icepyx

unknown

May 16, 2024

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	21
6	ICESat-2's Nested Variables	27
7	Visualizing ICESat-2 Elevations	39
8	Reading ICESat-2 Data in for Analysis	43
9	ICESat-2 AWS cloud data access	49
10	QUEST Example: Finding Argo and ICESat-2 data	53
11	icepyx Documentation (API)	59
12	icepyx-QUEST Documentation (API)	105
13	icepyx ChangeLog	111
14	Project Contributors	133
15	Contribution Guidelines	135
16	How to Contribute	139
17	Recognizing Contributions	143
18	icepyx Internals	147
19	QUEST Supported Datasets	149
20	icepyx Development Plan	151
21	Release Guide	153
22	Contributor Covenant Code of Conduct	157

23 ICESat-2 Resource Guide **161**

24 Contact Us **169**

25 Who is using icepyx? **171**

26 icepyx Adoption **173**

Bibliography **175**

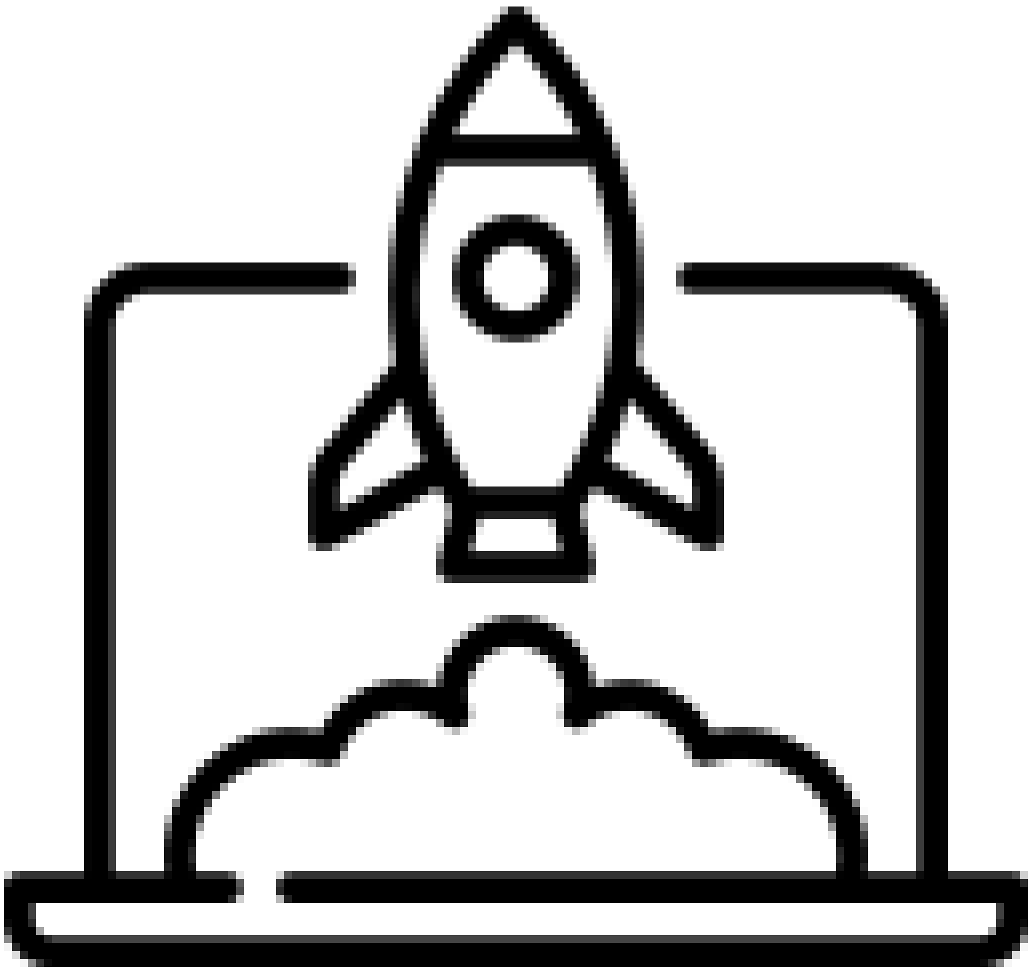
Python Module Index **179**

Index **181**

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

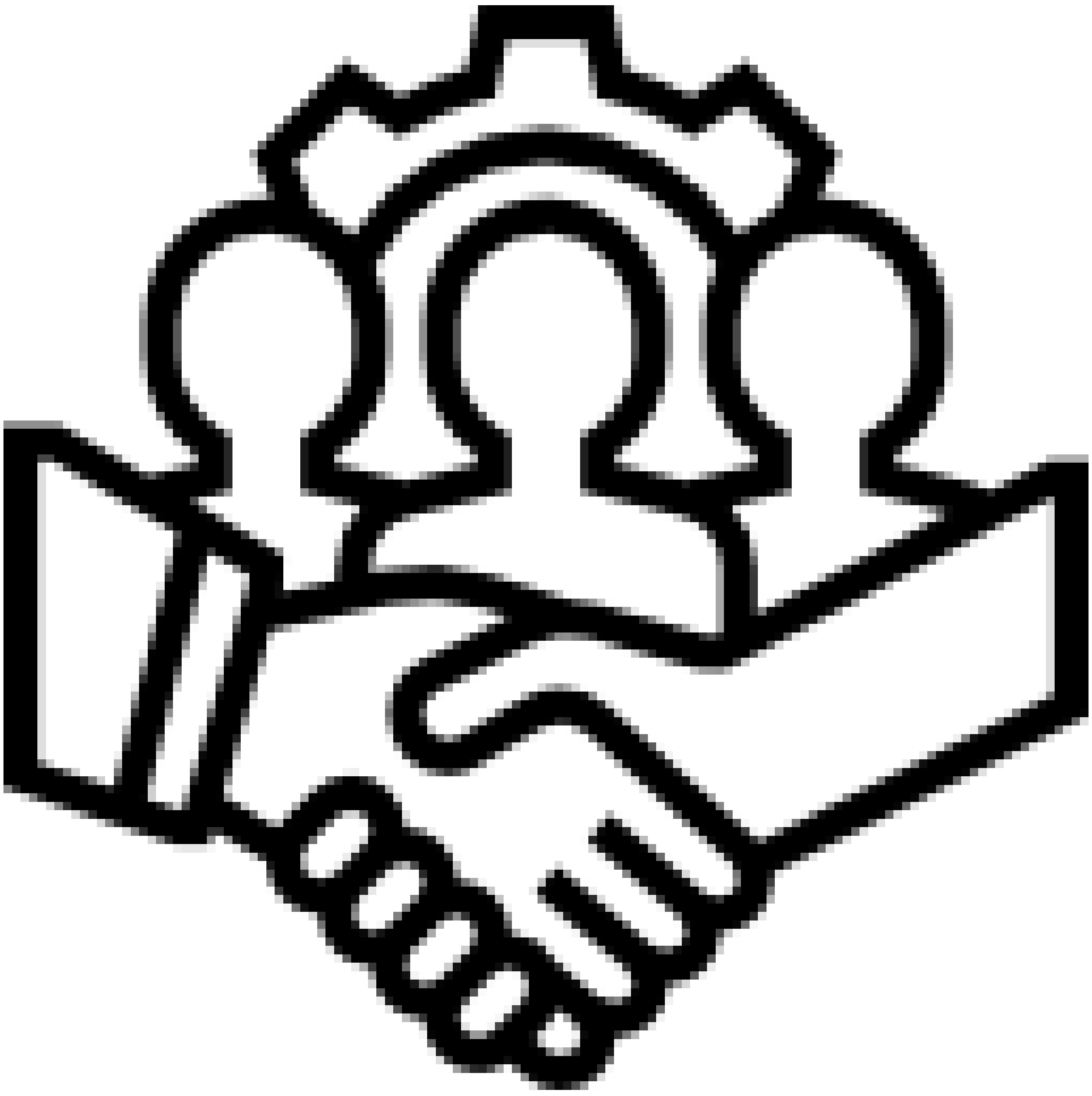
The user guide provides in-depth information on the tools and functionality available for obtaining and interacting with ICESat-2 data products.

Software Docs

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!

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

CHAPTER

ONE

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.

CHAPTER

TWO

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 setup.py and run

```
pip install -e.
```

CHAPTER THREE

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.

CHAPTER

FOUR

ACCESSING ICESAT-2 DATA

This notebook (`download`) illustrates the use of icepyx for programmatic ICESat-2 data query and download from the NASA NSIDC DAAC (NASA National Snow and Ice Data Center Distributed Active Archive Center). 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 NSIDC. 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 NSIDC API:

1. Define your parameters (spatial, temporal, dataset, etc.)
2. Query the NSIDC 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 Create an ICESat-2 data object with the 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 https://nsidc.org/data/icesat-2/products for a list of the available data products.
- `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 (currently shp, kml, and gpkg are 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. Per NSIDC requirements, geospatial polygon files may only contain one feature (polygon).

Then, for all non-gridded products (ATL<=13), 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']
```

Create the data object using our 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 too old a version (such that it is no longer hosted by NSIDC), 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')

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', version='002')
```

4.2.2 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, etc.), 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()
```

4.2.3 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 issued during ordering for each granule to which this applies (but no message is output for successfully subsetted granules, so don't worry!)

```
#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)
```

```
#print detailed information about the returned search results
region_a.granules.avail
```

4.2.4 Log in to NASA Earthdata

When downloading data from NSIDC, 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 make sure 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:

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

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their credentials using one of the three methods decribed above.

4.2.5 Additional Parameters and Subsetting

Once we have generated our session, we must build the required configuration parameters needed to actually download data. These will tell the system how we want to download the data. As with the CMR search parameters, these will be built automatically when you run `region_a.order_granules()`, but you can also create and view them with `region_a.reqparams`. The default parameters, given below, should work for most users.

- `page_size` = 2000. This is the number of granules we will request per order.
- `page_num` = 1. Determine the number of pages based on page size and the number of granules available. If no page_num is specified, this calculation is done automatically to set page_num, which then provides the number of individual orders we will request given the number of granules.
- `request_mode` = ‘async’
- `agent` = ‘NO’
- `include_meta` = ‘Y’

More details about the configuration parameters

`request_mode` is “asynchronous” by default, which allows concurrent requests to be queued and processed without the need for a continuous connection between you and the API endpoint. In contrast, using a “synchronous” `request_mode` means that the request relies on a direct, continous connection between you and the API endpoint. Outputs are directly downloaded, or “streamed”, to your working directory. For this tutorial, we will set the request mode to asynchronous.

Use the streaming `request_mode` with caution: While it can be beneficial to stream outputs directly to your local directory, note that timeout errors can result depending on the size of the request, and your request will not be queued in the system if NSIDC is experiencing high request volume. For best performance, NSIDC recommends setting `page_size=1` to download individual outputs, which will eliminate extra time needed to zip outputs and will ensure faster processing times per request.

Recall that we queried the total number and volume of granules prior to applying customization services. `page_size` and `page_num` can be used to adjust the number of granules per request up to a limit of 2000 granules for asynchronous, and 100 granules for synchronous (streaming). For now, let’s select 9 granules to be processed in each zipped request. For ATL06, the granule size can exceed 100 MB so we want to choose a granule count that provides us with a reasonable zipped download size.

```
print(region_a.reqparams)
# region_a.reqparams['page_size'] = 9
# print(region_a.reqparams)
```

Subsetting

In addition to the required parameters (CMRparams and reqparams) that are submitted with our order, for ICESat-2 data products we can also submit subsetting parameters to NSIDC. For a deeper dive into subsetting, please see our Subsetting Tutorial Notebook, which covers subsetting in more detail, including how to get a list of subsetting options, how to build your list of subsetting parameters, and how to generate a list of desired variables (most datasets have more than 200 variable fields!), including using pre-built default lists (these lists are still in progress and we welcome contributions!).

Subsetting utilizes the NSIDC's built in subsetter to extract only the data you are interested (spatially, temporally, variables of interest, etc.). The advantages of using the NSIDC's subsetter include:

- easily reproducible downloads, particularly when coupled with an icepyx query object
- smaller file size, meaning faster downloads, less storage required, and no need to subset the data on your own
- still easy to go back and order more data/variables with the same or similar search parameters
- no extraneous data means you can move directly to analysis and easily navigate your dataset

Certain subset parameters are specified by default unless `subset=False` is included as an input to `order_granules()` or `download_granules()` (which calls `order_granules()` under the hood). A separate, companion notebook tutorial covers subsetting in more detail, including how to get a list of subsetting options, how to build your list of subsetting parameters, and how to generate a list of desired variables (most products have more than 200 variable fields!), including using pre-built default lists (these lists are still in progress and we welcome contributions!).

As for the CMR and required parameters, default subset parameters can be built and viewed using `subsetparams`. Where an input spatial file is used, rather than a bounding box or manually entered polygon, the spatial file will be used for subsetting (unless subset is set to False) but not show up in the `subsetparams` dictionary.

icepyx also makes it easy to take advantage of the reformatting (e.g. file format conversion) options offered by NSIDC. These are covered in more detail in the Subsetting Tutorial Notebook.

```
region_a.subsetparams()
```

4.2.6 Place the order

Then, we can send the order to NSIDC using the order_granules function. Information about the granules ordered and their status will be printed automatically. Status information can also be emailed to the address associated with your EarthData account when the `email` kwarg is set to `True`. Additional information on the order, including request URLs, can be viewed by setting the optional keyword input 'verbose' to True.

```
region_a.order_granules()
# region_a.order_granules(verbose=True, subset=False, email=False)
```

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

4.2.7 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. Additional information is again available by using the optional boolean keyword `verbose`.

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

Credits

- original notebook by: Jessica Scheick
- notebook contributors: Amy Steiker and Tyler Sutterley
- source material: NSIDC Data Access Notebook by Amy Steiker and Bruce Wallin and 2020 Hackweek Data Access Notebook by Jessica Scheick and Amy Steiker

CHAPTER

FIVE

SUBSETTING ICESAT-2 DATA

This notebook (`download`) illustrates the use of icepyx for subsetting ICESat-2 data ordered through the 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 (aka 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), temporal (i.e. certain months of a time series), and/or certain variables. This process of extracting the data we are going to use is called subsetting.

In the case of ICESat-2 data coming from the 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 (ATL09) for an area along West Greenland (Disko Bay).

```
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')
```

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their credentials.

5.2 Discover Subsetting Options

You can see what subsetting options are available for a given product by calling `show_custom_options()`. The options are presented as a series of headings followed by available values in square brackets. Headings are:

- **Subsetting Options**: whether or not temporal and spatial subsetting are available for the data product
- **Data File Formats (Reformatting Options)**: return the data in a format other than the native hdf5 (submitted as a key=value kwarg to `order_granules(format='NetCDF4-CF')`)
- **Data File (Reformatting) Options Supporting Reprojection**: return the data in a reprojected reference frame. These will be available for gridded ICESat-2 L3B data products.
- **Data File (Reformatting) Options NOT Supporting Reprojection**: data file formats that cannot be delivered with reprojection
- **Data Variables (also Subsettable)**: a dictionary of variable name keys and the paths to those variables available in the product

```
region_a.show_custom_options(dictview=True)
```

By default, spatial and temporal subsetting based on your initial inputs is applied to your order unless you specify `subset=False` to `order_granules()` or `download_granules()` (which calls `order_granules` under the hood if you have not already placed your order) functions. Additional subsetting options must be specified as keyword arguments to the order/download functions.

Although some file format conversions and reprojections are possible using the `format`, `projection`,and `projection_parameters` keywords, the rest of this tutorial will focus on variable subsetting, which is provided with the `Coverage` keyword.

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 NSIDC 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.

Thus, this notebook uses a default list of wanted variables to showcase subsetting and refers the user to the aforementioned Jupyter Notebook for a more thorough exploration of ICESat-2 product variables.

5.3.1 Determine what variables are available for your data product

There are multiple ways to get a complete list of available variables. To increase readability, some display options (2 and 3, below) show the 200+ variable + path combinations as a dictionary where the keys are variable names and the values are the paths to that variable.

1. `region_a.order_vars.avail`, a list of all valid path+variable strings
2. `region_a.show_custom_options(dictview=True)`, all available subsetting options
3. `region_a.order_vars.parse_var_list(region_a.order_vars.avail)`, a dictionary of variable:paths key:value pairs

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

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

Taking advantage of the NSIDC subsetter 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 (e.g. you can keep the same spatial and temporal bounds but change the variable list). Related tools (e.g. `captoolkit`*) will let you easily merge files if you're uncomfortable merging them during read-in for processing.*

5.4.1 Building the default wanted variable list

```
region_a.order_vars.wanted
```

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

5.5 Applying variable subsetting to your order and download

In order to have your wanted variable list included with your order, you must pass it as a keyword argument to the `subsetparams()` attribute or the `order_granules()` or `download_granules()` (which calls `order_granules` under the hood if you have not already placed your order) functions.

```
region_a.subsetparams(Coverage=region_a.order_vars.wanted)
```

Or, you can put the `Coverage` parameter directly into `order_granules`: `region_a.order_granules(Coverage=region_a.order_vars.wanted)`

However, then you cannot view your subset parameters (`region_a.subsetparams`) prior to submitting your order.

```
region_a.order_granules()# <-- you do not need to include the 'Coverage' kwarg to
                             # order if you have already included it in a call to␣
↪subsetparams
```

```
region_a.download_granules('/home/jovyan/icepyx/dev-notebooks/vardata') # <-- you do not␣
↪need to include the 'Coverage' kwarg to
                             # download if you have already submitted it with your order
```

5.5.1 *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.

5.5.2 Check the variable list in your downloaded file

Compare the available variables associated with the full product relative to those in your downloaded data file.

```
# put the full filepath to a data file here. You can get this in JupyterHub by␣
↪navigating to the file,
# right clicking, and selecting copy path. Then you can paste the path in the quotes␣
↪below.
fn = ''
```

5.6 Check the downloaded data

Get all `latitude` variables in your downloaded file:

```
varname = 'latitude'

varlist = []
def IS2h5walk(vname, h5node):
    if isinstance(h5node, h5py.Dataset):
        varlist.append(vname)
```

(continues on next page)

(continued from previous page)

```
    return

with h5py.File(fn,'r') as h5pt:
    h5pt.visititems(IS2h5walk)

for tvar in varlist:
    vpath,vn = os.path.split(tvar)
    if vn==varname: print(tvar)
```

5.6.1 Compare to the variable paths available in the original data

```
region_a.order_vars.parse_var_list(region_a.order_vars.avail)[0][varname]
```

Credits

- notebook contributors: Zheng Liu, Jessica Scheick, and Amy Steiker
- some source material: NSIDC Data Access Notebook by Amy Steiker and Bruce Wallin

CHAPTER

SIX

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, whether that's via order and download (e.g. via NSIDC DAAC) or remote (e.g. via the cloud).
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 `.order_vars` property of a Query object
2. Access via the `.vars` 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 `.order_vars` 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')
```

```
# Accessing Variables
region_a.order_vars
```

```
# Showing the variable paths
region_a.order_vars.avail()
```

6.2.2 2. Access via the `.vars` property of a Read object

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

```
# Accessing Variables
reader.vars
```

```
# Showing the variable paths
# reader.vars.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 a 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 availble 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, either available from the NSIDC or 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#l"

# 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')
```

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their credentials.

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 heirarchy (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.order_vars.avail` will return a list of all valid path+variable strings.

```
region_a.order_vars.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.order_vars.parse_var_list(region_a.order_vars.avail())` will return a dictionary of variable:paths key:value pairs.

```
region_a.order_vars.parse_var_list(region_a.order_vars.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.order_vars.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.vars.avail()
reader.vars.parse_var_list(reader.vars.avail())
reader.vars.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.order_vars.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.order_vars.wanted
```

```
region_a.order_vars.append(defaults=True)
pprint(region_a.order_vars.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.order_vars.append(keyword_list=[''])
```

6.3.3 Modifying your wanted variable list

Generating and modifying your variable request list, which is stored in `region_a.order_vars.wanted`, is controlled by the `append` and `remove` functions that operate on `region_a.order_vars.wanted`. The input options to `append` are as follows (the full documentation for this function can be found by executing `help(region_a.order_vars.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.order_vars.wanted` to None
- `var_list` (as above)
- `beam_list` (as above)

- `keyword_list` (as above)

```
region_a.order_vars.remove(all=True)
pprint(region_a.order_vars.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.order_vars.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.order_vars.append(var_list=['latitude','longitude'])
pprint(region_a.order_vars.wanted)
```

Example 1.2: specify beams and variable

Add `latitude` for only `gt1l` and `gt2l`

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

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

Example 1.3: add/remove selected beams+variables

Add `latitude` for `gt3l` and remove it for `gt2l`

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

Example 1.4: keyword_list

Add latitude and longitude for all beams and with keyword land_ice_segments

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

Example 1.5: target a specific variable + path

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

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

Example 1.6: add variables not specific to beams/profiles

Add rgt under orbit_info.

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

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

Remove all paths for gt1l and gt3r

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

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

Generate a reasonable variable list prior to download

```
region_a.order_vars.remove(all=True)
region_a.order_vars.append(defaults=True)
pprint(region_a.order_vars.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.order_vars.append(var_list=['latitude','longitude'])
pprint(region_a.order_vars.wanted)
```

Example 2.2: specify beams/profiles and variable

Add latitude for only profile_1 and profile_2

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

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

Example 2.3: add/remove selected beams+variables

Add latitude for profile_3 and remove it for profile_2

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

Example 2.4: `keyword_list`

Add `latitude` for all profiles and with keyword `low_rate`

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

Example 2.5: target a specific variable + path

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

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

Example 2.6: add variables not specific to beams/profiles

Add `rgt` under `orbit_info`.

```
region_a.order_vars.append(keyword_list=['orbit_info'],var_list=['rgt'])
pprint(region_a.order_vars.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.order_vars.wanted`, such as `'orbit_info/rgt'`, are not duplicated.

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

Example 2.8: add all possible values for variables+paths

Append all `longitude` paths and all variables/paths with keyword `high_rate`. Simlarly 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.order_vars.append(var_list=['longitude'])
region_a.order_vars.append(keyword_list=['high_rate'])
pprint(region_a.order_vars.wanted)
```

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

Remove all paths for `profile_1` and `profile_3`

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

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

Generate a reasonable variable list prior to download

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

6.3.7 Using your wanted variable list

Now that you have your wanted variables list, you need to use it within your icepyx object (`Query` or `Read`) will automatically use it.

With a `Query` object

In order to have your wanted variable list included with your order, you must pass it as a keyword argument to the `subsetparams()` attribute or the `order_granules()` or `download_granules()` (which calls `order_granules` under the hood if you have not already placed your order) functions.

```
region_a.subsetparams(Coverage=region_a.order_vars.wanted)
```

Or, you can put the `Coverage` parameter directly into `order_granules`: `region_a.order_granules(Coverage=region_a.order_vars.wanted)`

However, then you cannot view your subset parameters (`region_a.subsetparams`) prior to submitting your order.

```
region_a.order_granules()# <-- you do not need to include the 'Coverage' kwarg to
                             # order if you have already included it in a call to␣
↪subsetparams
```

```
region_a.download_granules('/home/jovyan/icepyx/dev-notebooks/vardata') # <-- you do not␣
↪need to include the 'Coverage' kwarg to
                             # download if you have already submitted it with your order
```

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

CHAPTER

SEVEN

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.

Import packages

```
import icepyx as ipx
```

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)
```

(continues on next page)

(continued from previous page)

```
print(list(set(region.avail_granules(cycles=True)[0]))) #region.cycles
print(list(set(region.avail_granules(tracks=True)[0]))) #region.tracks
```

7.1 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()
```

7.2 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 splited 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
```

7.2.1 Plot elevation for individual RGT

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

```
rgtmap
```

7.2.2 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)
```

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their credentials.

7.2.3 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

CHAPTER

EIGHT

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 delimeter 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

```python
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)
```

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their credentials.

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/myfavoriteicsat-2file.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)
```

Read Module Update

Previously, icepyx required two additional conditions: 1) a `product` argument and 2) that your files either matched the default `filename_pattern` or that the user provided their own `filename_pattern`. These two requirements have been removed. `product` is now read directly from the file metadata (the root group's `short_name` attribute). Flexibility to specify multiple files via the `filename_pattern` has been replaced with the glob string feature, and by allowing a list of filepaths as an argument.

The `product` and `filename_pattern` arguments are now deprecated and were removed in icepyx version 1.0.0.

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

CHAPTER

NINE

ICESAT-2 AWS CLOUD DATA ACCESS

This notebook (`download`) illustrates the use of icepyx for accessing ICESat-2 data currently available through the AWS (Amazon Web Services) us-west2 hub 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 NSIDC, 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, sharing your code, etc.) 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 list of 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.avail()
```

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 it too time intensive with the s3 data, so instead the the product / version of the data product is read from the file and all possible variables associated with that product/version are reporting as available. As long as you are using the NSIDC provided s3 paths provided via Earthdata search 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 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, so 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
```

Important Authentication Update

Previously, icepyx required you to explicitly use the `.earthdata_login()` function to login. Running this function is deprecated and will result in an error, as icepyx will call the login function as needed. The user will still need to provide their 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.vars.avail()
```

Next, we append our desired variable to the `wanted_vars` list:

```
reader.vars.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 recieve 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 which are enabled to use the fastest low level data readers. Not all of these pieces fully developed right now, but the progress being made it exciting and there is lots of room for contribution!

Credits

- notebook by: Jessica Scheick and Rachel Wegener

CHAPTER

TEN

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 anologously 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 [$\degree$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 [$\degree$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)
```

CHAPTER

ELEVEN

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

GenQuery([spatial_extent, date_range, ...])	Base class for querying data.
Query([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-latitute, 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
[boolean, 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.

See also:

`Query`
`Quest`

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.

```
>>> aoi = str(Path('./doc/source/example_notebooks/supporting_files/simple_test_
↪poly.gpkg').resolve())
>>> reg_a_dates = ['2019-02-22','2019-02-28']
>>> reg_a = GenQuery(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)
```

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

Methods

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

Attributes

`dates`	Return an array showing the date range of the query object.
`end_time`	Return the end time specified for the end date.
`spatial`	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
`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-latitute, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].
`start_time`	Return the start time specified for the start date.
`temporal`	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
[string] 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
[string, default most recent version] Product version, given as a 3 digit string. If no version is given, the current version is used. Example: "006"

cycles
[string or a list of strings, default all available orbital cycles] Product cycle, given as a 2 digit string. If no cycle is given, all available cycles are used. Example: "04"

tracks
[string or a list of strings, default all available reference ground tracks (RGTs)] Product track, given as a 4 digit string. 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.

Returns

query object

See also:

GenQuery

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 v006
('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.

```
>>> aoi = str(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', aoi, reg_a_dates)
>>> print(reg_a)
Product ATL06 v006
('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']
```

__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

__init__([product, spatial_extent, ...])	
avail_granules([ids, cycles, tracks, cloud])	Obtain information about the available granules for the query object's parameters.
download_granules(path[, verbose, subset, ...])	Downloads the data ordered using order_granules.
earthdata_login([uid, email, s3token])	Authenticate with NASA Earthdata to enable data ordering and download.
latest_version()	A reference function to is2ref.latest_version.
order_granules([verbose, subset, email])	Place an order for the available granules for the query object.
product_all_info()	Display all metadata about the product of interest (the collection).
product_summary_info()	Display a summary of selected metadata for the specified version of the product of interest (the collection).
show_custom_options([dictview])	Display customization/subsetting options available for this product.
subsetparams(**kwargs)	Display the subsetting key:value pairs that will be submitted.
visualize_elevation()	Visualize elevation requested from OpenAltimetry API using datashader based on cycles https://holoviz.org/tutorial/Large_Data.html
visualize_spatial_extent()	Creates a map displaying the input spatial extent

Attributes

CMRparams	Display the CMR key:value pairs that will be submitted.
auth	Authentication object returned from earthaccess.login() which stores user authentication.
cycles	Return the unique ICESat-2 orbital cycle.
dataset	Legacy property included to provide depracation warning.
dates	Return an array showing the date range of the query object.
end_time	Return the end time specified for the end date.
granules	Return the granules object, which provides the underlying funtionality for searching, ordering, and downloading granules for the specified product.
order_vars	Return the order variables object.
product	Return the short name product ID string associated with the query object.
product_version	Return the product version of the data object.
reqparams	Display the required key:value pairs that will be submitted.
s3login_credentials	A dictionary which stores login credentials for AWS s3 access.
session	Earthaccess session object for connecting to Earthdata resources.
spatial	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
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-latitute, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].
start_time	Return the start time specified for the start date.
temporal	Return the Temporal object containing date/time range information for the query object.
tracks	Return the unique ICESat-2 Reference Ground Tracks

11.1.2 Attributes

Query.CMRparams	Display the CMR key:value pairs that will be submitted.
Query.cycles	Return the unique ICESat-2 orbital cycle.
Query.dates	Return an array showing the date range of the query object.
Query.end_time	Return the end time specified for the end date.
Query.granules	Return the granules object, which provides the underlying funtionality for searching, ordering, and downloading granules for the specified product.
Query.order_vars	Return the order variables object.
Query.product	Return the short name product ID string associated with the query object.
Query.product_version	Return the product version of the data object.
Query.reqparams	Display the required key:value pairs that will be submitted.
Query.spatial	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
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-latitute, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].
Query.subsetparams(**kwargs)	Display the subsetting key:value pairs that will be submitted.
Query.start_time	Return the start time specified for the start date.
Query.temporal	Return the Temporal object containing date/time range information for the query object.
Query.tracks	Return the unique ICESat-2 Reference Ground Tracks

icepyx.Query.CMRparams

property Query.**CMRparams**

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

Examples

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

icepyx.Query.cycles

property Query.**cycles**

Return the unique ICESat-2 orbital cycle.

Examples

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

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.

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**

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 funtionality 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**

Return the order variables object. This instance is generated when data is ordered from the NSIDC.

See also:

variables.Variables

Examples

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

icepyx.Query.product

property Query.**product**

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

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.

Examples

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

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'],␣
↪version='4')
>>> reg_a.product_version
'004'
```

icepyx.Query.reqparams

property Query.**reqparams**

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

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.reqparams
{'short_name': 'ATL06', 'version': '006', 'page_size': 2000}
```

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.order_granules()
>>> reg_a.reqparams
{'short_name': 'ATL06', 'version': '006', 'page_size': 2000, 'page_num': 1,
↪'request_mode': 'async', 'include_meta': 'Y', 'client_string': 'icepyx'}
```

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-latitute,

... 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.

See also:

`Spatial.extent`
`Spatial.extent_type`
`Spatial.extent_as_gdf`

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

icepyx.Query.subsetparams

`Query.subsetparams(**kwargs)`

Display the subsetting key:value pairs that will be submitted. It generates the dictionary if it does not already exist and returns an empty dictionary if subsetting is set to False during ordering.

Parameters

****kwargs**
[key-value pairs] Additional parameters to be passed to the subsetter. By default temporal and spatial subset keys are passed. Acceptable key values are ['format','projection','projection_parameters','Coverage']. At this time (2020-05), only variable ('Coverage') parameters will be automatically formatted.

See also:

order_granules

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.subsetparams()
{'time': '2019-02-20T00:00:00,2019-02-28T23:59:59',
'bbox': '-55.0,68.0,-48.0,71.0'}
```

icepyx.Query.start_time

property `Query.start_time`

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**

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 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

Query.avail_granules([ids, cycles, tracks, ...])	Obtain information about the available granules for the query object's parameters.
Query.download_granules(path[, verbose, ...])	Downloads the data ordered using order_granules.
Query.earthdata_login([uid, email, s3token])	Authenticate with NASA Earthdata to enable data ordering and download.
Query.latest_version()	A reference function to is2ref.latest_version.
Query.order_granules([verbose, subset, email])	Place an order for the available granules for the query object.
Query.product_all_info()	Display all metadata about the product of interest (the collection).
Query.product_summary_info()	Display a summary of selected metadata for the specified version of the product of interest (the collection).
Query.show_custom_options([dictview])	Display customization/subsetting options available for this product.
Query.visualize_spatial_extent()	Creates a map displaying the input spatial extent
Query.visualize_elevation()	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
[boolean, default False] Indicates whether the function should return a list of granule IDs.

cycles
[boolean, default False] Indicates whether the function should return a list of orbital cycles.

tracks
[boolean, default False] Indicates whether the function should return a list of RGTs.

cloud
[boolean, 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-premesis 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)': 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_006_01.h5', 'ATL06_20190222010344_08490205_006_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*, *verbose=False*, *subset=True*, *restart=False*, ***kwargs*)

Downloads the data ordered using order_granules.

Parameters

path
[string] String with complete path to desired download location.

verbose
[boolean, default False] Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of verbose.

subset
[boolean, 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)

restart
[boolean, default false] If previous download was terminated unexpectedly. Run again with restart set to True to continue.

****kwargs**
[key-value pairs] Additional parameters to be passed to the subsetter. By default temporal and spatial subset keys are passed. Acceptable key values are ['format','projection','projection_parameters','Coverage']. The variable 'Coverage' list should be constructed using the *order_vars.wanted* attribute of the object. At this time (2020-05), only variable ('Coverage') parameters will be automatically formatted.

See also:

`granules.download`

icepyx.Query.earthdata_login

`Query.earthdata_login(uid=None, email=None, s3token=None, **kwargs) → None`

Authenticate with NASA Earthdata to enable data ordering and download. Credential storage details are described in the EathdataAuthMixin class section.

Note: This method is deprecated and will be removed in a future release. It is no longer required to explicitly run *.earthdata_login()*. Authentication will be performed by the module as needed.

Parameters

uid
[string, default None] Deprecated keyword for Earthdata login user ID.

email
[string, default None] Deprecated keyword for backwards compatibility.

s3token
[boolean, default None] Deprecated keyword to generate AWS s3 ICESat-2 data access credentials

kwargs
[key:value pairs] Keyword arguments to be passed into earthaccess.login().

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.earthdata_login()
Enter your Earthdata Login username: ___________________
```

EARTHDATA_USERNAME and EARTHDATA_PASSWORD are not set in the current environment, try setting them or use a different strategy (netrc, interactive) No .netrc found in /Users/username

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()
'006'
```

icepyx.Query.order_granules

`Query.order_granules`(*verbose=False*, *subset=True*, *email=False*, ***kwargs*)

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

Parameters

verbose
[boolean, default False] Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of verbose.

subset
[boolean, 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)

email: boolean, default False
Have NSIDC auto-send order status email updates to indicate order status as pending/completed. The emails are sent to the account associated with your Earthdata account.

****kwargs**
[key-value pairs] Additional parameters to be passed to the subsetter. By default temporal and spatial subset keys are passed. Acceptable key values are ['format','projection','projection_parameters','Coverage']. The variable 'Coverage' list should be constructed using the *order_vars.wanted* attribute of the object. At this time (2020-05), only variable ('Coverage') parameters will be automatically formatted.

See also:

`granules.place_order`

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.order_granules()
order ID: [###############]
[order status output]
error messages:
[if any were returned from the NSIDC subsetter, e.g. No data found that matched␣
↪subset constraints.]
.
.
.
Retry request status is: complete
```

icepyx.Query.product_all_info

`Query.product_all_info()`

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

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).

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'],␣
↪version='006')
>>> reg_a.product_summary_info()
title :  ATLAS/ICESat-2 L3A Land Ice Height V006
short_name :  ATL06
version_id :  006
time_start :  2018-10-14T00:00:00.000Z
coordinate_system :  CARTESIAN
summary :  This data set (ATL06) provides geolocated, land-ice surface heights␣
↪(above the WGS 84 ellipsoid, ITRF2014 reference frame), 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 Ice, Cloud and land Elevation Satellite-2 (ICESat-2)␣
↪observatory.
orbit_parameters :  {}
```

icepyx.Query.show_custom_options

`Query.show_custom_options(`*dictview=False*`)`

Display customization/subsetting options available for this product.

Parameters

dictview
[boolean, default False] Show the variable portion of the custom options list as a dictionary with key:value pairs representing variable:paths-to-variable rather than as a long list of full variable paths.

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.show_custom_options(dictview=True)
Subsetting options
[{'id': 'ICESAT2',
'maxGransAsyncRequest': '2000',
'maxGransSyncRequest': '100',
'spatialSubsetting': 'true',
'spatialSubsettingShapefile': 'true',
'temporalSubsetting': 'true',
'type': 'both'}]
Data File Formats (Reformatting Options)
['TABULAR_ASCII', 'NetCDF4-CF', 'Shapefile', 'NetCDF-3']
Reprojection Options
[]
Data File (Reformatting) Options Supporting Reprojection
['TABULAR_ASCII', 'NetCDF4-CF', 'Shapefile', 'NetCDF-3', 'No reformatting']
Data File (Reformatting) Options NOT Supporting Reprojection
[]
Data Variables (also Subsettable)
['ancillary_data/atlas_sdp_gps_epoch',
'ancillary_data/control',
'ancillary_data/data_end_utc',
.
.
.
'quality_assessment/gt3r/signal_selection_source_fraction_3']
```

icepyx.Query.visualize_spatial_extent

Query.**visualize_spatial_extent**()

Creates a map displaying the input spatial extent

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 objects] Holoviews data visualization elements

11.2 Read Class

11.2.1 Constructor

Read(data_source[, glob_kwargs, ...])	Data object to read ICESat-2 data into the specified formats.

icepyx.Read

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

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

Parameters

data_source
[string, 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!

product
[string] 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 **Deprecation warning:** This argument is no longer required and has been deprecated. The dataset product is read from the file metadata.

filename_pattern
[string, default None] String that shows the filename pattern as previously required for Intake's path_as_pattern argument. The default describes files downloaded directly from NSIDC (subsetted and non-subsetted) for most products (e.g. ATL06). The ATL11 filename pattern from NSIDC is: 'ATL{product:2}_{rgt:4}{orbitsegment:2}_{cycles:4}_{version:3}_{revision:2}.h5'. **Deprecation warning:** This argument is no longer required and has been deprecated.

catalog
[string, default None] Full path to an Intake catalog for reading in data. If you still need to create a catalog, leave as default. **Deprecation warning:** This argument has been deprecated. Please use the data_source argument to pass in valid data.

Returns

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*, *product=None*, *filename_pattern=None*, *catalog=None*)

Methods

`__init__`(data_source[, glob_kwargs, ...])	
`earthdata_login`([uid, email, s3token])	Authenticate with NASA Earthdata to enable data ordering and download.
`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.
`session`	Earthaccess session object for connecting to Earthdata resources.
`vars`	Return the variables object associated with the data being read in.

11.2.2 Attributes

`Read.filelist`	Return the list of files represented by this Read object.
`Read.product`	Return the product associated with the Read object.
`Read.vars`	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.vars

property Read.**vars**

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.vars
<icepyx.core.variables.Variables at [location]>
```

11.2.3 Methods

Read.load()	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

`Variables`([vartype, path, product, version, ...])	Get, create, interact, and manipulate lists of variables and variable paths contained in ICESat-2 products.

icepyx.Variables

class icepyx.**Variables**(*vartype=None*, *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

vartype
[string] This argument is deprecated. The vartype will be inferred from data_source. One of ['order', 'file'] to indicate the source of the input variables. This field will be auto-populated when a variable object is created as an attribute of a query object.

path
[string, 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
[string, 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
[string, default None] Properly formatted string specifying a valid version of the ICESat-2 product.

avail
[dictionary, default None] Dictionary (key:values) of available variable names (keys) and paths (values).

wanted
[dictionary, 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__(*vartype=None*, *path=None*, *product=None*, *version=None*, *avail=None*, *wanted=None*, *auth=None*)

Methods

`__init__`([vartype, path, product, version, ...])	
`append`([defaults, var_list, beam_list, ...])	Add to the list of desired variables using user specified beams and variable list.
`avail`([options, internal])	Get the list of available variables and variable paths from the input data product
`earthdata_login`([uid, email, s3token])	Authenticate with NASA Earthdata to enable data ordering and download.
`parse_var_list`(varlist[, tiered, tiered_vars])	Parse a list of path strings into tiered lists and names of variables
`remove`([all, var_list, beam_list, keyword_list])	Remove the variables and paths from the wanted list using user specified beam, keyword,

Attributes

`auth`	Authentication object returned from earthaccess.login() which stores user authentication.
`path`	
`product`	
`s3login_credentials`	A dictionary which stores login credentials for AWS s3 access.
`session`	Earthaccess session object for connecting to Earthdata resources.
`version`	

11.3.2 Methods

`Variables.avail`([options, internal])	Get the list of available variables and variable paths from the input data product
`Variables.parse_var_list`(varlist[, tiered, ...])	Parse a list of path strings into tiered lists and names of variables
`Variables.append`([defaults, var_list, ...])	Add to the list of desired variables using user specified beams and variable list.
`Variables.remove`([all, var_list, beam_list, ...])	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.order_vars.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 of strings] List of full variable paths to be parsed.

tiered
[boolean, 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
[boolean, 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.order_vars.parse_var_list(reg_a.order_vars.avail())
>>> var_dict
{'atlas_sdp_gps_epoch': ['ancillary_data/atlas_sdp_gps_epoch'],
.
.
.
```

(continues on next page)

(continued from previous page)

```
'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',
```

(continues on next page)

(continued from previous page)

```
'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
[boolean, default False] Include the variables in the default variable list. Defaults are defined per-data product. When specified in conjuction with a var_list, default variables not on the user- specified list will be added to the order.

var_list
[list of strings, 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 of strings, 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 of strings, default None] A list of subdirectory names (keywords), from any heirarchy level within the data structure, to select variables within the product that include that keyword in their path. A list of availble 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.order_vars.append(beam_list=['gt1r'])
```

To include the default variables:

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

To add specific variables in orbit_info

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

To add all variables and paths in ancillary_data

```
>>> reg_a.order_vars.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
[boolean, default False] Remove all variables and paths from the wanted list.

var_list
[list of strings, 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 of strings, 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 of strings, default None] A list of subdirectory names (keywords), from any heirarchy 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.order_vars.remove(all=True)
```

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

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

To remove specific variables in orbit_info

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

To remove all variables and paths in ancillary_data

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

11.4 Components

11.4.1 APIformatting

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

Bases: `object`

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

Parameters

partype
[string] Type of parameter list. Must be one of ['CMR','required','subset']

values
[dictionary, default None] Dictionary of already-formatted parameters, if there are any, to avoid re-creating them.

reqtype
[string, 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

`check_req_values`()

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

`check_values`()

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

property `fmted_keys`

Returns the dictionary of formated keys associated with the parameter object.

property `poss_keys`

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.

Parameters

params
[dictionaries] Unlimited number of dictionaries to combine

Returns

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
[dictionary]

Returns

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 method *earthdata_login()* is included for backwards compatibility.

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.

`earthdata_login`(*uid=None*, *email=None*, *s3token=None*, ***kwargs*) → None

Authenticate with NASA Earthdata to enable data ordering and download. Credential storage details are described in the EathdataAuthMixin class section.

Note: This method is deprecated and will be removed in a future release. It is no longer required to explicitly run *.earthdata_login()*. Authentication will be performed by the module as needed.

Parameters

uid
[string, default None] Deprecated keyword for Earthdata login user ID.

email
[string, default None] Deprecated keyword for backwards compatibility.

s3token
[boolean, default None] Deprecated keyword to generate AWS s3 ICESat-2 data access credentials

kwargs
[key:value pairs] Keyword arguments to be passed into earthaccess.login().

Examples

```
>>> reg_a = ipx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> reg_a.earthdata_login()
Enter your Earthdata Login username: ___________________
```

EARTHDATA_USERNAME and EARTHDATA_PASSWORD are not set in the current environment, try setting them or use a different strategy (netrc, interactive) No .netrc found in /Users/username

`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.

Returns

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
[boolean, default False] Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of verbose.

path
[string] String with complete path to desired download directory and location.

restart
[boolean, 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.

See also:

`query.Query.download_granules`

Notes

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

`get_avail`(*CMRparams*, *reqparams*, *cloud=False*)

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

Parameters

CMRparams
[dictionary] Dictionary of properly formatted CMR search parameters.

reqparams
[dictionary] Dictionary of properly formatted parameters required for searching, ordering, or downloading from NSIDC.

cloud
[deprecated, boolean, default False] CMR metadata is always collected for the cloud system.

See also:

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

Notes

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

`place_order`(*CMRparams*, *reqparams*, *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
[dictionary] Dictionary of properly formatted CMR search parameters.

reqparams
[dictionary] Dictionary of properly formatted parameters required for searching, ordering, or downloading from NSIDC.

subsetparams
[dictionary] Dictionary of properly formatted subsetting parameters. An empty dictionary is passed as input here when subsetting is set to False in query methods.

verbose
[boolean, default False] Print out all feedback available from the order process. Progress information is automatically printed regardless of the value of verbose.

subset
[boolean, 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
[string, default None] String of the full filename and path when the spatial input is a file.

See also:

`query.Query.order_granules`

Notes

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

`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: boolean, default True
Return a list of the available granule IDs for the granule dictionary

cycles
[boolean, default False] Return a list of the available orbital cycles for the granule dictionary

tracks
[boolean, default False] Return a list of the available Reference Ground Tracks (RGTs) for the granule dictionary

dates
[boolean, default False] Return a list of the available dates for the granule dictionary.

cloud
[boolean, 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 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: string
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: string
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')
'006'
```

11.4.5 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
[array] An array of bounding coordinates.

property extent_as_gdf

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

Returns

extent_gdf
[GeoDataframe] A GeoDataframe containing the spatial region of interest.

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
```

```
>>> 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

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

`fmt_for_EGI()`

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

string
Properly formatted json string for submission to EGI (NSIDC API).

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

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

Parameters

extent_type
[string] 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

boolean
indicating whether or not the spatial extent crosses the dateline.

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

Return a geodataframe of the spatial extent

Parameters

extent_type
[string] One of 'bounding_box' or 'polygon', indicating what type of input the spatial extent is

spatial_extent
[string 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-latitute, 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
[boolean, default False] Indication for whether the spatial_extent string is a filename or coordinate list

Returns

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

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 ((-48.00000 68.00000, -48.00000 71.000...
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: string
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).

11.4.6 temporal

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

Bases: `object`

property end

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

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

Returns

boolean (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: string

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

- “YYYY-MM-DD”
- “YYYY-DOY”

Returns

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(str)
A date range provided in the form of a list of datetime.dates. List must be of length 2.

start_time: string or datetime.time
end_time: string or datetime.time

Returns

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: string, datetime.time, None
end_time: string, datetime.time, None

Returns

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.datetime)
A date range provided in the form of a list of datetimes. List must be of length 2.

start_time: None, string, datetime.time
end_time: None, string, datetime.time
NOTE: If start and/or end times are given,
they will be **ignored in favor of the time from the start/end datetime.datetime objects.**

Returns

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, datetime.datetime, datetime.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: string or datetime.time
end_time: string or datetime.time

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: string, datetime.time, None
end_time: string, datetime.time, None

Returns

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))
```

11.4.7 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.

11.4.8 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
[ipx.Query object, default None] icepy Query class object.

product
[string] ICESat-2 product ID

spatial_extent: list or string, default None
as in the ipx.Query object

date_range
[list of 'YYYY-MM-DD' strings, default None] as in the ipx.Query object

cycle
[string, default all available orbital cycles, default None] as in the ipx.Query object

track
[string, default all available reference ground tracks (RGTs), default None] as in the ipx.Query object

See also:

`ipx.Query`

`generate_OA_parameters()` → list

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

Returns

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

`grid_bbox`(*binsize=5*) → list

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

Returns

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

`make_request`(*base_url*, *payload*)

Make HTTP request

Parameters

base_url
[string] OpenAltimetry URL

See also:

`request_OA_data`

`parallel_request_OA()` → array

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
[dask.Array] A dask array containing the ICESat-2 data.

`query_icesat2_filelist()` → tuple

Query list of ICESat-2 files for each bounding box

Returns

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

`request_OA_data`(*paras*) → array

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
[da.array] A dask array containing the ICESat-2 elevation data.

`viz_elevation`() -> (<class 'holoviews.core.spaces.DynamicMap'>, <class 'holoviews.core.layout.Layout'>)

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 objects] Holoviews data visualization elements

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

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
[list] A list of file names from latest n ICESat-2 cycles

`icepyx.core.visualization.gran_paras`(*filename*) → list

Returns a list of granule information for file name string.

Parameters

filename: String
ICESat-2 file name

Returns

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

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

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

Parameters

message
[string] Message to indicate users the options

CHAPTER

TWELVE

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

`GenQuery`([spatial_extent, date_range, ...])	Base class for querying data.
`Quest`(spatial_extent, date_range[, ...])	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

Returns

quest object

See also:

`GenQuery`

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
```

`__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

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

Attributes

`dates`	Return an array showing the date range of the query object.
`end_time`	Return the end time specified for the end date.
`spatial`	Return the spatial object, which provides the underlying functionality for validating and formatting geospatial objects.
`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-latitute, ... upper-right-longitude, upper-right-latitude]. Polygon data is [longitude1, latitude1, longitude2, latitude2, ... longitude_n,latitude_n, longitude1,latitude1].
`start_time`	Return the start time specified for the start date.
`temporal`	Return the Temporal object containing date/time range information for the query object.

12.1.2 Methods

`Quest.add_icesat2`(product[, start_time, ...])	Adds ICESat-2 datasets to QUEST structure.
`Quest.add_argo`([params, presRange])	Adds Argo (including Argo-BGC) to QUEST structure.
`Quest.search_all`(**kwargs)	Searches for requred dataset within platform (i.e. ICESat-2, Argo) of interest.
`Quest.download_all`([path])	Downloads requested dataset(s).
`Quest.save_all`(path)	Saves all datasets according to their respective *.save()* functionality.

icepyx.Quest.add_icesat2

`Quest.`**`add_icesat2`**(*product*, *start_time=None*, *end_time=None*, *version=None*, *cycles=None*, *tracks=None*, *files=None*, ***kwargs*) → None

Adds ICESat-2 datasets to QUEST structure.

Parameters

For details on inputs, see the Query documentation.

Returns

None

See also:

`icepyx.core.GenQuery`
`icepyx.core.Query`

icepyx.Quest.add_argo

`Quest.add_argo`(*params=['temperature'], presRange=None*) → None

Adds Argo (including Argo-BGC) to QUEST structure.

Parameters

For details on inputs, see the Argo dataset script documentation.

Returns

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 requred 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.

CHAPTER THIRTEEN

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 1.0.0)

13.1.1 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 4 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Rachel Wegener
- Whyjay Zheng +
- allcontributors[bot]
- Kelsey Bisson
- Zach Fair
- Romina Piunno

13.2 Version 0.8.1

13.2.1 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 1 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick

13.3 Version 0.8.0

13.3.1 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 3 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Rachel Wegener +
- Sarah Hall +

13.4 Version 0.7.0

13.4.1 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 2 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- allcontributors[bot]

13.5 Version 0.6.4

13.5.1 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 3 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Wei Ji
- allcontributors[bot]

13.6 Version 0.6.3

13.6.1 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 Jupyer 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 3 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Kelsey Bisson
- allcontributors[bot] +

13.7 Version 0.6.2

13.7.1 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 1 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick

13.8 Version 0.6.0 + 0.6.1

13.8.1 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 7 people contributed to this release. People with a "+" by their names contributed for the first time.

- Bruce Wallin
- Jessica Scheick
- Romina Piunno +
- Tyler Sutterley
- Wei Ji
- Zach Fair +
- learn2phoenix +

13.9 Version 0.5.0

13.9.1 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 3 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Wei Ji
- trey-stafford +

13.10 Version 0.4.1

13.10.1 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 1 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick

13.11 Version 0.4.0

13.11.1 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 5 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Kelsey Bisson
- Tian Li
- Tyler Sutterley +
- Wei Ji

13.12 Version 0.3.4

13.12.1 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 2 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Wei Ji

13.13 Version 0.3.3

13.13.1 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 4 people contributed to this release. People with a "+" by their names contributed for the first time.

- Bruce Wallin +
- Jessica Scheick
- Landung "Don" Setiawan +
- Wei Ji +

13.14 Version 0.3.2

13.14.1 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 7 people contributed to this release. People with a "+" by their names contributed for the first time.

- Amy Steiker +
- Anthony Arendt +
- Facundo Sapienza +
- Jessica Scheick
- Kelsey Bisson +
- Tian Li +
- alexdibella +

13.15 Version 0.3.1

13.15.1 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 noteobok
- 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.

13.16 Version 0.2-alpha

13.16.1 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.

13.17 Version 0.1-alpha

13.17.1 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.

CHAPTER FOURTEEN

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!

CHAPTER

FIFTEEN

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*.

CHAPTER

SIXTEEN

HOW TO CONTRIBUTE

On this page we briefly provide basic 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 loading into your workspace. For help navigating git and GitHub, see this guide. GitHub also provides a lot of great how-to materials for navigating and contributing.

16.2 Every time you contribute

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 or on GitHub directly). After you do this, commit your changes and add a descriptive commit message.

3. After committing the changes, push them 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. More, smaller pull requests are easier to debug and merge than fewer large ones, so 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](https://pre-commit.com/). 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.

CHAPTER

SEVENTEEN

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 describered 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 alphebetical 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 accomodate 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.

CHAPTER

EIGHTEEN

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 explicity 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()
```

CHAPTER NINETEEN

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.

CHAPTER

TWENTY

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 reseachers 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.

CHAPTER

TWENTYONE

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 merg (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.

CHAPTER

TWENTYTWO

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.

CHAPTER

TWENTYTHREE

ICESAT-2 RESOURCE GUIDE

This guide contains information regarding available resources for working with ICESat-2 datasets, both specifically (e.g. for ICESat-2 data) and more broadly (e.g. point cloud analysis of LiDAR datasets). It includes resources formally developed by/with support from NASA as well as individual and community efforts stemming from personal interest to ongoing research workflows.

Please feel free to add your project or another resource to this guide by submitting a pull request. We reserve the right to reject suggested resources that fall outside the scope of icepyx.

23.1 Other Ways to Access ICESat-2 Data

icepyx aims to provide intuitive, object-based methods for finding, obtaining, visualizing, and analyzing ICESat-2 data as part of an open, reproducible workflow that leverages existing tools wherever possible (see *Complementary GitHub Repositories*) and can be run locally, using high performance computing, or in the cloud using Pangeo. A few other options available for querying, visualizing, and downloading ICESat-2 data files are:

- NSIDC (DAAC) Data Access
  - Select "ICESat-2 Data Sets" from the left hand menu. Choose your dataset (ATL##). Then, use the spatial and temporal filters to narrow your list of granules available for download.
- OpenAltimetry
  - Collaboration between NSIDC, Scripps, and San Diego Supercomputer Center
  - A web tool to visualize and download ICESat and ICESat-2 surface heights
  - Data may be subsetted by data product, reference ground track (RGT), and beam
  - Currently available ICESat-2 datasets are: ATL06 (land ice height), ATL07 (sea ice height), ATL08 (land/vegetation height), ATL13 (water surface height)

23.2 Software Packages for Working with ICESat-2 Data

icepyx is but one of several software packages designed to improve user experience with ICESat-2 data. The links below highlight other packages active in the community.

23.2.1 Open-Source Packages

ICESat-2 can be tricky to process for the first time, especially if working with the ATL03 data. Software packages have been developed to make ICESat-2 data analysis easier for new and experienced users. Here, we highlight some commonly-used software packages developed by the science community. Most of these can be used alongside Icepyx to facilitate ICESat-2 data processing. Most of these packages are callable through Python, though others may require access to other software. Keep this in mind before attempting to use any package or plugin.

- SlideRule
  - collaboration between the ICESat-2 science team and University of Washington
  - A Python client to process ICESat-2 ATL03 data prior to download.
  - Aggregates ATL03 data into line segments of user-defined length, creating a customized form of the ATL06 product.
  - Data may also be subsetted based on spatial bounds and photon classification.
- IceFlow
  - by National Snow and Ice Data Center (NSIDC)
  - A Python library designed to simplify the co-registration of cryospheric datasets.
  - Matches georeferencing parameters across data sets, allowing a user to derive a time series across multiple datasets for a given region.
  - Currently valid datasets include ICESat, ICESat-2, and Operation IceBridge.
- PhoREAL
  - by Applied Research Laboratories, University of Texas at Austin
  - A Python-based toolbox that may also be run as a GUI (Windows only).
  - Allows for quick processing of ATL03/08 data, which may then be used to generate 2-D plots of ICESat-2 surface heights.
  - Users may also convert processed data to .las, .csv, and .kml file formats.

23.2.2 Complementary GitHub Repositories

Here we describe a selection of publicly available Python code posted on GitHub with applicability for working with ICESat-2 data. This includes repositories that are more broadly designed for working with LiDAR/point cloud datasets in general. These repositories represent independent but complimentary projects that we hope to make easily interoperable within icepyx in order to maximize capabilities and minimize duplication of efforts. Conversations about how to best accomplish this have been ongoing since the conception of icepyx, and we welcome everyone to join the conversation (please see our *contact page*).

Note: This list is a compilation of publicly available GitHub repositories and includes some annotations to reflect how they relate to icepyx. Please check each repository's licensing information before using or modifying their code. Additional resources having to do specifically with obtaining ICESat-2 data are noted in the last section of this document.

- captoolkit
  - by Fernando Paolo, Johan Nilsson, Alex Gardner
  - NASA's JPL Cryosphere Altimetry Processing Toolkit
  - Set of command line utilities to process, reduce, change format, etc. altimetry data from ICESat-2 and several other altimeters (e.g. ERS, CryoSat-2, IceBridge)

  - Includes utilities to read and extract variables of interest, compute and apply various corrections (e.g. tides, inverse barometer), detrend and correct data, do a variety of geographic computations and manipulations (e.g. raster math, masking, slope/aspect), and tile/grid/reduce data
  - We envision making captoolkit's utilities available as part of the icepyx ecosystem in order for users to quickly obtain and pre-process/correct/process ICESat-2 data.
- Icesat2-viz
  - by Aimee Barciauskas-bgse
  - Exploration for visualizing ICESat-2 data products; focused on 3-D visualization using mapbox tools
  - We hope to take advantage of Icesat2-viz's work to provide 3-D visualizations of ICESat-2 data to expand on the 2-D visualization options currently available within icepyx.
- Nsidc-subsetter
  - by Tyler Sutterley
  - Retrieve IceBridge, ICESat, and ICESat-2 data using the NSIDC subsetter API
  - Command line tool
  - Download data and convert it into a georeferenced format (e.g. geojson, kml, or shapefile)
  - We envision use of Nsidc-subsetter to improve interoperability between icepyx and the NSIDC subsetter API. Currently, icepyx has very limited subsetting capabilities that are not easy to access or find more information about.
- pointCollection
  - by Ben Smith
  - Efficiently organize and manipulate a database of points using this set of utilities
  - Access data fields using dot syntax and quickly index subsets of previously downloaded data
  - We hope to capitalize on some of the concepts of data access, indexing, and processing presented in pointCollection to improve our interfacing with ICESat-2 data within icepyx.

23.2.3 MATLAB Packages

- PhotonLabeler
  - by Lonesome Malambo
  - A MATLAB-based user interface that allows for manual interpretation of ICESat-2 photons.
  - Users may classify photons based on surface type, signal/noise likelihood, or other user-defined labels.
  - Derives simple statistics for any user-defined photon classifications.

23.3 Resources Developed For and During Hackweeks

Previous hackweeks gave participants the opportunity to develop codes to help download and/or analyze ICESat-2 data. Many of these projects are inactive, but they may provide useful workflows for users to work with.

23.3.1 First ICESat-2 Cryospheric Hackweek at the University of Washington (June 2019)

This June 2019 event resulted in the production of a series of tutorials, developed primarily by members of the ICESat-2 Science Team and early data users, aimed at educating the cryospheric community in obtaining and using ICESat-2 datasets. During the actual Hackweek, teams of researchers and data scientists developed a series of interesting projects related to their interests/research. Many of these resources were drawn from in the initial development of *icepyx*.

Tutorials

The available tutorials, most of which contain one or more Jupyter Notebooks to illustrate concepts, are listed below. Additional information for citing (including licensing) and running (e.g. through a Pangeo Binder) these tutorials can be found at the above link. They are published on Zenodo.

1. Overview of the ICESat-2 mission (slides)
2. Introduction to Open Science and Reproducible Research
3. Access and Customize ICESat-2 Data via NSIDC API
4. Intro to HDF5 and Reduction of ICESat-2 Data Files
5. Clouds and ICESat-2 Data Filtering
6. Gridding and Filtering of ICESat/ICESat-2 Elevation Change Data
7. ICESat-2 for Sea Ice
8. Geospatial Data Exploration, Analysis, and Visualization
9. Correcting ICESat-2 data and related applications
10. Numerical Modeling

Projects

Though in many cases preliminary, these project repositories c an provide useful starting points to develop effective cryospheric workflows where more formal examples and functionality have not yet been developed.

Sea Ice

- Floes are Swell
  - Calculate chord length (CLD) and lead width (LWD)
- Segtrax
  - Create trajectories of sea ice motion (creates Python trajectory class)

Glaciers and Ice Sheets

- Crackup
  - Investigating small-scale features such as crevasses and water depth

- GlacierSat2
  - Constrain surface types (e.g. wet vs. dry snow) using ICESat-2 data over the Juneau Icefield, working towards looking at seasonal elevation changes
- WaterNoice
  - Detection of hydrologic features (e.g. meltwater ponds, firn aquifer seeps, blue ice megadunes, icebergs, etc.) in ATL06 land ice product
- SnowBlower/blowing snow
  - Evaluate the blowing snow flag and look at blowing snow models
- Cross-trak (xtrak)
  - Interpolation between ICESat-2 tracks
  - Create gridded elevation data from multiple ICESat-2 tracks
- Ground2Float
  - Identify grounding zones using ICESat-2 data (using the slope-break method)
- Topohack
  - Resolve topography over complex terrain

23.3.2 Second [Virtual] ICESat-2 Cryospheric Hackweek Facilitated by the University of Washington

The 2020 ICESat-2 Cryospheric Science Hackweek was the first virtual Hackweek held by the University of Washington. While originally planned as a five-day, in-person workshop, the event was shifted to a fully virtual/remote setting in light of stay-at-home orders and travel restrictions in place to curb the spread of COVID-19.

To accomodate multiple time zones and limit the daily duration of online tutorial sessions, the event was spread out over the course of ten days. The first week had three half-days of interactive tutorials/lectures. The second week had four days that included some interactive tutorials/lectures and scheduled times where instructors were available to help participants with a facilitated exploration of datasets and hands-on software development.

This June 2020 event resulted in the production of a series of tutorials, developed by volunteer instructors and presented during the event. During the actual Hackweek, teams of researchers and data scientists developed a series of interesting projects related to their interests/research.

Tutorials

The tutorials from this event live within a dedicated GitHub repository and are published on Zenodo. You can run the tutorials by following the instructions here. The published tutorial repo also includes links to presentation slides and videos of the recorded presentations.

Tutorial Topics:

1. Introductory Session 1. ICESat-2 Mission: Satellite, Sensor, and Data 1. Git and GitHub 1. Jupyter and iPython 1. Geospatial Analysis with Python 1. Introduction to ICESat-2 Sea Ice and Land Ice Products and Data Access 1. Programmatic ICESat-2 data access 1. Introduction to HDF5 and ICESat-2 data files 1. Land ice applications 1. Sea ice applications 1. Science data generation 1. Machine learning

Projects

Though in many cases preliminary, these project repositories can provide useful starting points to develop effective cryospheric workflows where more formal examples and functionality have not yet been developed.

- icepyx
  - Contributions to icepyx included new example notebooks, packaging on Python Package Index, installation instructions, and automated code coverage reports.

Sea Ice

- leading to phytoplankton
  - Obtain and visualize coincident ICESat-2, Sentinal-2, and Argo Float data
  - Many members of this group still meet regularly (2 years out!) and are creating a template to add new coincident datasets to icepyx.
  - Group members (including newer members) contribute to icepyx as collaborative developers and code reviewers.
- overcast
  - Build tools to merge data and explore the effects of sea ice leads on clouds in the Arctic

Glaciers and Ice Sheets

- Seasonal Snow
  - Compare ICESat-2 data with high resolution DEMs over complex (mountainous) terrain
- unsupervised
  - unsupervised surface classification of ATL03 photons
- FirnAndMelt
- CloudMask
  - Fetch, classify, and label ICESat-2 data
  - Still an ongoing collaboration??
- crossovers
  - processing of non-overlapping ICESat-2 tracks
- surface_velocity
  - Infer surface ice velocity from repeat passes and across beams.
  - Continued work resulted in a poster at AGU Fall Meeting 2020
- Assimilation
  - Compare ICESat-2 elevations with multiple DEM raster data types.
  - Quantify errors and compare results regionally
  - Contributed additional authentication methods to icepyx and provided initial code for what eventually became the Visualization module.

23.3.3 Third ICESat-2 Hackweek Facilitated by the University of Washington eScience Institute

This event is currently being planned, and will take place 21-25 March 2022. Please see the event website for more information.

Tutorials

The tutorials for this event will be available to download for interactive use as well as pre-rendered on the event's website. The website will be linked here once it is live, and final information posted here after the event.

Projects

Projects will be listed here after the event.

CHAPTER

TWENTYFOUR

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

CHAPTER

TWENTYFIVE

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

CHAPTER TWENTYSIX

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.

[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:.

[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] Scheick, J, Bisson, K, Fair, Z, Piunno, 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.

[3] 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.

[4] 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.

[5] Scheick, J., Arendt, Anthony, Haley, Charley, Henderson, Scott, Koh, Jane, Setiawan, Don, Alterman, Naomi, Meyer, Joachim, Cristea, Nicoleta, Schweiger, Axel, Barciauskas, Aimee, Smith, Ben, Piunno, Romina, Shapero, 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.

[6] Scheick, Jessica, Bisson, Kelsey, Fair, Zachary, Piunno, 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.

[7] 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.

[8] Tilling, 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] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Ideström, 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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, 87
icepyx.core.auth, 89
icepyx.core.granules, 90
icepyx.core.is2ref, 93
icepyx.core.spatial, 93
icepyx.core.temporal, 97
icepyx.core.validate_inputs, 101
icepyx.core.visualization, 101

INDEX

Symbols

`__init__()` (*icepyx.GenQuery method*), 61
`__init__()` (*icepyx.Query method*), 63
`__init__()` (*icepyx.Quest method*), 106
`__init__()` (*icepyx.Read method*), 79
`__init__()` (*icepyx.Variables method*), 81

A

`about_product()` (*in module icepyx.core.is2ref*), 93
`add_argo()` (*icepyx.Quest method*), 108
`add_icesat2()` (*icepyx.Quest method*), 107
`append()` (*icepyx.Variables method*), 85
`auth` (*icepyx.core.auth.EarthdataAuthMixin property*), 89
`AuthenticationError`, 89
`avail()` (*icepyx.Variables method*), 83
`avail_granules()` (*icepyx.Query method*), 72

B

`build_params()` (*icepyx.core.APIformatting.Parameters method*), 87

C

`check_dateline()` (*in module icepyx.core.spatial*), 95
`check_req_values()` (*icepyx.core.APIformatting.Parameters method*), 87
`check_s3bucket()` (*in module icepyx.core.validate_inputs*), 101
`check_valid_date_range()` (*in module icepyx.core.temporal*), 97
`check_values()` (*icepyx.core.APIformatting.Parameters method*), 87
`CMRparams` (*icepyx.Query property*), 65
`combine_params()` (*in module icepyx.core.APIformatting*), 87
`convert_string_to_date()` (*in module icepyx.core.temporal*), 98
`cycles` (*icepyx.Query property*), 66
`cycles()` (*in module icepyx.core.validate_inputs*), 101

D

`dates` (*icepyx.Query property*), 66
`download()` (*icepyx.core.granules.Granules method*), 90
`download_all()` (*icepyx.Quest method*), 108
`download_granules()` (*icepyx.Query method*), 73

E

`earthdata_login()` (*icepyx.core.auth.EarthdataAuthMixin method*), 89
`earthdata_login()` (*icepyx.Query method*), 74
`EarthdataAuthMixin` (*class in icepyx.core.auth*), 89
`end` (*icepyx.core.temporal.Temporal property*), 97
`end_time` (*icepyx.Query property*), 66
`extent` (*icepyx.core.spatial.Spatial property*), 93
`extent_as_gdf` (*icepyx.core.spatial.Spatial property*), 94
`extent_file` (*icepyx.core.spatial.Spatial property*), 94
`extent_type` (*icepyx.core.spatial.Spatial property*), 94
`extract_product()` (*in module icepyx.core.is2ref*), 93
`extract_version()` (*in module icepyx.core.is2ref*), 93

F

`filelist` (*icepyx.Read property*), 80
`files_in_latest_n_cycles()` (*in module icepyx.core.visualization*), 103
`fmt_for_CMR()` (*icepyx.core.spatial.Spatial method*), 94
`fmt_for_EGI()` (*icepyx.core.spatial.Spatial method*), 94
`fmted_keys` (*icepyx.core.APIformatting.Parameters property*), 87

G

`generate_OA_parameters()` (*icepyx.core.visualization.Visualize method*), 101
`GenQuery` (*class in icepyx*), 59
`geodataframe()` (*in module icepyx.core.spatial*), 95
`get_avail()` (*icepyx.core.granules.Granules method*), 91
`gran_IDs()` (*in module icepyx.core.granules*), 92
`gran_paras()` (*in module icepyx.core.visualization*), 103
`Granules` (*class in icepyx.core.granules*), 90
`granules` (*icepyx.Query property*), 67

grid_bbox() (*icepyx.core.visualization.Visualize method*), 102
gt2spot() (*in module icepyx.core.is2ref*), 93

I

icepyx.core.APIformatting
    module, 87
icepyx.core.auth
    module, 89
icepyx.core.granules
    module, 90
icepyx.core.is2ref
    module, 93
icepyx.core.spatial
    module, 93
icepyx.core.temporal
    module, 97
icepyx.core.validate_inputs
    module, 101
icepyx.core.visualization
    module, 101
info() (*in module icepyx.core.granules*), 92

L

latest_version() (*icepyx.Query method*), 74
latest_version() (*in module icepyx.core.is2ref*), 93
load() (*icepyx.Read method*), 80

M

make_request() (*icepyx.core.visualization.Visualize method*), 102
module
    icepyx.core.APIformatting, 87
    icepyx.core.auth, 89
    icepyx.core.granules, 90
    icepyx.core.is2ref, 93
    icepyx.core.spatial, 93
    icepyx.core.temporal, 97
    icepyx.core.validate_inputs, 101
    icepyx.core.visualization, 101

O

order_granules() (*icepyx.Query method*), 75
order_vars (*icepyx.Query property*), 67

P

parallel_request_OA() (*icepyx.core.visualization.Visualize method*), 102
Parameters (*class in icepyx.core.APIformatting*), 87
parse_var_list() (*icepyx.Variables static method*), 83
place_order() (*icepyx.core.granules.Granules method*), 91
poss_keys (*icepyx.core.APIformatting.Parameters property*), 87
prod_version() (*in module icepyx.core.validate_inputs*), 101
product (*icepyx.Query property*), 68
product (*icepyx.Read property*), 80
product_all_info() (*icepyx.Query method*), 76
product_summary_info() (*icepyx.Query method*), 76
product_version (*icepyx.Query property*), 68

Q

Query (*class in icepyx*), 62
query_icesat2_filelist() (*icepyx.core.visualization.Visualize method*), 102
Quest (*class in icepyx*), 105

R

Read (*class in icepyx*), 78
remove() (*icepyx.Variables method*), 86
reqparams (*icepyx.Query property*), 68
request_OA_data() (*icepyx.core.visualization.Visualize method*), 102

S

s3login_credentials (*icepyx.core.auth.EarthdataAuthMixin property*), 90
save_all() (*icepyx.Quest method*), 109
search_all() (*icepyx.Quest method*), 108
session (*icepyx.core.auth.EarthdataAuthMixin property*), 90
show_custom_options() (*icepyx.Query method*), 76
Spatial (*class in icepyx.core.spatial*), 93
spatial (*icepyx.Query property*), 69
spatial_extent (*icepyx.Query property*), 69
start (*icepyx.core.temporal.Temporal property*), 97
start_time (*icepyx.Query property*), 70
subsetparams() (*icepyx.Query method*), 70

T

Temporal (*class in icepyx.core.temporal*), 97
temporal (*icepyx.Query property*), 71
to_string() (*in module icepyx.core.APIformatting*), 88
tracks (*icepyx.Query property*), 71
tracks() (*in module icepyx.core.validate_inputs*), 101

U

user_check() (*in module icepyx.core.visualization*), 103

V

validate_bounding_box() (*in module icepyx.core.spatial*), 96

validate_date_range_date() (*in module icepyx.core.temporal*), 98
validate_date_range_datestr() (*in module icepyx.core.temporal*), 99
validate_date_range_datetime() (*in module icepyx.core.temporal*), 99
validate_date_range_dict() (*in module icepyx.core.temporal*), 100
validate_polygon_file() (*in module icepyx.core.spatial*), 96
validate_polygon_list() (*in module icepyx.core.spatial*), 96
validate_polygon_pairs() (*in module icepyx.core.spatial*), 96
validate_times() (*in module icepyx.core.temporal*), 100
Variables (*class in icepyx*), 81
vars (*icepyx.Read property*), 80
Visualize (*class in icepyx.core.visualization*), 101
visualize_elevation() (*icepyx.Query method*), 77
visualize_spatial_extent() (*icepyx.Query method*), 77
viz_elevation() (*icepyx.core.visualization.Visualize method*), 103