
icepyx

Author name not set

Mar 16, 2026

GETTING STARTED

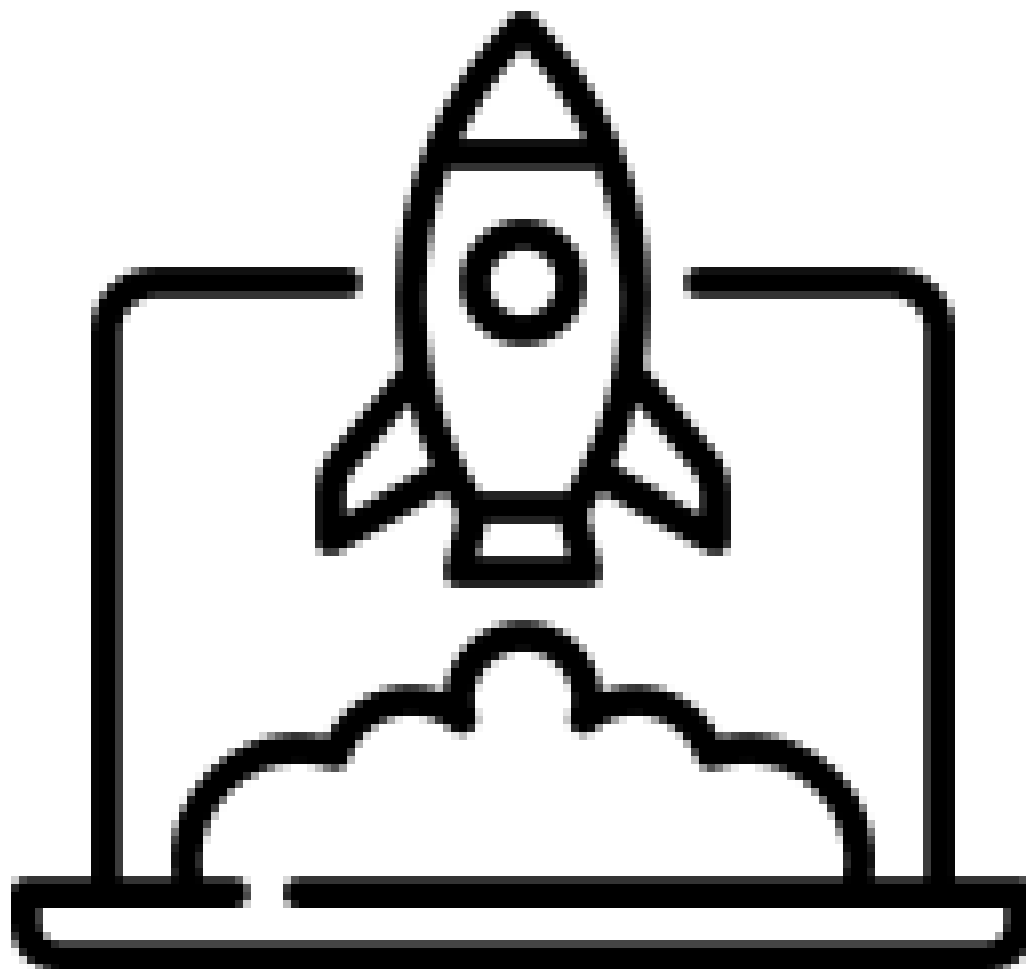
1	Origin and Purpose	7
2	Installation	9
3	Citing icepyx	11
4	Accessing ICESat-2 Data	13
5	Subsetting ICESat-2 Data	19
6	ICESat-2's Nested Variables	23
7	Visualizing ICESat-2 Elevations	33
8	Reading ICESat-2 Data in for Analysis	37
9	ICESat-2 AWS cloud data access	43
10	QUEST Example: Finding Argo and ICESat-2 data	47
11	icepyx Documentation (API)	53
12	icepyx-QUEST Documentation (API)	97
13	icepyx ChangeLog	101
14	Project Contributors	125
15	Contribution Guidelines	127
16	How to Contribute	131
17	Recognizing Contributions	135
18	icepyx Internals	137
19	QUEST Supported Datasets	139
20	icepyx Development Plan	141
21	Release Guide	143
22	Contributor Covenant Code of Conduct	147

23 ICESat-2 Resource Guide	151
24 Contact Us	153
25 Who is using icepyx?	155
26 icepyx Adoption	157
Bibliography	159
Python Module Index	163
Index	165

Python tools for obtaining and working with ICESat-2 data

Quick Links: [Installation](#) | [Citation](#) | [Examples](#) | [Source Code](#) | [Contact](#)

icepyx is both a software library and a community composed of ICESat-2 data users, developers, and the scientific community. We are working together to develop a shared library of resources - including existing resources, new code, tutorials, and use-cases/examples - that simplify the process of querying, obtaining, analyzing, and manipulating ICESat-2 datasets to enable scientific discovery. To further enhance data discovery, we have developed the QUEST module to facilitate querying of ICESat-2 data and complimentary Argo oceanographic data, with additional dataset support expected in the future.



Getting Started

New to ICESat-2 or icepyx? Learn how to install icepyx and use it to jumpstart your project today. Check out our gallery of examples, too!

[Installation Instructions](#)



User Guide

The user guide provides in-depth information on the tools and functionality available for obtaining and interacting with ICESat-2 data products.

Software Docs



QUEST

Query, Unify, Explore SpatioTemporal (QUEST) is a module that extends icepyx functionality to other datasets.

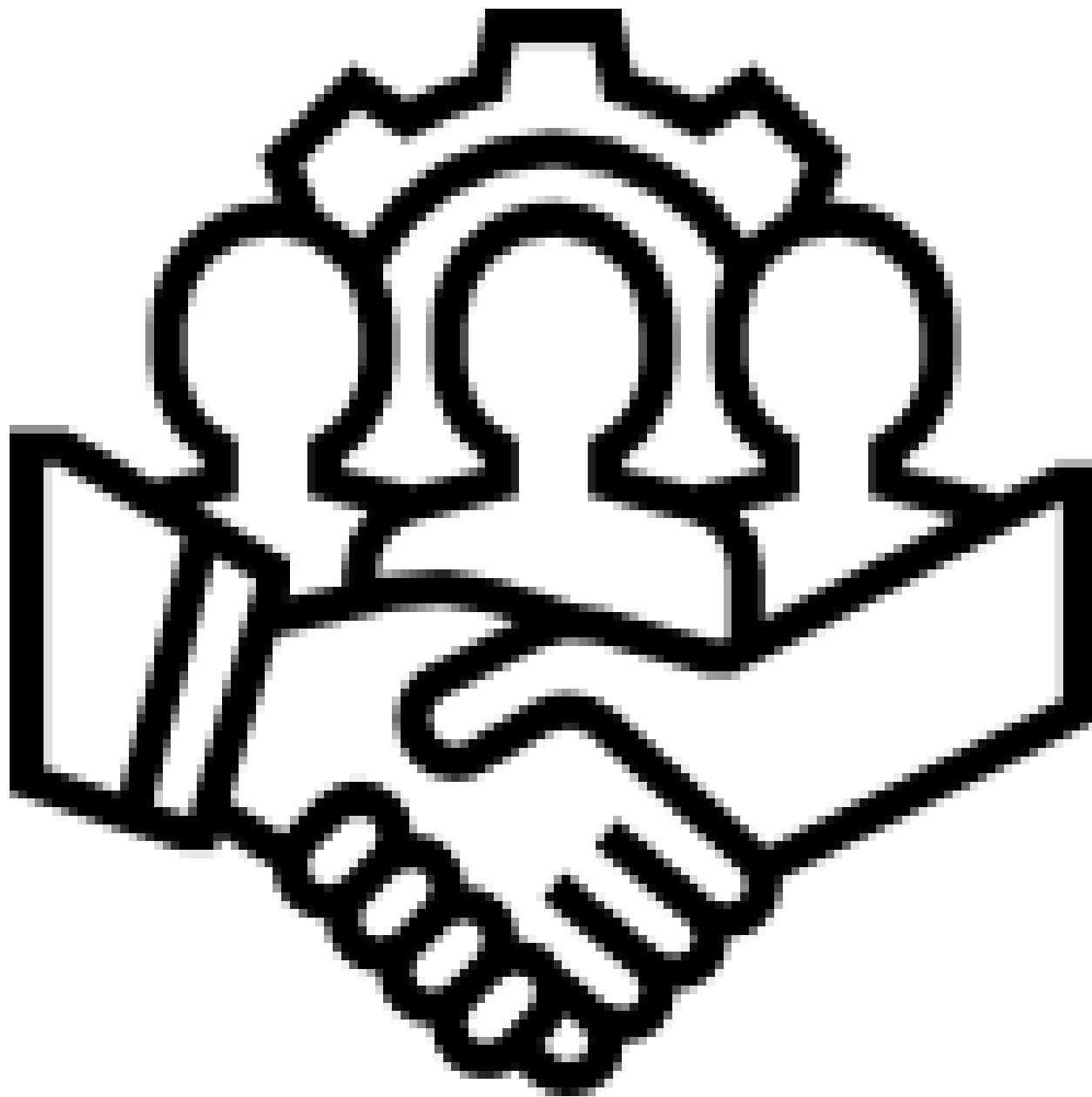
[Start your QUEST!](#)



Development Guide

Have an idea or an ancillary dataset to contribute to icepyx? Go here for information on best practices for developing and contributing to icepyx.

[Development Guide](#)

**Get in Touch**

icepyx is more than just software! We're a community of data producers, managers, and users who collaborate openly and share code and skills for every step along the entire data pipeline. Find resources for your questions here!

Get Involved!



ICE, CLOUD, AND LAND ELEVATION SATELLITE-2

ICESat-2 Resources

Curious about other tools for working with ICESat-2 data? Want to share your resource? Check out the amazing work already in progress!

ICESat-2 Resource Guide

ORIGIN AND PURPOSE

icepyx is both a software library and a community composed of ICESat-2 data users, developers, and the scientific community. We are working together to develop a shared library of resources - including existing resources, new code, tutorials, and use-cases/examples - that simplify the process of querying, obtaining, analyzing, and manipulating ICESat-2 and (via the QUEST module) relevant ancillary datasets to enable scientific discovery.

icepyx aims to provide a clearinghouse for code, functionality to improve interoperability, documentation, examples, and educational resources that tackle disciplinary research questions while minimizing the amount of repeated effort across groups utilizing similar datasets. icepyx also hopes to foster collaboration, open-science, and reproducible workflows by integrating and sharing resources.

Many of the underlying tools from which icepyx was developed began as Jupyter Notebooks developed for and during the cryosphere-themed ICESat-2 Hackweek at the University of Washington in June 2019 or as scripts written and used by the ICESat-2 Science Team members. Originally called `icesat2py`, the project combined and generalized these scripts into a unified framework, adding examples, documentation, and testing where necessary and making them accessible for everyone. icepyx is now a domain-agnostic, standalone software package and community (under the broader [icesat2py GitHub organization](#)) that continues to build functionality for obtaining and working with ICESat-2 data products locally and in the cloud. It also improves interoperability for ICESat-2 datasets with other open-source tools. Our [resources guide](#) provides additional information on both the foundational documents for icepyx and closely related libraries for working with ICESat-2 data.

INSTALLATION

2.1 Quickstart

The simplest (and recommended) way to install icepyx is by using the [mamba](#) package manager (or [conda](#), which can be used in place of any of the mamba commands shown here). The command below takes care of setting up a virtual environment and installs icepyx along with all the necessary dependencies:

```
mamba env create --name icepyx-env --channel conda-forge icepyx
```

To activate the virtual environment, you can do:

```
mamba activate icepyx-env
```

2.2 Using mamba

If you already have a virtual mamba/conda environment set up and activated, you can install the latest stable release of icepyx from [conda-forge](#) like so:

```
mamba install icepyx
```

To upgrade an installed version of icepyx to the latest stable release, do:

```
mamba update icepyx
```

2.3 Using pip

Alternatively, you can also install icepyx using [pip](#).

```
pip install icepyx
```

Windows users will need to first install [Fiona](#); please look at the instructions there. Windows users may consider installing Fiona using pipwin

```
pip install pipwin  
pipwin install Fiona
```

2.4 For the latest features

Currently, conda-forge and pip packages are generated with each tagged release. This means it is possible that these methods will not install the latest merged features of icepyx. In this case, icepyx is also available for use via the GitHub repository. The contents of the repository can be downloaded as a [zipped file](#) or cloned.

To use icepyx this way, fork this repo to your own account, then git clone the repo onto your system. To clone the repository:

```
git clone https://github.com/icesat2py/icepyx.git
```

Provided the location of the repo is part of your \$PYTHONPATH, you should simply be able to add `import icepyx` to your Python document. Alternatively, in a command line or terminal, navigate to the folder in your cloned repository containing `pyproject.toml` and run

```
pip install -e.
```

CITING ICEPYX

3.1 icepyx

This community and software is developed with the goal of supporting science applications. Thus, our contributors (including those who have developed the packages used within icepyx) and maintainers justify their efforts and demonstrate the impact of their work through citations.

If you have used icepyx in your work, please consider citing our library. We encourage you to use a version-specific citation and DOI (available from [Zenodo](#)) to improve reproducibility and let users know the state of the software at the time of your analysis.

A non-versioned citation of icepyx:

The icepyx Developers, (2023). icepyx: Python tools for obtaining and working with ICESat-2 data. Zenodo. <https://doi.org/10.5281/zenodo.7729175>

A bibtex version for users working in Latex:

```
@Misc{icepyx,  
author =    {{The icepyx Developers}},  
organization = {icesat2py},  
title =    {{icepyx: Python} tools for obtaining and working with {ICESat-2} data},  
year =     {2023},  
doi = "https://doi.org/10.5281/zenodo.7729175",  
publisher = {Zenodo},  
url = "https://github.com/icesat2py/icepyx",  
}
```

For more information on the “icepyx Developers”, please see our [Attribution Guidelines](#). See our docs for a [full list of contributors](#) and their contribution types.

3.2 icepyx Dependencies

If you have used one of the included packages to extend your data analysis capabilities within icepyx, please consider additionally citing that work, because it represents an independent software contribution to the open-source community. SciPy provides a [helpful resource](#) for citing packages within the SciPy ecosystem (including Matplotlib, NumPy, pandas, and SciPy). Links to citation information for other commonly used packages are below.

- [fiona](#)
- [GeoPandas](#)
- [Pangeo](#)
- [shapely](#)

- xarray

3.3 ICESat-2 Data

ICESat-2 data citation depends on the exact dataset used. Citation information for each data product can be found through the [NSIDC website](#).

ACCESSING ICESAT-2 DATA

This notebook ([download](#)) illustrates the use of `icepyx` for programmatic ICESat-2 data query and download from the NASA National Snow and Ice Data Center Distributed Active Archive Center (NASA NSIDC DAAC). A complimentary notebook demonstrates in greater detail the [subsetting](#) options available when ordering data.

Import packages, including `icepyx`

```
import icepyx as ipx
import os
import shutil
%matplotlib inline
```

4.1 Quick-Start Guide

The entire process of getting ICESat-2 data (from query to download) can ultimately be accomplished in two minimal lines of code:

```
region_a = ipx.Query(short_name, spatial_extent, date_range)
region_a.download_granules(path)
```

where the function inputs are described in more detail below.

The rest of this notebook explains the required inputs used above, optional inputs not available in the minimal example, and the other data search and visualization tools built in to `icepyx` that make it easier for the user to find, explore, and download ICESat-2 data programmatically from the NASA NSIDC DAAC. The detailed steps outlined and the methods showcased below are meant to give the user more control over the data they find and download (including options to order/download only the relevant portions of a data granule), some of which are called using default values behind the scenes if the user simply skips to the `download_granules` step.

4.2 Key Steps for Programmatic Data Access

There are several key steps for accessing data from the [NASA Harmony API](#):

1. Define your parameters (spatial, temporal, dataset, etc.)
2. Query the NASA Harmony API to find out more information about the dataset
3. Define additional parameters (e.g. subsetting/customization options)
4. Order your data
5. Download your data

`icepyx` streamlines this process into a minimal number of lines of code.

4.2.1 Login to NASA Earthdata

When downloading data from the NASA NSIDC DAAC, all users must login using a valid (free) Earthdata account. The process of authenticating is handled by icepyx by creating and handling the required authentication to interface with the data at the DAAC (including ordering and download). Authentication is completed as login-protected features are accessed. In order to allow icepyx to login for us, we still have to ensure that we have made our Earthdata credentials available for icepyx to find.

There are multiple ways to provide your Earthdata credentials via icepyx. Behind the scenes, icepyx is using the [earthaccess library](#). The [earthaccess documentation](#) automatically tries three primary mechanisms for logging in, all of which are supported by icepyx:

- through an interactive, in-notebook login (used below); passwords are not shown plain text with this option
- with `EARTHDATA_USERNAME` and `EARTHDATA_PASSWORD` environment variables (these are the same as the ones you might have set for icepyx previously)
- with stored credentials in a `.netrc` file (not recommended for security reasons)

4.2.2 Define desired search parameters

There are three required inputs, depending on how you want to search for data. Two are required in all cases:

- `short_name` = the data product of interest, known as its “short name”. See [ICESat-2 Products](#) for a list of the available data products. Note that Quick Looks products are not currently available.
- `spatial_extent` = a region of interest to search within. This can be entered as a bounding box, polygon vertex coordinate pairs, or a polygon geospatial file (ESRI Shapefile (.zip or .shz), kml (.kml), and GeoJSON (*.json or .geojson) are currently supported).
 - bounding box: Given in decimal degrees for the lower left longitude, lower left latitude, upper right longitude, and upper right latitude.
 - polygon vertices: Given as longitude, latitude coordinate pairs of decimal degrees with the last entry a repeat of the first.
 - polygon file: A string containing the full file path and name.

NOTE: The input keyword for `short_name` was updated in the code from `dataset` to `product` to match common usage. This should not affect users providing positional inputs as demonstrated in this tutorial.

NOTE: You can submit at most one bounding box or a list of lonlat polygon coordinates per object instance. See the [CMR API documentation](#) for geospatial polygon file requirements.

Then, for all non-gridded products, you must include AT LEAST one of the following inputs (temporal or orbital constraints):

- `date_range` = the date range for which you would like to search for results. The following formats are accepted:
 - A list of two ‘YYYY-MM-DD’ strings separated by a comma
 - A list of two ‘YYYY-DOY’ strings separated by a comma
 - A list of two `datetime.date` or `datetime.datetime` objects
 - Dict with the following keys:
 - * `start_date`: start date; type can be `datetime.datetime`, `datetime.date`, or strings (format ‘YYYY-MM-DD’ or ‘YYYY-DOY’)
 - * `end_date`: end date; type can be `datetime.datetime`, `datetime.date`, or strings (format ‘YYYY-MM-DD’ or ‘YYYY-DOY’)

- `cycles` = Which orbital cycle to use, input as a numerical string or a list of strings. If no input is given, this value defaults to all available cycles within the search parameters. An orbital cycle refers to the 91-day repeat period of the ICESat-2 orbit.
- `tracks` = Which [Reference Ground Track \(RGT\)](#) to use, input as a numerical string or a list of strings. If no input is given, this value defaults to all available RGTs within the spatial and temporal search parameters.

Below are examples of each type of spatial extent and temporal input and an example using orbital parameters. Please choose and run only one of the input option cells to set your spatial and temporal parameters.

```
# bounding box
short_name = 'ATL06'
spatial_extent = [-55, 68, -48, 71]
date_range = ['2019-02-20', '2019-02-28']
```

```
# polygon vertices (here equivalent to the bounding box, above)
short_name = 'ATL06'
spatial_extent = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]
date_range = ['2019-02-20', '2019-02-28']
```

```
# bounding box with 'YYYY-DOY' date range (equivalent to 'YYYY-MM-DD' date ranges above)
short_name = 'ATL06'
spatial_extent = [-55, 68, -48, 71]
date_range = ['2019-051', '2019-059']
```

```
# polygon vertices with datetime.datetime date ranges
import datetime as dt

start_dt = dt.datetime(2019, 2, 20, 0, 10, 0)
end_dt = dt.datetime(2019, 2, 28, 14, 45, 30)
short_name = 'ATL06'
spatial_extent = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]
date_range = [start_dt, end_dt]
```

```
# bounding box with dict containing date ranges
short_name = 'ATL06'
spatial_extent = [-55, 68, -48, 71]
date_range = {"start_date": start_dt, "end_date": '2019-02-28'}
```

```
# #polygon geospatial file (metadata match but no subset match)
# short_name = 'ATL06'
# spatial_extent = './supporting_files/data-access_PineIsland/glims_polygons.kml'
# date_range = ['2019-02-22', '2019-02-28']

# #polygon geospatial file (subset and metadata match)
# short_name = 'ATL06'
# spatial_extent = './supporting_files/data-access_PineIsland/glims_polygons.shp'
# date_range = ['2019-10-01', '2019-10-05']

#polygon geospatial file (same area as other examples; subset and metadata match)
short_name = 'ATL06'
spatial_extent = './supporting_files/simple_test_poly.gpkg'
date_range = ['2019-10-01', '2019-10-05']
```

4.2.3 Create the data object using inputs

```
region_a = ipx.Query(short_name, spatial_extent, date_range)
```

```
# using orbital parameters with one of the above data products + spatial parameters
region_a = ipx.Query(short_name, spatial_extent,
    cycles=['03','04','05','06','07'], tracks=['0849','0902'])

print(region_a.product)
print(region_a.product_version)
print(region_a.cycles)
print(region_a.tracks)
```

These properties include visualization of the spatial extent on a map. The style of map you will see depends on whether or not you have a certain library, `geoviews`, installed. Under the hood, this is because the `proj` library must be installed with `conda` (it is not available from `PyPI`) to support some `geoviews` dependencies. With `geoviews`, this plotting function returns an interactive map. Otherwise, your spatial extent will plot on a static map using `matplotlib`.

```
# print(region_a.spatial_extent)
region_a.visualize_spatial_extent()
```

Formatted parameters and function calls allow us to see the the properties of the data object we have created.

```
print(region_a.product)
print(region_a.temporal) # .dates, .start_time, .end_time can also be used for a piece_
↳of this information
# print(region_a.dates)
# print(region_a.start_time)
# print(region_a.end_time)
print(region_a.cycles)
print(region_a.tracks)
print(region_a.product_version)
region_a.visualize_spatial_extent()
```

There are also several optional inputs to allow the user finer control over their search. Start and end time are only valid inputs on a temporally limited search, and they are ignored if your `date_range` input is a `datetime.datetime` object.

- `start_time` = start time to search for data on the start date. If no input is given, this defaults to 00:00:00.
- `end_time` = end time for the end date of the temporal search parameter. If no input is given, this defaults to 23:59:59.

Times must be input as 'HH:mm:ss' strings or `datetime.time` objects.

- `version` = What version of the data product to use, input as a numerical string. If no input is given, this value defaults to the most recent version of the product specified in `short_name`.

NOTE Version 002 is used as an example in the below cell. However, using it will cause 'no results' errors in granule ordering for some search parameters. These issues have been resolved in later versions of the data products, so it is best to use the most recent version where possible. Similarly, if you try to order/download a retired version no longer accessible at the NASA NSIDC DAAC, you will get a "no data matched your request" error. Thus, you will need to update the version associated with `region_a` and rerun the next cell for the rest of this notebook to run.

```
region_a = ipx.Query(short_name, spatial_extent, date_range, \
    start_time='03:30:00', end_time='21:30:00', version='002')
```

(continues on next page)

(continued from previous page)

```
print(region_a.product)
print(region_a.dates)
print(region_a.product_version)
print(region_a.spatial)
print(region_a.temporal)
```

Alternatively, you can also just create the query object without creating named variables first:

```
region_a = ipx.Query('ATL06', [-55, 68, -48, 71],
    ['2019-02-01', '2019-02-28'], start_time='00:00:00', end_time='23:59:59')

print(region_a.product)
print(region_a.dates)
print(region_a.product_version)
print(region_a.spatial)
print(region_a.temporal)
```

4.2.4 More information about your query object

In addition to viewing the stored object information shown above (e.g., product short name, start and end date and time, version), we can also request summary information about the data product itself or confirm that we have manually specified the latest version.

```
region_a.product_summary_info()
print(region_a.latest_version())
```

If the summary does not provide all of the information you are looking for, or you would like to see information for previous versions of the data product, all available metadata for the collection product is available in a readable format.

```
region_a.product_all_info()
```

4.2.5 Querying a data product

In order to search the product collection for available data granules, we need to build our search parameters. This is done automatically behind the scenes when you run `region_a.avail_granules()`, but you can also build and view them by calling `region_a.CMRparams`. These are formatted as a dictionary of key-value pairs according to the [CMR documentation](#).

```
#build and view the parameters that will be submitted in our query
region_a.CMRparams
```

Now that our parameter dictionary is constructed, we can search the CMR database for the available granules. Granules returned by the CMR metadata search are automatically stored within the data object. The search completed at this level relies completely on the granules' metadata. As a result, some (and in rare cases all) of the granules returned may not actually contain data in your specified region, particularly if the region is small or located near the boundaries of a given granule. If this is the case, the subsetter will not return any data when you actually place the order. A warning message will be included in the `order.status` output for each granule to which this applies.

```
#search for available granules and provide basic summary info about them
region_a.avail_granules()
```

```
#get a list of granule IDs for the available granules
region_a.avail_granules(ids=True)
```

4.2.6 Subsetting

For a deeper dive into subsetting, please see our [Subsetting Tutorial Notebook](#), which covers subsetting in more detail.

Subsetting utilizes the NASA Harmony subsetting service to spatially and temporally extract the data you are interested in. The advantages of using Harmony include:

- easily reproducible downloads, particularly when coupled with an icepyx query object
- smaller file size, meaning faster downloads and less storage required
- easily order more data with the same or similar search parameters
- quickly move to analysis and navigate your dataset

4.2.7 Place the order

Then, we can send the order to Harmony using the `order_granules` function. Information about the granules ordered and their status will be printed automatically.

```
order = region_a.order_granules()
# region_a.order_granules(subset=False)
order
```

```
order.status()
```

4.2.8 Download the order

Finally, we can download our order to a specified directory (which needs to have a full path but doesn't have to point to an existing directory) and the download status will be printed as the program runs.

```
path = './download'
region_a.download_granules(path)
```

Credits

- original notebook by: Jessica Scheick
- notebook contributors: Amy Steiker, Tyler Sutterley, and Theresa Andersen
- source material: [NSIDC Data Access Notebook](#) by Amy Steiker and Bruce Wallin and [2020 Hackweek Data Access Notebook](#) by Jessica Scheick and Amy Steiker

SUBSETTING ICESAT-2 DATA

This notebook (download) illustrates the use of icepyx for subsetting ICESat-2 data ordered through the NASA NSIDC DAAC. We'll show how to find out what subsetting options are available and how to specify the subsetting options for your order.

For more information on using icepyx to find, order, and download data, see our complimentary [ICESat-2 Data Access Notebook](#).

Questions? Be sure to check out the FAQs throughout this notebook, indicated as italic headings.

5.1 What is SUBSETTING anyway?

Anyone who's worked with geospatial data has probably encountered subsetting. Typically, we search for data wherever it is stored and download the chunks (granules, scenes, passes, swaths, etc.) that contain something we are interested in. Then, we have to extract from each chunk the pieces we actually want to analyze. Those pieces might be geospatial (i.e., an area of interest) or temporal (i.e., certain months of a time series). This process of extracting the data we are going to use is called subsetting.

In the case of ICESat-2 data from the NASA NSIDC DAAC, we can do this subsetting step on the data prior to download, reducing our number of data processing steps and resulting in smaller, faster downloads and storage.

Import packages, including icepyx

```
import icepyx as ipx

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

import h5py
import os, json
from pprint import pprint
```

Create a query object and log in to Earthdata

For this example, we'll be working with a sea ice product (ATL07) for an area along West Greenland (Disko Bay).

```
region_a = ipx.Query('ATL07', [-55, 68, -48, 71], ['2019-02-22', '2019-02-28'], \
                    start_time='00:00:00', end_time='23:59:59')
```

5.2 Discover customization options

You can see the customization options for a given product by calling `show_custom_options()`. The options are presented as a dictionary of key-value pairs. Three options are currently available:

- `bboxSubset`: bounding box subsetting
- `shapeSubset`: polygon subsetting
- `temporalSubset`: temporal subsetting

`outputFormats` indicates that only HDF5 is a supported output format. `variableSubset`, `concatenate`, and `reproject` are currently unavailable (set to false).

Note that these subsetting options are available for all L2-L3A products. Subsetting options are not currently supported for L3B or Quick Looks products.

```
region_a.show_custom_options()
```

By default, spatial and temporal subsetting based on your initial inputs is applied to your order. This will be true no matter if you use the `order_granules()` function or the `download_granules()` function (which calls `.order_granules_under()` the hood if you have not already placed your order). If you don't want your order to be spatially subset, you can use the `subset=False` argument in either `.order_granules()` or `.download_granules()`. Additional subsetting options must be specified as keyword arguments to the order/download functions.

5.2.1 Why do I have to provide spatial bounds to icepyx even if I don't use them to subset my data order?

Because they're still needed for the granule level search. Spatial inputs are usually required for any data search, on any platform, even if your search parameters cover the entire globe.

The spatial information you provide is used to search the data repository and determine which granules might contain data over your area of interest. When you use that spatial information for subsetting, it's actually asking the NASA Harmony subsetter to extract the appropriate data from each granule. Thus, even if you set `subset=False` and download entire granules, you still need to provide some inputs on what geographic area you'd like data for.

5.3 About data variables in a query object

A given ICESat-2 product may have over 200 variable + path combinations. `icepyx` includes a custom `Variables` module that is "aware" of the ATLAS sensor and how the ICESat-2 data products are stored. The [ICESat-2 Data Variables Example](#) provides a detailed set of examples on how to use `icepyx`'s built in `Variables` module.

While variable subsetting is not supported for ICESat-2 data, you can refer to the aforementioned Jupyter Notebook to learn how to interact with ICESat-2 variables after requesting your data.

5.4 Why not just download all the data and subset locally? What if I need more granules?

Taking advantage of the NASA Harmony subsetting service is a great way to reduce your download size and thus your download time and the amount of storage required, especially if you're storing your data locally during analysis. By downloading your data using `icepyx`, it is easy to go back and get additional data with the same, similar, or different parameters. Related tools (e.g., `captoolkit`) will let you easily merge files if you're uncomfortable merging them during read-in for processing.

```
short_name = 'ATL06'
spatial_extent = './supporting_files/simple_test_poly.gpkg'
date_range = ['2019-10-01', '2019-10-05']
```

```
region_a = ipx.Query(short_name, spatial_extent
,
  cycles=['03','04','05','06'], tracks=['0849','0902'])

print(region_a.product)
print(region_a.product_version)
print(region_a.cycles)
print(region_a.tracks)
print(region_a.spatial_extent)
```

```
region_a.visualize_spatial_extent()
```

We can also print a list of available granules for our query:

```
region_a.avail_granules(cloud=True)
```

5.5 Applying granule subsetting to your order and downloading the results

```
order = region_a.order_granules(subset=True)
order
```

5.5.1 Checking an order status

```
order.status()
```

5.5.2 Downloading subsetting granules

```
files = order.download_granules("./data")
```

5.5.3 Why does the subsetter say no matching data was found?

Sometimes, granules (“files”) returned in our initial search end up not containing any data in our specified area of interest. This is because the initial search is completed using summary metadata for a granule. You’ve likely encountered this before when viewing available imagery online: your spatial search turns up a bunch of images with only a few border or corner pixels, maybe even in no data regions, in your area of interest. Thus, when you go to extract the data from the area you want (i.e., spatially subset it), you don’t get any usable data from that image.

5.6 Handling large orders

By default, the Harmony subsetter will only process the first 300 granules for large orders, placing them into a “pre-viewing” status. This allows users to check that results look correct. Once the job has completed its preview, which includes the first 100 granules, then we can resume the order if we are satisfied that our request is correct. The following guidance is commented out by default but can be uncommented to test this large order behavior.

```
# short_name = 'ATL06'  
# spatial_extent = './supporting_files/simple_test_poly.gpkg'  
# date_range = ['2018-10-01', '2020-02-05']  
  
# region_a = ipx.Query(short_name, spatial_extent, date_range)  
  
# order = region_a.order_granules(subset=True)  
# order
```

This order includes 311 input granules, and therefore it is automatically placed into a previewing state. We can inspect the status of this order and wait until it moves to a “paused” state, once the initial 100 granules are complete.

```
# order.status()
```

If we are satisfied with the order, then we can resume processing:

```
# order.resume()  
# order
```

5.7 Working with the downloaded data

Now that the subsetting files have been downloaded, we can now work with them using the `icepyx Read` class. See the [Reading ICESat-2 Data in for Analysis](#) notebook for more information.

5.7.1 Credits

- notebook contributors: Zheng Liu, Jessica Scheick, Amy Steiker, and Theresa Andersen
- some source material: [NSIDC Data Access Notebook](#) by Amy Steiker and Bruce Wallin

ICESAT-2'S NESTED VARIABLES

This notebook ([download](#)) illustrates the use of `icepyx` for managing lists of available and wanted ICESat-2 data variables. The two use cases for variable management within your workflow are:

1. During the data access process via remote cloud access.
2. When reading in data to a Python object (whether from local files or the cloud).

A given ICESat-2 product may have over 200 variable + path combinations. `icepyx` includes a custom `Variables` module that is “aware” of the ATLAS sensor and how the ICESat-2 data products are stored. The module can be accessed independently and can also be accessed as a component of a `Query` object or `Read` object.

This notebook illustrates in detail how the `Variables` module behaves. We use the module independently and also show how powerful it is directly in the `icepyx` workflow using a `Query` data access example. Module usage using `Query` is analogous through an `icepyx` ICESat-2 `Read` object. More detailed example workflows specifically for the `query` and `read` tools within `icepyx` are available as separate Jupyter Notebooks.

Questions? Be sure to check out the FAQs throughout this notebook, indicated as italic headings.

6.1 Why do ICESat-2 products need a custom variable manager?

It can be confusing and cumbersome to comb through the 200+ variable and path combinations contained in ICESat-2 data products. An `hdf5` file is built like a folder with files in it. Opening an ICESat-2 file can be like opening a new folder with over 200 files in it and manually searching for only ones you want!

The `icepyx` `Variables` module makes it easier for users to quickly find and extract the specific variables they would like to work with across multiple beams, keywords, and variables and provides reader-friendly formatting to browse variables. A future development goal for `icepyx` includes developing an interactive widget to further improve the user experience. For data read-in, additional tools are available to target specific beam characteristics (e.g. strong versus weak beams).

Import packages, including `icepyx`

```
import icepyx as ipx
from pprint import pprint
```

6.2 Creating or Accessing ICESat-2 Variables

There are three ways to create or access an ICESat-2 `Variables` object in `icepyx`:

1. Access via the `.variables` property of a `Query` object
2. Access via the `.variables` property of a `Read` object
3. Create a stand-alone ICESat-2 `Variables` object using a local file, cloud file, or a product name

An example of each of these is shown below.

6.2.1 1. Access Variables via the .variables property of a Query object

```
region_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-22', '2019-02-28'], \
                    start_time='00:00:00', end_time='23:59:59')
```

```
# Showing the variable paths, prior icepyx versions utilized order_vars()
region_a.variables.avail()
```

6.2.2 2. Access via the .variables property of a Read object

```
path_root = '/full/path/to/your/data/'
reader = ipx.Read(path_root)
```

```
# Accessing Variables
reader.variables.parse_var_list(region_a.variables.avail())
```

6.2.3 3. Create a stand-alone Variables object

You can also generate an independent Variables object. This can be done using either:

1. The filepath to a local or cloud file you'd like a variables list for
2. The product name (and optionally version) of an ICESat-2 product

Note: Cloud data access requires a valid Earthdata login; you will be prompted to log in if you are not already authenticated.

Create a variables object from a filepath:

```
filepath = '/full/path/to/your/data.h5'
v = ipx.Variables(path=filepath)
```

```
# v.avail()
```

Create a variables object from a product. The version argument is optional.

```
v = ipx.Variables(product='ATL03')
```

```
v.avail()
```

```
v = ipx.Variables(product='ATL03', version='006')
```

```
v.avail()
```

Now that you know how to create or access Variables, the remainder of this notebook showcases the functions available for building and modifying variables lists. Remember, the example shown below uses a Query object, but the same methods are available if you are using a Read object or a Variables object.

6.3 Interacting with ICESat-2 Data Variables

Each variables instance (which is actually an associated Variables class object) contains two variable list attributes. One is the list of possible or available variables (`avail` attribute) and is unmutable, or unchangeable, as it is based on the input product specifications or files. The other is the list of variables you'd like to actually have (in your downloaded file or data object) from all the potential options (`wanted` attribute) and is updateable.

Thus, your `avail` list depends on your data source and whether you are accessing or reading data, while your `wanted` list may change for each analysis you are working on or depending on what variables you want to see.

The variables parameter has methods to:

- get a list of all available variables from the file (`avail()` method).
- append new variables to the wanted list (`append()` method).
- remove variables from the wanted list (`remove()` method).

We'll showcase the use of all of these methods and attributes below using an `icepyx.Query` object. Usage is identical in the case of an `icepyx.Read` object. More detailed example workflows specifically for the `query` and `read` tools within `icepyx` are available as separate Jupyter Notebooks.

Create a query object and log in to Earthdata

For this example, we'll be working with a land ice product (ATL06) for an area along West Greenland (Disko Bay). A second option for an atmospheric product (ATL09) that uses profiles instead of the ground track (`gt`) categorization is also provided.

```
region_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-22', '2019-02-28'], \
                    start_time='00:00:00', end_time='23:59:59')
```

```
## Uncomment and run the code in this cell to use the second variable subsetting suite,
↳ of examples,
## with the beam specifier containing "profile" instead of "gt#1"

# region_a = ipx.Query('ATL09', [-55, 68, -48, 71], ['2019-02-22', '2019-02-28'], \
#                 start_time='00:00:00', end_time='23:59:59')
```

6.3.1 ICESat-2 data variables

ICESat-2 data is natively stored in a nested file format called `hdf5`. Much like a directory-file system on a computer, each variable (file) has a unique path through the hierarchy (directories) within the file. Thus, some variables (e.g. `'latitude'`, `'longitude'`) have multiple paths (one for each of the six beams in most products).

Determine what variables are available

`region_a.variables.avail` will return a list of all valid path+variable strings.

```
region_a.variables.avail()
```

To increase readability, you can use built in functions to show the 200+ variable + path combinations as a dictionary where the keys are variable names and the values are the paths to that variable. `region_a.variables.parse_var_list(region_a.variables.avail())` will return a dictionary of variable:paths key:value pairs.

```
region_a.variables.parse_var_list(region_a.variables.avail())
```

By passing the boolean `options=True` to the `avail` method, you can obtain lists of unique possible variable inputs (`var_list` inputs) and path subdirectory inputs (`keyword_list` and `beam_list` inputs) for your data product. These can be helpful for building your wanted variable list.

```
region_a.variables.avail(options=True)
```

Remember

You can run these same methods no matter how you created or accessed your ICESat-2 Variables. So the methods in this section could be equivalently be accessed using a Read object, or by directly accessing a file on your computer:

```
# Using a Read object
reader.variables.avail()
reader.variables.parse_var_list(reader.variables.avail())
reader.variables.avail(options=True)

# Using a file on your computer
v = Variables(path='/my/file.h5')
v.avail()
v.parse_var_list(v.avail())
v.avail(options=True)
```

6.3.2 Building your wanted variable list

Now that you know which variables and path components are available, you need to build a list of the ones you'd like included. There are several options for generating your initial list as well as modifying it, giving the user complete control.

The options for building your initial list are:

1. Use a default list for the product (not yet fully implemented across all products. Have a default variable list for your field/product? Submit a pull request or post it as an issue on [GitHub!](#))
2. Provide a list of variable names
3. Provide a list of profiles/beams or other path keywords, where “keywords” are simply the unique subdirectory names contained in the full variable paths of the product. A full list of available keywords for the product is displayed in the error message upon entering `keyword_list=['']` into the `append` function (see below for an example) or by running `region_a.variables.avail(options=True)`, as above.

Note: all products have a short list of “mandatory” variables/paths (containing spacecraft orientation and time information needed to convert the data’s `delta_time` to a readable `datetime`) that are automatically added to any built list. If you have any recommendations for other variables that should always be included (e.g. uncertainty information), please let us know!

Examples of using each method to build and modify your wanted variable list are below.

```
region_a.variables.wanted
```

```
region_a.variables.append(defaults=True)
pprint(region_a.variables.wanted)
```

The keywords available for this product are shown in the error message upon entering a blank `keyword_list`, as seen in the next cell.

```
region_a.variables.append(keyword_list=[' '])
```

6.3.3 Modifying your wanted variable list

Generating and modifying your variable request list, which is stored in `region_a.variables.wanted`, is controlled by the `append` and `remove` functions that operate on `region_a.variables.wanted`. The input options to `append` are as follows (the full documentation for this function can be found by executing `help(region_a.variables.append)`).

- `defaults` (default `False`) - include the default variable list for your product (not yet fully implemented for all products; please submit your default variable list for inclusion!)
- `var_list` (default `None`) - list of variables (entered as strings)
- `beam_list` (default `None`) - list of beams/profiles (entered as strings)
- `keyword_list` (default `None`) - list of keywords (entered as strings); use `keyword_list=[' ']` to obtain a list of available keywords

Similarly, the options for `remove` are:

- `all` (default `False`) - reset `region_a.variables.wanted` to `None`
- `var_list` (as above)
- `beam_list` (as above)
- `keyword_list` (as above)

```
region_a.variables.remove(all=True)
pprint(region_a.variables.wanted)
```

6.3.4 Examples (Overview)

Below are a series of examples to show how you can use `append` and `remove` to modify your wanted variable list. For clarity, `region_a.variables.wanted` is cleared at the start of many examples. However, multiple `append` and `remove` commands can be called in succession to build your wanted variable list (see Examples 3+).

There are two example tracks. The first is for land ice (ATL06) data that is separated into beams. The second is for atmospheric data (ATL09) that is separated into profiles. Both example tracks showcase the same functionality and are provided for users of both data types.

6.3.5 Example Track 1 (Land Ice - run with ATL06 dataset)

Example 1.1: choose variables

Add all latitude and longitude variables across all six beam groups. Note that the additional required variables for time and spacecraft orientation are included by default.

```
region_a.variables.append(var_list=['latitude', 'longitude'])
pprint(region_a.variables.wanted)
```

Example 1.2: specify beams and variable

Add latitude for only gt1l and gt2l

```
region_a.variables.remove(all=True)
pprint(region_a.variables.wanted)
```

```
var_dict = region_a.variables.append(beam_list=['gt1l', 'gt2l'], var_list=['latitude'])
pprint(region_a.variables.wanted)
```

Example 1.3: add/remove selected beams+variables

Add latitude for gt3l and remove it for gt2l

```
region_a.variables.append(beam_list=['gt3l'], var_list=['latitude'])
region_a.variables.remove(beam_list=['gt2l'], var_list=['latitude'])
pprint(region_a.variables.wanted)
```

Example 1.4: keyword_list

Add latitude and longitude for all beams and with keyword land_ice_segments

```
region_a.variables.append(var_list=['latitude', 'longitude'], keyword_list=['land_ice_
↳ segments'])
pprint(region_a.variables.wanted)
```

Example 1.5: target a specific variable + path

Remove gt1r/land_ice_segments/longitude (but keep gt1r/land_ice_segments/latitude)

```
region_a.variables.remove(beam_list=['gt1r'], var_list=['longitude'], keyword_list=[
↳ 'land_ice_segments'])
pprint(region_a.variables.wanted)
```

Example 1.6: add variables not specific to beams/profiles

Add rgt under orbit_info.

```
region_a.variables.append(keyword_list=['orbit_info'], var_list=['rgt'])
pprint(region_a.variables.wanted)
```

Example 1.7: add all variables+paths of a group

In addition to adding specific variables and paths, we can filter all variables with a specific keyword as well. Here, we add all variables under orbit_info. Note that paths already in region_a.order_vars.wanted, such as 'orbit_info/rgt', are not duplicated.

```
region_a.variables.append(keyword_list=['orbit_info'])
pprint(region_a.variables.wanted)
```

Example 1.8: add all possible values for variables+paths

Append all longitude paths and all variables/paths with keyword `land_ice_segments`.

Similarly to what is shown in Example 4, if you submit only one `append` call as `region_a.variables.append(var_list=['longitude'], keyword_list=['land_ice_segments'])` rather than the two `append` calls shown below, you will only add the variable `longitude` and only paths containing `land_ice_segments`, not ALL paths for `longitude` and ANY variables with `land_ice_segments` in their path.

```
region_a.variables.append(var_list=['longitude'])
region_a.variables.append(keyword_list=['land_ice_segments'])
pprint(region_a.variables.wanted)
```

Example 1.9: remove all variables+paths associated with a beam

Remove all paths for `gt1l` and `gt3r`

```
region_a.variables.remove(beam_list=['gt1l', 'gt3r'])
pprint(region_a.variables.wanted)
```

Example 1.10: generate a default list for the rest of the tutorial

Generate a reasonable variable list prior to download

```
region_a.variables.remove(all=True)
region_a.variables.append(defaults=True)
pprint(region_a.variables.wanted)
```

6.3.6 Example Track 2 (Atmosphere - run with ATL09 dataset commented out at the start of the notebook)**Example 2.1: choose variables**

Add all latitude and longitude variables

```
region_a.variables.append(var_list=['latitude', 'longitude'])
pprint(region_a.variables.wanted)
```

Example 2.2: specify beams/profiles and variable

Add latitude for only `profile_1` and `profile_2`

```
region_a.variables.remove(all=True)
pprint(region_a.variables.wanted)
```

```
var_dict = region_a.variables.append(beam_list=['profile_1', 'profile_2'], var_list=[
↪ 'latitude'])
pprint(region_a.variables.wanted)
```

Example 2.3: add/remove selected beams+variables

Add latitude for profile_3 and remove it for profile_2

```
region_a.variables.append(beam_list=['profile_3'],var_list=['latitude'])
region_a.variables.remove(beam_list=['profile_2'], var_list=['latitude'])
pprint(region_a.variables.wanted)
```

Example 2.4: keyword_list

Add latitude for all profiles and with keyword low_rate

```
region_a.variables.append(var_list=['latitude'],keyword_list=['low_rate'])
pprint(region_a.variables.wanted)
```

Example 2.5: target a specific variable + path

Remove 'profile_1/high_rate/latitude' (but keep 'profile_3/high_rate/latitude')

```
region_a.variables.remove(beam_list=['profile_1'], var_list=['latitude'], keyword_list=[
↪ 'high_rate'])
pprint(region_a.variables.wanted)
```

Example 2.6: add variables not specific to beams/profiles

Add rgt under orbit_info.

```
region_a.variables.append(keyword_list=['orbit_info'],var_list=['rgt'])
pprint(region_a.variables.wanted)
```

Example 2.7: add all variables+paths of a group

In addition to adding specific variables and paths, we can filter all variables with a specific keyword as well. Here, we add all variables under orbit_info. Note that paths already in region_a.variables.wanted, such as 'orbit_info/rgt', are not duplicated.

```
region_a.variables.append(keyword_list=['orbit_info'])
pprint(region_a.variables.wanted)
```

Example 2.8: add all possible values for variables+paths

Append all longitude paths and all variables/paths with keyword high_rate. Similarly to what is shown in Example 4, if you submit only one append call as region_a.order_vars.append(var_list=['longitude'], keyword_list=['high_rate']) rather than the two append calls shown below, you will only add the variable longitude and only paths containing high_rate, not ALL paths for longitude and ANY variables with high_rate in their path.

```
region_a.variables.append(var_list=['longitude'])
region_a.variables.append(keyword_list=['high_rate'])
pprint(region_a.variables.wanted)
```

Example 2.9: remove all variables+paths associated with a profile

Remove all paths for profile_1 and profile_3

```
region_a.variables.remove(beam_list=['profile_1','profile_3'])
pprint(region_a.variables.wanted)
```

Example 2.10: generate a default list for the rest of the tutorial

Generate a reasonable variable list prior to download

```
region_a.variables.remove(all=True)
region_a.variables.append(defaults=True)
pprint(region_a.variables.wanted)
```

6.3.7 Using your wanted variable list

NOTE: Variable subsetting is not yet available in Harmony for ICESat-2 products.

With a Read object

Calling the `load()` method on your Read object will automatically look for your wanted variable list and use it. Please see the [read-in example Jupyter Notebook](#) for a complete example of this usage.

With a local filepath

One of the benefits of using a local filepath in variables is that it allows you to easily inspect the variables that are available in your file. Once you have a variable of interest from the `avail` list, you could read that variable in with another library, such as `xarray`. The example below demonstrates this assuming an ATL06 ICESat-2 file.

```
# filepath = '/full/path/to/my/ATL06_file.h5'
v = ipx.Variables(path=filepath)
v.avail()
# Browse paths and decide you need `gt1l/land_ice_segments/`
```

```
import xarray as xr

xr.open_dataset(filepath, group='gt1l/land_ice_segments/', engine='h5netcdf')
```

You'll notice in this workflow you are limited to viewing data only within a particular group. Icepyx also provides functionality for merging variables within or even across files. See the [read-in example Jupyter Notebook](#) for more details about these features of icepyx.

Credits

- based on the subsetting notebook by: Jessica Scheick and Zheng Liu

VISUALIZING ICESAT-2 ELEVATIONS

This notebook (download) demonstrates interactive ICESat-2 elevation visualization by requesting data from [OpenAltimetry](#) based on metadata provided by [icepyx](#). We will show how to plot spatial extent and elevation interactively.

Some of this notebook is currently non-functional

Visualizations requiring the [OpenAltimetry API](#) are currently unavailable (since ~October 2023). The API changed and we haven't yet updated this notebook correspondingly.

Import packages

```
import icepyx as ipx
```

7.1 Create an ICESat-2 query object

Set the desired parameters for your data visualization.

For details on minimum required inputs, please refer to [IS2_data_access](#). If you are using a spatial extent input other than a bounding box for your search, it will automatically be converted to a bounding box for the purposes of visualization ONLY (your query object will not be affected).

```
# bounding box  
# Larsen C Ice Shelf  
short_name = 'ATL06'  
date_range = ['2020-7-1', '2020-8-1']  
spatial_extent = [-67, -70, -59, -65]  
cycles = ['03']  
tracks = ['0948', '0872', '1184', '0186', '1123', '1009', '0445', '0369']
```

```
# # polygon vertices  
# short_name = 'ATL06'  
# date_range = ['2019-02-20', '2019-02-28']  
# spatial_extent = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]
```

```
# # polygon geospatial file  
# short_name = 'ATL06'  
# date_range = ['2019-10-01', '2019-10-05']  
# spatial_extent = './supporting_files/data-access_PineIsland/glims_polygons.shp'
```

```
region = ipx.Query(short_name, spatial_extent, date_range)
```

```
print(region.product)
print(region.dates)
print(region.start_time)
print(region.end_time)
print(region.product_version)
print(list(set(region.avail_granules(cycles=True)[0]))) #region.cycles
print(list(set(region.avail_granules(tracks=True)[0]))) #region.tracks
```

7.2 Visualize spatial extent

By calling function `visualize_spatial_extent`, it will plot the spatial extent in red outline overlaid on a basemap, try zoom-in/zoom-out to see where is your interested region and what the geographic features look like in this region.

```
region.visualize_spatial_extent()
```

7.3 Visualize ICESat-2 elevation using OpenAltimetry API

Note: this function currently only supports products ATL06, ATL07, ATL08, ATL10, ATL12, ATL13

Now that we have produced an interactive map showing the spatial extent of ICESat-2 data to be requested from NSIDC using icepyx, what if we want to have a quick check on the ICESat-2 elevations we plan to download from NSIDC? [OpenAltimetry API](#) provides a nice way to achieve this. By sending metadata (product, date, bounding box, trackId) of each ICESat-2 file to the API, it can return elevation data almost instantaneously. The major drawback is requests are limited to 5x5 degree spatial bounding box selection for most of the ICESat-2 L3A products [ATL06](#), [ATL07](#), [ATL08](#), [ATL10](#), [ATL12](#), [ATL13](#). To solve this issue, if you input spatial extent exceeds the 5 degree maximum in either horizontal dimension, your input spatial extent will be split into 5x5 degree lat/lon grids first, use icepyx to query the metadata of ICESat-2 files located in each grid, and send each request to OpenAltimetry. Data sampling rates are 1/50 for ATL06 and 1/20 for other products.

There are multiple ways to access icepyx's visualization module. This option assumes you are visualizing the data as part of a workflow that will result in a data download. Alternative options for accessing the OpenAltimetry-based visualization module directly are provided at the end of this example.

```
cyclemap, rgtmap = region.visualize_elevation()
cyclemap
```

7.3.1 Plot elevation for individual RGT

The visualization tool also provides the option to view elevation data by latitude for each ground track.

```
rgtmap
```

7.3.2 Move on to data downloading from NSIDC if these are the products of interest

For more details on the data ordering and downloading process, see [ICESat-2_DAAC_DataAccess_Example](#)

```
region.order_granules()

#view a short list of order IDs
region.granules.orderIDs
```

(continues on next page)

(continued from previous page)

```
path = 'your data directory'  
region.download_granules(path)
```

7.3.3 Alternative Access Options to Visualize ICESat-2 elevation using OpenAltimetry API

You can also view elevation data by importing the visualization module directly and initializing it with your query object or a list of parameters:

```
from icepyx.core.visualization import Visualize
```

- passing your query object directly to the visualization module

```
region2 = ipx.Query(short_name, spatial_extent, date_range)  
vis = Visualize(region2)
```

- creating a visualization object directly without first creating a query object

```
vis = Visualize(product=short_name, spatial_extent=spatial_extent, date_range=date_range)
```

Credits

- Notebook by: [Tian Li](#), [Jessica Scheick](#) and [Wei Ji](#)
- Source material: [READ_ATL06_DEM Notebook](#) by [Tian Li](#) and [Friedrich Knuth](#)

READING ICESAT-2 DATA IN FOR ANALYSIS

This notebook ([download](#)) illustrates the use of `icepyx` for reading ICESat-2 data files, loading them into a data object. Currently the default data object is an Xarray Dataset, with ongoing work to provide support for other data object types.

For more information on how to order and download ICESat-2 data, see the [icepyx data access tutorial](#).

8.1 Motivation

Most often, when you open a data file, you must specify the underlying data structure and how you'd like the information to be read in. A simple example of this, for instance when opening a csv or similarly delimited file, is letting the software know if the data contains a header row, what the data type is (string, double, float, boolean, etc.) for each column, what the delimiter is, and which columns or rows you'd like to be loaded. Many ICESat-2 data readers are quite manual in nature, requiring that you accurately type out a list of string paths to the various data variables.

`icepyx` simplifies this process by relying on its awareness of ICESat-2 specific data file variable storage structure. Instead of needing to manually iterate through the beam pairs, you can provide a few options to the Read object and `icepyx` will do the heavy lifting for you (as detailed in this notebook).

8.2 Approach

If you're interested in what's happening under the hood: `icepyx` uses the `xarray` library to read in each of the requested variables of the dataset. `icepyx` formats each requested variable and then merges the read-in data from each of the variables to create a single data object. The use of `xarray` is powerful, because the returned data object can be used with relevant `xarray` processing tools.

Import packages, including `icepyx`

```
import icepyx as ipx
```

8.3 Quick-Start Guide

For those who might be looking into playing with this (but don't want all the details/explanations)

```
path_root = '/full/path/to/your/ATL06_data/'
reader = ipx.Read(path_root)
```

```
reader.vars.append(beam_list=['gt1l', 'gt3r'], var_list=['h_li', "latitude", "longitude"
↪])
```

```
ds = reader.load()
ds
```

```
ds.plot.scatter(x="longitude", y="latitude", hue="h_li", vmin=-100, vmax=2000)
```

8.4 Key steps for loading (reading) ICESat-2 data

Reading in ICESat-2 data with icepyx happens in a few simple steps:

1. Let icepyx know where to find your data (this might be local files or urls to data in cloud storage)
2. Create an icepyx Read object
3. Make a list of the variables you want to read in (does not apply for gridded products)
4. Load your data into memory (or read it in lazily, if you're using Dask)

We go through each of these steps in more detail in this notebook.

8.4.1 Step 0: Get some data if you haven't already

Here are a few lines of code to get you set up with a few data files if you don't already have some on your local system.

```
region_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-22', '2019-02-28'], \
                    start_time='00:00:00', end_time='23:59:59')
```

```
region_a.download_granules(path=path_root)
```

8.4.2 Step 1: Set data source path

Provide a full path to the data to be read in (i.e. opened). Currently accepted inputs are:

- a string path to directory - all files from the directory will be opened
- a string path to single file - one file will be opened
- a list of filepaths - all files in the list will be opened
- a glob string (see [glob](#)) - any files matching the glob pattern will be opened

```
path_root = '/full/path/to/your/data/'
```

```
# filepath = path_root + 'ATL06-20181214041627-Sample.h5'
```

```
# list_of_files = ['/my/data/ATL06/processed_ATL06_20190226005526_09100205_006_02.h5',
#                 '/my/other/data/ATL06/processed_ATL06_20191202102922_10160505_006_01.h5
#                 '→']
```

Glob Strings

`glob` is a Python library which allows users to list files in their file systems whose paths match a given pattern. Icepyx uses the `glob` library to give users greater flexibility over their input file lists.

`glob` works using `*` and `?` as wildcard characters, where `*` matches any number of characters and `?` matches a single character. For example:

- `/this/path/*.h5`: refers to all `.h5` files in the `/this/path` folder (Example matches: `"/this/path/processed_ATL03_20191130221008_09930503_006_01.h5"` or `"/this/path/myfavoriteicsat2file.h5"`)
- `/this/path/*ATL07*.h5`: refers to all `.h5` files in the `/this/path` folder that have `ATL07` in the filename. (Example matches: `"/this/path/ATL07-02_20221012220720_03391701_005_01.h5"` or `"/this/path/processed_ATL07.h5"`)
- `/this/path/ATL??/*.h5`: refers to all `.h5` files that are in a subfolder of `/this/path` and a subdirectory of `ATL` followed by any 2 characters (Example matches: `"/this/path/ATL03/processed_ATL03_20191130221008_09930503_006_01.h5"`, `"/this/path/ATL06/myfile.h5"`)

See the glob documentation or other online explainer tutorials for more in depth explanation, or advanced glob paths such as character classes and ranges.

Recursive Directory Search

glob will not by default search all of the subdirectories for matching filepaths, but it has the ability to do so.

If you would like to search recursively, you can achieve this by either:

1. passing the `recursive` argument into `glob_kwargs` and including `**` in your filepath
2. using glob directly to create a list of filepaths

Each of these two methods are shown below.

Method 1: passing the recursive argument into `glob_kwargs`

```
ipx.Read('/path/to/**/folder', glob_kwargs={'recursive': True})
```

You can use `glob_kwargs` for any additional argument to Python's builtin `glob.glob` that you would like to pass in via icepyx.

Method 2: using glob directly to create a list of filepaths

```
import glob
```

```
list_of_files = glob.glob('/path/to/**/folder', recursive=True)
ipx.Read(list_of_files)
```

8.4.3 Step 2: Create an icepyx read object

Using the `data_source` described in Step 1, we can create our Read object.

```
reader = ipx.Read(data_source=path_root)
```

The Read object now contains the list of matching files that will eventually be loaded into Python. You can inspect its properties, such as the files that were located or the identified product, directly on the Read object.

```
reader.filelist
```

```
reader.product
```

8.4.4 Step 3: Specify variables to be read in

To load your data into memory or prepare it for analysis, icepyx needs to know which variables you'd like to read in. If you've used icepyx to download data from NSIDC with variable subsetting (which is the default), then you may already be familiar with the icepyx `Variables` module and how to create and modify lists of variables. We showcase a specific case here, but we encourage you to check out [the icepyx Variables example](#) for a thorough trip through how to create and manipulate lists of ICESat-2 variable paths (examples are provided for multiple data products).

If you want to see a [likely very long] list of all path + variable combinations available to you, this unmutable (unchangeable) list is generated by default from the first file in your list (so not all variables may be contained in all of the files, depending on how you are accessing the data).

```
reader.vars.avail()
```

To make things easier, you can use icepyx's built-in default list that loads commonly used variables for your non-gridded data product, or create your own list of variables to be read in. icepyx will determine what variables are available for you to read in by creating a list from one of your source files. If you have multiple files that you're reading in, icepyx will automatically generate a list of filenames and take the first one to get the list of available variables.

Thus, if you have different variables available across files (even from the same data product), you may run into issues and need to come up with a workaround (we can help you do so!). We anticipate most users will have the minimum set of variables they are seeking to load available across all data files, so we're not currently developing this feature. Please get in touch if it would be a helpful feature for you or if you encounter this problem!

You may create a variable list for gridded ICESat-2 products. However, all variables in the file will still be added to your `DataSet`. (This is an area we're currently exploring on expanding - please let us know if you're working on this and would like to contribute!)

For a basic case, let's say we want to read in height, latitude, and longitude for all beam pairs. We create our variables list as

```
reader.vars.append(var_list=['h_li', "latitude", "longitude"])
```

Then we can view a dictionary of the variables we'd like to read in.

```
reader.vars.wanted
```

Don't forget - if you need to start over, and re-generate your wanted variables list, it's easy!

```
reader.vars.remove(all=True)
```

8.4.5 Step 4: Loading your data

Now that you've set up all the options, you're ready to read your ICESat-2 data into memory!

```
ds = reader.load()
```

Within a Jupyter Notebook, you can get a summary view of your data object.

ATTENTION: icepyx loads your data by creating an Xarray DataSet for each input granule and then merging them. In some cases, the automatic merge fails and needs to be handled manually. In these cases, icepyx will return a warning with the error message from the failed Xarray merge and a list of per-granule DataSets

This can happen if you unintentionally provide the same granule multiple times with different filenames or in segmented products where the `rgt+cycle` automatically generated `gran_idx` values match. In this latter case, you can simply provide unique `gran_idx` values for each `DataSet` in `ds` and run `import xarray as xr` and `ds_merged = xr.merge(ds)` to create one merged `DataSet`.

```
ds
```

8.5 On to data analysis!

From here, you can begin your analysis. Ultimately, icepyx aims to include an Xarray extension with ICESat-2 aware functions that allow you to do things like easily use only data from strong beams. That functionality is still in development. For fun, we've included a basic plot made with Xarray's built in functionality.

```
ds.plot.scatter(x="longitude", y="latitude", hue="h_li", vmin=-100, vmax=2000)
```

A developer note to users: our next steps will be to create an xarray extension with ICESat-2 aware functions (like “get_strong_beams”, etc.). Please let us know if you have any ideas or already have functions developed (we can work with you to add them, or add them for you!).

8.5.1 Credits

- original notebook by: Jessica Scheick
- notebook contributors: Wei Ji and Tian

ICESAT-2 AWS CLOUD DATA ACCESS

This notebook (download) illustrates the use of icepyx for accessing ICESat-2 data currently available through the Amazon Web Services (AWS) us-west2 s3 data bucket.

9.1 Notes

1. ICESat-2 data became publicly available on the cloud on 29 September 2022. Thus, access methods and example workflows are still being developed by the NASA NSIDC DAAC, and the underlying code in icepyx will need to be updated now that these data (and the associated metadata) are available. We appreciate your patience and contributions (e.g., reporting bugs and sharing your code) during this transition!
2. This example and the code it describes are part of ongoing development. Current limitations to using these features are described throughout the example, as appropriate.
3. You **MUST** be working within an AWS instance. Otherwise, you will get a permissions error.

9.2 Querying for data and finding s3 urls

```
import icepyx as ipx
```

```
# Make sure the user sees important warnings if they try to read a lot of data from the  
↳ cloud  
import warnings  
warnings.filterwarnings("always")
```

We will start the way we often do: by creating an icepyx Query object.

```
short_name = 'ATL03'  
spatial_extent = [-45, 58, -35, 75]  
date_range = ['2019-11-30', '2019-11-30']
```

```
reg = ipx.Query(short_name, spatial_extent, date_range)
```

9.2.1 Get the granule s3 urls

With this query object, you can get a list of available granules. This function returns a list containing the granule IDs and a list of the corresponding urls. Use `cloud=True` to get the needed s3 urls.

```
gran_ids = reg.avail_granules(ids=True, cloud=True)  
gran_ids
```

9.3 Determining variables of interest

There are several ways to view available variables. One is to use the existing Query object:

```
reg.order_vars
```

Another way is to use the Variables module:

```
ipx.Variables(product=short_name).avail()
```

We can also do this using a specific s3 filepath from the Query object:

```
ipx.Variables(path=gran_ids[1][0]).avail()
```

From any of these methods, we can see that `h_ph` is a variable for this data product. So, we will read that variable in the next step.

9.3.1 A note on listing variables using s3 urls

We can use the Variables module with an s3 url to explore available data variables the same way we do with local files. An important difference, however, is how the available variables list is created. When reading a local file, the Variables module will traverse the entire file and search for variables that are present in that file. This method is too time intensive with the s3 data, so instead the product/version of the data product is read from the file and all possible variables associated with that product/version are reported as available. As long as you are using the Earthdata Search s3 paths provided by the NASA NSIDC DAAC and the Query object, these lists will be the same.

9.3.2 A note on authentication

Notice that accessing cloud data requires two layers of authentication: 1) authenticating with your Earthdata Login and 2) authenticating for cloud access. These both happen behind the scenes without the need for users to provide any explicit commands.

Icepyx uses `earthaccess` to generate your s3 data access token, which will be valid for *one* hour. Icepyx will also renew the token for you after an hour; if viewing your token over the course of several hours, you may notice the values will change.

If you do want to see your s3 credentials, you can access them using:

```
# #uncommenting the line below will print your temporary aws login credentials
# reg.s3login_credentials
```

9.4 Choose a data file and access the data

Note: If you get a PermissionDenied Error when trying to read in the data, you may not be sending your request from an AWS hub in us-west2. We're currently working on how to alert users if they will not be able to access ICESat-2 data in the cloud for this reason.

We are ready to read our data! We do this by creating a reader object and using the s3 url returned from the Query object.

```
# the first index, [1], gets us into the list of s3 urls
# the second index, [0], gets us the first entry in that list.
s3url = gran_ids[1][0]
# s3url = 's3://nsidc-cumulus-prod-protected/ATLAS/ATL03/004/2019/11/30/ATL03_
→20191130221008_09930503_004_01.h5'
```

Create the Read object

```
reader = ipx.Read(s3url)
```

This reader object gives us yet another way to view available variables.

```
reader.variables.avail()
```

Next, we append our desired variable to the wanted variables (`wanted_vars`):

```
reader.variables.append(var_list=['h_ph'])
```

Finally, we load the data:

```
%%time  
# This may take 5-10 minutes  
reader.load()
```

9.4.1 Some important caveats

While the cloud data reading is functional within icepyx, it is very slow. Approximate timing shows it takes ~6 minutes of load time per variable per file from s3. Because of this, you will receive a warning if you try to load either more than three variables or two files at once.

The slow load speed is a demonstration of the many steps involved in making cloud data actionable - the data supply chain needs optimized source data, efficient low level data readers, and high level libraries that are enabled to use the fastest low level data readers. Not all of these pieces are fully developed right now, but the progress being made is exciting and there is lots of room for contribution!

Credits

- original notebook by: Jessica Scheick and Rachel Wegener
- notebook contributors: Theresa Andersen

QUEST EXAMPLE: FINDING ARGO AND ICESAT-2 DATA

In this notebook, we are going to find Argo and ICESat-2 data over a region of the Pacific Ocean. Normally, we would require multiple data portals or Python packages to accomplish this. However, thanks to the [QUEST \(Query, Unify, Explore SpatioTemporal\)](#) module, we can use `icepyx` to find both!

```
# Basic packages
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
from pprint import pprint

# icepyx and QUEST
import icepyx as ipx
```

10.1 Define the Quest Object

QUEST builds off of the general querying process originally designed for ICESat-2, but makes it applicable to other datasets.

Just like the ICESat-2 Query object, we begin by defining our Quest object. We provide the following bounding parameters:

- `spatial_extent`: Data is constrained to the given box over the Pacific Ocean.
- `date_range`: Only grab data from April 18-19, 2022 (to keep download sizes small for this example).

```
# Spatial bounds, given as SW/NE corners
spatial_extent = [-154, 30, -143, 37]

# Start and end dates, in YYYY-MM-DD format
date_range = ['2022-04-18', '2022-04-19']

# Initialize the QUEST object
reg_a = ipx.Quest(spatial_extent=spatial_extent, date_range=date_range)

print(reg_a)
```

Notice that we have defined our spatial and temporal domains, but we do not have any datasets in our QUEST object. The next section leads us through that process.

10.2 Getting the data

Let's first query the ICESat-2 data. If we want to extract information about the water column, the ATL03 product is likely the desired choice.

- `short_name`: ATL03

```
# ICESat-2 product
short_name = 'ATL03'

# Add ICESat-2 to QUEST datasets
reg_a.add_icesat2(product=short_name)
print(reg_a)
```

Let's see the available files over this region.

```
pprint(reg_a.datasets['icesat2'].avail_granules(ids=True))
```

Note the ICESat-2 functions shown here are the same as those used for direct icepyx queries. The user is referred to other [example workbooks](#) for detailed explanations about icepyx functionality.

Accessing ICESat-2 data requires Earthdata login credentials. When running the `download_all()` function below, an authentication check will be passed when attempting to download the ICESat-2 files.

Now let's grab Argo data using the same constraints. This is as simple as using the below function.

```
# Add argo to the desired QUEST datasets
reg_a.add_argo()
```

When accessing Argo data, the variables of interest will be organized as vertical profiles as a function of pressure. By default, only temperature is queried, so the user should supply a list of desired parameters using the code below. The user may also limit the pressure range of the returned data by passing `presRange="0,200"`.

Note: Our example shows only physical Argo float parameters, but the process is identical for including BGC float parameters.

```
# Customized variable query to retrieve salinity instead of temperature
reg_a.add_argo(params=['salinity'])
```

Additionally, a user may view or update the list of requested Argo and Argo-BGC parameters at any time through `reg_a.datasets['argo'].params`. If a user submits an invalid parameter ("temp" instead of "temperature", for example), an `AssertionError` will be raised. `reg_a.datasets['argo'].presRange` behaves analogously for limiting the pressure range of Argo data.

```
# update the list of argo parameters
reg_a.datasets['argo'].params = ['temperature', 'salinity']

# show the current list
reg_a.datasets['argo'].params
```

As for ICESat-2 data, the user can interact directly with the Argo data object (`reg_a.datasets['argo']`) to search or download data outside of the `Quest.search_all()` and `Quest.download_all()` functionality shown below.

The approach to directly search or download Argo data is to use `reg_a.datasets['argo'].search_data()`, and `reg_a.datasets['argo'].download()`. In both cases, the existing parameters and pressure ranges are used unless the user passes new `params` and/or `presRange` kwargs, respectively, which will directly update those values (stored attributes).

With our current setup, let's see what Argo parameters we will get.

```
# see what argo parameters will be searched for or downloaded
reg_a.datasets['argo'].params
```

```
reg_a.datasets['argo'].search_data()
```

Now we can access the data for both Argo and ICESat-2! The below function will do this for us.

Important: The Argo data will be compiled into a Pandas DataFrame, which must be manually saved by the user as demonstrated below. The ICESat-2 data is saved as processed HDF-5 files to the directory provided.

```
path = './quest/downloaded-data/'

# Access Argo and ICESat-2 data simultaneously
reg_a.download_all(path=path)
```

We now have one available Argo profile, containing temperature and pressure, in a Pandas DataFrame. BGC Argo is also available through QUEST, so we could add more variables to this list.

If the user wishes to add more profiles, parameters, and/or pressure ranges to a pre-existing DataFrame, then they should use `reg_a.datasets['argo'].download(keep_existing=True)` to retain previously downloaded data and have the new data added.

The `reg_a.download_all()` function also provided a file containing ICESat-2 ATL03 data. Recall that because these data files are very large, we focus on only one file for this example.

The below workflow uses the icepyx Read module to quickly load ICESat-2 data into an Xarray DataSet. To read in multiple files, see the [icepyx Read tutorial](#) for how to change your input source.

```
filename = 'processed_ATL03_20220419002753_04111506_006_02.h5'

reader = ipx.Read(data_source=path+filename)
```

```
# decide which portions of the file to read in
reader.vars.append(beam_list=['gt2l'],
                  var_list=['h_ph', "lat_ph", "lon_ph", 'signal_conf_ph'])
```

```
ds = reader.load()
ds
```

To make the data more easily plottable, let's convert the data into a Pandas DataFrame. Note that this method is memory-intensive for ATL03 data, so users are suggested to look at small spatial domains to prevent the notebook from crashing. Here, since we only have data from one granule and ground track, we have sped up the conversion to a dataframe by first removing extra data dimensions we don't need for our plots. Several of the other steps completed below using Pandas have analogous operations in Xarray that would further reduce memory requirements and computation times.

```
is2_pd =(ds.squeeze()
         .reset_coords()
         .drop_vars(["source_file", "data_start_utc", "data_end_utc", "gran_idx"])
         .to_dataframe()
         )
```

```
is2_pd
```

```
# Create a new dataframe with only "ocean" photons, as indicated by the "ds_surf_type"
↳flag
is2_pd = is2_pd.reset_index(level=[0,1])
is2_pd_ocean = is2_pd[is2_pd.ds_surf_type==1].drop(columns="photon_idx")
is2_pd_ocean
```

```
# Set Argo data as its own DataFrame
argo_df = reg_a.datasets['argo'].argodata
```

```
# Convert both DataFrames into GeoDataFrames
is2_gdf = gpd.GeoDataFrame(is2_pd_ocean,
                           geometry=gpd.points_from_xy(is2_pd_ocean['lon_ph'], is2_pd_
↳ocean['lat_ph']),
                           crs='EPSG:4326'
)
argo_gdf = gpd.GeoDataFrame(argo_df,
                             geometry=gpd.points_from_xy(argo_df.lon, argo_df.lat),
                             crs='EPSG:4326'
)
```

To view the relative locations of ICESat-2 and Argo, the below cell uses the `explore()` function from GeoPandas. The time variables cause errors in the function, so we will drop those variables first.

Note that for large datasets like ICESat-2, loading the map might take a while.

```
# Drop time variables that would cause errors in explore() function
is2_gdf = is2_gdf.drop(['delta_time', 'atlas_sdp_gps_epoch'], axis=1)
```

```
# Plot ICESat-2 track (medium/high confidence photons only) on a map
m = is2_gdf[is2_gdf['signal_conf_ph']>=3].explore(column='rgt', tiles='Esri.WorldImagery
↳',
                                                name='ICESat-2')

# Add Argo float locations to map
argo_gdf.explore(m=m, name='Argo', marker_kwds={"radius": 6}, color='red')
```

While we're at it, let's plot temperature and pressure profiles for each of the Argo floats in the area.

```
# Plot vertical profile of temperature vs. pressure for all of the floats
fig, ax = plt.subplots(figsize=(12, 6))
for pid in np.unique(argo_df['profile_id']):
    argo_df[argo_df['profile_id']==pid].plot(ax=ax, x='temperature', y='pressure',
↳label=pid)
plt.gca().invert_yaxis()
plt.xlabel('Temperature [°C]')
plt.ylabel('Pressure [hPa]')
plt.ylim([750, -10])
plt.tight_layout()
```

Lastly, let's look at some near-coincident ICESat-2 and Argo data in a multi-panel plot.

```
# Only consider ICESat-2 signal photons
is2_pd_signal = is2_pd_ocean[is2_pd_ocean['signal_conf_ph']>=0]
```

(continues on next page)

(continued from previous page)

```

## Multi-panel plot showing ICESat-2 and Argo data

# Calculate Extent
lons = [-154, -143, -143, -154, -154]
lats = [30, 30, 37, 37, 30]
lon_margin = (max(lons) - min(lons)) * 0.1
lat_margin = (max(lats) - min(lats)) * 0.1

# Create Plot
fig,([ax1,ax2],[ax3,ax4]) = plt.subplots(2, 2, figsize=(12, 6))

# Plot Relative Global View
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world.plot(ax=ax1, color='0.8', edgecolor='black')
argo_df.plot.scatter(ax=ax1, x='lon', y='lat', s=25.0, c='green', zorder=3, alpha=0.3)
is2_pd_signal.plot.scatter(ax=ax1, x='lon_ph', y='lat_ph', s=10.0, zorder=2, alpha=0.3)
ax1.plot(lons, lats, linewidth=1.5, color='orange', zorder=2)
ax1.set_xlim(-160, -100)
ax1.set_ylim(20, 50)
ax1.set_aspect('equal', adjustable='box')
ax1.set_xlabel('Longitude', fontsize=18)
ax1.set_ylabel('Latitude', fontsize=18)

# Plot Zoomed View of Ground Tracks
argo_df.plot.scatter(ax=ax2, x='lon', y='lat', s=50.0, c='green', zorder=3, alpha=0.3)
is2_pd_signal.plot.scatter(ax=ax2, x='lon_ph', y='lat_ph', s=10.0, zorder=2, alpha=0.3)
ax2.plot(lons, lats, linewidth=1.5, color='orange', zorder=1)
ax2.set_xlim(min(lons) - lon_margin, max(lons) + lon_margin)
ax2.set_ylim(min(lats) - lat_margin, max(lats) + lat_margin)
ax2.set_aspect('equal', adjustable='box')
ax2.set_xlabel('Longitude', fontsize=18)
ax2.set_ylabel('Latitude', fontsize=18)

# Plot ICESat-2 along-track vertical profile. A dotted line notes the location of a
↔ nearby Argo float
is2 = ax3.scatter(is2_pd_signal['lat_ph'], is2_pd_signal['h_ph']+13.1, s=0.1)
ax3.axvline(34.43885, linestyle='--', linewidth=3, color='black')
ax3.set_xlim([34.3, 34.5])
ax3.set_ylim([-20, 5])
ax3.set_xlabel('Latitude', fontsize=18)
ax3.set_ylabel('Approx. IS-2 Depth [m]', fontsize=16)
ax3.set_yticklabels(['15', '10', '5', '0', '-5'])

# Plot vertical ocean profile of the nearby Argo float
argo_df.plot(ax=ax4, x='temperature', y='pressure', linewidth=3)
# ax4.set_yscale('log')
ax4.invert_yaxis()
ax4.get_legend().remove()
ax4.set_xlabel('Temperature [°C]', fontsize=18)
ax4.set_ylabel('Argo Pressure', fontsize=16)

```

(continues on next page)

(continued from previous page)

```
plt.tight_layout()

# Save figure
#plt.savefig('/icepyx/quest/figures/is2_argo_figure.png', dpi=500)
```

Recall that the Argo data must be saved manually. The dataframe associated with the Quest object can be saved using `reg_a.save_all(path)`

```
reg_a.save_all(path)
```

ICEPYX DOCUMENTATION (API)

icepyx class diagram illustrating the library’s public-facing classes, their attributes and methods, and their relationships. Additional UML diagrams, including a more detailed, developer UML class diagram showing hidden parameters, are available on [GitHub in the icepyx/doc/source/user_guide/documentation/ directory](#). Diagrams are updated automatically after a pull request (PR) is approved and before it is merged to the development branch.

11.1 Query Class

11.1.1 Constructors

<code>GenQuery</code> ([spatial_extent, date_range, ...])	Base class for querying data.
<code>Query</code> ([product, spatial_extent, date_range, ...])	Query and get ICESat-2 data

icepyx.GenQuery

class icepyx.**GenQuery**(*spatial_extent=None, date_range=None, start_time=None, end_time=None, **kwargs*)

Base class for querying data.

Generic components of query object that specifically handles spatio-temporal constraints applicable to all datasets. Extended by Query (ICESat-2) and Quest (other products).

Parameters

- **spatial_extent** (*list of coordinates or string (i.e. file name)*) – Spatial extent of interest, provided as a bounding box, list of polygon coordinates, or geospatial polygon file. NOTE: Longitude values are assumed to be in the range -180 to +180, with 0 being the Prime Meridian (Greenwich). See `xdateline` for regions crossing the date line. You can submit at most one bounding box or list of polygon coordinates. Per NSIDC requirements, geospatial polygon files may only contain one feature (polygon). Bounding box coordinates should be provided in decimal degrees as [lower-left-longitude, lower-left-latitude, upper-right-longitude, upper-right-latitude]. Polygon coordinates should be provided as coordinate pairs in decimal degrees as [(longitude1, latitude1), (longitude2, latitude2), ... (longitude_n, latitude_n), (longitude1, latitude1)] or [longitude1, latitude1, longitude2, latitude2, ... longitude_n, latitude_n, longitude1, latitude1]. Your list must contain at least four points, where the first and last are identical. Geospatial polygon files are entered as strings with the full file path and must contain only one polygon with the area of interest. Currently supported formats are: kml, shp, and gpkg
- **date_range** (*list or dict, as follows*) – Date range of interest, provided as start and end dates, inclusive. Accepted input date formats are:
 - YYYY-MM-DD string

- YYYY-DOY string
- `datetime.date` object (if times are included)
- `datetime.datetime` objects (if no times are included)

where YYYY = 4 digit year, MM = 2 digit month, DD = 2 digit day, DOY = 3 digit day of year. Date inputs are accepted as a list or dictionary with `start_date` and `end_date` keys. Currently, a list of specific dates (rather than a range) is not accepted. TODO: allow searches with a list of dates, rather than a range.

- **`start_time`** (`str`, `datetime.time`, `default None`) – Start time in UTC/Zulu (24 hour clock). Input types are an HH:mm:ss string or `datetime.time` object where HH = hours, mm = minutes, ss = seconds. If None is given (and a `datetime.datetime` object is not supplied for `date_range`), a default of 00:00:00 is applied.
- **`end_time`** (`str`, `datetime.time`, `default None`) – End time in UTC/Zulu (24 hour clock). Input types are an HH:mm:ss string or `datetime.time` object where HH = hours, mm = minutes, ss = seconds. If None is given (and a `datetime.datetime` object is not supplied for `date_range`), a default of 23:59:59 is applied. If a `datetime.datetime` object was created without times, the `datetime` package defaults will apply over those of `icepyx`.
- **`xdateline`** (`bool`, `default None`) – Keyword argument to enforce spatial inputs that cross the International Date Line. Internally, this will translate your longitudes to 0 to 360 to construct the correct, valid Shapely geometry.

WARNING: This will allow your request to be properly submitted and visualized. However, this flag WILL NOT automatically correct for incorrectly ordered spatial inputs.

Examples

Initializing Query with a bounding box

```
>>> reg_a_bbox = [-55, 68, -48, 71]
>>> reg_a_dates = ['2019-02-20', '2019-02-28']
>>> reg_a = GenQuery(reg_a_bbox, reg_a_dates)
>>> print(reg_a)
Extent type: bounding_box
Coordinates: [-55.0, 68.0, -48.0, 71.0]
Date range: (2019-02-20 00:00:00, 2019-02-28 23:59:59)
```

Initializing Query with a list of polygon vertex coordinate pairs.

```
>>> reg_a_poly = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]
>>> reg_a_dates = ['2019-02-20', '2019-02-28']
>>> reg_a = GenQuery(reg_a_poly, reg_a_dates)
>>> print(reg_a)
Extent type: polygon
Coordinates: [-55.0, 68.0, -55.0, 71.0, -48.0, 71.0, -48.0, 68.0, -55.0, 68.0]
Date range: (2019-02-20 00:00:00, 2019-02-28 23:59:59)
```

Initializing Query with a geospatial polygon file.

```
>>> from pathlib import Path
>>> aoi = Path('./doc/source/example_notebooks/supporting_files/simple_test_poly.
→gpkg').resolve()
>>> reg_a_dates = ['2019-02-22', '2019-02-28']
>>> reg_a = GenQuery(str(aoi), reg_a_dates)
>>> print(reg_a)
Extent type: polygon
Coordinates: [-55.0, 68.0, -55.0, 71.0, -48.0, 71.0, -48.0, 68.0, -55.0, 68.0]
Date range: (2019-02-22 00:00:00, 2019-02-28 23:59:59)
```

 **See also**

Query, Quest

`__init__(spatial_extent=None, date_range=None, start_time=None, end_time=None, **kwargs)`

Methods

`__init__([spatial_extent, date_range, ...])`

Attributes

<code>dates</code>	Return an array showing the date range of the query object.
<code>end_time</code>	Return the end time specified for the end date.
<code>spatial</code>	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
<code>spatial_extent</code>	Return an array showing the spatial extent of the query object. Spatial extent is returned as an input type (which depends on how you initially entered your spatial data) followed by the geometry data. Bounding box data is [lower-left-longitude, lower-left-latitude, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n, latitude_n, longitude1, latitude1].
<code>start_time</code>	Return the start time specified for the start date.
<code>temporal</code>	Return the Temporal object containing date/time range information for the query object.

icepyx.Query

`class icepyx.Query(product=None, spatial_extent=None, date_range=None, start_time=None, end_time=None, version=None, cycles=None, tracks=None, auth=None, **kwargs)`

Query and get ICESat-2 data

ICESat-2 Data object to query, obtain, and perform basic operations on available ICESat-2 data products using temporal and spatial input parameters. Allows the easy input and formatting of search parameters to match the NASA NSIDC DAAC and (development goal-not yet implemented) conversion to multiple data types. Expands the superclass GenQuery.

See the doc page for GenQuery for details on temporal and spatial input parameters.

Parameters

- **product** (*str*) – ICESat-2 data product ID, also known as “short name” (e.g. ATL03). Available data products can be found at: <https://nsidc.org/data/icesat-2/data-sets>

- **version** (*str*, *default most recent version*) – Product version, given as a 3 digit string. If no version is given, the current version is used. Example: “007”
- **cycles** (*str or list[str]*, *default all available orbital cycles*) – Product cycle, given as a 2 digit string, or a list of 2-digit strings. If no cycle is given, all available cycles are used. Example: “04”
- **tracks** (*str or list[str]*, *default all available reference ground tracks (RGTs)*) – Product track, given as a 4 digit string, or a list of 4-digit strings. If no track is given, all available reference ground tracks are used. Example: “0594”
- **auth** (*earthaccess.auth.Auth*, *default None*) – An earthaccess authentication object. Available as an argument so an existing earthaccess.auth.Auth object can be used for authentication. If not given, a new auth object will be created whenever authentication is needed.

Return type

query object

Examples

Initializing Query with a bounding box.

```
>>> reg_a_bbox = [-55, 68, -48, 71]
>>> reg_a_dates = ['2019-02-20', '2019-02-28']
>>> reg_a = Query('ATL06', reg_a_bbox, reg_a_dates)
>>> print(reg_a)
Product ATL06 v007
('bounding_box', [-55.0, 68.0, -48.0, 71.0])
Date range ['2019-02-20', '2019-02-28']
```

Initializing Query with a list of polygon vertex coordinate pairs.

```
>>> reg_a_poly = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]
>>> reg_a_dates = ['2019-02-20', '2019-02-28']
>>> reg_a = Query('ATL06', reg_a_poly, reg_a_dates)
>>> reg_a.spatial_extent
('polygon', [-55.0, 68.0, -55.0, 71.0, -48.0, 71.0, -48.0, 68.0, -55.0, 68.0])
```

Initializing Query with a geospatial polygon file.

```
>>> from pathlib import Path
>>> aoi = Path('./doc/source/example_notebooks/supporting_files/simple_test_poly.
↳gpkg').resolve()
>>> reg_a_dates = ['2019-02-22', '2019-02-28']
>>> reg_a = Query('ATL06', str(aoi), reg_a_dates)
>>> print(reg_a)
Product ATL06 v007
('polygon', [-55.0, 68.0, -55.0, 71.0, -48.0, 71.0, -48.0, 68.0, -55.0, 68.0])
Date range ['2019-02-22', '2019-02-28']
```

See also[GenQuery](#)

`__init__`(*product=None, spatial_extent=None, date_range=None, start_time=None, end_time=None, version=None, cycles=None, tracks=None, auth=None, **kwargs*)

Methods

<code>__init__</code> ([<i>product, spatial_extent, ...</i>])	
<code>avail_granules</code> ([<i>ids, cycles, tracks, cloud</i>])	Obtain information about the available granules for the query object's parameters.
<code>download_granules</code> (<i>path</i> [, <i>overwrite, subset, ...</i>])	Download the granules for the order, blocking until they are ready if necessary.
<code>latest_version</code> ()	A reference function to <code>is2ref.latest_version</code> .
<code>order_granules</code> ([<i>subset, skip_preview</i>])	Place an order for the available granules for the query object.
<code>product_all_info</code> ()	Display all metadata about the product of interest (the collection).
<code>product_summary_info</code> ()	Display a summary of selected metadata for the specified version of the product of interest (the collection).
<code>show_custom_options</code> ()	Display customization/subsetting options available for this product.
<code>skip_preview</code> ()	Skips preview for current order, if it is in preview state.
<code>visualize_elevation</code> ()	Visualize elevation requested from OpenAltimetry API using datashader based on cycles https://holoviz.org/tutorial/Large_Data.html
<code>visualize_spatial_extent</code> ()	Creates a map displaying the input spatial extent

Attributes

<code>CMRparams</code>	value pairs that will be submitted.
<code>REQUEST_RETRY_INTERVAL_SECONDS</code>	
<code>auth</code>	Authentication object returned from <code>earthaccess.login()</code> which stores user authentication.
<code>concept_id</code>	
<code>cycles</code>	Return the unique ICESat-2 orbital cycle.
<code>dates</code>	Return an array showing the date range of the query object.
<code>end_time</code>	Return the end time specified for the end date.
<code>granules</code>	Return the granules object, which provides the underlying functionality for searching, ordering, and downloading granules for the specified product.
<code>order_vars</code>	This used to print the list of variables for subsetting, Harmony doesn't provide that for IS2 datasets.
<code>product</code>	Return the short name product ID string associated with the query object.
<code>product_version</code>	Return the product version of the data object.
<code>s3login_credentials</code>	A dictionary which stores login credentials for AWS s3 access.
<code>session</code>	Earthaccess session object for connecting to Earth-data resources.

continues on next page

Table 5 – continued from previous page

<i>spatial</i>	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
<i>spatial_extent</i>	Return an array showing the spatial extent of the query object. Spatial extent is returned as an input type (which depends on how you initially entered your spatial data) followed by the geometry data. Bounding box data is [lower-left-longitude, lower-left-latitude, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n, latitude_n, longitude1, latitude1].
<i>start_time</i>	Return the start time specified for the start date.
<i>temporal</i>	Return the Temporal object containing date/time range information for the query object.
<i>tracks</i>	Return the unique ICESat-2 Reference Ground Tracks
<i>variables</i>	

11.1.2 Attributes

<i>Query.CMRparams</i>	value pairs that will be submitted.
<i>Query.cycles</i>	Return the unique ICESat-2 orbital cycle.
<i>Query.dates</i>	Return an array showing the date range of the query object.
<i>Query.end_time</i>	Return the end time specified for the end date.
<i>Query.granules</i>	Return the granules object, which provides the underlying functionality for searching, ordering, and downloading granules for the specified product.
<i>Query.order_vars</i>	This used to print the list of variables for subsetting, Harmony doesn't provide that for IS2 datasets.
<i>Query.product</i>	Return the short name product ID string associated with the query object.
<i>Query.product_version</i>	Return the product version of the data object.
<i>Query.spatial</i>	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
<i>Query.spatial_extent</i>	Return an array showing the spatial extent of the query object. Spatial extent is returned as an input type (which depends on how you initially entered your spatial data) followed by the geometry data. Bounding box data is [lower-left-longitude, lower-left-latitude, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n, latitude_n, longitude1, latitude1].
<i>Query.start_time</i>	Return the start time specified for the start date.
<i>Query.temporal</i>	Return the Temporal object containing date/time range information for the query object.
<i>Query.tracks</i>	Return the unique ICESat-2 Reference Ground Tracks

icepyx.Query.CMRparams

property Query.CMRparams: **CMRParamsWithBbox** | **CMRParamsWithPolygon**

value pairs that will be submitted. It generates the dictionary if it does not already exist.

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.CMRparams
{'concept_id': 'C3564876127-NSIDC_CPRD',
 'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
 'bounding_box': '-55.0,68.0,-48.0,71.0'}
```

Type

Display the CMR key

icepyx.Query.cycles

property Query.cycles

Return the unique ICESat-2 orbital cycle.

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.cycles
['No orbital[cycle] parameters set']
```

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], cycles=['03', '04'], tracks=[
→ '0849', '0902'])
>>> reg_a.cycles
['03', '04']
```

icepyx.Query.dates

property Query.dates: **list[str]**

Return an array showing the date range of the query object. Dates are returned as an array containing the start and end datetime objects, inclusive, in that order.

i Examples

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.dates
['2019-02-20', '2019-02-28']
```

```
>>> reg_a = GenQuery([-55, 68, -48, 71])
>>> reg_a.dates
['No temporal parameters set']
```

icepyx.Query.end_time**property** Query.**end_time**: `list[str] | str`

Return the end time specified for the end date.

Examples

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.end_time
'23:59:59'
```

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'], end_
↳time='10:20:20')
>>> reg_a.end_time
'10:20:20'
```

```
>>> reg_a = GenQuery([-55, 68, -48, 71])
>>> reg_a.end_time
['No temporal parameters set']
```

icepyx.Query.granules**property** Query.**granules**

Return the granules object, which provides the underlying functionality for searching, ordering, and downloading granules for the specified product. Users are encouraged to use the built-in wrappers rather than trying to access the granules object themselves.

See also

[avail_granules](#), [order_granules](#), [download_granules](#), `granules.Granules`

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.granules
<icepyx.core.granules.Granules at [location]>
```

icepyx.Query.order_vars**property** Query.**order_vars**: *Variables* | *None*

This used to print the list of variables for subsetting, Harmony doesn't provide that for IS2 datasets. we do need to implement a class that gets the variables even if it'sm only for listing.

icepyx.Query.product**property** Query.**product**

Return the short name product ID string associated with the query object.

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.product
'ATL06'
```

icepyx.Query.product_version**property** Query.**product_version**

Return the product version of the data object.

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.product_version
'007'
```

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'], ↵
↵version='6')
>>> reg_a.product_version
'006'
```

icepyx.Query.spatial**property** Query.**spatial**

Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects. The spatial object has several properties to enable user access to the stored spatial extent in multiple formats.

↪ See also

`spatial.Spatial.spatial_extent`, `spatial.Spatial.extent_type`, `spatial.Spatial.extent_file`, `spatial.Spatial`

Examples

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.spatial
<icepyx.core.spatial.Spatial at [location]>
```

```
>>> print(reg_a.spatial)
Extent type: bounding_box
Coordinates: [-55.0, 68.0, -48.0, 71.0]
```

icepyx.Query.spatial_extent**property** Query.spatial_extent

Return an array showing the spatial extent of the query object. Spatial extent is returned as an input type (which depends on how you initially entered your spatial data) followed by the geometry data. Bounding box data is [lower-left-longitude, lower-left-latitude,

... upper-right-longitude, upper-right-latitude].

Polygon data is [longitude1, latitude1, longitude2, latitude2,
... longitude_n,latitude_n, longitude1,latitude1].

Returns

- *tuple of length 2*
- *First tuple element is the spatial type (“bounding box” or “polygon”).*
- *Second tuple element is the spatial extent as a list of coordinates.*

Examples

```
# Note: coordinates returned as float, not int >>> reg_a = GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.spatial_extent ('bounding_box', [-55.0, 68.0, -48.0, 71.0])
```

```
>>> reg_a = GenQuery([(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)], [
→ '2019-02-20', '2019-02-28'])
>>> reg_a.spatial_extent
('polygon', [-55.0, 68.0, -55.0, 71.0, -48.0, 71.0, -48.0, 68.0, -55.0, 68.0])
```

```
# NOTE Is this where we wanted to put the file-based test/example? # The test file path is: examples/supporting_files/simple_test_poly.gpkg
```

See also

Spatial.extent, Spatial.extent_type, Spatial.extent_as_gdf

icepyx.Query.start_time**property** Query.start_time: list[str] | str

Return the start time specified for the start date.

Examples

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.start_time
'00:00:00'
```

```
>>> reg_a = ipx.GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'], start_
→time='12:30:30')
>>> reg_a.start_time
'12:30:30'
```

```
>>> reg_a = GenQuery([-55, 68, -48, 71])
>>> reg_a.start_time
['No temporal parameters set']
```

icepyx.Query.temporal

property Query.temporal: *Temporal* | list[str]

Return the Temporal object containing date/time range information for the query object.

See also

temporal.Temporal.start, temporal.Temporal.end, temporal.Temporal

Examples

```
>>> reg_a = GenQuery([-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> print(reg_a.temporal)
Start date and time: 2019-02-20 00:00:00
End date and time: 2019-02-28 23:59:59
```

```
>>> reg_a = GenQuery([-55, 68, -48, 71], cycles=['03', '04', '05', '06', '07'],
→tracks=['0849', '0902'])
>>> print(reg_a.temporal)
['No temporal parameters set']
```

icepyx.Query.tracks

property Query.tracks

Return the unique ICESat-2 Reference Ground Tracks

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.tracks
['No orbital[tracks] parameters set']
```

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], cycles=['03', '04'], tracks=[
→'0849', '0902'])
>>> reg_a.tracks
['0849', '0902']
```

11.1.3 Methods

<code>Query.avail_granules([ids, cycles, tracks, ...])</code>	Obtain information about the available granules for the query object's parameters.
<code>Query.download_granules(path[, overwrite, ...])</code>	Download the granules for the order, blocking until they are ready if necessary.
<code>Query.latest_version()</code>	A reference function to <code>is2ref.latest_version</code> .
<code>Query.order_granules([subset, skip_preview])</code>	Place an order for the available granules for the query object.
<code>Query.product_all_info()</code>	Display all metadata about the product of interest (the collection).
<code>Query.product_summary_info()</code>	Display a summary of selected metadata for the specified version of the product of interest (the collection).
<code>Query.show_custom_options()</code>	Display customization/subsetting options available for this product.
<code>Query.visualize_spatial_extent()</code>	Creates a map displaying the input spatial extent
<code>Query.visualize_elevation()</code>	Visualize elevation requested from OpenAltimetry API using datashader based on cycles https://holoviz.org/tutorial/Large_Data.html

icepyx.Query.avail_granules

`Query.avail_granules(ids=False, cycles=False, tracks=False, cloud=False)`

Obtain information about the available granules for the query object's parameters. By default, a complete list of available granules is obtained and stored in the object, but only summary information is returned. Lists of granule IDs, cycles and RGTs can be obtained using the boolean triggers.

Parameters

- **ids** (*bool*, *default False*) – Indicates whether the function should return a list of granule IDs.
- **cycles** (*bool*, *default False*) – Indicates whether the function should return a list of orbital cycles.
- **tracks** (*bool*, *default False*) – Indicates whether the function should return a list of RGTs.
- **cloud** (*bool*, *default False*) – Indicates whether the function should return data available in the cloud. Note: except in rare cases while data is in the process of being appended to, data available in the cloud and for download via on-premises will be identical.

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.avail_granules()
{'Number of available granules': 4,
 'Average size of granules (MB)': np.float64(55.166646003723145),
 'Total size of all granules (MB)': 220.66658401489258}

>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-23'])
>>> reg_a.avail_granules(ids=True)
```

```

[['ATL06_20190221121851_08410203_007_01.h5', 'ATL06_20190222010344_08490205_007_
→01.h5']]
>>> reg_a.avail_granules(cycles=True)
[['02', '02']]
>>> reg_a.avail_granules(tracks=True)
[['0841', '0849']]

```

icepyx.Query.download_granules

Query.**download_granules**(*path*, *overwrite=False*, *subset=True*, *skip_preview=False*)

Download the granules for the order, blocking until they are ready if necessary.

Parameters

- **path** (*str* or *Path*) – The directory where granules should be saved.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files (default is False).
- **subset** (*bool*, *default True*) – Apply subsetting to the data order using harmony, returning only data that meets the subset parameters. Spatial and temporal subsetting based on the input parameters happens by default when subset=True, but additional subsetting options are available. Spatial subsetting returns all data that are within the area of interest (but not complete granules. This eliminates false-positive granules returned by the metadata-level search)
- **skip_preview** (*bool*, *default False*) – If True, bypass the preview state when we order subsetting queries that exceed 300 granules.

Returns

A list of the downloaded file paths if successful, otherwise None.

Return type

list or None

icepyx.Query.latest_version

Query.**latest_version**()

A reference function to `is2ref.latest_version`.

Determine the most recent version available for the given product.

Examples

```

>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.latest_version()
'007'

```

icepyx.Query.order_granules

Query.**order_granules**(*subset=True*, *skip_preview=False*)

Place an order for the available granules for the query object.

Parameters

- **subset** (*bool*, *default True*) – Apply subsetting to the data order using harmony, returning only data that meets the subset parameters. Spatial and temporal subsetting based on

the input parameters happens by default when `subset=True`, but additional subsetting options are available. Spatial subsetting returns all data that are within the area of interest (but not complete granules. This eliminates false-positive granules returned by the metadata-level search)

- **skip_preview** (*bool*, *default False*) – If True, bypass the preview state when we order subsetting queries that exceed 300 granules.

Return type

DataOrder

➔ See also

`harmony.place_order`

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.order_granules()
Harmony job ID: 931355e8-0005-4dff-9c76-7903a5be283d
[order status output]
Harmony provided these error messages:
[if any were returned from the harmony subsetter, e.g. No data found that
↪matched subset constraints.]
Your harmony order is: complete
```

icepyx.Query.product_all_info

`Query.product_all_info()`

Display all metadata about the product of interest (the collection).

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> reg_a.product_all_info()
{very long prettily-formatted dictionary output}
```

icepyx.Query.product_summary_info

`Query.product_summary_info()`

Display a summary of selected metadata for the specified version of the product of interest (the collection).

i Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'],
↪version='007')
>>> reg_a.product_summary_info()
title : ATLAS/ICESat-2 L3A Land Ice Height V007
short_name : ATL06
version_id : 007
```

```

time_start : 2018-10-14T00:00:00.000Z
coordinate_system : CARTESIAN
summary : ATL06 contains geolocated land-ice surface heights above the WGS84_
↳ellipsoid, plus ancillary parameters that can be used to interpret and assess_
↳the quality of the height estimates. The data were acquired by the Advanced_
↳Topographic Laser Altimeter System (ATLAS) instrument on board the ICESat-2_
↳observatory.
orbit_parameters : {}

```

icepyx.Query.show_custom_options

Query.show_custom_options()

Display customization/subsetting options available for this product.

Return type

Dict[str, Any]

icepyx.Query.visualize_spatial_extent

Query.visualize_spatial_extent()

Creates a map displaying the input spatial extent

i Examples

```

>>> reg_a = ipx.Query('ATL06', 'path/spatialfile.shp', ['2019-02-22', '2019-02-28'])
>>> reg_a.visualize_spatial_extent
[visual map output]

```

icepyx.Query.visualize_elevation

Query.visualize_elevation()

Visualize elevation requested from OpenAltimetry API using datashader based on cycles https://holoviz.org/tutorial/Large_Data.html

Returns

map_cycle, map_rgt + lineplot_rgt – Holoviews data visualization elements

Return type

Holoviews objects

11.2 Read Class

11.2.1 Constructor

Read(data_source[, glob_kwargs, out_obj_type])

Data object to read ICESat-2 data into the specified formats.

icepyx.Read

class icepyx.Read(*data_source*, *glob_kwargs*={}, *out_obj_type*=None)

Data object to read ICESat-2 data into the specified formats. Provides flexibility for reading nested hdf5 files into common analysis formats.

Parameters

- **data_source** (*str*, *pathlib.Path*, *list*) – A string, `pathlib.Path` object, or list which specifies the files to be read. The string can be either: 1) the path of a single file 2) the path to a directory or 3) a [glob string](<https://docs.python.org/3/library/glob.html>). The List must be a list of strings, each of which is the path of a single file.
- **glob_kwargs** (*dict*, *default* {}) – Additional arguments to be passed into the [glob.glob()](<https://docs.python.org/3/library/glob.html#glob.glob>)function
- **out_obj_type** (*object*, *default* `xarray.Dataset`) – The desired format for the data to be read in. Currently, only `xarray.Dataset` objects (default) are available. Please ask us how to help enable usage of other data objects!

Return type

read object

Examples

```
Reading a single file >>> ipx.Read('/path/to/data/processed_ATL06_20190226005526_09100205_006_02.h5')
# doctest: +SKIP
```

```
Reading all files in a directory >>> ipx.Read('/path/to/data/') # doctest: +SKIP
```

```
Reading files that match a particular pattern (here, all .h5 files that start with processed_ATL06_). >>>
ipx.Read('/path/to/data/processed_ATL06_*.h5') # doctest: +SKIP
```

```
Reading a specific list of files >>> list_of_files = [ ... '/path/to/data/processed_ATL06_20190226005526_09100205_006_02.h5'
... '/path/to/more/data/processed_ATL06_20191202102922_10160505_006_01.h5', ... ] >>>
ipx.Read(list_of_files) # doctest: +SKIP
```

`__init__`(*data_source*, *glob_kwargs*={}, *out_obj_type*=None)

Methods

`__init__`(*data_source*[], *glob_kwargs*, ...)

`load`()

Create a single Xarray Dataset containing the data from one or more files and/or ground tracks.

Attributes

`auth`

Authentication object returned from `earthaccess.login()` which stores user authentication.

`filelist`

Return the list of files represented by this Read object.

`product`

Return the product associated with the Read object.

`s3login_credentials`

A dictionary which stores login credentials for AWS s3 access.

continues on next page

Table 10 – continued from previous page

<code>session</code>	Earthaccess session object for connecting to Earth-data resources.
<code>variables</code>	Return the variables object associated with the data being read in.

11.2.2 Attributes

<code>Read.filelist</code>	Return the list of files represented by this Read object.
<code>Read.product</code>	Return the product associated with the Read object.
<code>Read.variables</code>	Return the variables object associated with the data being read in.

icepyx.Read.filelist

property `Read.filelist`

Return the list of files represented by this Read object.

icepyx.Read.product

property `Read.product`

Return the product associated with the Read object.

icepyx.Read.variables

property `Read.variables`

Return the variables object associated with the data being read in. This instance is generated from the source file or first file in a list of input files (when source is a directory).

See also

`variables.Variables`

Examples

```
>>> reader = ipx.Read(path_root, "ATL06", pattern)
>>> reader.variables
<icepyx.core.variables.Variables at [location]>
```

11.2.3 Methods

<code>Read.load()</code>	Create a single Xarray Dataset containing the data from one or more files and/or ground tracks.
--------------------------	---

icepyx.Read.load

Read.load()

Create a single Xarray Dataset containing the data from one or more files and/or ground tracks. Uses icepyx's ICESat-2 data product awareness and Xarray's *combine_by_coords* function.

All items in the wanted variables list will be loaded from the files into memory. If you do not provide a wanted variables list, a default one will be created for you.

11.3 Variables Class

11.3.1 Constructor

<code>Variables([path, product, version, avail, ...])</code>	Get, create, interact, and manipulate lists of variables and variable paths contained in ICESat-2 products.
--	---

icepyx.Variables

class icepyx.Variables(*path=None, product=None, version=None, avail=None, wanted=None, auth=None*)

Get, create, interact, and manipulate lists of variables and variable paths contained in ICESat-2 products.

Parameters

- **path** (*str, default None*) – The path to a local Icesat-2 file. The variables list will contain the variables present in this file. Either path or product are required input arguments.
- **product** (*str, default None*) – Properly formatted string specifying a valid ICESat-2 product. The variables list will contain all available variables for this product. Either product or path are required input arguments.
- **version** (*str, default None*) – Properly formatted string specifying a valid version of the ICESat-2 product.
- **avail** (*dict, default None*) – Dictionary (key:values) of available variable names (keys) and paths (values).
- **wanted** (*dict, default None*) – As avail, but for the desired list of variables
- **auth** (*earthaccess.auth.Auth, default None*) – An earthaccess authentication object. Available as an argument so an existing earthaccess.auth.Auth object can be used for authentication. If not given, a new auth object will be created whenever authentication is needed.

`__init__`(*path=None, product=None, version=None, avail=None, wanted=None, auth=None*)

Methods

<code>__init__</code> ([path, product, version, avail, ...])	
<code>append</code> ([defaults, var_list, beam_list, ...])	Add to the list of desired variables using user specified beams and variable list.
<code>avail</code> ([options, internal])	Get the list of available variables and variable paths from the input data product
<code>parse_var_list</code> (varlist[, tiered, tiered_vars])	Parse a list of path strings into tiered lists and names of variables

continues on next page

Table 14 – continued from previous page

<code>remove([all, var_list, beam_list, keyword_list])</code>	Remove the variables and paths from the wanted list using user specified beam, keyword,
---	---

Attributes

<code>auth</code>	Authentication object returned from <code>earthaccess.login()</code> which stores user authentication.
<code>path</code>	
<code>product</code>	
<code>s3login_credentials</code>	A dictionary which stores login credentials for AWS s3 access.
<code>session</code>	Earthaccess session object for connecting to Earth-data resources.
<code>version</code>	

11.3.2 Attributes

<code>Variables.path</code>
<code>Variables.product</code>
<code>Variables.version</code>

`icepyx.Variables.path`

property `Variables.path`

`icepyx.Variables.product`

property `Variables.product`

`icepyx.Variables.version`

property `Variables.version`

11.3.3 Methods

<code>Variables.avail([options, internal])</code>	Get the list of available variables and variable paths from the input data product
<code>Variables.parse_var_list(varlist[, tiered, ...])</code>	Parse a list of path strings into tiered lists and names of variables
<code>Variables.append([defaults, var_list, ...])</code>	Add to the list of desired variables using user specified beams and variable list.
<code>Variables.remove([all, var_list, beam_list, ...])</code>	Remove the variables and paths from the wanted list using user specified beam, keyword,

icepyx.Variables.avail

`Variables.avail(options=False, internal=False)`

Get the list of available variables and variable paths from the input data product

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'], ↵
↳version='5')
>>> reg_a.variables.avail()
['ancillary_data/atlas_sdp_gps_epoch',
'ancillary_data/control',
'ancillary_data/data_end_utc',
'ancillary_data/data_start_utc',
.
.
.
'quality_assessment/gt3r/signal_selection_source_fraction_3']
```

icepyx.Variables.parse_var_list

`static Variables.parse_var_list(varlist, tiered=True, tiered_vars=False)`

Parse a list of path strings into tiered lists and names of variables

Parameters

- **varlist** (*list[str]*) – List of full variable paths to be parsed.
- **tiered** (*bool, default True*) – Whether to return the paths (sans variable name) as a nested list of component strings (e.g. `[[‘orbit_info’, ‘ancillary_data’, ‘gt1l’], [‘none’, ‘none’, ‘land_ice_segments’]]`) or a single list of path strings (e.g. `[‘orbit_info’, ‘ancillary_data’, ‘gt1l/land_ice_segments’]`)
- **tiered_vars** (*bool, default False*) – Whether or not to append a list of the variable names to the nested list of component strings (e.g. `[[‘orbit_info’, ‘ancillary_data’, ‘gt1l’], [‘none’, ‘none’, ‘land_ice_segments’], [‘sc_orient’, ‘atlas_sdp_gps_epoch’, ‘h_li’]]`)

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'], ↵
↳version='1')
>>> var_dict, paths = reg_a.variables.parse_var_list(reg_a.variables.avail())
>>> var_dict
{'atlas_sdp_gps_epoch': ['ancillary_data/atlas_sdp_gps_epoch'],
.
.
.
'latitude': ['gt1l/land_ice_segments/latitude',
'gt1r/land_ice_segments/latitude',
'gt2l/land_ice_segments/latitude',
```

```

'gt2r/land_ice_segments/latitude',
'gt3l/land_ice_segments/latitude',
'gt3r/land_ice_segments/latitude'],
.
.
.
}
>>> var_dict.keys()
dict_keys(['atlas_sdp_gps_epoch', 'control', 'data_end_utc', 'data_start_utc',
'end_cycle', 'end_delta_time', 'end_geoseg', 'end_gpssow', 'end_gpsweek',
'end_orbit', 'end_region', 'end_rgt', 'granule_end_utc', 'granule_start_utc',
'qa_at_interval', 'release', 'start_cycle', 'start_delta_time', 'start_geoseg',
'start_gpssow', 'start_gpsweek', 'start_orbit', 'start_region', 'start_rgt',
'version', 'dt_hist', 'fit_maxiter', 'fpb_maxiter', 'maxiter', 'max_res_ids',
'min_dist', 'min_gain_th', 'min_n_pe', 'min_n_sel', 'min_signal_conf', 'n_hist',
'nhist_bins', 'n_sigmas', 'proc_interval', 'qs_lim_bsc', 'qs_lim_hrs', 'qs_lim_
→hsigma',
'qs_lim_msw', 'qs_lim_snr', 'qs_lim_sss', 'rbin_width', 'sigma_beam', 'sigma_tx',
't_dead', 'atl06_quality_summary', 'delta_time', 'h_li', 'h_li_sigma', 'latitude
→',
'longitude', 'segment_id', 'sigma_geo_h', 'fpb_mean_corr', 'fpb_mean_corr_sigma',
'fpb_med_corr', 'fpb_med_corr_sigma', 'fpb_n_corr', 'med_r_fit', 'tx_mean_corr',
tx_med_corr', 'dem_flag', 'dem_h', 'geoid_h', 'dh_fit_dx', 'dh_fit_dx_sigma', '
dh_fit_dy', 'h_expected_rms', 'h_mean', 'h_rms_misfit', 'h_robust_sprd',
'n_fit_photons', 'n_seg_pulses', 'sigma_h_mean', 'signal_selection_source',
'signal_selection_source_status', 'snr', 'snr_significance', 'w_surface_window_
→final',
ckgrd', 'bsnow_conf', 'bsnow_h', 'bsnow_od', 'cloud_flg_asr', 'cloud_flg_atm',
→'dac',
'e_bckgrd', 'layer_flag', 'msw_flag', 'neutat_delay_total', 'r_eff', 'solar_
→azimuth',
'solar_elevation', 'tide_earth', 'tide_equilibrium', 'tide_load', 'tide_ocean',
'tide_pole', 'ref_azimuth', 'ref_coelv', 'seg_azimuth', 'sigma_geo_at', 'sigma_
→geo_r',
igma_geo_xt', 'x_atc', 'y_atc', 'bckgrd_per_m', 'bin_top_h', 'count', 'ds_
→segment_id',
'lat_mean', 'lon_mean', 'pulse_count', 'segment_id_list', 'x_atc_mean', 'record_
→number',
'reference_pt_lat', 'reference_pt_lon', 'signal_selection_status_all',
'signal_selection_status_backup', 'signal_selection_status_confident', 'crossing_
→time',
'cycle_number', 'lan', 'orbit_number', 'rgt', 'sc_orient', 'sc_orient_time',
'qa_granule_fail_reason', 'qa_granule_pass_fail', 'signal_selection_source_
→fraction_0',
'signal_selection_source_fraction_1', 'signal_selection_source_fraction_2',
'signal_selection_source_fraction_3'])
>>> import numpy
>>> numpy.unique(paths)
array(['ancillary_data', 'bias_correction', 'dem', 'fit_statistics',
'geophysical', 'ground_track', 'gt1l', 'gt1r', 'gt2l', 'gt2r',
'gt3l', 'gt3r', 'land_ice', 'land_ice_segments', 'none',
'orbit_info', 'quality_assessment', 'residual_histogram',
'segment_quality', 'signal_selection_status'], dtype='<U23')

```

icepyx.Variables.append

`Variables.append(defaults=False, var_list=None, beam_list=None, keyword_list=None)`

Add to the list of desired variables using user specified beams and variable list. A pregenerated default variable list can be used by setting defaults to True. Note: The calibrated backscatter `cab_prof` is not in the default list for ATL09

Parameters

- **defaults** (*bool*, *default False*) – Include the variables in the default variable list. Defaults are defined per-data product. When specified in conjunction with a `var_list`, default variables not on the user- specified list will be added to the order.
- **var_list** (*list[str]*, *default None*) – A list of variables to request, if not all available variables are wanted. A list of available variables can be obtained by entering `var_list=['']` into the function.
- **beam_list** (*list[str]*, *default None*) – A list of beam strings, if only selected beams are wanted (the default value of None will automatically include all beams). For ATL09, acceptable values are ['profile_1', 'profile_2', 'profile_3']. For ATL11, acceptable values are ['pt1', 'pt2', 'pt3']. For all other products, acceptable values are ['gt1l', 'gt1r', 'gt2l', 'gt2r', 'gt3l', 'gt3r'].
- **keyword_list** (*list[str]*, *default None*) – A list of subdirectory names (keywords), from any hierarchy level within the data structure, to select variables within the product that include that keyword in their path. A list of available keywords can be obtained by entering `keyword_list=['']` into the function.

Notes

See also the [IS2_data_access2-subsetting](#) example notebook

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
```

To add all variables related to a specific ICESat-2 beam

```
>>> reg_a.variables.append(beam_list=['gt1r'])
```

To include the default variables:

```
>>> reg_a.variables.append(defaults=True)
```

To add specific variables in orbit_info

```
>>> reg_a.variables.append(keyword_list=['orbit_info'], var_list=['sc_orient_time
→'])
```

To add all variables and paths in ancillary_data

```
>>> reg_a.variables.append(keyword_list=['ancillary_data'])
```

icepyx.Variables.remove

Variables.**remove**(*all=False, var_list=None, beam_list=None, keyword_list=None*)

Remove the variables and paths from the wanted list using user specified beam, keyword, and variable lists.

Parameters

- **all** (*bool, default False*) – Remove all variables and paths from the wanted list.
- **var_list** (*list[str], default None*) – A list of variables to request, if not all available variables are wanted. A list of available variables can be obtained by entering `var_list=['']` into the function.
- **beam_list** (*list[str], default None*) – A list of beam strings, if only selected beams are wanted (the default value of None will automatically include all beams). For ATL09, acceptable values are ['profile_1', 'profile_2', 'profile_3']. For ATL11, acceptable values are ['pt1', 'pt2', 'pt3']. For all other products, acceptable values are ['gt1l', 'gt1r', 'gt2l', 'gt2r', 'gt3l', 'gt3r'].
- **keyword_list** (*list[str], default None*) – A list of subdirectory names (keywords), from any hierarchy level within the data structure, to select variables within the product that include that keyword in their path.

Notes

See also the [IS2_data_access2-subsetting](#) example notebook

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
```

To clear the list of wanted variables

```
>>> reg_a.variables.remove(all=True)
```

To remove all variables related to a specific ICESat-2 beam

```
>>> reg_a.variables.remove(beam_list=['gt1r'])
```

To remove specific variables in orbit_info

```
>>> reg_a.variables.remove(keyword_list=['orbit_info'], var_list=['sc_orient_time
→'])
```

To remove all variables and paths in ancillary_data

```
>>> reg_a.variables.remove(keyword_list=['ancillary_data'])
```

11.4 Components

11.4.1 APIformatting

Generate and format information for submitting to API (CMR and NSIDC).

`class icepyx.core.APIformatting.Parameters(partype, values=None, reqtype=None)`

Bases: `Generic[T]`

Build and update the parameter lists needed to submit a data order

Parameters

- **partype** (*str*) – Type of parameter list. Must be one of ['CMR', 'required', 'subset']
- **values** (*dict*, *default None*) – Dictionary of already-formatted parameters, if there are any, to avoid re-creating them.
- **reqtype** (*str*, *default None*) – For *partype*=='required', indicates which parameters are required based on the type of query. Must be one of ['search', 'download']

`build_params(**kwargs)`

Build the parameter dictionary of formatted key:value pairs for submission to NSIDC in the data request.

Parameters

****kwargs** – Keyword inputs containing the needed information to build the parameter list, depending on parameter type, if the already formatted key:value is not submitted as a kwarg. May include optional keyword arguments to be passed to the subsetter. Valid keywords are time, bbox OR Boundingshape, format, projection, projection_parameters, and Coverage.

Keyword argument inputs for 'CMR' may include: start, end, extent_type, spatial_extent Keyword argument inputs for 'required' may include: product or short_name, version, page_size, page_num, request_mode, include_meta, client_string Keyword argument inputs for 'subset' may include: geom_filepath, start, end, extent_type, spatial_extent

Return type

`None`

`check_req_values()`

Check that all of the required keys have values, if the key was passed in with the values parameter.

Return type

`bool`

`check_values()`

Check that the non-required keys have values, if the key was passed in with the values parameter.

Return type

`bool`

fnted_keys

Enable the Parameters class' fnted_keys property to be typechecked correctly.

See: <https://github.com/microsoft/pyright/issues/3071#issuecomment-1043978070>

partype: `T`

property poss_keys: `dict[str, list[str]]`

Returns a list of possible input keys for the given parameter object. Possible input keys depend on the parameter type (partype).

`icepyx.core.APIformatting.combine_params(*param_dicts)`

Combine multiple dictionaries into one.

Merging is performed in sequence using `dict.update()`; dictionaries later in the list overwrite those earlier.

Parameters

params (*dictionaries*) – Unlimited number of dictionaries to combine

Return type

A single dictionary of all input dictionaries combined

Examples

```
>>> CMRparams = {'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
→ 'bounding_box': '-55,68,-48,71'}
>>> reqparams = {'short_name': 'ATL06', 'version': '002', 'page_size': 2000,
→ 'page_num': 1}
>>> ipx.core.APIformatting.combine_params(CMRparams, reqparams)
{'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
'bounding_box': '-55,68,-48,71',
'short_name': 'ATL06',
'version': '002',
'page_size': 2000,
'page_num': 1}
```

`icepyx.core.APIformatting.to_string(params)`

Combine a parameter dictionary into a single url string

Parameters

params (*dict*)

Return type

url string of input dictionary (not encoded)

Examples

```
>>> CMRparams = {'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
...             'bounding_box': '-55,68,-48,71'}
>>> reqparams = {'short_name': 'ATL06', 'version': '002', 'page_size': 2000,
→ 'page_num': 1}
>>> params = ipx.core.APIformatting.combine_params(CMRparams, reqparams)
>>> ipx.core.APIformatting.to_string(params)
'temporal=2019-02-20T00:00:00Z,2019-02-28T23:59:59Z&bounding_box=-55,68,-48,71&
→short_name=ATL06&version=002&page_size=2000&page_num=1'
```

11.4.2 EarthdataAuthMixin

exception `icepyx.core.auth.AuthenticationError`

Bases: `Exception`

Raised when an error is encountered while authenticating Earthdata credentials

class `icepyx.core.auth.EarthdataAuthMixin(auth=None)`

Bases: `object`

This mixin class generates the needed authentication sessions and tokens, including for NASA Earthdata cloud access. Authentication is completed using the [earthaccess library](<https://nsidc.github.io/earthaccess/>). Methods for authenticating are:

1. Storing credentials as environment variables (`$EARTHDATA_LOGIN` and `$EARTHDATA_PASSWORD`)
2. Entering credentials interactively

3. Storing credentials in a .netrc file (not recommended for security reasons)

More details on using these methods is available in the [earthaccess documentation](https://nsidc.github.io/earthaccess/tutorials/restricted-datasets/#auth).

This class can be inherited by any other class that requires authentication. For example, the *Query* class inherits this one, and so a Query object has the *.session* property.

The class can be created without any initialization parameters, and the properties will be populated when they are called. It can alternately be initialized with an `earthaccess.auth.Auth` object, which will then be used to create a session or `s3login_credentials` as they are called.

Parameters

auth (*earthaccess.auth.Auth*, *default None*) – Optional parameter to initialize an object with existing credentials.

Examples

```
>>> a = EarthdataAuthMixin()
>>> a.session
>>> a.s3login_credentials
```

property auth

Authentication object returned from `earthaccess.login()` which stores user authentication.

property s3login_credentials

A dictionary which stores login credentials for AWS s3 access. This property is accessed if using AWS cloud data.

Because s3 tokens are only good for one hour, this function will automatically check if an hour has elapsed since the last token use and generate a new token if necessary.

property session

Earthaccess session object for connecting to Earthdata resources.

11.4.3 granules

class icepyx.core.granules.Granules

Bases: *EarthdataAuthMixin*

Interact with ICESat-2 data granules. This includes finding, ordering, and downloading them as well as (not yet implemented) getting already downloaded granules into the query object.

Return type

Granules object

download(verbose, path, restart=False)

Downloads the data for the object's orderIDs, which are generated by ordering data from the NSIDC.

Parameters

- **verbose** (*bool*, *default False*) – Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of verbose.
- **path** (*str*) – String with complete path to desired download directory and location.
- **restart** (*bool*, *default False*) – Restart your download if it has been interrupted. If the kernel has been restarted, but you successfully completed your order, you will

need to re-initialize your query class object and can then skip immediately to the `download_granules` method with `restart=True`.

Notes

This function is used by `query.Query.download_granules()`, which automatically feeds in the required parameters.

See also

`query.Query.download_granules`

`get_avail(CMRparams, cloud=False)`

Get a list of available granules for the query object's parameters. Generates the *avail* attribute of the granules object.

Parameters

- **CMRparams** (`CMRParamsWithBbox` | `CMRParamsWithPolygon`) – Dictionary of properly formatted CMR search parameters.
- **cloud** (`bool`, default: `False`) – CMR metadata is always collected for the cloud system. Deprecated since version 1.2: This parameter is ignored.

Notes

This function is used by `query.Query.avail_granules()`, which automatically feeds in the required parameters.

See also

`APIformatting.Parameters`, `query.Query.avail_granules`

`place_order(CMRparams, subsetparams, verbose, subset=True, geom_filepath=None)`

Place an order for the available granules for the query object. Adds the list of zipped files (orders) to the granules data object (which is stored as the *granules* attribute of the query object). You must be logged in to Earthdata to use this function.

Parameters

- **CMRparams** (`CMRParamsWithBbox` | `CMRParamsWithPolygon`) – Dictionary of properly formatted CMR search parameters.
- **subsetparams** (*dict*) – Dictionary of properly formatted subsetting parameters. An empty dictionary is passed as input here when subsetting is set to `False` in query methods.
- **verbose** (*bool*, default *False*) – Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of `verbose`.
- **subset** (*bool*, default *True*) – Apply subsetting to the data order from the NSIDC, returning only data that meets the subset parameters. Spatial and temporal subsetting based

on the input parameters happens by default when `subset=True`, but additional subsetting options are available. Spatial subsetting returns all data that are within the area of interest (but not complete granules. This eliminates false-positive granules returned by the metadata-level search)

- **geom_filepath** (*str*, *default None*) – String of the full filename and path when the spatial input is a file.

Notes

This function is used by `query.Query.order_granules()`, which automatically feeds in the required parameters.

See also

`query.Query.order_granules`

`icepyx.core.granules.gran_IDS(grans, ids=False, cycles=False, tracks=False, dates=False, cloud=False)`

Returns a list of granule information for each granule dictionary in the input list of granule dictionaries. Granule info may be from a list of those available from NSIDC (for ordering/download) or a list of granules present on the file system.

Parameters

- **grans** (*list of dictionaries*) – List of input granule json dictionaries. Must have key “producer_granule_id”
- **ids** (*bool*, *default True*) – Return a list of the available granule IDs for the granule dictionary
- **cycles** (*bool*, *default False*) – Return a list of the available orbital cycles for the granule dictionary
- **tracks** (*bool*, *default False*) – Return a list of the available Reference Ground Tracks (RGTs) for the granule dictionary
- **dates** (*bool*, *default False*) – Return a list of the available dates for the granule dictionary.
- **cloud** (*bool*, *default False*) – Return a a list of AWS s3 urls for the available granules in the granule dictionary.

`icepyx.core.granules.info(grans)`

Return some basic summary information about a set of granules for an query object. Granule info may be from a list of those available from NSIDC (for ordering/download) or a list of granules present on the file system.

11.4.4 harmony

`class icepyx.core.harmony.HarmonyApi`

Bases: `EarthdataAuthMixin`

A client for interacting with the NASA Harmony API.

harmony_client

The Harmony API client.

Type

harmony.Client

job_ids

List of job IDs that have been placed with the Harmony API.

Type

list[str]

check_order_status(*job_id*)

Check the status of a submitted Harmony job.

Parameters**job_id** (*str*)**Return type**

dict[str, Any]

download_granules(*download_dir*, *overwrite=False*)

Download all granules associated with current order.

This method retrieves and downloads granules for all job IDs stored in *self.job_ids*, saving them to the specified directory.**Parameters**

- **download_dir** (*Path*) – The directory where granules should be saved.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files (default is False).

Returns

A list of file paths to the downloaded granules.

Return type

list of Path

get_capabilities(*concept_id*)

Retrieve the capabilities of a dataset given its concept ID.

Parameters**concept_id** (*str*) – The concept ID of the dataset.**Returns**

A dictionary containing dataset capabilities.

Return type

dict

pause_order(*job_id*)

Pauses an order that is currently processing.

Parameters**job_id** (*str*) – The ID of the Harmony job to resume.**Returns**

A dictionary containing the order status and related metadata.

Return type

dict

place_order(*concept_id*, *spatial=None*, *temporal=None*, *shape=None*, *granule_name=[]*, *skip_preview=False*)

Submit an order to Harmony and wait for it to complete.

Parameters

- **concept_id** (*str*) – The concept ID of the dataset.
- **spatial** (*harmony.BBox, str, harmony.request.WKT, or None, optional*) – The spatial extent for the order.
- **temporal** (*HarmonyTemporal or None, optional*) – The temporal range for the order.
- **shape** (*str or None, optional*) – A spatial shape file for filtering.
- **granule_name** (*list[str], optional*) – Specific granule names to include in the order.
- **skip_preview** (*bool, optional*) – Whether to bypass preview mode if the order exceeds 300 granules.

Returns

The Harmony job ID.

Return type

str

resume_order(*job_id*)

Resume processing of an order that is paused.

Parameters

job_id (*str*) – The ID of the Harmony job to resume.

Returns

A dictionary containing the order status and related metadata.

Return type

dict

skip_preview(*job_id*)

Resume processing of an order that is in the “PREVIEW” state.

If a subsetting order exceeds 300 granules, Harmony places it in a paused state called “preview,” where only a few granules are processed initially. This method resumes processing by first pausing and then resuming the order.

Parameters

job_id (*str*) – The ID of the Harmony job to resume.

Returns

A dictionary containing the order status and related metadata.

Return type

dict

class icepyx.core.harmony.HarmonyTemporal

Bases: *TypedDict*

start: *datetime*

stop: *datetime*

11.4.5 is2ref

`icepyx.core.is2ref.about_product(prod)`

Ping Earthdata to get metadata about the product of interest (the collection).

➔ See also

`query.Query.product_all_info`

`icepyx.core.is2ref.extract_product(filepath, auth=None)`

Read the product type from the metadata of the file. Valid for local or s3 files, but must provide an auth object if reading from s3. Return the product as a string.

Parameters

- **filepath** (*str*) – local or remote location of a file. Could be a local string or an s3 filepath
- **auth** (*earthaccess.auth.Auth*, *default None*) – An earthaccess authentication object. Optional, but necessary if accessing data in an s3 bucket.

`icepyx.core.is2ref.extract_version(filepath, auth=None)`

Read the version from the metadata of the file. Valid for local or s3 files, but must provide an auth object if reading from s3. Return the version as a string.

Parameters

- **filepath** (*str*) – local or remote location of a file. Could be a local string or an s3 filepath
- **auth** (*earthaccess.auth.Auth*, *default None*) – An earthaccess authentication object. Optional, but necessary if accessing data in an s3 bucket.

`icepyx.core.is2ref.gt2spot(gt, sc_orient)`

`icepyx.core.is2ref.latest_version(product)`

Determine the most recent version available for the given product.

📘 Examples

```
>>> latest_version('ATL03')
'007'
```

11.4.6 orders

class `icepyx.core.orders.DataOrder(job_id, type, granules, harmony_client)`

Bases: `object`

A class representing an order for Harmony data processing.

Parameters

- **job_id** (*str*)
- **type** (*str*)
- **granules** (*List[Any] | Granules*)
- **harmony_client** (*Any*)

job_id

The ID of the Harmony job.

Type

str

Return type

str

type

The type of order (e.g., “subset”).

Type

str

granules

A list of granules included in the order.

Type

list

harmony_client

The Harmony API client used for interacting with the service.

Type

object

HARMONY_BASE_URL = 'https://harmony.earthdata.nasa.gov/workflow-ui/'

download(*path*, *overwrite=False*)

Download the granules for the order, blocking until they are ready if necessary.

Parameters

- **path** (*str* or *Path*) – The directory where granules should be saved.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files (default is False).

Returns

A list of downloaded granules

Return type

list or None

download_granules(*path*, *overwrite=False*)

Download the granules for the order.

Parameters

- **path** (*str* or *Path*) – The directory where granules should be saved.
- **overwrite** (*bool*, *optional*) – Whether to overwrite existing files (default is False).

Returns

A list of downloaded granules

Return type

list or None

job_id()

Get the job ID of the order.

Returns

The Harmony job ID.

Return type

str

pause()

Pause the order.

Returns

The response from the Harmony API if the order is paused, otherwise None.

Return type

dict or None

resume()

Resume the order if it has been paused or is on a “preview” state.

Returns

The response from the Harmony API if the order is resumed, otherwise None.

Return type

dict or None

skip_preview()

Resume the order if it has been paused due to exceeding granule limits.

Returns

The response from the Harmony API if the order is resumed, otherwise None.

Return type

dict or None

status()

Retrieve the status of the order.

Returns

A dictionary containing the order status and related metadata.

Return type

dict

11.4.7 spatial

class icepyx.core.spatial.Spatial(*spatial_extent*, ***kwarg*)

Bases: object

property extent

Return the coordinates of the spatial extent of the Spatial object.

The result will be returned as an array. For input geometry files with multiple features, the boundary of the the unary union of all features is returned.

Returns**spatial extent** – An array of bounding coordinates.**Return type**

array

property extent_as_gdf

Return the spatial extent of the query object as a GeoPandas GeoDataframe.

Returns**extent_gdf** – A GeoDataframe containing the spatial region of interest.

Return type

geopandas.GeoDataFrame

property extent_file

Return the path to the geospatial polygon file containing the Spatial object's spatial extent. If the spatial extent did not come from a file (i.e. user entered list of coordinates), this will return None.

Examples

```
>>> reg_a = Spatial([-55, 68, -48, 71])
>>> reg_a.extent_file
```

```
>>> from pathlib import Path
>>> reg_a = Spatial(str(Path('./doc/source/example_notebooks/supporting_files/
↳ simple_test_poly.gpkg').resolve()))
>>> reg_a.extent_file
./doc/source/example_notebooks/supporting_files/simple_test_poly.gpkg
```

property extent_type

Return the extent type of the Spatial object as a string.

Examples

```
>>> reg_a = Spatial([-55, 68, -48, 71])
>>> reg_a.extent_type
'bounding_box'
```

```
>>> reg_a = Spatial([(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)])
>>> reg_a.extent_type
'polygon'
```

fmt_for_CMR()

Format the spatial extent for NASA's Common Metadata Repository (CMR) API.

CMR spatial inputs must be formatted a specific way. This method formats the given spatial extent to be a valid submission. For large/complex polygons, this includes simplifying the polygon (NOTE: currently not all polygons are simplified enough). Coordinates will be properly ordered, and the required string formatting applied. For small regions, a buffer may be added.

Returns

Properly formatted string of spatial data for submission to CMR API.

Return type

string

fmt_for_EGI()

WARNING: This method is deprecated. Use `fmt_for_CMR` instead. Format the spatial extent input into a subsetting key value for submission to EGI (the NSIDC DAAC API).

EGI spatial inputs must be formatted a specific way. This method formats the given spatial extent to be a valid submission. DevGoal: For large/complex polygons, this includes simplifying the polygon. Coordinates will be properly ordered, and the required string formatting applied.

Returns

Properly formatted json string for submission to CMR

Return type

string

`icepyx.core.spatial.check_dateline(extent_type, spatial_extent)`

Check if a bounding box or polygon input cross the dateline.

Parameters

- **extent_type** (*str*) – One of ‘bounding_box’ or ‘polygon’, indicating what type of input the spatial extent is
- **spatial_extent** (*list*) – A list containing the spatial extent as coordinates in decimal degrees of [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].

Returns

indicating whether or not the spatial extent crosses the dateline.

Return type

boolean

`icepyx.core.spatial.geodataframe(extent_type, spatial_extent, file=False, xdateline=None)`

Return a geodataframe of the spatial extent

Parameters

- **extent_type** (*str*) – One of ‘bounding_box’ or ‘polygon’, indicating what type of input the spatial extent is
- **spatial_extent** (*str or list*) – A list containing the spatial extent OR a string containing a filename. If file is False, spatial_extent should be a list of coordinates in decimal degrees of [lower-left-longitude, lower-left-latitude, upper-right-longitude, upper-right-latitude] or [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].

If file is True, spatial_extent is a string containing the full file path and filename to the file containing the desired spatial extent.
- **file** (*bool, default False*) – Indication for whether the spatial_extent string is a filename or coordinate list

Returns

gdf – Returns a GeoPandas GeoDataFrame containing the spatial extent. The GeoDataFrame will have only one entry unless a geospatial file was submitted.

Return type

`geopandas.GeoDataFrame`

See also

`icepyx.Query`

Examples

```
>>> reg_a = ipx.Query('ATL06', [-55, 68, -48, 71], ['2019-02-20', '2019-02-28'])
>>> gdf = geodataframe(reg_a.spatial.extent_type, reg_a.spatial.extent)
```

```
>>> gdf.geometry
0 POLYGON ((-48 68, -48 71, -55 71, -55 68, -48 ...
Name: geometry, dtype: geometry
```

`icepyx.core.spatial.validate_bounding_box(spatial_extent)`

Validates the `spatial_extent` parameter as a bounding box.

If the `spatial_extent` is a valid bounding box, returns a tuple containing the Spatial object parameters for the bounding box; otherwise, throw an error containing the reason the bounding box is invalid.

Parameters

`spatial_extent` (*list* or *np.ndarray*) – A list or `np.ndarray` of strings, numerics, or tuples representing bounding box coordinates in decimal degrees.

Must be provided in the order: [lower-left-longitude, lower-left-latitude, upper-right-longitude, upper-right-latitude]

`icepyx.core.spatial.validate_polygon_file(spatial_extent)`

Validates the `spatial_extent` parameter as a polygon from a file.

If the `spatial_extent` parameter contains a valid polygon, returns a tuple containing the Spatial object parameters for the polygon;

otherwise, throw an error containing the reason the polygon/polygon file is invalid.

Parameters

`spatial_extent` (*str*) – A string representing a geospatial polygon file (kml, shp, gpkg) * must provide full file path * recommended for file to only contain 1 polygon.

- if multiple polygons, only the first polygon is selected at this time.

`icepyx.core.spatial.validate_polygon_list(spatial_extent)`

Validates the `spatial_extent` parameter as a polygon from a list of coordinates.

If the `spatial_extent` is a valid polygon, returns a tuple containing the Spatial object parameters for the polygon;

otherwise, throw an error containing the reason the polygon is invalid.

Parameters

`spatial_extent` (*list* or *np.ndarray*) – A list or `np.ndarray` of strings, numerics, or tuples representing polygon coordinates, provided as coordinate pairs in decimal degrees in the order: [longitude1, latitude1, longitude2, latitude2, longitude_n,latitude_n, longitude1,latitude1]

If the first and last coordinate pairs are NOT equal, the polygon will be closed automatically (last point will be connected to the first point).

`icepyx.core.spatial.validate_polygon_pairs(spatial_extent)`

Validates the `spatial_extent` parameter as a polygon from coordinate pairs.

If the `spatial_extent` is a valid polygon, returns a tuple containing the Spatial object parameters for the polygon;

otherwise, throw an error containing the reason the polygon is invalid.

Parameters

`spatial_extent` (*list* or *np.ndarray*) – A list or `np.ndarray` of tuples representing polygon coordinate pairs in decimal degrees in the order: [(longitude1, latitude1), (longitude2, latitude2), (longitude_n,latitude_n), (longitude1,latitude1)]

If the first and last coordinate pairs are NOT equal, the polygon will be closed automatically (last point will be connected to the first point).

11.4.8 temporal

class icepyx.core.temporal.Temporal(*date_range*, *start_time=None*, *end_time=None*)

Bases: `object`

Parameters

- **date_range** (*list* | *dict*)
- **start_time** (*str* | *time* | *None*)
- **end_time** (*str* | *time* | *None*)

property end: `datetime`

Return the end date and time of the Temporal object as a `datetime.datetime` object.

Examples

```
>>> tmp_a = Temporal(["2016-01-01", "2020-01-01"])
>>> tmp_a.end
datetime.datetime(2020, 1, 1, 23, 59, 59)
```

property start: `datetime`

Return the start date and time of the Temporal object as a `datetime.datetime` object.

Examples

```
>>> tmp_a = Temporal(["2016-01-01", "2020-01-01"])
>>> tmp_a.start
datetime.datetime(2016, 1, 1, 0, 0)
```

icepyx.core.temporal.**check_valid_date_range**(*start*, *end*)

Helper function for checking if a date range is valid.

AssertionError is raised if the start date is later than the end date.

Parameters

- **start** (*datetime.datetime*, *datetime.date*) – Starting date of date range to check.
- **end** (*datetime.datetime*, *datetime.date*) – Ending date of date range to check

Return type

`bool` (True if date range is valid, False otherwise)

Examples

```
>>> start = dt.datetime.strptime("2016-01-01", "%Y-%m-%d")
>>> end = dt.datetime.strptime("2020-01-01", "%Y-%m-%d")
>>> drange = check_valid_date_range(start, end)
>>> drange
```

```
>>> drange = check_valid_date_range(end, start)
AssertionError: Your date range is invalid; end date MUST be on or after the start_
↳date.
```

`icepyx.core.temporal.convert_string_to_date(date)`

Converts a string to a datetime object. Throws an error if an invalid format is passed in.

Parameters

date (*str*) –

A string representation for the date value. Current supported date formats are:

- "YYYY-MM-DD"
- "YYYY-DOY"

Return type

`datetime.date` object, representing the date from the string parameter.

Examples

```
>>> mmdd = "2016-01-01"
>>> converted = convert_string_to_date(mmdd)
>>> converted
datetime.date(2016, 1, 1)
```

```
>>> doy = "2020-40"
>>> converted = convert_string_to_date(doy)
>>> converted
datetime.date(2020, 2, 9)
```

`icepyx.core.temporal.validate_date_range_date(date_range, start_time=None, end_time=None)`

Validates a date range provided in the form of a list of `datetime.date` objects.

Combines the start and end dates with their respective start and end times to create complete start and end `datetime.datetime` objects.

Parameters

- **date_range** (`list[date]`) – A date range provided in the form of a list of `datetime.dates`. List must be of length 2.
- **start_time** (`str | time | None`)
- **end_time** (`str | time | None`)

Return type

Start and end datetimes as `datetime.datetime` objects

Examples

```
>>> drange = [dt.date(2016, 1, 1), dt.date(2020, 1, 1)]
>>> valid_drange = validate_date_range_date(drange, "00:10:00", "21:00:59")
>>> valid_drange
(datetime.datetime(2016, 1, 1, 0, 10), datetime.datetime(2020, 1, 1, 21, 0, 59))
```

`icepyx.core.temporal.validate_date_range_datestr(date_range, start_time=None, end_time=None)`

Validates a date range provided in the form of a list of strings.

Combines the start and end dates with their respective start and end times to create complete start and end `datetime.datetime` objects.

Parameters

- **date_range** (`list[str]`) –

A date range provided in the form of a list of strings

Strings must be of formats accepted by `validate_inputs_temporal.convert_string_to_date()`.

List must be of length 2.

- **start_time** (`str | time | None`)
- **end_time** (`str | time | None`)

Return type

Start and end dates and times as `datetime.datetime` objects

Examples

```
>>> daterange = validate_date_range_datestr(["2016-01-01", "2020-01-01"])
>>> daterange
(datetime.datetime(2016, 1, 1, 0, 0), datetime.datetime(2020, 1, 1, 23, 59, 59))
```

`icepyx.core.temporal.validate_date_range_datetime(date_range, start_time=None, end_time=None)`

Validates a date range provided in the form of a list of datetimes.

Parameters

- **date_range** (`list[datetime]`) – A date range provided in the form of a list of datetimes. List must be of length 2.
- **NOTE** (If start and/or end times are given,)
- **objects.** (they will be ignored in favor of the time from the start/end `datetime.datetime`)
- **start_time** (`str | time | None`)
- **end_time** (`str | time | None`)

Return type

Start and end dates and times as `datetime.datetime` objects

Examples

```
>>> drange = [dt.datetime(2016, 1, 14, 1, 0, 0), dt.datetime(2020, 2, 9, 13, 10, 1)]
>>> valid_drange = validate_date_range_datetime(drange)
>>> valid_drange
(datetime.datetime(2016, 1, 14, 1, 0), datetime.datetime(2020, 2, 9, 13, 10, 1))
```

`icepyx.core.temporal.validate_date_range_dict(date_range, start_time=None, end_time=None)`

Validates a date range provided in the form of a dict with the following keys:

Parameters

- **date_range** (`dict[str, str | date]`) – A date range provided in the form of a dict. `date_range` must contain only the following keys:

- `start_date`: start date, type can be of `dt.datetime`, `dt.date`, or string

- `end_date`: end date, type can be of `dt.datetime`, `dt.date`, or string

Keys MUST have the exact names/formatting above or a `ValueError` will be thrown by this function.

If the values are of type `dt.datetime` and were created without times, the `datetime` package defaults of all 0s are used and the `start_time`/`end_time` parameters will be ignored!

- **start_time** (`str | time | None`)
- **end_time** (`str | time | None`)

Return type

`tuple[datetime, datetime]`

Returns

- *Start and end datetimes as `datetime.datetime` objects*
- *(by combining the start/end dates with their respective start/end times, if the dict type is not `datetime`)*

Examples

```
>>> drange = {"start_date": "2016-01-01", "end_date": "2020-01-01"}
>>> valid_drange = validate_date_range_dict(drange, "00:00:00", "23:59:59")
>>> valid_drange
(datetime.datetime(2016, 1, 1, 0, 0), datetime.datetime(2020, 1, 1, 23, 59, 59))
```

`icepyx.core.temporal.validate_times(start_time, end_time)`

Validates the start and end times passed into `__init__` and returns them as `datetime.time` objects.

NOTE: If start and/or end times are not provided (are of type `None`), the defaults are `00:00:00` and `23:59:59`, respectively.

Parameters

- **start_time** (`str, datetime.time, None`)
- **end_time** (`string, datetime.time, None`)

Return type

`start_time, end_time` as `datetime.time` objects

Examples

```
>>> val_time = validate_times("00:00:00", "23:59:59")
>>> val_time
(datetime.time(0, 0), datetime.time(23, 59, 59))
```

11.4.9 validate_inputs

`icepyx.core.validate_inputs.check_s3bucket(path)`

Check if the given path is an s3 path. Raise a warning if the data being referenced is not in the NSIDC bucket

`icepyx.core.validate_inputs.cycles(cycle)`

Check if the submitted cycle is valid, and warn the user if not available.

`icepyx.core.validate_inputs.prod_version(latest_vers, version)`

Check if the submitted product version is valid, and warn the user if a newer version is available.

`icepyx.core.validate_inputs.tracks(track)`

Check if the submitted RGT is valid, and warn the user if not available.

11.4.10 visualize

Interactive visualization of spatial extent and ICESat-2 elevations

class `icepyx.core.visualization.Visualize(query_obj=None, product=None, spatial_extent=None, date_range=None, cycles=None, tracks=None)`

Bases: `object`

Object class to quickly visualize elevation data for select ICESat-2 products (ATL06, ATL07, ATL08, ATL10, ATL12, ATL13) based on the query parameters defined by the `icepyx.Query` object. Provides interactive maps that show product elevation on a satellite basemap.

Parameters

- **query_obj** (`icepyx.Query`, *default None*) – `icepyx.Query` class object.
- **product** (*str*) – ICESat-2 product ID
- **spatial_extent** (*list or str*, *default None*) – as in the `icepyx.Query` object
- **date_range** (*list of 'YYYY-MM-DD' strings*, *default None*) – as in the `icepyx.Query` object
- **cycle** (*str*, *default all available orbital cycles*, *default None*) – as in the `icepyx.Query` object
- **track** (*str*, *default all available reference ground tracks (RGTs)*, *default None*) – as in the `icepyx.Query` object

➔ See also

`icepyx.Query`

generate_OA_parameters()

Get metadata from file lists in each 5*5 bounding box.

Returns

paras_list – A list of parameters for OpenAltimetry API query, including the reference ground track (RGT), cycle number, datetime, bounding box, and product name.

Return type

`list`

grid_bbox(*binsize=5*)

Split bounding box into 5 x 5 grids when latitude/longitude range exceeds the default OpenAltimetry 5*5 degree spatial limits

Returns

bbox_list – A list of bounding boxes with a maximum size of 5*5 degree

Return type

list

make_request(*base_url, payload*)

Make HTTP request

Parameters

base_url (*str*) – OpenAltimetry URL

 **See also**

[request_OA_data](#)

parallel_request_OA()

Requests elevation data from OpenAltimetry API in parallel. Currently supports OA_Products ['ATL06','ATL07','ATL08','ATL10','ATL12','ATL13']

For ATL03 Photon Data, OA only supports single date request according to: <https://openaltimetry.org/data/swagger-ui/#/Public/getATL08DataByDate>, with geospatial limitation of 1 degree lat/lon. Visualization of ATL03 data is not implemented within this module at this time.

Returns

OA_data_da – A dask array containing the ICESat-2 data.

Return type

dask.Array

query_icesat2_filelist()

Query list of ICESat-2 files for each bounding box

Returns

filelist_tuple – A tuple of non-empty lists of bounding boxes and corresponding ICESat-2 file lists

Return type

tuple

request_OA_data(*paras*)

Request data from OpenAltimetry based on API: <https://openaltimetry.org/data/swagger-ui/#/>

Parameters

paras (*list*) – A single parameter list for an OpenAltimetry API data request.

Returns

OA_darr – A dask array containing the ICESat-2 elevation data.

Return type

da.array

viz_elevation()

Visualize elevation requested from OpenAltimetry API using datashader based on cycles https://holoviz.org/tutorial/Large_Data.html

Returns

map_cycle, map_rgt + lineplot_rgt – Holoviews data visualization elements

Return type

Holoviews objects

`icepyx.core.visualization.files_in_latest_n_cycles(files, cycles, n=1)`

Get list of file names from latest n ICESat-2 cycles

Parameters

- **files** (*list*) – A list of file names.
- **cycles** (*list*) – A list of available ICESat-2 cycles
- **n** (*int, default 1*) – Number of latest ICESat-2 cycles to pick

Returns

viz_file_list – A list of file names from latest n ICESat-2 cycles

Return type

list

`icepyx.core.visualization.gran_paras(filename)`

Returns a list of granule information for file name string.

Parameters

filename (*str*) – ICESat-2 file name

Returns

gran_paras_list – A list of parameters including RGT, cycle, and datetime of ICESat-2 data granule

Return type

list

`icepyx.core.visualization.user_check(message)`

Check if user wants to proceed visualization when the API request number exceeds 200

Parameters

message (*str*) – Message to indicate users the options

11.5 Types

ICEPYX-QUEST DOCUMENTATION (API)

QUEST and icepyx share the top-level GenQuery class. The documentation for GenQuery are within [icepyx.Query](#).

Class diagram illustrating the QUEST component's of icepyx's public-facing classes, their attributes and methods, and their relationships. Additional UML diagrams, including a more detailed, developer UML class diagram showing hidden parameters, are available on [GitHub in the icepyx/doc/source/user_guide/documentation/ directory](#). Diagrams are updated automatically after a pull request (PR) is approved and before it is merged to the development branch.

12.1 Quest Class

12.1.1 Constructors

<code>GenQuery([spatial_extent, date_range, ...])</code>	Base class for querying data.
<code>Quest(spatial_extent, date_range[, ...])</code>	QUEST - Query Unify Explore SpatioTemporal - object to query, obtain, and perform basic operations on datasets (i.e. Argo, BGC Argo, MODIS, etc) for combined analysis with ICESat-2 data products.

icepyx.Quest

class icepyx.Quest(*spatial_extent, date_range, start_time=None, end_time=None, proj='default'*)

QUEST - Query Unify Explore SpatioTemporal - object to query, obtain, and perform basic operations on datasets (i.e. Argo, BGC Argo, MODIS, etc) for combined analysis with ICESat-2 data products. A new dataset can be added using the `dataset.py` template. QUEST expands the icepyx GenQuery superclass.

See the doc page for GenQuery for details on temporal and spatial input parameters.

Parameters

proj (*proj4 string*) – Geospatial projection. Not yet implemented

Return type

quest object

Examples

Initializing Quest with a bounding box.

```
>>> reg_a_bbox = [-55, 68, -48, 71]
>>> reg_a_dates = ['2019-02-20', '2019-02-28']
>>> reg_a = Quest(spatial_extent=reg_a_bbox, date_range=reg_a_dates)
>>> print(reg_a)
Extent type: bounding_box
```

```
Coordinates: [-55.0, 68.0, -48.0, 71.0]
Date range: (2019-02-20 00:00:00, 2019-02-28 23:59:59)
Data sets: None
```

Add datasets to the quest object.

```
>>> reg_a.datasets = {'ATL07':None, 'Argo':None}
>>> print(reg_a)
Extent type: bounding_box
Coordinates: [-55.0, 68.0, -48.0, 71.0]
Date range: (2019-02-20 00:00:00, 2019-02-28 23:59:59)
Data sets: ATL07, Argo
```

➔ See also

[GenQuery](#)

`__init__(spatial_extent, date_range, start_time=None, end_time=None, proj='default')`

Tells QUEST to initialize data given the user input spatiotemporal data.

Methods

<code>__init__(spatial_extent, date_range[, ...])</code>	Tells QUEST to initialize data given the user input spatiotemporal data.
<code>add_argo([params, presRange])</code>	Adds Argo (including Argo-BGC) to QUEST structure.
<code>add_icesat2(product[, start_time, end_time, ...])</code>	Adds ICESat-2 datasets to QUEST structure.
<code>download_all([path])</code>	Downloads requested dataset(s).
<code>save_all(path)</code>	Saves all datasets according to their respective <code>.save()</code> functionality.
<code>search_all(**kwargs)</code>	Searches for required dataset within platform (i.e. ICESat-2, Argo) of interest.

Attributes

<code>dates</code>	Return an array showing the date range of the query object.
<code>end_time</code>	Return the end time specified for the end date.
<code>spatial</code>	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.

continues on next page

Table 3 – continued from previous page

<code>spatial_extent</code>	Return an array showing the spatial extent of the query object. Spatial extent is returned as an input type (which depends on how you initially entered your spatial data) followed by the geometry data. Bounding box data is [lower-left-longitude, lower-left-latitude, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n, latitude_n, longitude1, latitude1].
<code>start_time</code>	Return the start time specified for the start date.
<code>temporal</code>	Return the Temporal object containing date/time range information for the query object.

12.1.2 Methods

<code>Quest.add_icesat2</code> (product[, start_time, ...])	Adds ICESat-2 datasets to QUEST structure.
<code>Quest.add_argo</code> ([params, presRange])	Adds Argo (including Argo-BGC) to QUEST structure.
<code>Quest.search_all</code> (**kwargs)	Searches for required dataset within platform (i.e. ICESat-2, Argo) of interest.
<code>Quest.download_all</code> ([path])	Downloads requested dataset(s).
<code>Quest.save_all</code> (path)	Saves all datasets according to their respective <code>.save()</code> functionality.

icepyx.Quest.add_icesat2

`Quest.add_icesat2`(product, start_time=None, end_time=None, version=None, cycles=None, tracks=None, files=None, **kwargs)

Adds ICESat-2 datasets to QUEST structure.

Parameters

- **inputs** (For details on)
- **documentation.** (see the Query)

Return type

None

➔ See also

`icepyx.core.GenQuery`, `icepyx.core.Query`

icepyx.Quest.add_argo

`Quest.add_argo`(params=['temperature'], presRange=None)


Adds Argo (including Argo-BGC) to QUEST structure.

Parameters

- **inputs** (For details on)
- **documentation.** (see the Argo dataset script)

Return type

None

 **See also**`quest.dataset_scripts.argo, icepyx.query.GenQuery` **Examples**

```
# example with profiles available >>> reg_a = Quest([-154, 30,-143, 37], ['2022-04-12', '2022-04-26']) >>>
reg_a.add_argo(params=["temperature", "salinity"])
```

icepyx.Quest.search_all`Quest.search_all(**kwargs)`

Searches for required dataset within platform (i.e. ICESat-2, Argo) of interest.

Parameters

****kwargs** (*default None*) – Optional passing of keyword arguments to supply additional search constraints per datasets. Each key must match the dataset name (e.g. “icesat2”, “argo”) as in `quest.datasets.keys()`, and the value is a dictionary of acceptable keyword arguments and values allowable for the `search_data()` function for that dataset. For instance: `icesat2 = {"IDs":True}`, `argo = {"presRange": "10,500"}`.

icepyx.Quest.download_all`Quest.download_all(path="", **kwargs)`

Downloads requested dataset(s).

Parameters

****kwargs** (*default None*) – Optional passing of keyword arguments to supply additional search constraints per datasets. Each key must match the dataset name (e.g. “icesat2”, “argo”) as in `quest.datasets.keys()`, and the value is a dictionary of acceptable keyword arguments and values allowable for the `search_data()` function for that dataset. For instance: `icesat2 = {"verbose":True}`, `argo = {"keep_existing":True}`.

icepyx.Quest.save_all`Quest.save_all(path)`Saves all datasets according to their respective `.save()` functionality.**Parameters**

path (*str*) – Path at which to save the dataset files.

ICEPYX CHANGELOG

This is the list of changes made to icepyx in between each release. Full details can be found in the [commit logs](#).

13.1 Latest Release (Version 2.0.2)

13.1.1 What's new in 2.0.2 (13 March 2026)

These are the changes in icepyx 2.0.2 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- add issue templates for bugs and features (#728)

Bug fixes

- fix tests failing due to version updates at NSIDC and update requirements (#727)
- fix download function to order granules if needed (#726)
 - updates the download function to first order data if the user has not explicitly run the order function, returning to the intended behavior of icepyx, as in v1.
- fix version in query, add tests (#741)

Deprecations

- None

Maintenance

- [pre-commit.ci] pre-commit autoupdate (#725)
- Bump the github-actions group with 2 updates (#724)
- [pre-commit.ci] pre-commit autoupdate (#738)
- Bump the github-actions group across 1 directory with 6 updates (#739)
- modify tests to handle retired data product versions (#746)
- [pre-commit.ci] pre-commit autoupdate (#743)

Documentation

- update references citing icepyx through Feb 2026 (#745)
- update traffic through March 2026 (#747)

Other

- get doctests running as part of test actions and update to v007 in tests (#713)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.2 Version 2.0.1

13.2.1 What’s new in 2.0.1 (21 November 2025)

These are the changes in icepyx 2.0.1 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- Update fsspec parameters for cloud reads (#677)

Bug fixes

- Resolve GeoPandas library unary_union warnings (#681)
- update kwargs in pypistats calls to get the script working again (#714)
- [bug fix] remove incorrect passing of “no orbital parameters set” string to “readable_granule_name” (#712)
 - re-add cycle and track validation removed in #657
 - simplify track and cycle properties
 - change cycle and track return to None if None is input
 - add test for track submission
- Updating earthaccess and fixing a few minor bugs. (#715)
 - update earthaccess to latest version
 - fix order url generation
 - fix deprecated calls to *vars* in data read-in notebook

Maintenance

- [pre-commit.ci] pre-commit autoupdate (#671)
- [pre-commit.ci] pre-commit autoupdate (#693)
- Bump codecov/codecov-action from 5.4.0 to 5.4.2 in the github-actions group (#679)
- [pre-commit.ci] pre-commit autoupdate (#700)
- Bump the github-actions group with 3 updates (#707)
- [pre-commit.ci] pre-commit autoupdate and add fetch depth to integration test action (#702)
- Test actions updates and cleanup (#696)

- Bump the github-actions group, remove composite action, pin actions to full length commit SHA (#710)
 - Remove composite action, inline into workflows directly: Dependabot did not update the actions/setup-python version in the composite action
 - Pin actions to full length commit SHA version (more secure): Xref [<https://docs.zimor.sh/audits/#unpinned-uses>] (<https://docs.zimor.sh/audits/#unpinned-uses>) and [<https://docs.github.com/en/actions/reference/security/secure-use#using-third-party-actions>] (<https://docs.github.com/en/actions/reference/security/secure-use#using-third-party-actions>)[pre-commit.ci] pre-commit autoupdate (#711)
- Bump the github-actions group with 2 updates (#718)
- [pre-commit.ci] pre-commit autoupdate (#719)

Documentation

- docs: add chuckwondo as a contributor for review (#667)
- Add intersphinx mappings (#680)
- docs: add emmanuel-ferdman as a contributor for code (#697)
- migrate IS2 resource guide to central community page (#706)
- add quest paper to docs (#709)
- [docs] traffic updates Apr-Oct 2025 (#716)

Other

- Separate jobs for publishing to TestPyPI and PyPI (#687)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.3 Version 2.0.0

13.3.1 What’s new in 2.0.0 (16 May 2025)

These are the changes in icepyx 2.0.0 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

Warning

This is major release and contains a few breaking changes. Full details will be described in the documentation.

NSIDC (the NASA DAAC that hosts ICESat-2 data) is transitioning their API endpoint from the on-premises EGI system to the cloud-based Harmony services. icepyx has been updated to access these new services; Certain capabilities are temporarily going away

- Data reformatting: users will only get HDF5 files back from the new API.
- Variable subsetting: The new cloud subsetter doesn’t support variable subsetting (yet), therefore all variables will be included in subsetted orders.

Find more details on all these changes, including their progress, on GitHub issues and discussions.

New Features

- Adding support for NASA's Harmony subsetter (#657)
 - Data orders can be previewed, stopped and resumed using the library or the Harmony UI.
 - The Harmony trajectory subsetter automatically pauses orders with more than 300 granules, we need to manually resume them or pass the `skip_preview=True` parameter to the order.
 - Reference: <https://harmony.earthdata.nasa.gov/>
- Order class: when an order is placed, icepyx will return an order class instance that will help us keep track of the order progress, pause and eventually download the data. We can still use the Query class to download, however using the order instance is probably simpler and more intuitive. (#657)

Bug fixes

- None

Deprecations

- EGI API is deprecated (Closes #530)
 - PR #657 implements NASA Harmony support to replace the on-premises EGI service.
 - data reformatting and variable subsetting are not supported in the new API. - Users will only get HDF5 files back from the new API.
 - `order_vars()` is deprecated in favor of `variables()`.
 - `fmt_for_EGI()` is deprecated.
 - Any EGI specific code is deprecated and has been removed in this release.
- Drop support for Python 3.9 and 3.10 (#655)
- Removed existing deprecations for v1.x (#640)

Maintenance

- Separate CI unit tests and integration tests with their own confest (#657)
- Fixed some integration tests that were failing due to the new Harmony API #657)
- Updated ruff action (#654)

Documentation

- Updated notebooks to reflect the new Harmony API usage (#657)
 - The updated notebooks include instructions for making full and subset orders, and show how to use the new Harmony API.
 - The top level API works the same, users only need to update to v2.0.0 and the code should work (unless using variable subsetting or reformatting)
 - Notebooks are in the `doc/source/example_notebooks` folder, and are also available on the Readthedocs documentation page.
- Include quest tile in index.rst (#497)

Other

- None

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.4 Version 1.3.0

13.4.1 What’s new in 1.3.0 (13 September 2024)

These are the changes in icepyx 1.3.0 See [icepyx ChangeLog](#) for a full changelog including other versions of icepyx.

Warning

This is anticipated to be the last release in v1 of icepyx.

NSIDC (the NASA DAAC that hosts ICESat-2 data) is transitioning their API endpoint from the on-premises EGI system to the cloud-based Harmony services. icepyx is being updated to access these new services; the only change users will experience concerns services that are no longer available (e.g. reformatting).

The developers plan to use this version change to implement additional breaking changes concerning the Query module, transitioning it’s functionality to use the earthaccess library instead.

Find more details on all these changes, including their progress, on GitHub issues and discussions.

Bug fixes

- Bugfix: catch multiple types of exceptions (#560)

Deprecations

- Drop support for Python 3.7 and 3.8 (#608)
- Add v1.x deprecation warning (#603)

Maintenance

- Avoid running tests that require creds on PRs (#563)
 - enables PRs from forks to pass CI tests and be merged
- Skip linting and formatting Jupyter Notebooks for now (#567)
- Keep GitHub Actions up to date with GitHub’s Dependabot (#555)
- Bump the github-actions group with 3 updates (#568)
- Add codespell spelling/typo linter and fix errors (#556)
- Validate pyproject.toml (#557)
- Enable currently-ignored linting rules E721, E722, F403 (#573)
- Add ruff rules C4 to check comprehensions (#570)
- Add ruff rules SIM to simplify Python code (#572)
- More ruff & pre-commit rules (#546)

- Fix minor typo (#589)
- Extract constant URLs to own module (#591)
- Clarify some comments / docstrings (#592)
- Migrate unit and integration tests from TravisCI to GitHub Actions (#580)
- Reinstate `test_download_granules_without_subsetting` (#581)
 - Ensure that granules can be ordered from NSIDC and downloaded with the `subset=False` option.
 - Also fix a possible race condition in case the NSIDC order status is completed right at the start (e.g. when order was cached/done already).
- Bump `sangonzal/repository-traffic-action` from 0.1.5 to 1 in the `github-actions` group (#596)
- `[pre-commit.ci]` pre-commit autoupdate (#597)
- bump `sangonzal/repository-traffic-action` version to latest working one (fixes #596) (#601)
- Replace integration test review trigger with manual trigger (#595)

Documentation

- Update viz doc to clarify its current status (partial functionality - incompatible with OA API changes) (#540)
- Add quickstart to README (#549)
- docs: add cclauss as a contributor for maintenance and review (#583)
- docs: add jrenrut as a contributor for bug, and code (#586)
- docs: add mfisher87 as a contributor for bug, code, and 4 more (#585)
- Tweak format and wording of contributor doc (#566)
- `[docs]` traffic updates May-Aug 2024 (#606)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.5 Version 1.2.0

13.5.1 What’s new in 1.2.0 (14 August 2024)

These are the changes in `icepyx` 1.2.0 See [icepyx ChangeLog](#) for a full changelog including other versions of `icepyx`.

New Features

- Replace `setup.py` with equivalent `pyproject.toml` (#539)
- Fix continuous delivery & docs to account for `setup.py` -> `pyproject.toml` change (#541)

Bug fixes

- Switch to set comparison to fix a test dependent on unreliable order (#550)

Maintenance

- update docstring tests for numpy 2.0 (#537)
- Add Zenodo badge and update all-contributors badge (#536)
- Autofix flake8 ignores E711, E712, E714, F401, F841 with Ruff (#542)
- Switch to ruff (#543)

Documentation

- fix bib entry (#529)
- [docs] update is2 resources (#535)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.6 Version 1.1.0

13.6.1 What’s new in 1.1.0 (5 June 2024)

These are the changes in icepyx 1.1.0 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- fix s3url grabbing for .nc files (#507)
 - reformat some long-lines as well
- clean up the read module after adding cloud-read capabilities (#502)
 - update tests to match new parse_source function
- pre-commit check for files over 5MB (#143)
- Clean up APIformatting module and fix ATL11 temporal kwarg submission (#515)
 - add atl11 exception for cmr and required temporal parameters
 - remove unused “default” key in _get_possible_keys
 - move “short_name” and “version” keys from the CMR list to the required list
 - utilize EarthdataAuthMixin for Granules (was still explicitly passing session)
 - Touched files were also cleaned up for linting.

Bug fixes

- update doctest for newest xarray repr (fix failing test) (#513)
- add dask[dataframe] to requirements (#519)
- improve netcdf multi-group reading and handle metadata QC issue (#516)
 - ATL11 v006, ATL14 v003, and ATL15 v003 have a meaningless string set as the version. A temporary fix gets the most recent version number from CMR to use instead for those products.
 - adds handling to read in multiple variable groups from a netcdf

Deprecations

- None

Maintenance

- update codecov uploader to start getting coverage reports (#496)
- fix black linter failures resulting from psf/black action update (#501)
- reformat all code using the pre-commit ci (#503)
- [pre-commit.ci] pre-commit autoupdate (#506, #520, #524)
- pkg_resources deprecation: stop calling setup.py directly (#500)
- add dependency needed to build packages (#511)
- [pre-commit.ci] pre-commit autoupdate of black (#510)
- minor edits to language and badges (#494)

Documentation

- update 2023 icepyx citations (#499)
- Updating the “QUEST Available Datasets” page (#509)
 - Updated links and information relating to Argo and the QUEST example notebook.
- add quest to pyreverse uml generation + separate in API docs (#498)
 - generate umls for quest module
 - make separate API docs pages for icepyx and quest
 - add manual trigger for uml creation to allow updates after post-approval changes
 - simplify highest level API TOC to show only class levels
 - add previous/next navigation buttons to API docs
- docs: add whyjz as a contributor for bug, code, and review (#522)
- Added a brief mention to QUEST in icepyx docs intro. (#514)
- [docs] update icepyx citations through Apr 2024 (#525)
- [docs] traffic updates Jan-May 2024 (#526)
 - removes a few outstanding duplicates in 2021 data

Other

- None

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.7 Version 1.0.0

13.7.1 What's new in 1.0.0 (5 January 2024)

These are the changes in icepyx 1.0.0 See [icepyx ChangeLog](#) for a full changelog including other versions of icepyx.

New (and Updated) Features

- update Read input arguments (#444)
 - add filelist and product properties to Read object
 - deprecate filename_pattern and product class Read inputs
 - transition to data_source input as a string (including glob string) or list
 - update tutorial with changes and user guidance for using glob
- enable QUEST kwarg handling (#452)
 - add kwarg acceptance for data queries and download_all in quest
 - Add QUEST dataset page to RTD
- Variables as an independent class (#451)
 - Refactor Variables class to be user facing functionality
- Expand Variables class to read s3 urls (#464)
 - expand extract_product and extract_version to check for s3 url
 - add cloud notes to variables notebook
- add argo functionality to QUEST (#427)
 - add argo.py dataset functionality and implementation through QUEST
 - demonstrate QUEST usage via example notebook
 - add save to QUEST DataSet class template
- Expand icepyx to read s3 data (#468)

Bug fixes

- temporarily disable OpenAltimetry API tests (#459)
 - add OA API warning
 - comment out tests that use OA API
- fix spot number calculation (#458)
- Update read module coordinate dimension manipulations to use new xarray index (#473)
- Fix behind EDL tests (#480)
- fix permissions for publishing to pypi (#487)

Deprecations

- deprecate filename_pattern and product class Read inputs (#444)
- remove *file* input arg and *_source* property from query (and improve some formatting) (#479)

Maintenance

- update QUEST and GenQuery classes for argo integration (#441)
- format all code files using black (#476)
- update tests to data version 006 and resolve flake8 errors on edited files (#478)
- update github actions and add black linter for PRs (#475)
 - update pypi action to use OIDC trusted publisher mgmt
 - generalize the flake8 action to a general linting action and add black
 - put flake8 config parameters into a separate file (.flake8)
 - update versions of actions/pre-commit hooks
 - specify uml updates only need to run on PRs to development
 - do not run uml updates on PRs into main #449)
 - update docs config files to be compliant
 - temporarily ignore many flake8 error codes until legacy files are updated
- Convert deprecation warnings to errors and remove associated checks #482

Documentation

- Fix a broken link in IS2_data_access.ipynb (#456)
- docs: add rwegener2 as a contributor for bug, code, and 6 more (#460)
- docs: add jpswinski as a contributor for review (#461)
- docs: add whyjz as a contributor for tutorial (#462)
- add newest icepyx citations (#455)
- traffic updates Aug-Dec 2023 (#477)
- docs: add lheagy as a contributor for mentoring, and review (#481)
- docs: add rtilling as a contributor for ideas (#484)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

- Kelsey Bisson
- Zach Fair
- Romina Piunno

13.8 Version 0.8.1

13.8.1 What’s new in 0.8.1 (14 November 2023)

These are the changes in icepyx 0.8.1 See [icepyx ChangeLog](#) for a full changelog including other versions of icepyx.

Bug fixes

- fix the ATL08 delta_time dimension read error (#470)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.9 Version 0.8.0

13.9.1 What’s new in 0.8.0 (12 September 2023)

These are the changes in icepyx 0.8.0 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- create temporal module and add input types and testing (#327)
 - create temporal module
 - create temporal testing module
 - add support for more temporal input types (datetime objects) and formats (dict)
 - temporal docstring, user guide updates
 - updated example notebook for new temporal inputs
 - update temporal info in data access tutorial example notebook
 - GitHub action UML generation auto-update
- Refactor authentication (#435)
 - modularize authentication using a mixin class
 - add docstrings and update example notebooks
 - add tests
- add atl23 (new product) to lists and tests (#445)

Deprecations

- Remove intake catalog from Read module (#438)
 - delete is2cat.py and references
 - remove intake and related modules
- Raise warning for use of catalog in Read module (#446)

Maintenance

- update codecov action and remove from deps (#421)
- is2ref tests for product formatting and default var lists (#424)
- get s3urls for all data products and update doctests to v006 (#426)
 - Always send CMR query to provider NSIDC_CPRD to make sure s3 urls are returned.
- Traffic updates 2023 Feb-Aug (#442)

Documentation

- update install instructions (#409)
 - add s3fs as requirement to make cloud access default
 - transition to recommending mamba over conda
- add release guide to docs (#255)
- docs maintenance and pubs/citations update (#422)
 - add JOSS to bib and badges
 - switch zenodo links to nonversioned icepyx

Other

- JOSS submission (#361)
 - Matches Release v0.6.4_JOSS per #420 plus a few editorial edits available via the pubs/joss branch.
- update and clarify authorship, citation, and attribution policies (#419)
 - add CITATION.cff file
 - update citation docs with Zenodo doi and ‘icepyx Developers’ as author

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.10 Version 0.7.0

13.10.1 What’s new in 0.7.0 (20 March 2023)

These are the changes in icepyx 0.7.0 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- transition icepyx to using earthaccess library for authentication (#410)
 - add earthaccess (formerly earthdata) as dependency
 - remove Earthdata module
 - update earthdata_login function to use earthaccess
 - update data access and other example notebooks
 - update earthdata and NSIDC login tests
 - simplify cloud access example auth handling using earthaccess

Bug fixes

- update read-in module for ATL11 (#398)
 - remove sc_orient from atl11 required vars list
 - introduce new function to determine if data product uses ground tracks, paths, or profiles and generalize code accordingly
 - add some custom treatments for 2d delta_times and cases when there is a gt and spot

- add atl11 path parsing and docs
- handle merging given non-unique ref_pt coordinates
- change simplify to preserve topology (#404)
- update oa viz for polygons after spatial module change (#390)
 - reactivate atl10 and atl12 viz tests and update sizes
 - disable ATL13 viz tests
- manual solution for getting ATL15 s3 urls via icepyx (#413)
- fix NSIDC login tests (#418)

Deprecations

- icepyx.core.earthdata module

Maintenance

- update license and copyright info (#408)
- update uml action to not install requirements (#406)
- update traffic data and action (#401)
 - add updating pip and upgrading pypistats to gh action
- update tracking data and actions (#393)
 - update action versions in github workflows
 - only pip install needed packages instead of all requirements

Documentation

- add ravindraK08 as a contributor for review (#405)
- add icesat2py clarification note (#395)
- update contact page - discontinue regular meetings (#396)
- add JOSS and update conda badges (#388)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.11 Version 0.6.4

13.11.1 What’s new in 0.6.4 (28 October 2022)

These are the changes in icepyx 0.6.4 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- create spatial class, automatic polygon closure functionality (#322)
- add cross-date line flag and handling (including in function to create geodataframe) (#331)

Bug fixes

- fixed several geospatial/cross-date line issues (see New Features)

Deprecations

- in *ipx.Query.avail_granules* and *ipx.core.granules.gran_IDs*, the *s3urls* keyword has been changed to *cloud*.

Maintenance

- add existing contributors using the AllContributors bot (#332-333, #335-360, #365-368, #370-371)
- clean up after adding all contribs via bot (#334)
- add provider flag to CMR request (needed with new cloud availability) (#380)
- update traffic data and scripts (#363, #369, #383)
- enable support for Python 3.10 and 3.11 (#372)
- update json file with subsetting options to v005 of ATL06 (#387)

Documentation

- update bib file and contribs formatting (#364)
- update cloud tutorial to work with publicly accessible cloud data (#381)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.12 Version 0.6.3

13.12.1 What’s new in 0.6.3 (28 July 2022)

These are the changes in icepyx 0.6.3 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- create merge index during data read-in (#305)
 - implement *photon_id* index (commented) and linear *photon_idx*
- add all contributors bot
- start adding other contributors with bot and debug allcontributors setup (#314)

Bug fixes

- address *readable_granule_name* implementation error (#292)
 - place one order per *granule_id* rather than submitting a list

Maintenance

- functionize numpy datetime conversion (#305)

Documentation

- Tracking docs updates (#307)
 - change citation heading to ‘citing icepyx’ for clarity
 - separate downloads and citations in tracking section
 - add no-tracking note
- simplify and update development plan (#306)
 - simplify and update development plan
 - note we’ve not activated discussions
 - fix IceFlow link in citations file
- add how-to guide and clarify contributions in icepyx docs (#319)
 - added text to include ancillary data within icepyx
 - added Jupyter notebook considerations
 - added GitHub instructions for new users
 - added development panel on landing page
 - Edited contribution guidelines
 - created how to develop file

This file focuses on the basic introductory GitHub steps as well as best practices with code and working with icepyx locally

- added contribution links, and QUEST idea

Added in text links for the contribution page, as well as a placeholder for ‘adding an ancillary dataset’ to the contribution page. Will need to add a link for a tutorial on ‘how to add a dataset’ in an upcoming release.

- add other hackweek repos
- allcontrib bot and simplify instructions

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.13 Version 0.6.2

13.13.1 What’s new in 0.6.2 (17 March 2022)

These are the changes in icepyx 0.6.2. See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

Bug fixes

- debug behind login Travis CI tests (#291)
- manually remove ‘z’ from datetime string (#294)

Maintenance

- update action add-and-commit v8 and remove branch/ref keywords (#290)
- add read-in functionality for deeply nested variables (e.g. ATL08) (#281)
- icepyx tracking (traffic and pypi) updates (#295)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.14 Version 0.6.0 + 0.6.1

13.14.1 What’s new in 0.6.0 (4 March 2022)

These are the changes in icepyx 0.6.0 (0.6.1) See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

Release 0.6.0 turned into release 0.6.1 due to a tagging + timing issue where the last pull request was merged after the tag was created. Increasing the version to 0.6.1 enabled us to quickly troubleshoot and fix our available builds, but a new release log was not created.

New Features

- add ability to only order one page by specifying ‘page-num’ (#87)
- add environmental variable option for Earthdata credentials (#200)
- Super query - turn Query into a superclass called GenQuery (#249)
- Update or skip all existing docstring tests (#259)
 - add docstring tests to pytest ini
- splashy front page (#260)
- QUEST (Query_Unify_Explore_SpatioTemporal) - Colocated data framework (#273)
 - Introduce framework to implement icepyx functionality (query & download) to other data sets

Bug fixes

- update docstring test config params (#256)
- update commit action version in traffic action and data (#258)
- update OA data sizes for CI tests (#262)
- comment out tests failing due to OA issue (#268)

Maintenance

- use Cmr-search-after instead of paging (#87)
- minor tweaks for gridded product access (#276)
 - check if data product is gridded before parsing filename for gran_IDs
 - update required inputs for query object based on product
 - add try to variables module for gridded products required variables list

Documentation

- Use MyST-NB to render jupyter notebook and markdown files and other docs fixes (#196)
- automate pypi statistics tracking and figure updates
- clarify which branch badges in README point to (#267)
- improve example notebooks in docs (#265)
 - Turn examples into a dedicated section
 - update and standardize example headings and levels
 - update notebook links to rtd rendering (from GH)
 - add download links to example notebooks
- Update readthedocs build dependencies (#261)
- Updating ICESat-2 Resources Pages (#263)
 - Updated resource guides
 - Created separate resource pages

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.15 Version 0.5.0

13.15.1 What’s new in 0.5.0 (8 December 2021)

These are the changes in icepyx 0.5.0 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- Add ICESat-2 data read-in functionality (#222) * includes read module and is2cat (catalog) module, example notebook, and docs * basic tests file for read module validation functions * add basic build catalog function to read data object * add ability to get list of variables from a file * add variables example notebook; trim variables module details out of subsetting example * update examples from 2020 Hackweek tutorials
- preliminary AWS access (#213) * add basic cloud data access capabilities * add weak check for AWS instance

Bug fixes

- Fix failing test_visualization_date_range check for ATL07 (#241)
- remove extra cell causing errors in example notebook

Deprecations

- None

Maintenance

- add github action to add binder badge to PRs (#229 and #233)
- update links for travis badge (#234)
- changed pytest to run in verbose mode so that it is more clear which parametrized test is failing.

- By default, no email status updates to users when ordering granules (#240)
- Set default page size for orders to 2000 per NSIDC recommendation (#239)
- update add and commit GitHub Action version (#244)
- update citation list and traffic/download data (#245)

Documentation

- GitHub action UML generation auto-update (#244)

Other

- None

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.16 Version 0.4.1

13.16.1 What’s new in 0.4.1 (01 December 2021)

These are the changes in icepyx 0.4.1 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- GitHub action to automatically update uml diagrams (#208)

Bug fixes

- check errors raised by empty query object from bounding box split in openaltimetry visualization (#220)
- updated product_summary_info function to latest fields returned by CMR (product_id -> title)

Deprecations

- *query.dataset* is now deprecated in favor of *query.product*.

Maintenance

- improved variable naming for clarity and in line with common usage (#211)
- add tests that require an active NSIDC Earthdata session (#209)
- update tracking metrics and limit traffic action to parent repo (#221)
- remove extra code block from example notebook (#225)

Documentation

- improve query docstrings (#212)

Other

- add research notice to readme (#206)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.17 Version 0.4.0

13.17.1 What’s new in 0.4.0 (13 May 2021)

These are the changes in icepyx 0.4.0. See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- use geoviews to visualize query spatial extent interactively when possible (#176)
- add ability to search by orbital parameters (RGT, cycle) (#148)
- pre-download elevation data visualization with OpenAltimetry API (#144)

Bug fixes

- Fix Sphinx warnings related to whitespace (#197)

Maintenance

- apply black formatting (#201)
- improvements to formatting and management of request parameters (use strings to prevent url formatting)

Documentation

- Added Conda and pip badges (#198)
- New example to showcase new visualization module

Other

- Add geoviews as extra optional dependency in setup.py (#193)
- Bulk rename master to main (#194)
- surface NSIDC order and http download errors more effectively to user (#195)
- switch to using a miniconda environment install for travis testing

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.18 Version 0.3.4

13.18.1 What’s new in v0.3.4 (23 March 2021)

These are the changes in icepyx 0.3.4 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- None

Bug fixes

- None

Deprecations

- None

Maintenance

- fix read the docs build with automatic versioning (#182)
- update tracking metrics and automate github action figure updates (#184)

Documentation

- Add quick instructions for installing icepyx via conda (#187)

Other

- None

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.19 Version 0.3.3

13.19.1 What’s new in v0.3.3 (11 March 2021)

These are the changes in icepyx 0.3.3 See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- Add ability to access ATL11 Annual Land Ice Height data (#161)

Bug fixes

- re-fix socket error (#163)

Deprecations

- None

Maintenance

- Update .gitignore (#180)
- add second requirements file to rtd.yml (#178)
- add manifest and update requirements files pre-conda-forge release (#175)
- Setup GitHub Action for publishing to PyPI and TestPyPI (#174)

- Add automatic versioning and required files (#168)
- add bib file to RTD configuration (#164)

Documentation

- None

Other

- Add automatic versioning and required files (#168)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.20 Version 0.3.2

13.20.1 What’s new in v0.3.2 (1 December 2020)

This is a summary of the changes in icepyx v0.3.2. See *icepyx ChangeLog* for a full changelog including other versions of icepyx. Note that during this time period we transitioned to master + development branches, with mandatory squash commits to the development branch from working branches in order to simplify the git history.

New Features

- tracking tools set up
- bibliography of icepyx uses

Bug fixes

- resolve normal projection KeyError that resulted from a DAAC change to capabilities.xml
- allow and validate numpy inputs for query objects

Deprecations

- None

Maintenance

- update Travis trigger to test PRs submitted from forks

Documentation

- section on tracking and usage statistics
- add current path to *pip install -e* instructions

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.21 Version 0.3.1

13.21.1 What's new in v0.3.1 (10 September 2020)

This is a summary of the changes in icepyx v0.3.1. See *icepyx ChangeLog* for a full changelog including other versions of icepyx. Note that during this time period we transitioned to master + development branches, with mandatory squash commits to the development branch from working branches in order to simplify the git history.

New Features

- allow data querying using tracks and cycles
- transition to use of *query* class object
- add Black pre-commit hook and flake8 for code formatting and style consistency
- created a development branch, enabling master to be the stable release branch
- add icepyx release to PyPI, thereby enabling non-dev installs with pip
- add code coverage badge for testing
- enable alternative Earthdata authentication with netrc
- automatically unzip downloaded files into a single directory
- save order IDs and enable restart of download for previously ordered data
- option to suppress order status emails from NSIDC
- display variables in a dictionary format
- overall, the variables class was overhauled: generalized, improved, and tested

Bug fixes

- update bounding box assertions to allow crossing dateline
- add try/except for gaierror
- automatically order polygon vertices properly for submission to CMR and NSIDC APIs
- fix index error due to NSIDC metadata changes
- skip straight to variable subsetting without needing to manually run data search first

Deprecations

- *icesat2data* class is deprecated. The existing functionality to search and obtain data has been migrated to the *query* class. A new class will be created for subsequent steps of working with data.
- inclusive flag for *variable.append* and *variable.remove* methods has been removed

Maintenance

- add PyPI building to Travis for new releases
- update class architecture diagram and add to documentation page
- refactor test suite into multiple modules

Documentation

- update and improve installation instructions (especially for Windows users)
- review and update all docstrings (including examples)
- move examples to top level directory for easy finding (and make development notebooks harder to find)
- create subsetting workflow example Jupyter notebook
- improve explanations in introductory example notebook
- reorganized documentation structure to be more intuitive (and categorized)

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.22 Version 0.2-alpha

13.22.1 What’s new in v0.2-alpha (6 May 2020)

These are the changes in pandas v0.2-alpha See *icepyx ChangeLog* for a full changelog including other versions of icepyx.

New Features

- Ongoing work to refactor the `icesat2data` class into a more pythonic, modular structure
 - Create *Earthdata* login class object and call as an attribute of an `icesat2data` object
 - Move API (NSIDC and CMR) formatting functions to a separate module, *APIformatting*
 - Create ICESat-2 reference function module, *is2ref*
 - Create *Granules* class to get/order/download granules and call as an attribute of the `icesat2data` object
 - Create *Variables* class to interface with ICESat-2 nested variables
 - Create *Parameters* class for managing API inputs within *APIformatting* module
- allow installation with pip and git

Bug fixes

- Polygon handling will now put polygon coordinates into the correct order for submitting to CMR API

Deprecations

- `icesat2data` class was refactored - access to some functionality changed

Maintenance

- Update examples to work with refactored code
- Update and expand tests for refactored code

Documentation

- Generate and include a UML diagram
- Update documentation to reflect refactored code
 - Separate into icesat2data API and component classes

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

13.23 Version 0.1-alpha

13.23.1 What’s new in v0.1-alpha (7 April 2020)

This was the first official “release” of icepyx, after it had been in development since Fall 2019.

This changelog captures the general features of icepyx functionality at the time of this initial release, rather than providing a detailed account of all development steps and changes that were made.

Features

- Functionality to query and order data from NSIDC using their built-in API and NASA’s CMR API
- Visualization of input spatial parameters
- Enable subsetting using NSIDC subsetter
- Variable and variable path viewing and manipulation
- Set up continuous integration testing with Travis

Bug fixes

- No known bugs at release

Deprecations

- is2class became icesat2data

Documentation

- Example usage notebooks - using *icepyx* to access ICESat-2 data - subsetting using the NSIDC subsetter - comparing ATLAS altimeter and DEM data in Colombia
- Generate documentation using Sphinx and automate building/updating to ReadtheDocs

Other

- Develop attribution and contribution guidelines
- Provide ICESat-2 Resources Guide

Contributors

A total of 0 people contributed to this release. People with a “+” by their names contributed for the first time.

PROJECT CONTRIBUTORS

The following people have made contributions to the project (in alphabetical order) and are considered “The icepyx Developers”. Thanks goes to these wonderful people ([emoji key](#)):

This project follows the [all-contributors](#) specification. Contributions of any kind welcome!

CONTRIBUTION GUIDELINES

Thank you for your interest in contributing to icepyx! We welcome and invite contributions of any size from anyone at any career stage and with any amount of coding experience. Since this is a community-based project, we're thankful for your contributions to the utility and success of this project.

Here we provide a set of guidelines and information for contributing to icepyx. This project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

15.1 Ways to Contribute

- Share your use cases and examples (as Jupyter Notebooks, scripts, etc.)
- Submit bug reports and feature requests
- Write code for everyone to use
- Fix typos
- Improve documentation and testing

The best way to report a problem, request a feature, find out if others are working on a similar problem or application, or let us know you'd like to contribute is to find the *Issues* tab and check if your problem/suggestion has already been reported. If so, please provide any additional information in the ongoing discussion. Otherwise, feel free to create a new issue and submit your problem or suggestions.

15.2 Requesting a Feature

Find the *Issues* tab at the top of GitHub repository and click the *New Issue* button. Please give your suggestion a clear title and let us know if this is something you'd like to work on and contribute.

15.3 Reporting a Bug

Find the *Issues* tab at the top of GitHub repository and click the *New Issue* button. Give your issue a clear title and describe the steps required to recreate it in as much detail as possible. If you can, include a small example that reproduces the error. More information and minimal examples will help us resolve issues faster.

15.4 Questions and Help

Please do not create issues to ask for help. A faster way to reach the community is through our Science/ICESat-2 subcategory on the [Pangeo discourse](#) page. We are excited to have you join an existing conversation or start a new post! Please note that a GitHub login is required to post on the discourse page.

15.4.1 Other Resources

- Check out our *ICESat-2 Open-Source Resources Guide* for a host of tools and code for getting and working with ICESat-2 data
- A great set of [interactive tutorials](#) for learning and practicing using git
- Let us know about the helpful tools you've found (*Questions and Help*)!

15.5 Adding Examples

We are delighted you'd like to contribute your `icepyx` example! Examples may be in the form of executable scripts or interactive Jupyter Notebooks. Please make sure that each example has a descriptive name so someone not familiar with your project understands its general behavior. Fully working examples should be submitted using a pull request to the "development" branch, following the steps outlined below for *Contributing Code*.

15.6 Contributing Code

We follow a standard git workflow for code changes and additions. All submitted code, including our own, goes through the pull request process; no changes are pushed directly to the *main* or *development* branches. This allows our continuous integration (testing) process to ensure that the code is up to our standards and passes all of our tests (i.e. doesn't break what's already there and working). By having a *development* branch for daily work, we enable the *main* branch to remain stable between releases even as new features are being added.

15.6.1 First Steps

Before you begin writing code, please first check out our issues page. Someone may already be working on the same problem, and you may be able to contribute directly to their efforts. If not, create a new issue to describe what you plan to do.

15.6.2 General Guidelines

- Make each pull request as small and simple as possible. Unrelated changes should be submitted as multiple pull requests.
- Larger changes should be broken down into their basic components and integrated separately.
- Bug fixes should be their own pull requests.
- Do not commit changes to files irrelevant to your pull request, such as *.gitignore*
- Write descriptive commit and pull request messages. Someone looking at the code a decade from now should know what you worked on from your commit message.
- Be kind and encouraging to all contributors; be willing to accept constructive criticism to improve your code.
- Review of pull requests takes time, particularly if the pull request is large and/or the commit messages are ambiguous.

15.6.3 Basic Steps to Contribute

We encourage users to follow the [git pull request workflow](#). For more detailed steps, please see *How to Contribute*.

For users that would like to add a dataset to the QUEST module, we are currently developing a Jupyter notebook to guide users through the necessary steps.

15.6.4 Licensing

icepyx is licensed under the [BSD-3 license](#). Contributed code will also be licensed under BSD-3. If you did not write the code yourself, it is your responsibility to ensure that the existing license is compatible and included in the contributed files or you have documented permission from the original author to relicense the code.

15.7 Improving Documentation and Testing

Found a typo in the documentation or have a suggestion to make it clearer? Consider letting us know by creating an issue or (better yet!) submitting a fix. This is a great, low stakes way to practice the pull request process!

Discovered a currently untested case? Please share your test, either by creating an issue or submitting a pull request to add it to our suite of test cases.

15.8 Attribution for Contributions

We appreciate any and all contributions made to icepyx, direct or indirect, large or small. To learn more about how you will be recognized for your contributions, please see our [Attribution Guidelines](#).

HOW TO CONTRIBUTE

On this page we provide brief instructions for contributing to icepyx. We encourage users to follow the [git pull request workflow](#). For contribution examples, please see [contribution guidelines](#).

16.1 Contributing for the first time

1. If you don't have one, sign up for a GitHub account (visit <https://github.com/> and 'sign up for GitHub account').
2. Clone the icepyx repo: Open a terminal window. Navigate to the folder on your computer where you want to store icepyx. For example,

```
cd /Users/YOURNAMEHERE/documents/ICESat-2
```

Within this folder, clone the icepyx repo by executing

```
git clone https://github.com/icesat2py/icepyx.git
```

You should receive confirmation in terminal of the files downloading into your workspace. For help navigating git and GitHub, see this [guide](#). GitHub also provides great how-to materials for navigating and contributing.

16.2 Every time you contribute

Important

If you do not have write permissions on this repository, start with creating a fork by clicking the Fork button in GitHub. All actions from here on should be performed in your fork.

1. To add new content, you need to create a new branch. You can do this on GitHub by clicking the down arrow next to 'development' and making a new branch (you can give it whatever name you want - the naming doesn't matter much as it will only be a temporary branch).
2. Navigate to the new branch you created. Make any edits or additions on this branch (this can be done locally with git or on GitHub directly). After you do this, commit your changes and add a descriptive commit message.
3. Push your new commits to GitHub if you were working locally. Then, open a pull request to merge your branch into the development branch. Members of the icepyx community will review your changes and may ask you to make some more.
4. If this is your first PR, someone on the icepyx team should add you to the contributors list. icepyx follows the [all-contributors](#) specification using the contributors bot to add new contributors. You are also welcome to add yourself by adding a comment with the text:

```
@all-contributors add @[GitHub_handle] for a, b, and c
```

where a, b, etc. is a list of the appropriate [contribution emojis](#). The bot will open a separate PR to add the contributor or new contribution types!

5. Repeat these steps, creating a new branch and ultimately a pull request for each change.

Tip

More, smaller pull requests are easier to debug and merge than fewer large ones, so please create pull requests regularly!

16.3 Steps for working with icepyx locally

Each time you begin working on icepyx (and especially each time you are about to create a new branch), update your local copy of icepyx with

```
git pull https://github.com/icesat2py/icepyx.git
```

to ensure you have the most up to date version of icepyx in your library.

If you are modifying portions of code, you will need to run

```
pip install -e.
```

within your Python environment to use your real-time edited version of the code during runtime.

16.4 Setting up a Development Work Environment

icepyx uses a few tools to ensure that files have consistent formatting and run tests. You can easily install the ones most frequently used by creating a new mamba (or conda) environment (from the home level of your local copy of the icepyx repo) with

```
mamba env create --name icepyx-env --channel conda-forge -f requirements-dev.txt -f ↵  
↵requirements.txt
```

and then (1) running `pre-commit install` to let git know about pre-commit and (2) pip installing icepyx as described above and below.

One of the tools installed with “requirements-dev.txt” is called [pre-commit](#). We have included a set of pre-commit formatting hooks that we strongly encourage all contributors to use. These hooks will check the files you are committing for format consistency, reformatting the files if necessary. You can tell files were reformatted if you get a message showing one of the checks failed. In this case, you will need to re-commit your changes until all pre-commit hooks pass (i.e. a failed pre-commit check results in no git commit). Pre-commit will also run on icepyx PRs using the pre-commit CI (continuous integration). As with other automations happening in PRs, you’ll want to make sure you pull the changes back to your local version before making new commits.

16.5 Considerations with Jupyter Notebook

If you are working in Jupyter Notebook, in addition to manually installing your working version in your Python environment with

```
pip install -e.
```

you will need to dynamically reload icepyx within your notebook by executing

```
%load_ext autoreload
import icepyx as ipx
%autoreload 2
```

in a notebook cell. This allows the Jupyter Notebook to detect and use changes you've made to the underlying code.

RECOGNIZING CONTRIBUTIONS

We are extremely grateful to everyone who has contributed to the success of the icepyx community and software. This document outlines our goals to give appropriate attribution to all contributors to icepyx in ways that are fair and diverse and supportive of professional goals. We define *contributions* broadly as:

Efforts towards achieving icepyx's goals, including (1) writing code, tests, or documentation, (2) development of example workflows, (3) development, significant contributions, or maintenance of a tailored package that broadens the functionality of icepyx, (4) feedback and suggestions, (5) community building, (6) etc.

We recognize contributions in the following ways.

Note: These policies are not set in stone and may be changed to accommodate the growth of the project or the preferences of the community.

17.1 Contributors List

This project follows the [all-contributors](#) specification. When you contribute to icepyx for the first time or in a new way, you or a maintainer can use the *All Contributors bot to open a PR* <<https://allcontributors.org/docs/en/bot/usage>> to recognize your contribution. Comment on an existing PR with *@all-contributors please add @<username> for <contributions>*. This will add you (or your new contribution type) to the CONTRIBUTORS.rst file located in the top level directory; the file is packaged and distributed with icepyx, so each release has a record of contributors and their contribution types.

17.2 Changelog

Each release includes a changelog of updates. Everyone who has made a commit since the last release is listed, with new contributors indicated. This list is automatically generated using a Sphinx extension; where available, full names are used. If the user's full name is not available on GitHub, their GitHub handle is used.

17.3 Example Workflows

Many of the example workflows included within icepyx were developed by individuals or small teams for educational or research purposes. We encourage example developers to provide proper recognition for these efforts both within the notebook itself and by adding contributors to the *Contributors List* for attribution as described herein.

17.4 Version Releases on Zenodo

Each new release of icepyx is [archived on Zenodo](#).

Following the collaborative approach of [The Turing Way](#), we aim to encourage community leadership and shared ownership of icepyx. To this end, beginning with version 0.6.4 (the full adoption of the all-contributors specification) we collectively represent the icepyx authors in citations (including Zenodo releases) as “The icepyx Developers”.

As described above, a complete list of contributors and their contribution types is available via the [Contributors List](#).

** A note about releases <v0.6.4: Prior version releases adhere to authorship guidelines in place at the time, listing individual contributors who had manually added their names to *CONTRIBUTORS.rst*. Authorship order was alphabetical by last name, except in cases where a substantial contribution was made by one or more contributors to a given release. **

17.5 Peer-Reviewed Publications (Papers)

We will occasionally prepare manuscripts describing our software and its uses for submission to peer-reviewed journals. These efforts are typically “in addition to” contributions made directly to icepyx (community or repository) and thus may have specific author lists. To be eligible for authorship on a peer-reviewed publication, contributors must:

1. Contribute to the development (including code, documentation, and examples) of icepyx. Substantial non-code contributions may constitute eligibility for authorship.
2. Contribute ideas, participate in authorship discussions (see next paragraph), write, read, and review the manuscript in a timely manner, and provide feedback (acknowledgement of review is sufficient, but we’d prefer more).

Author order will be determined based on co-author discussion, led by the publication preparation leader, ideally during the initial planning stages of manuscript preparation (i.e. as soon as an idea matures into a potential manuscript and before writing begins). Authorship will continue to be evaluated throughout the manuscript preparation process. Discussions will consider authorship norms (e.g. How does author order convey participation and prestige? How critical is first authorship to career advancement for each member of the team? Do an individual’s contributions meet authorship criteria or are they more suited to acknowledgements?). Author order determination may also consider metrics such as the number of commits since the last major release with an associated paper (`git shortlog vX.0.0...HEAD -sne`), contributions that do not have associated commits, and contributions to the preparation of the manuscript.

17.6 Motivation and References

Concepts and models of attribution, credit, contribution, and authorship can vary across time, application, and communities. [FORCE11](#) has an entire [Attribution Working Group](#) dedicated to working on attribution for research products. [URSSI](#) hosted a workshop in 2019 ([report](#)) to identify core issues and propose solutions to challenges around software credit. For software, current best practices (e.g.) emphasize the importance of having a document such as this one to describe an individual community’s policies for credit, authorship, and attribution. This document is an effort to describe icepyx’s policies, with an awareness that they may change to accommodate community growth, best practices, and feedback.

We do not attempt to identify contribution levels through the number of commits made to the repository (e.g. `git shortlog -sne`) or active engagement on GitHub (e.g. through issues, discussions, and pull requests) and Discourse. The latter is difficult to quantify, and the use of squash merges into the development branch can mask the relative complexity of various contributions and does not necessarily capture significant conceptual contributions.

Copyright notice: Preparation of this document and our credit policies was inspired in part by these [authorship guidelines](#) provided by [Fatiando a Terra](#) and [The Turing Way](#). We encourage potential contributors to consider the resources provided by the [NASA High Mountain Asia Team \(HiMAT\)](#) and established or emerging best practices in their community. Please get in touch if you would like to discuss updates to this contribution recognition policy.

ICEPYX INTERNALS

18.1 Authentication

Authentication in `icepyx` is handled using a `Mixin` class. A `Mixin` class is a class which defines functionality that may be desired by multiple other classes within a library. For example, at this time the `Query`, `Variables`, and `Read` classes need to be able to authenticate. Instead of defining the same properties and functionality twice, `icepyx` has an `EarthdataAuthMixin` class that is inherited by any modules that need an Earthdata login.

Property Access

Even though they aren't explicitly defined in the `init` method, properties like `.session` are accessible on a `Query` object because they are inherited. The code that indicates this to Python is `EarthdataAuthMixin.__init__(self)`.

For example:

```
import icepyx as ipx

region_a = ipx.Query('ATL06', [-45, 74, -44, 75], ['2019-11-30', '2019-11-30'], \
                    start_time='00:00:00', end_time='23:59:59')

# authentication can be accessed via the Query object
region_a.session
region_a.s3login_credentials
```

Adding authentication to a new class

To add authentication to an additional `icepyx` class, one needs to add the `Mixin` to the class. To do this:

1. Add the `EarthdataAuthMixin` class to the class constructor (and import the mixin)
2. Add the `EarthdataAuthMixin` `init` method within the `init` method of the new class `EarthdataAuthMixin.__init__(self)`
3. Access the properties using the **public** properties (Ex. `self.session`, not `self._session`.)

A minimal example of the new class (saved in `icepyx/core/newclass.py`) would be:

```
from icepyx.core.auth import EarthdataAuthMixin

class MyNewClass(EarthdataAuthMixin):
    def __init__(self):
        self.mynewclassproperty = True

        EarthdataAuthMixin.__init__(self)
```

(continues on next page)

(continued from previous page)

```
def my_exciting_new_method(self):  
    # This method requires login  
    s = self.session  
    print(s)  
    return 'We authenticated inside the method!'
```

The class would then be accessible with:

```
from icepyx.core.newclass import MyNewClass  
  
n = MyNewClass()  
  
n.session  
n.my_exciting_new_method()
```

QUEST SUPPORTED DATASETS

On this page, we outline the datasets that are supported by the QUEST module. Click on the links for each dataset to view information about the API and sensor/data platform used.

19.1 List of Datasets

19.1.1 Argo

The Argo mission involves a series of floats that are designed to capture vertical ocean profiles of temperature, salinity, and pressure down to ~2000 m. Some floats are in support of BGC-Argo, which also includes data relevant for biogeochemical applications: oxygen, nitrate, chlorophyll, backscatter, and solar irradiance.

For interested readers, a preprint outlining the QUEST module and its application to Argo data access is available [here](#).

[Argo Workflow Example](#)

QUEST uses the Argovis API to access Argo data, so users are encouraged to use the following citation:

Tucker, T., D. Giglio, M. Scanderbeg, and S.S.P. Shen, 2020. Argovis: A Web Applications for Fast Delivery, Visualization, and Analysis of Argo data. *J. Atmos. Oceanic Technol.*, 37, 401-416, <https://doi.org/10.1175/JTECH-D-19-0041.1>

Citations for individual Argo datasets may be found at this link: <https://argovis.colorado.edu/about>

19.2 Adding a Dataset to QUEST

Want to add a new dataset to QUEST? No problem! QUEST includes a template script (`dataset.py`) that may be used to create your own querying module for a dataset of interest.

Once you have developed a script with the template, you may request for the module to be added to QUEST via GitHub. Please see the How to Contribute page [How to Contribute](#) for instructions on how to contribute to icepyx.

Detailed guidelines on how to construct your dataset module are currently a work in progress.

ICEPYX DEVELOPMENT PLAN

This page provides a high-level overview of focus areas for icepyx’s ongoing and future development. This list is intentionally general and not exhaustive. Instead, specific development tasks and new functionality implementations are driven by individual developers/teams.

Our ongoing efforts are tracked as issues on our GitHub [issue tracker](#). We invite you to join the active discussions happening there.

20.1 Major Themes for Development

20.1.1 Enhancing User Interactivity and Visualization

icepyx aims to continually reduce the need for researchers to rewrite “routine” code by enabling easy end-to-end data visualization and providing a simple, community-based framework for reproducibility.

20.1.2 Open Science Example Use Cases

Research is the primary driver for development of icepyx functionality. We encourage you to use icepyx as a framework for finding and processing your ICESat-2 data, from designing your analysis to writing code to analyze your data to generating presentation-quality figures. We welcome example use cases from all disciplines. Some topics currently being investigated using ICESat-2 data:

- snow height in non-glaciated regions
- subsurface ocean structure (including bathymetry)
- vegetation canopy height
- glacier ice velocity
- sea level change
- archaeological site discovery
- phytoplankton concentrations under sea ice
- iceberg detection

Please [contact us](#) if you have any questions or would like to submit an example workflow showcasing your research!

20.1.3 Data Analysis and Interaction

Many data analysis techniques (filtering, corrections, trend detection, feature detection, statistics, machine learning, etc.) are used by researchers to analyze ICESat-2 data products. As part of the broader Python ecosystem, relevant libraries that specialize in these techniques are easily incorporated into a single work environment. In addition, creating ICESat-2 specific extensions for Xarray and Pandas data structures will enhance our ability to utilize these analysis tools by providing easy ways to manipulate ICESat-2 data in the appropriate input type required by each library. Workflows

showcasing complex analyses to answer pressing science questions provide an opportunity for new researchers to build on existing work.

20.1.4 Validation and Integration with Other Products

The complexity of multiple data access systems, often with different metadata formats and API access types, presents a challenge for finding and integrating diverse datasets. Driven by researcher use cases, icepyx contains a consistent framework for adding a new product/sensor to an existing data analysis pipeline, improving researcher ability to easily compare diverse datasets across varying sensor types and spatial and temporal scales.

20.2 Modifying the Development Plan

Everyone is invited to review and propose new themes for the Development Plan. icepyx is continually evolving and its direction is driven by your feedback and contributions.

If you'd like to add a theme to this development plan, please submit your idea in [GitHub Discussions](#) to solicit community feedback. Once there is agreement on your idea, submit a pull request to update the Development Plan, including a link to the discussion.

RELEASE GUIDE

Interested in the process for creating a new icepyx release? Here is a guide outlining the process and how to fix common mistakes.

21.1 Create a Release Log

Create a new branch from the development branch. You'll create and update the release documents on this branch.

In `doc/source/user_guide/changelog` is a file called `template.rst`. Make a *copy* of this file and update the copy's filename to your version release number. We follow standard [semantic versioning](#) practices.

Create an entry for the current "Latest Release":

```
Version 0.x.y
-----
.. toctree::
   :maxdepth: 2

v0.x.y
```

Add your new version to the `doc/source/user_guide/changelog/index.rst` as the "Latest Release".

```
Latest Release (Version 0.8.0)
-----

.. toctree::
   :maxdepth: 2

v0.8.0
```

Now, populate your new release file by filling in the template. You will probably need to make the release date a few days out to allow time for review and merging.

There are no strict rules for how you generate the content for the file. One method is to use a `git log` command to display the commit history of the development branch since the last release. If you're using `git` in terminal, checkout development and make sure your local development branch is up-to-date (`git pull`). Then run `git log 'v0.x.y'...HEAD` where 'v0.x.y' is the current/latest release. You can sort and edit the commit messages as needed to populate the changelog.

Add your new changelog file, commit and push your changes, and head to GitHub to open a Pull Request (PR).

21.2 Create a Release Pull Request to the Development Branch

On GitHub, create a PR from your release branch into the development branch. Once the PR is reviewed and all the tests pass, you or your reviewer can squash and merge the PR into the development branch.

Now you're ready to update main and actually package your new release!

21.3 Create a Pull Request from Development to Main

The changelog is completed, we're not waiting for any more PRs to be merged, and we're ready to share the newest version of icepyx with the world. Create a PR to merge the development branch into main (so main will now be your base branch). If any tests fail, you may need to do some debugging. This will involve submitting a new PR to development with whatever debugging changes you've made. Once merged into development, any changes will automatically be reflected in this step's PR, and the tests will rerun automatically.

With an approving review and passed tests in hand, you're ready to push the new release! Unlike when you merge new features into development with a squash merge, for this step you'll want to use a plain old merge (the button says "Create a Merge Commit"). This makes it easy to keep development and main even instead of diverging due to a series of merge commits. [This website](#) does a great job explaining the how and why of not using a squash merge here.

However, if you forget and squash merge, never fear. You can simply revert the commit and begin again from the beginning of this step.

21.4 Update the Development Branch Head

We want to make sure at this point that the development and main branches are even. You can do this with a git API, but the way to do it using git in terminal is:

```
git pull
git checkout development
git merge main
git push origin development:development
```

If you have to create a merge commit message, STOP! You've done something wrong and need to go back to the previous step. Creating the merge commit will make main and development diverge and the repo maintainers sad.

21.5 Tag the Release

Last, but potentially most importantly, we need to tag and create the release. This step will trigger the package to be built and update the distribution available from conda and PyPI. It will also publish the new release on Zenodo. GitHub makes releases easy - on the repo's home page, simply select "Releases" from the right hand side and then the "Draft a New Release" button. Add a new tag with the version number of your release, making sure it points to the main branch (by default, GitHub will suggest the development branch!) Fill out the form and create the release.

If you tag the release too soon (and there end up being more commits), or point it to the wrong branch/commit, never fear. You can delete the release from GitHub with the click of a button. If you want to reuse the version tag though (you most likely do), you'll first have to remove the tag locally and push the updated (deleted) tag to GitHub:

```
git push --delete origin tagname
```

See [this guide](#) on how to delete local and remote git tags.

Then you can go back to the beginning of this step to create a new tag and release. Alternatively, you may be better off yanking the previous release (but leaving the tag) and increasing your patch number in a new tag+release. This may be necessary if you have a failing release already on PyPI.

21.6 Finishing Up

If all went according to plan, you should see your most recent version of icepyx available from PyPI within a few moments. It won't happen immediately, as they need to properly build the installation files. To make the latest release available via conda-forge, a few bots will run and let the feedstock maintainers know when it's ready or if there are any issues. Then they can manually approve the merge to the feedstock repo and the new release will be available in a few minutes.

Congratulations! You released a new version of icepyx! Share the good news on Twitter or Slack and appreciate your hard work and contributions to open-source development.

CONTRIBUTOR COVENANT CODE OF CONDUCT

22.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

22.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

22.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

22.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

22.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at jbscheick-at-gmail-dot-com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

22.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

22.6.1 1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

22.6.2 2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

22.6.3 3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

22.6.4 4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the project community.

22.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at <https://www.contributor-covenant.org/faq>.
Translations are available at <https://www.contributor-covenant.org/translations>.

ICESAT-2 RESOURCE GUIDE

The ICESat-2 Community now maintains the [Awesome ICESat-2 Resource Guide](#). We encourage you to check out the list of resources and archive there.

CONTACT US

- Need help installing, running, or using *icepyx*? Ask for help on [Discourse](#) or [GitHub Discussions](#).
- Found a bug? Post an issue on [GitHub](#)!
- Want to request or contribute a feature? Share your idea on [GitHub Discussions](#).
- Have a question or want to know more? Join us for a virtual meeting (see below).
- Want to get involved? Do one or more of the above, or reach out to one of the dev team members individually. We're excited to hear your thoughts and provide help!

Absolutely NO software development is necessary to join and contribute to our community. We look forward to meeting you!

24.1 Virtual Meetings

Our team (developers, users, scientists, educators) consists primarily of volunteers. We meet on an as-needed basis via Zoom to provide support, troubleshoot issues, and plan development and collaboration efforts.

Our meetings are open to all, with upcoming meeting information available via [Discourse](#), [GitHub Discussions](#), or by request. The *QUEST (Query Unify Explore SpatioTemporal)* team meets weekly on Mondays to co-work on integrating additional sensors into the *icepyx* workflow. Please contact us if you are interested in joining the QUEST team.

24.2 Ongoing Efforts

In addition to the ongoing development of *icepyx* itself, the ICESat-2 community continues to grow through a number of related initiatives, workshops, and events:

- [CryoCloud](#)
- [ICESat-2 Hackweeks](#)

WHO IS USING ICEPYX?

How is icepyx being used by the ICESat-2 data user community?

Is your team or project using icepyx but not listed below? Please add your organization to the appropriate list with a link to your project/product (or *get in touch* and we'll add it)!

25.1 Projects and Organizations

Projects and organizations that use icepyx.

- NSIDC
- IceFlow
- University of Washington e-Science institute
- ICESat-2 Hackweeks
- SlideRule
- Colorado School of Mines Glaciology Laboratory

25.2 Publications About icepyx

Peer-reviewed publications about icepyx software

icepyx in the open-source landscape

25.3 Presentations and Materials Featuring icepyx

Presentations that feature or explain icepyx

25.4 Publications Utilizing icepyx

Research that utilizes icepyx for ICESat-2 data

ICEPYX ADOPTION

Estimating usage of open-source software is a fundamentally difficult task, and “easy” metrics like number of downloads [have the potential to be misleading](#).

We are excited by the enthusiastic adoption of `icepyx` by the ICESat-2 data user community, and despite these limitations in data tracking metrics, we have begun (November 2020) to track aggregate user downloads and page views as shown below.

Although technologies exist, to respect user privacy and international regulations (without requiring browser cookies), we intentionally do not track the IP addresses of users accessing our code or documentation. As a result, we are unable to view usage statistics for specific pages/examples or repeat visitors. If you find certain materials especially helpful, we'd appreciate *hearing from you!*

26.1 GitHub Traffic

Clones and views of the `icepyx` library directly on GitHub.

26.2 PyPI Downloads

Non-mirrored downloads of `icepyx` from the [Python Package Index](#) (e.g. using `pip install icepyx`).

Icon images from [Flaticon](#) (by Freepik, Pixel perfect, and Eucalyp) and [NASA](#).

BIBLIOGRAPHY

- [1] Scheick, Jessica, Leong, Wei Ji, Bisson, Kelsey, Arendt, Anthony, Bhushan, Shashank, Fair, Zachary, Hagen, Norland Raphael, Henderson, Scott, Knuth, Friedrich, Li, Tian, Liu, Zheng, Piunno, Romina, Ravinder, Nitin, Setiawan, Landung "Don", Sutterley, Tyler, Swinski, JP, and Anubhav. icepyx: querying, obtaining, analyzing, and manipulating ICESat-2 datasets. *Journal of Open Source Software*, 8(84):4912, 2023. URL: <https://doi.org/10.21105/joss.04912>, doi:10.21105/joss.04912.
- [2] Scheick, Jessica, Piunno, Romina, Fair, Zachary, Tilling, Rachel, Bella, A. Di, Abib, Nicole, and Bisson, Kelsey. Software to enable ocean discoveries: a case study with ICESat-2 and Argo. *Geoscience Data Journal*, 12(3):e291, 2025. J. Scheick and R. Piunno contributed equally to this work. URL: <https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/gdj3.291>, arXiv:<https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/gdj3.291>, doi:<https://doi.org/10.1002/gdj3.291>.
- [1] Bednar, James A. and Durant, Martin. The Pandata scalable open-source analysis stack. In *Proceedings of the 22nd Python in Science Conference (SciPy 2023)*, volume, 85–92. 2023. URL: https://conference.scipy.org/proceedings/scipy2023/pdfs/james_bednar.pdf, doi:.
- [2] Chudley, Thomas R. and Howat, Ian M. pDEMtools: conveniently search, download, and process Arctic DEM and REMA products. *Journal of Open Source Software*, 9(102):7149, October 2024. doi:10.21105/joss.07149.
- [3] Shean, David, Swinski, J.p., Smith, Ben, Sutterley, Tyler, Henderson, Scott, Ugarte, Carlos, Lidwa, Eric, and Neumann, Thomas. SlideRule: enabling rapid, scalable, open science for the NASA ICESat-2 mission and beyond. *Journal of Open Source Software*, 8(81):4982, 2023. URL: <https://doi.org/10.21105/joss.04982>, doi:10.21105/joss.04982.
- [4] Tankersley, Matthew D. PolarToolkit: Python Tools for Convenient, Reproducible, and Open Polar Science. *Journal of Open Source Software*, 9(100):6502, August 2024. doi:10.21105/joss.06502.
- [1] Arendt, Anthony, Scheick, Jessica, Shean, David, Buckley, Ellen, Grigsby, Shane, Haley, Charley, Heagy, Lindsey, Mohajerani, Yara, Neumann, Tom, Nilsson, Johan, Markus, Thorsten, Paolo, Fernando S., Perez, Fernando, Petty, Alek, Schweiger, Axel, Smith, Benjamin, Steiker, Amy, Alvis, Sebastian, Henderson, Scott, Holschuh, Nick, Liu, Zheng, and Sutterley, Tyler. 2020 ICESat-2 Hackweek Tutorials. August 2020. URL: <https://doi.org/10.5281/zenodo.3966463>, doi:10.5281/zenodo.3966463.
- [2] Henderson, Scott, Meyer, Joachim, Scheick, Jessica, Arendt, Anthony, Barciauskas, Aimee, Barrett, Andy, Bhushan, Shashank, Ji, Wei, Piunno, Romina, Setiawan, Don, Snow, Tasha, Sutterley, Tyler, and Swinski, JP. ICESAT-2 HackWeek/website-2024: 2024 Jupyter Book. November 2024. URL: <https://doi.org/10.5281/zenodo.14052799>, doi:10.5281/zenodo.14052799.
- [3] Meyer, Joachim, Arndt, Philipp, Arendt, Anthony, YoungHyunKoo, Swinski, JP, Henderson, Scott, Scheick, Jessica, Barrett, Andy, Steiker, Amy, Snow, Tasha, Piunno, Romina, Sutterley, Tyler, Yekeen, Shamsudeen Temitope, King, Michalea, M, Jonathan, Barciauskas, Aimee, Williams, Jullian, Zikan, Karina (Inka), Ji, Wei, Lowndes, Julia Stewart, Sauthoff, Wilson, Fair, Zach, and liuzheng-arctic. Icesat-2hackweek/icesat-2-hackweek-2023: 2023. January 2024. URL: <https://doi.org/10.5281/zenodo.10519966>, doi:10.5281/zenodo.10519966.

- [4] Scheick, J, Bisson, K, Fair, Z, Piuanno, R, Leong, WJ, Lopez, L, and Hall, S. icepyx as an icebreaker: starting conversations and building competencies in open science. Invited abstract and poster. American Geophysical Union Fall Meeting, Chicago, IL, USA. 12-16 December 2022., 2022. doi:10.5281/zenodo.7837428.
- [5] Scheick, J., Arendt, A., Heagy, L., Paolo, F., Perez, F., and Steiker, A. \texttt icepyx: developing community and software around ICESat-2 Data. 2020. Abstract and eLightning (poster + presentation). American Geophysical Union Fall Meeting, virtual, USA. 1-17 December 2020.
- [6] Scheick, J., Arendt, A., Heagy, L., and Perez, F. Introducing icepyx, an open source Python library for obtaining and working with ICESat-2 data. 2019. Abstract and poster. American Geophysical Union Fall Meeting, San Francisco, California, USA. 9-13 December 2019. doi:10.1002/essoar.10501423.1.
- [7] Scheick, J., Arendt, Anthony, Haley, Charley, Henderson, Scott, Koh, Jane, Setiawan, Don, Alterman, Naomi, Meyer, Joachim, Cristea, Nicoleta, Schweiger, Axel, Barciauskas, Aimee, Smith, Ben, Piuanno, Romina, Shapiro, Daniel, Fair, Zachary, Arndt, Philipp, Leong, Wei Ji, Sutterley, Tyler, Snow, Tasha, Beig, Mikala, Besso, Hannah, Liu, Zheng, Joughin, Ian, Bisson, Kelsey, and Sauthoff, Wilson. ICESat-2 Hackweek Website. April 2022. If you use this book, please cite it as below. URL: <https://doi.org/10.5281/zenodo.6462479>, doi:10.5281/zenodo.6462479.
- [8] Scheick, Jessica, Bisson, Kelsey, Fair, Zachary, Piuanno, Romina, Abib, Nicole, Di Bella, Alessandro, and Tilling, Rachel. On a QUEST (Query, Unify, Explore SpatioTemporal) to accelerate ICESat-2 applications in ocean science via icepyx. January 2024. URL: <https://doi.org/10.5281/zenodo.10563003>, doi:10.5281/zenodo.10563003.
- [9] Scheick, Jessica, Bisson, Kelsey, Li, Tian, Leong, Wei Ji, and Arendt, Anthony. Collaborative computational resource development around ICESat-2 data: the icepyx community and library. *Earth and Space Science Open Archive*, pages 9, 2022. Abstract and poster (presented by Wei Ji Leong). American Geophysical Union Fall Meeting, New Orleans, LA, USA. 13-17 December 2021. URL: <https://doi.org/10.1002/essoar.10511316.1>, doi:10.1002/essoar.10511316.1.
- [10] Wegener, Rachel, Lopez, Luis, Steiker, Amy, and Scheick, Jessica. Using icepyx to access ICESat-2 data. November 2023. GEDI/ICESat-2 Workshop, 2023 Space and Sustainability Colloquium (Espacio y Sostenibilidad). Sociedad Latinoamericana en Percepción Remota y Sistemas de Información Espacial (SELPER), Guadalajara, Mexico, 15-16 November 2023. URL: <https://nasa-openscapes.github.io/2023-ssc/>.
- [1] Al-Attabi, Zaid, Narayan, Siddharth, Campbell, Anthony, Wuppukondur, Ananth, Babanawo, Daystar, Xu, Yicheng, Thomas, Nathan, Fatoyinbo, Lola, and Lagomasino, David. Sensitivity of Flood Extent and Population Exposure to Variations in Bathymetry Across a Shallow Continental Shelf Lagoon. *Journal of Geophysical Research: Oceans*, 131(3):e2025JC022956, March 2026. doi:10.1029/2025JC022956.
- [2] Bisson, K. M. and Cael, B. B. How are under ice phytoplankton related to sea ice in the Southern Ocean? *Geophysical Research Letters*, 48(21):e2021GL095051, 2021. doi:10.1029/2021GL095051.
- [3] Eidam, E. F., Bisson, K., Wang, C., Walker, C., and Gibbons, A. ICESat-2 and ocean particulates: A roadmap for calculating lemp Kd from space-based lidar photon profiles. *Remote Sensing of Environment*, 311:114222, September 2024. doi:10.1016/j.rse.2024.114222.
- [4] Eidam, Emily, Walker, Catherine, Bisson, Kelsey, Paris, Matthew, and Cooper, Lillian. Novel application of ICESat-2 ATLAS data to determine coastal light attenuation as a proxy for suspended particulate matter. In *OCEANS 2022, Hampton Roads*, volume, 1–7. 2022. doi:10.1109/OCEANS47191.2022.9977084.
- [5] Fair, Zachary, Vuyovich, Carrie, Neumann, Thomas, Pflug, Justin, Shean, David, Enderlin, Ellyn M., Zikan, Karina, Besso, Hannah, Lundquist, Jessica, Deschamps-Berger, Cesar, and Treichler, Désirée. Review article: using spaceborne lidar for snow depth retrievals: recent findings and utility for hydrologic applications. *The Cryosphere*, 19(11):5671–5691, November 2025. doi:10.5194/tc-19-5671-2025.
- [6] Fernando, Garrido. Mapping the diversity of agricultural systems in the Cuellaje Sector, Cotacachi, Ecuador using ATL08 for the ICESat-2 Mission and machine learning techniques. In Gervasi, Osvaldo, Murgante, Beniamino, Misra, Sanjay, Garau, Chiara, Blečić, Ivan, Taniar, David, Apduhan, Bernady O., Rocha, Ana Maria A. C., Tarantino, Eufemia, and Torre, Carmelo Maria, editors, *Computational Science and Its Applications –*

- ICCSA 2021*, volume 12957, 170–181. Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-87013-3_13.
- [7] Freer, B. I. D., Marsh, O. J., Hogg, A. E., Fricker, H. A., and Padman, L. Modes of Antarctic tidal grounding line migration revealed by Ice, Cloud, and land Elevation Satellite-2 (ICESat-2) laser altimetry. *The Cryosphere*, 17(9):4079–4101, 2023. URL: <https://tc.copernicus.org/articles/17/4079/2023/>, doi:10.5194/tc-17-4079-2023.
- [8] Golin, Ailé Selenne, Páez Campos, Hugo Ramiro, Guevara Ochoa, Cristian, Dávila, Claudia Fernanda, and Vives, Luis Sebastián. Assessing open-access digital elevation models for hydrological applications in a large scale plain: drainage networks, shallow water bodies and vertical accuracy. *Earth Surface Processes and Landforms*, 49(15):5269–5283, 2024. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/esp.6035>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/esp.6035>, doi:<https://doi.org/10.1002/esp.6035>.
- [9] Idestrom, Petter. Remote sensing of cryospheric surfaces: small scale surface roughness signatures in satellite altimetry data. Master's thesis, UmeåUniversity, Sweden, Sept. 2023. URL: <https://www.diva-portal.org/smash/get/diva2:1801057/FULLTEXT01.pdf>.
- [10] Kamath, Harsh G., Singh, Manmeet, Malviya, Neetiraj, Martilli, Alberto, He, Liu, Aliaga, Daniel, He, Cenlin, Chen, Fei, Magruder, Lori A., Yang, Zong-Liang, and Niyogi, Dev. GLOBal Building heights for Urban Studies (UT-GLOBUS) for city- and street- scale urban simulations: Development and first applications. *Scientific Data*, 11(1):886, August 2024. doi:10.1038/s41597-024-03719-w.
- [11] Leite, Rodrigo V., Amaral, Cibele, Neigh, Christopher S. R., Cosenza, Diogo N., Klauberg, Carine, Hudak, Andrew T., Aragão, Luiz, Morton, Douglas C., Coffield, Shane, McCabe, Tempest, and Silva, Carlos A. Leveraging the next generation of spaceborne Earth observations for fuel monitoring and wildland fire management. *Remote Sensing in Ecology and Conservation*, 11(2):133–155, April 2025. doi:10.1002/rse2.416.
- [12] Li, T., Dawson, G. J., Chuter, S. J., and Bamber, J. L. Mapping the grounding zone of Larsen C Ice Shelf, Antarctica, from ICESat-2 laser altimetry. *The Cryosphere*, 14(11):3629–3643, 2020. URL: <https://tc.copernicus.org/articles/14/3629/2020/>, doi:10.5194/tc-14-3629-2020.
- [13] Malambo, Lonesome and Popescu, Sorin. Mapping vegetation canopy height across the contiguous United States using ICESat-2 and ancillary datasets. *Remote Sensing of Environment*, 309:114226, August 2024. doi:10.1016/j.rse.2024.114226.
- [14] Mallinis, Giorgos, Verde, Natalia, Siachalou, Sofia, Latinopoulos, Dionisis, Akratos, Christos, and Kagalou, Ifigenia. Evaluation of multiple classifier systems for mapping different hierarchical levels of forest ecosystems in the mediterranean region using Sentinel-2, Sentinel-1, and ICESat-2 data. *Forests*, 2023. URL: <https://www.mdpi.com/1999-4907/14/11/2224>, doi:10.3390/f14112224.
- [15] Musthafa, Mohamed, Singh, Gulab, and Kumar, Praveen. Comparison of forest stand height interpolation of GEDI and ICESat-2 LiDAR measurements over tropical and sub-tropical forests in India. *Environmental Monitoring and Assessment*, 195(1):71, 2022. URL: <https://doi.org/10.1007/s10661-022-10657-w>, doi:10.1007/s10661-022-10657-w.
- [16] Narin, Omer Gokberk. Evaluation and integration of spaceborne altimetry missions for inland water level monitoring: a multi-country study. *Advances in Space Research*, 77(1):281–296, January 2026. doi:10.1016/j.asr.2025.09.078.
- [17] Snellink, Kamil. Assessing land ice height decrease of the Fleming Glacier using ICESat-2 satellite data: a 2019-2022 analysis. July 2023. Bachelor Thesis. Applied Earth Sciences, Department of Geoscience and Remote Sensing. URL: <https://repository.tudelft.nl/islandora/object/uuid:909190a5-b91d-4f67-8134-3f19756ed817?collection=education>.
- [18] Sothe, Camile, Gonsamo, Alemu, Lourenço, Ricardo B., Kurz, Werner A., and Snider, James. Spatially continuous mapping of forest canopy height in Canada by combining GEDI and ICESat-2 with PALSAR and Sentinel. *Remote Sensing*, 14(20):5158, Oct 2022. Attribution to icepyx from 15 Oct 2022 Twitter post, <https://twitter.com/rblourenco/status/1581320878511382528>. URL: <http://dx.doi.org/10.3390/rs14205158>, doi:10.3390/rs14205158.

- [19] Tian, Xiangxi and Shan, Jie. ICESat-2 controlled integration of GEDI and SRTM data for large-scale digital elevation model generation. *IEEE Transactions on Geoscience and Remote Sensing*, 62():1–14, 2024. doi:10.1109/TGRS.2024.3389821.
- [20] Tian, Xiangxi and Shan, Jie. Learning-Based Spatial Interpolation for Sparse Space Altimetry Measurements. *IEEE Transactions on Geoscience and Remote Sensing*, 63:1–14, 2025. doi:10.1109/TGRS.2025.3643406.
- [21] van Leeuwen, Gijs. The automated retrieval of supraglacial lake depth and extent from ICESat-2 photon clouds leveraging DBSCAN clustering. 2022. Utrecht University Masters Thesis. URL: https://studenttheses.uu.nl/bitstream/handle/20.500.12932/43402/MSC_thesis_ICESat2_GJvanLeeuwen.pdf?sequence=1.

PYTHON MODULE INDEX

i

- `icepyx.core.APIformatting`, 75
- `icepyx.core.auth`, 77
- `icepyx.core.granules`, 78
- `icepyx.core.harmony`, 80
- `icepyx.core.is2ref`, 83
- `icepyx.core.orders`, 83
- `icepyx.core.spatial`, 85
- `icepyx.core.temporal`, 89
- `icepyx.core.types`, 95
- `icepyx.core.validate_inputs`, 93
- `icepyx.core.visualization`, 93

Symbols

`__init__()` (*icepyx.GenQuery* method), 55
`__init__()` (*icepyx.Query* method), 56
`__init__()` (*icepyx.Quest* method), 98
`__init__()` (*icepyx.Read* method), 68
`__init__()` (*icepyx.Variables* method), 70

A

`about_product()` (in module *icepyx.core.is2ref*), 83
`add_argo()` (*icepyx.Quest* method), 99
`add_icesat2()` (*icepyx.Quest* method), 99
`append()` (*icepyx.Variables* method), 74
`auth` (*icepyx.core.auth.EarthdataAuthMixin* property), 78
AuthenticationError, 77
`avail()` (*icepyx.Variables* method), 72
`avail_granules()` (*icepyx.Query* method), 64

B

`build_params()` (*icepyx.core.APIformatting.Parameters* method), 76

C

`check_dateline()` (in module *icepyx.core.spatial*), 87
`check_order_status()` (*icepyx.core.harmony.HarmonyApi* method), 81
`check_req_values()` (*icepyx.core.APIformatting.Parameters* method), 76
`check_s3bucket()` (in module *icepyx.core.validate_inputs*), 93
`check_valid_date_range()` (in module *icepyx.core.temporal*), 89
`check_values()` (*icepyx.core.APIformatting.Parameters* method), 76
`CMRparams` (*icepyx.Query* property), 59
`combine_params()` (in module *icepyx.core.APIformatting*), 76
`convert_string_to_date()` (in module *icepyx.core.temporal*), 90
`cycles` (*icepyx.Query* property), 59
`cycles()` (in module *icepyx.core.validate_inputs*), 93

D

DataOrder (class in *icepyx.core.orders*), 83
`dates` (*icepyx.Query* property), 59
`download()` (*icepyx.core.granules.Granules* method), 78
`download()` (*icepyx.core.orders.DataOrder* method), 84
`download_all()` (*icepyx.Quest* method), 100
`download_granules()` (*icepyx.core.harmony.HarmonyApi* method), 81
`download_granules()` (*icepyx.core.orders.DataOrder* method), 84
`download_granules()` (*icepyx.Query* method), 65

E

EarthdataAuthMixin (class in *icepyx.core.auth*), 77
`end` (*icepyx.core.temporal.Temporal* property), 89
`end_time` (*icepyx.Query* property), 60
`extent` (*icepyx.core.spatial.Spatial* property), 85
`extent_as_gdf` (*icepyx.core.spatial.Spatial* property), 85
`extent_file` (*icepyx.core.spatial.Spatial* property), 86
`extent_type` (*icepyx.core.spatial.Spatial* property), 86
`extract_product()` (in module *icepyx.core.is2ref*), 83
`extract_version()` (in module *icepyx.core.is2ref*), 83

F

`filelist` (*icepyx.Read* property), 69
`files_in_latest_n_cycles()` (in module *icepyx.core.visualization*), 95
`fmt_for_CMR()` (*icepyx.core.spatial.Spatial* method), 86
`fmt_for_EGI()` (*icepyx.core.spatial.Spatial* method), 86
`fnted_keys` (*icepyx.core.APIformatting.Parameters* attribute), 76

G

`generate_OA_parameters()` (*icepyx.core.visualization.Visualize* method), 93
GenQuery (class in *icepyx*), 53
`geodataframe()` (in module *icepyx.core.spatial*), 87
`get_avail()` (*icepyx.core.granules.Granules* method), 79

- get_capabilities() (*icepyx.core.harmony.HarmonyApi* method), 81
- gran_ids() (*in module icepyx.core.granules*), 80
- gran_paras() (*in module icepyx.core.visualization*), 95
- Granules (*class in icepyx.core.granules*), 78
- granules (*icepyx.core.orders.DataOrder* attribute), 84
- granules (*icepyx.Query* property), 60
- grid_bbox() (*icepyx.core.visualization.Visualize* method), 93
- gt2spot() (*in module icepyx.core.is2ref*), 83
- ## H
- HARMONY_BASE_URL (*icepyx.core.orders.DataOrder* attribute), 84
- harmony_client (*icepyx.core.harmony.HarmonyApi* attribute), 80
- harmony_client (*icepyx.core.orders.DataOrder* attribute), 84
- HarmonyApi (*class in icepyx.core.harmony*), 80
- HarmonyTemporal (*class in icepyx.core.harmony*), 82
- ## I
- icepyx.core.APIformatting
module, 75
- icepyx.core.auth
module, 77
- icepyx.core.granules
module, 78
- icepyx.core.harmony
module, 80
- icepyx.core.is2ref
module, 83
- icepyx.core.orders
module, 83
- icepyx.core.spatial
module, 85
- icepyx.core.temporal
module, 89
- icepyx.core.types
module, 95
- icepyx.core.validate_inputs
module, 93
- icepyx.core.visualization
module, 93
- info() (*in module icepyx.core.granules*), 80
- ## J
- job_id (*icepyx.core.orders.DataOrder* attribute), 83
- job_id() (*icepyx.core.orders.DataOrder* method), 84
- job_ids (*icepyx.core.harmony.HarmonyApi* attribute), 81
- ## L
- latest_version() (*icepyx.Query* method), 65
- latest_version() (*in module icepyx.core.is2ref*), 83
- load() (*icepyx.Read* method), 70
- ## M
- make_request() (*icepyx.core.visualization.Visualize* method), 94
- module
- icepyx.core.APIformatting, 75
 - icepyx.core.auth, 77
 - icepyx.core.granules, 78
 - icepyx.core.harmony, 80
 - icepyx.core.is2ref, 83
 - icepyx.core.orders, 83
 - icepyx.core.spatial, 85
 - icepyx.core.temporal, 89
 - icepyx.core.types, 95
 - icepyx.core.validate_inputs, 93
 - icepyx.core.visualization, 93
- ## O
- order_granules() (*icepyx.Query* method), 65
- order_vars (*icepyx.Query* property), 60
- ## P
- parallel_request_0A() (*icepyx.core.visualization.Visualize* method), 94
- Parameters (*class in icepyx.core.APIformatting*), 75
- parse_var_list() (*icepyx.Variables* static method), 72
- partype (*icepyx.core.APIformatting.Parameters* attribute), 76
- path (*icepyx.Variables* property), 71
- pause() (*icepyx.core.orders.DataOrder* method), 85
- pause_order() (*icepyx.core.harmony.HarmonyApi* method), 81
- place_order() (*icepyx.core.granules.Granules* method), 79
- place_order() (*icepyx.core.harmony.HarmonyApi* method), 81
- poss_keys (*icepyx.core.APIformatting.Parameters* property), 76
- prod_version() (*in module icepyx.core.validate_inputs*), 93
- product (*icepyx.Query* property), 60
- product (*icepyx.Read* property), 69
- product (*icepyx.Variables* property), 71
- product_all_info() (*icepyx.Query* method), 66
- product_summary_info() (*icepyx.Query* method), 66
- product_version (*icepyx.Query* property), 61
- ## Q
- Query (*class in icepyx*), 55

- query_icesat2_filelist() (icepyx.core.visualization.Visualize method), 94
- Quest (class in icepyx), 97
- ## R
- Read (class in icepyx), 68
- remove() (icepyx.Variables method), 75
- request_OA_data() (icepyx.core.visualization.Visualize method), 94
- resume() (icepyx.core.orders.DataOrder method), 85
- resume_order() (icepyx.core.harmony.HarmonyApi method), 82
- ## S
- s3login_credentials (icepyx.core.auth.EarthdataAuthMixin property), 78
- save_all() (icepyx.Quest method), 100
- search_all() (icepyx.Quest method), 100
- session (icepyx.core.auth.EarthdataAuthMixin property), 78
- show_custom_options() (icepyx.Query method), 67
- skip_preview() (icepyx.core.harmony.HarmonyApi method), 82
- skip_preview() (icepyx.core.orders.DataOrder method), 85
- Spatial (class in icepyx.core.spatial), 85
- spatial (icepyx.Query property), 61
- spatial_extent (icepyx.Query property), 62
- start (icepyx.core.harmony.HarmonyTemporal attribute), 82
- start (icepyx.core.temporal.Temporal property), 89
- start_time (icepyx.Query property), 62
- status() (icepyx.core.orders.DataOrder method), 85
- stop (icepyx.core.harmony.HarmonyTemporal attribute), 82
- validate_date_range_datestr() (in module icepyx.core.temporal), 90
- validate_date_range_datetime() (in module icepyx.core.temporal), 91
- validate_date_range_dict() (in module icepyx.core.temporal), 91
- validate_polygon_file() (in module icepyx.core.spatial), 88
- validate_polygon_list() (in module icepyx.core.spatial), 88
- validate_polygon_pairs() (in module icepyx.core.spatial), 88
- validate_times() (in module icepyx.core.temporal), 92
- Variables (class in icepyx), 70
- variables (icepyx.Read property), 69
- version (icepyx.Variables property), 71
- Visualize (class in icepyx.core.visualization), 93
- visualize_elevation() (icepyx.Query method), 67
- visualize_spatial_extent() (icepyx.Query method), 67
- viz_elevation() (icepyx.core.visualization.Visualize method), 94
- ## T
- Temporal (class in icepyx.core.temporal), 89
- temporal (icepyx.Query property), 63
- to_string() (in module icepyx.core.APIformatting), 77
- tracks (icepyx.Query property), 63
- tracks() (in module icepyx.core.validate_inputs), 93
- type (icepyx.core.orders.DataOrder attribute), 84
- ## U
- user_check() (in module icepyx.core.visualization), 95
- ## V
- validate_bounding_box() (in module icepyx.core.spatial), 88
- validate_date_range_date() (in module icepyx.core.temporal), 90