icepyx

unknown

Oct 01, 2021

GETTING STARTED

Python tools for obtaining and working with ICESat-2 data

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.

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 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. This project combines and generalizes these scripts into a unified framework, adding examples, documentation, and testing where necessary and making them accessible for everyone. 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 way to install icepyx is by using the conda package manager. The command below takes care of setting up a virtual environment and installs icepyx along with all the necessary dependencies:

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

To activate the virtual environment, you can do:

```
conda activate icepyx-env
```

2.2 Using conda

If you already have a virtual conda environment set up and activated, you can install the latest stable release of icepyx from conda-forge like so:

```
conda install icepyx
```

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

```
conda 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
```

Currently, conda 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

EXAMPLE NOTEBOOKS

Listed below are example jupyter-notebooks:

{ “cells”: [{ “cell_type”: “markdown”, “id”: “1e29ff05”, “metadata”: {}, “source”: [“## Visualizing ICESat-2 Data\n”, “### Elevation Visualization Example Notebook\n”, “\n”, “This notebook 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. \n”, “\n”, “#### Credits\n”, “* Notebook by: Tian Li, Jessica Scheick and \n”, “Wei Ji\n”, “* Source material: READ_ATL06_DEM Notebook by Tian Li and Friedrich Knuth”] }, { “cell_type”: “markdown”, “id”: “6333399a”, “metadata”: {}, “source”: [“### Import packages”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “157dfbe8”, “metadata”: {}, “outputs”: [], “source”: [“import icepyx as ipx”] }, { “cell_type”: “markdown”, “id”: “57f2cfd8”, “metadata”: {}, “source”: [“### Create an ICESat-2 query object\n”, “Set the desired parameters for your data visualization.\n”, “\n”, “For details on minimum required inputs, please refer to ICESat-2_DAAC_DataAccess_Example. 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).”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “926496ad”, “metadata”: {}, “outputs”: [], “source”: [“#bounding box\n”, “#Larsen C Ice Shelf\n”, “short_name = ‘ATL06’\n”, “date_range = [‘2020-7-1’, ‘2020-8-1’]\n”, “spatial_extent = [-67, -70, -59, -65] \n”, “cycles = [‘03’]\n”, “tracks = [‘0948’, ‘0872’, ‘1184’, ‘0186’, ‘1123’, ‘1009’, ‘0445’, ‘0369’]”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “71142c20”, “metadata”: {}, “outputs”: [], “source”: [“# # polygon vertices\n”, “# short_name = ‘ATL06’\n”, “# date_range = [‘2019-02-20’,’2019-02-28’]\n”, “# spatial_extent = [(-55, 68), (-55, 71), (-48, 71), (-48, 68), (-55, 68)]”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “119dbcb0”, “metadata”: {}, “outputs”: [], “source”: [“# # polygon geospatial file\n”, “# short_name = ‘ATL06’\n”, “# date_range = [‘2019-10-01’,’2019-10-05’]\n”, “# spatial_extent = ‘./supporting_files/data-access_PineIsland/glims_polygons.shp’”] }, { “cell_type”: “markdown”, “id”: “8b537573”, “metadata”: {}, “source”: [“Create an ICESat-2 data object based on query parameters “] }, { “cell_type”: “code”, “execution_count”: null, “id”: “bbece76a-1776-489f-9ef5-78a704057bb4”, “metadata”: {}, “outputs”: [], “source”: [“region = ipx.Query(short_name, spatial_extent, date_range)”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “8fbae0e7”, “metadata”: {}, “outputs”: [], “source”: [“print(region.product)\n”, “print(region.dates)\n”, “print(region.start_time)\n”, “print(region.end_time)\n”, “print(region.product_version)\n”, “print(list(set(region.avail_granules(cycles=True)[0]))) #region.cycles\n”, “print(list(set(region.avail_granules(tracks=True)[0]))) #region.tracks”] }, { “cell_type”: “markdown”, “id”: “1b178836”, “metadata”: {}, “source”: [“### Visualize spatial extent \n”, “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.”] }, { “cell_type”: “code”, “execution_count”: null, “id”: “ec9777b0-b3e5-4cb5-85a3-35ad1ff982ac”, “metadata”: { “tags”: [] }, “outputs”: [], “source”: [“region.visualize_spatial_extent()”] }, { “cell_type”: “markdown”, “id”: “71ca513d”, “metadata”: {}, “source”: [“### Visualize ICESat-2 elevation using OpenAltimetry API\n”, “\n”, “#### **Note: this function currently only supports products `ATL06, ATL07, ATL08, ATL10, ATL12, ATL13`**\n”, “\n”, “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.\n", "\n", "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."] }, { "cell_type": "code", "execution_count": null, "id": "93a712af-ab1f-4d79-834d-c2735a5677a8", "metadata": { "tags": [] }, "outputs": [], "source": ["cyclemap, rgtmap = region.visualize_elevation()\n", "cyclemap"] }, { "cell_type": "markdown", "id": "9ee72a5c", "metadata": {}, "source": ["#### Plot elevation for individual RGT\n", "\n", "The visualization tool also provides the option to view elevation data by latitude for each ground track."] }, { "cell_type": "code", "execution_count": null, "id": "802e5c6e", "metadata": {}, "outputs": [], "source": ["rgtmap"] }, { "cell_type": "markdown", "id": "b7082edd", "metadata": {}, "source": ["### Move on to data downloading from NSIDC if these are the products of interest\n", "\n", "For more details on the data ordering and downloading process, see ICESat-2_DAAC_DataAccess_Example"] }, { "cell_type": "code", "execution_count": null, "id": "05f91e82", "metadata": {}, "outputs": [], "source": ["earthdata_uid = 'yourearthdataid'\n", "email = 'youremailaddress'\n", "region.earthdata_login(earthdata_uid, email)\n", "region.order_granules()\n", "\n", "#view a short list of order IDs\n", "region.granules.orderIDs\n", "\n", "path = 'your data directory'\n", "region.download_granules(path)"] }, { "cell_type": "markdown", "id": "textile-casting", "metadata": {}, "source": ["### Alternative Access Options to Visualize ICESat-2 elevation using OpenAltimetry API\n", "\n", "You can also view elevation data by importing the visualization module directly and initializing it with your query object or a list of parameters:\n", " `\n", " from icepyx.core.visualization import Visualize\n", " \n", " - passing your query object directly to the visualization module\n", " `\n", " region2 = ipx.Query(short_name, spatial_extent, date_range)\n", " vis = Visualize(region2)\n", " \n", " - creating a visualization object directly without first creating a query object\n", " `\n", " vis = Visualize(product=short_name, spatial_extent=spatial_extent, date_range=date_range)\n", " "] }], "metadata": { "kernelspec": { "display_name": "Python 3", "language": "python", "name": "python3" }, "language_info": { "codemirror_mode": { "name": "ipython", "version": 3 }, "file_extension": ".py", "mimetype": "text/x-python", "name": "python", "nbconvert_exporter": "python", "pygments_lexer": "ipython3", "version": "3.7.6" } }, "nbformat": 4, "nbformat_minor": 5 }

CHAPTER

FOUR

CITATION INFORMATION

4.1 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: Scheick, J. *et al.*, (2019). icepyx: Python tools for obtaining and working with ICESat-2 data. https://github.com/icesat2py/icepyx.

A bibtex version for users working in Latex:

```
  @Misc{icepyx,
author =    {Scheick, Jessica and others},
organization = {icesat2py},
title =     {{icepyx: Python} tools for obtaining and working with {ICESat-2} data},
year =      {2019--},
url = "https://github.com/icesat2py/icepyx"
  }
```

4.2 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

4.3 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

FIVE

ICEPYX DOCUMENTATION (API REFERENCE)

icepyx package diagram illustrating the library's public-facing, high-level package structure and their relationships.

icepyx class diagram illustrating the library's public-facing classes, their attributes and methods, and their relationships. A more detailed, developer UML class diagram showing hidden parameters is 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.

5.1 Query Class

5.1.1 Constructor

Query([product, spatial_extent, date_range, ...])	ICESat-2 Data object to query, obtain, and perform basic operations on available ICESat-2 data products using temporal and spatial input parameters.

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*, *files=None*)

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.

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
- **spatial_extent** (*list or string*) – Spatial extent of interest, provided as a bounding box, list of polygon coordinates, or geospatial polygon file. 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. DevGoal: adapt code so the polygon is automatically

closed if need be 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 of 'YYYY-MM-DD' strings*) – Date range of interest, provided as start and end dates, inclusive. The required date format is 'YYYY-MM-DD' strings, where YYYY = 4 digit year, MM = 2 digit month, DD = 2 digit day. Currently, a list of specific dates (rather than a range) is not accepted. DevGoal: accept date-time objects, dicts (with 'start_date' and 'end_date' keys, and DOY inputs). DevGoal: allow searches with a list of dates, rather than a range.
- **start_time** (*HH:mm:ss, default 00:00:00*) – Start time in UTC/Zulu (24 hour clock). If None, use default. DevGoal: check for time in date-range date-time object, if that's used for input.
- **end_time** (*HH:mm:ss, default 23:59:59*) – End time in UTC/Zulu (24 hour clock). If None, use default. DevGoal: check for time in date-range date-time object, if that's used for input.
- **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: "004"
- **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"
- **files** (*string, default None*) – A placeholder for future development. Not used for any purposes yet.

Returns

Return type query object

Examples

Initializing Query with a bounding box.

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

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 = icepyx.query.Query('ATL06', reg_a_poly, reg_a_dates)
>>> reg_a
<icepyx.core.query.Query at [location]>
```

Initializing Query with a geospatial polygon file.

```
>>> aoi = '/User/name/location/aoi.shp'
>>> reg_a_dates = ['2019-02-22','2019-02-28']
>>> reg_a = icepyx.query.Query('ATL06', aoi, reg_a_dates)
>>> reg_a
<icepyx.core.query.Query at [location]>
```

__init__(*product=None*, *spatial_extent=None*, *date_range=None*, *start_time=None*, *end_time=None*, *version=None*, *cycles=None*, *tracks=None*, *files=None*)

Methods

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

avail_granules([ids, cycles, tracks])	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)	Log in to NSIDC EarthData to access data.
latest_version()	Determine the most recent version available for the given product.
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	value pairs that will be submitted.
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.
file_vars	Return the file variables object.
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.

continues on next page

Table 3 – continued from previous page

product	Return the short name product ID string associated with the query object.
product_version	Return the product version of the data object.
reqparams	value pairs that will be submitted.
spatial_extent	Return an array showing the spatial extent of the query object.
start_time	Return the start time specified for the start date.
tracks	Return the unique ICESat-2 Reference Ground Tracks

5.1.2 Attributes

Query.CMRparams	value pairs that will be submitted.
Query.cycles	Return the unique ICESat-2 orbital cycle.
Query.dataset	Legacy property included to provide depracation warning.
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.file_vars	Return the file variables object.
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.reqparams	value pairs that will be submitted.
Query.spatial_extent	Return an array showing the spatial extent of the query object.
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.tracks	Return the unique ICESat-2 Reference Ground Tracks

icepyx.Query.CMRparams

property Query.**CMRparams**

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

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.CMRparams
{'short_name': 'ATL06',
'version': '002',
'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
'bounding_box': '-55,68,-48,71'}
```

Type Display the CMR key

icepyx.Query.cycles

property Query.**cycles**
Return the unique ICESat-2 orbital cycle.

Examples

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

icepyx.Query.dataset

property Query.**dataset**
Legacy property included to provide depracation warning.

See also:

product

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

icepyx.Query.end_time

property Query.**end_time**
Return the end time specified for the end date.

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.end_time
'23:59:59'
```

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'], end_time='10:20:20')
>>> reg_a.end_time
'10:20:20'
```

icepyx.Query.file_vars

property Query.**file_vars**

Return the file variables object. This instance is generated when files are used to create the data object (not yet implemented).

See also:

variables.Variables

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> reg_a.file_vars
<icepyx.core.variables.Variables at [location]>
```

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 = icepyx.query.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 = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> reg_a.order_vars
<icepyx.core.variables.Variables at [location]>
```

icepyx.Query.reqparams

property Query.**reqparams**

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

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.reqparams
{'page_size': 10, 'page_num': 1}
```

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> reg_a.order_granules()
>>> reg_a.reqparams
{'page_size': 10, 'page_num': 1, 'request_mode': 'async', 'include_meta': 'Y',
↪'client_string': 'icepyx'}
```

Type Display the required key

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 [[array of longitudes],[array of corresponding latitudes]].

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.spatial_extent
['bounding box', [-55, 68, -48, 71]]
```

```
>>> reg_a = icepyx.query.Query('ATL06',[(-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]]
```

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 = icepyx.query.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': '-55,68,-48,71'}
```

icepyx.Query.start_time

property Query.**start_time**
Return the start time specified for the start date.

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.start_time
'00:00:00'
```

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'], start_time='12:30:30')
>>> reg_a.start_time
'12:30:30'
```

icepyx.Query.tracks

property Query.**tracks**
Return the unique ICESat-2 Reference Ground Tracks

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.tracks
['0841', '0849', '0902', '0910']
```

5.1.3 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)	Log in to NSIDC EarthData to access data.
Query.latest_version()	Determine the most recent version available for the given product.
Query.order_granules([verbose, subset, email])	Place an order for the available granules for the query object.
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*)

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.

Examples

```
>>> reg_a = icepyx.query.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)': 48.975419759750004,
'Total size of all granules (MB)': 195.90167903900002}
```

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.avail_granules(ids=True)
>>> reg_a.avail_granules(cycles=True)
['02']
>>> reg_a.avail_granules(tracks=True)
['0841', '0849', '0902', '0910']
```

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*, *email*)

Log in to NSIDC EarthData to access data. Generates the needed session and token for most data searches and data ordering/download.

Parameters

- **uid** (*string*) – Earthdata login user ID
- **email** (*string*) – Email address. NSIDC will automatically send you emails about the status of your order.

See also:

`Earthdata.Earthdata`

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
```

icepyx.Query.latest_version

`Query.`**`latest_version`**()

Determine the most recent version available for the given product.

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.latest_version()
'003'
```

icepyx.Query.order_granules

Query.**order_granules**(*verbose=False*, *subset=True*, *email=True*, ***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 True*) – Have NSIDC auto-send order status email updates to indicate order status as pending/completed.
- ****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 = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> 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.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 = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28
↪'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> 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

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

icepyx.Query.visualize_elevation

Query.**visualize_elevation**()

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

Returns **map_cycle, map_rgt + lineplot_rgt** – Holoviews data visualization elements

Return type Holoviews objects

5.2 Query Components

5.2.1 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: dataset (data product), version, start, end, extent_type, spatial_extent Keyword argument inputs for 'required' may include: 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

Return type single dictionary of all input dictionaries combined

Examples

```
>>> CMRparams = {'short_name': 'ATL06', 'version': '002', 'temporal': '2019-02-
↪20T00:00:00Z,2019-02-28T23:59:59Z', 'bounding_box': '-55,68,-48,71'}
>>> reqparams = {'page_size': 10, 'page_num': 1}
>>> icepyx.core.APIformatting.combine_params(CMRparams, reqparams)
{'short_name': 'ATL06',
'version': '002',
'temporal': '2019-02-20T00:00:00Z,2019-02-28T23:59:59Z',
'bounding_box': '-55,68,-48,71',
'page_size': 10,
'page_num': 1}
```

icepyx.core.APIformatting.**to_string**(*params*)
Combine a parameter dictionary into a single url string

Parameters **params** (*dictionary*) –

Returns

Return type url string of input dictionary (not encoded)

Examples

```
>>> CMRparams = {'short_name': 'ATL06', 'version': '002', 'temporal': '2019-02-
↪20T00:00:00Z,2019-02-28T23:59:59Z', 'bounding_box': '-55,68,-48,71'}
>>> reqparams = {'page_size': 10, 'page_num': 1}
>>> params = icepyx.core.APIformatting.combine_params(CMRparams, reqparams)
>>> icepyx.core.APIformatting.to_string(params)
'short_name=ATL06&version=002&temporal=2019-02-20T00:00:00Z,2019-02-28T23:59:59Z&
↪bounding_box=-55,68,-48,71&page_size=10&page_num=1'
```

5.2.2 Earthdata

class icepyx.core.Earthdata.**Earthdata**(*uid*, *email*, *capability_url*, *pswd=None*)

Bases: object

Initiate an Earthdata session for interacting with the NSIDC DAAC.

Parameters

- **uid** (*string*) – Earthdata Login user name (user ID).
- **email** (*string*) – Complete email address, provided as a string.
- **password** (*string (encrypted)*) – Password for Earthdata registration associated with the uid.
- **capability_url** (*string*) – URL required to access Earthdata

Returns

Return type Earthdata session object after a successful login

login()

This function tries to log the user in to Earthdata with the information provided. It prompts the user for their Earthdata password, but will only store that information within the active session. If the login fails, it will ask the user to re-enter their username and password up to five times to try and log in.

Alternatively, you can create a .netrc file in your $HOME directory with the following line:

machine urs.earthdata.nasa.gov login <uid> password <password>

Where <uid> is your NASA Earthdata user ID and <password> is your password Then change the permissions of that file to 600 This will allow you to have read and write access to the file No other user can access the file

$ chmod 600 ~/.netrc

The function checks for this file to retrieve credentials, prior to prompting for manual input.

Examples

```
>>> icepyx.core.Earthdata.Earthdata.login('sam.smith','sam.smith@domain.com')
Earthdata Login password:  ........
```

5.2.3 geospatial

icepyx.core.geospatial.**geodataframe**(*extent_type*, *spatial_extent*, *file=False*)

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*) – A string of the spatial extent. If file is False, the string 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, the string is 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

See also:

icepyx.Query

Examples

```
>>> reg_a = icepyx.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-02-28'])
>>> gdf = geospatial.geodataframe(reg_a.extent_type, reg_a._spat_extent)
>>> gdf.geometry
0    POLYGON ((-55.00000 68.00000, -55.00000 71.000...
Name: geometry, dtype: geometry
```

5.2.4 granules

class icepyx.core.granules.**Granules**

Bases: object

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

Return type Granules object

download(*verbose*, *path*, *session=None*, *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.
- **session** (*requests.session object*) – A session object authenticating the user to download data using their Earthdata login information. The session object will automatically be passed from the query object if you have successfully logged in there.
- **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 log in to Earthdata and can then skip immediately to the download_granules method with restart=True.

Notes

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

See also:

query.Query.download_granules

get_avail(*CMRparams*, *reqparams*)

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.

Notes

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

See also:

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

place_order(*CMRparams*, *reqparams*, *subsetparams*, *verbose*, *subset=True*, *session=None*, *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)
- **session** (*requests.session object*) – A session object authenticating the user to order data using their Earthdata login information. The session object will automatically be passed from the query object if you have successfully logged in there.
- **geom_filepath** (*string, default None*) – String of the full filename and path when the spatial input is a file.

Notes

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

See also:

`query.Query.order_granules`

icepyx.core.granules.**gran_IDs**(*grans*, *ids=True*, *cycles=False*, *tracks=False*, *dates=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 list of granual dictionaries.

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.

5.2.5 is2ref

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

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

See also:

`query.Query.product_all_info`

5.2.6 validate_inputs

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.**spatial**(*spatial_extent*)

Validate the input spatial extent and return the needed parameters to the query object.

icepyx.core.validate_inputs.**temporal**(*date_range*, *start_time*, *end_time*)

Validate the input temporal parameters and return the needed parameters to the query object.

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

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

5.2.7 variables

class icepyx.core.variables.**Variables**(*vartype*, *avail=None*, *wanted=None*, *session=None*, *product=None*, *version=None*, *path=None*)

Bases: `object`

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

Parameters

- **vartype** (*string*) – 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.
- **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
- **session** (*requests.session object*) – A session object authenticating the user to download data using their Earthdata login information. The session object will automatically be passed from the query object if you have successfully logged in there.
- **product** (*string, default None*) – Properly formatted string specifying a valid ICESat-2 product
- **version** (*string, default None*) – Properly formatted string specifying a valid version of the ICESat-2 product
- **path** (*string, default None*) – For vartype file, a path to a directory of or single input data file (not yet implemented)

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 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 ICESat-2_DAAC_DataAccess2_Subsetting example notebook

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-
↪02-28'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
```

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'])
```

avail(*options=False*, *internal=False*)

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

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-
↪02-28'], version='1')
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> 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']
```

static parse_var_list(*varlist*)

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

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-
↪02-28'], version='1')
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
>>> 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'],
.
.
.
'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',
```

(continues on next page)

(continued from previous page)

```
'lat_mean', 'lon_mean', 'pulse_count', 'segment_id_list', 'x_atc_mean', 'record_
↪number',
'reference_pt_lat', 'reference_pt_lon', 'signal_selection_status_all',
'signal_selection_status_backup', 'signal_selection_status_confident',
↪'crossing_time',
'cycle_number', 'lan', 'orbit_number', 'rgt', 'sc_orient', 'sc_orient_time',
'qa_granule_fail_reason', 'qa_granule_pass_fail', 'signal_selection_source_
↪fraction_0',
'signal_selection_source_fraction_1', 'signal_selection_source_fraction_2',
'signal_selection_source_fraction_3'])
>>> import numpy
>>> numpy.unique(paths)
array(['ancillary_data', 'bias_correction', 'dem', 'fit_statistics',
'geophysical', 'ground_track', 'gt1l', 'gt1r', 'gt2l', 'gt2r',
'gt3l', 'gt3r', 'land_ice', 'land_ice_segments', 'none',
'orbit_info', 'quality_assessment', 'residual_histogram',
'segment_quality', 'signal_selection_status'], dtype='<U23')
```

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.

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 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 ICESat-2_DAAC_DataAccess2_Subsetting example notebook

Examples

```
>>> reg_a = icepyx.query.Query('ATL06',[-55, 68, -48, 71],['2019-02-20','2019-
↪02-28'])
>>> reg_a.earthdata_login(user_id,user_email)
Earthdata Login password:  ........
```

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'])
```

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

Return type list

grid_bbox(*binsize=5*) → 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** – A list of bounding boxes with a maximum size of 5*5 degree

Return type list

make_request(*base_url*, *payload*)
Make HTTP request

Parameters **base_url** (*string*) – OpenAltimetry URL

See also:

request_OA_data

parallel_request_OA() → dask.array.routines.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** – A dask array containing the ICESat-2 data.

Return type dask.Array

query_icesat2_filelist() → tuple
Query list of ICESat-2 files for each bounding box

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

Return type tuple

request_OA_data(*paras*) → dask.array.routines.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** – A dask array containing the ICESat-2 elevation data.

Return type da.array

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 data visualization elements

Return type Holoviews objects

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

Return type list

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

Return type list

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

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

Parameters **message** (*string*) – Message to indicate users the options

CHAPTER

SIX

ICEPYX CHANGELOG

This is the list of changes made to icepyx in between each release. Full details can be found in the commit logs.

6.1 Latest Release (Version 0.4.1)

6.1.1 What's new in 0.4.1 (01 October 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 frome xample notebook (#225)

Documentation

- improve query docstrings (#212)

Other

- add research notice to readme (#206)

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

6.2 Version 0.4.0

6.2.1 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

6.3 Version 0.3.4

6.3.1 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

6.4 Version 0.3.3

6.4.1 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

6.5 Version 0.3.2

6.5.1 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 +

6.6 Version 0.3.1

6.6.1 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 12 people contributed to this release. People with a "+" by their names contributed for the first time.

- Amy Steiker +
- Anna Valentine +
- Bidhyananda Yadav +
- Bruce Wallin +
- David Shean +
- Friedrich Knuth +
- Jessica Scheick
- Raphael Hagen
- Tom Johnson +
- Tyler Sutterley +
- Wei Ji +
- Zheng Liu

6.7 Version 0.2-alpha

6.7.1 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 3 people contributed to this release. People with a "+" by their names contributed for the first time.

- Jessica Scheick
- Scott Henderson +
- Zheng Liu

6.8 Version 0.1-alpha

6.8.1 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 6 people contributed to this release. People with a "+" by their names contributed for the first time.

- Anthony Arendt +
- Fernando Perez +
- Jessica Scheick
- Raphael Hagen +

- Shashank Bhushan +
- Zheng Liu +

CHAPTER

SEVEN

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.

7.1 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 some code 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.

7.2 Requesting a Feature

Find the *Issues* tab at the top of GitHub repository and click *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.

7.3 Reporting a Bug

Find the *Issues* tab at the top of GitHub repository and click *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.

7.4 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.

7.4.1 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
- The 2019 ICESat-2 Hackweek Tutorial repo and [in progress] 2020 ICESat-2 Hackweek Tutorial repo are great resources for learning basic Python and development skills, learning about ICESat-2 data, setting up a computational environment, and finding and analyzing ICESat-2 datasets.
- A great set of interactive tutorials for learning and practicing using git
- Let us know about the helpful tools you've found by posting on our discourse forum as indicated above (*Questions and Help*)!

7.5 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*.

7.6 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.

7.6.1 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.

7.6.2 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.

7.6.3 Basic Steps to Contribute

We encourage users to follow the git pull request workflow. In a nutshell, the series of steps required to add new code is: (first time only)

- Clone the repository
- Fork the repo to your personal GitHub account
- Add your fork as a remote
- Add yourself to *CONTRIBUTORS.rst* (see *Attribution for Contributions*)

(each time you are going to make changes)

- Update the development branch
- Create a new branch
- Make your changes and commit them to the branch
- Push your changes to your fork
- Make a pull request (on GitHub; pull requests will automatically be made against the development branch)
- Push any additional, relevant changes to the same pull request (this will happen automatically if you push the changes to the same branch from which you made the pull request)

7.6.4 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.

7.7 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.

7.8 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

EIGHT

ATTRIBUTION GUIDELINES

We are extremely grateful to everyone who has contributed to the success of the icepyx community, whether through direct contributions to or feedback about icepyx or as developers or maintainers of complimentary resources that are included within the icepyx ecosystem. 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. To do so, we define broadly *contributions* as:

> Efforts towards achieving icepyx's goals, including writing code, tests, or documentation, development of example workflows, development, significant contributions, or maintenance of a tailored package that broadens the functionality of icepyx, feedback and suggestions, community building, etc.

We use the terms "contributors", "developers", and "authors" interchangeably. We will recognize contributions in the following ways.

8.1 Contributors List

Anyone who has contributed a pull request to icepyx is welcome to add themselves to the `CONTRIBUTORS.rst` file located in the top level directory; the file is packaged and distributed with icepyx. This process is optional, but is the easiest way for us to say "thank you" to everyone who has helped this project.

8.2 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.

8.3 Version Release on Zenodo

When new releases of icepyx are archived on Zenodo, anyone who has contributed to icepyx will be invited to be an author. The list of potential authors will be generated using the *Contributors List*. Thus, if you have contributed to icepyx and would like to be included as an author, you *must* add your full name, affiliation ("Unaffiliated" is acceptable), and ORCID (optional) to `CONTRIBUTORS.rst`.

Author order will be determined based on co-author discussion during preparation of the version release, led by one or more of the members of the lead development team (Anthony Arendt, Lindsey Heagy, Fernando Perez, Jessica Scheick). Metrics for guiding the determination of author order will include the number of commits made to the repository (`git shortlog -sne`) and active engagement on GitHub (e.g. through issues and pull requests) and Discourse. Author order may also be modified on a case-by-case basis by consensus of the lead development team and top contributors.

If you do not wish to be included in the author list for Zenodo version releases, please add a note (e.g. "do not include in Zenodo") to your entry.

8.4 Scientific Publications (Papers)

Authorship on scientific papers currently constitutes an important metric for assessing scientific merit and contribution and is often directly linked to career advancement. We aim to write academic papers for our software and its uses. Ideally, we will publish on an early version of the software and subsequently on major releases, use cases, or to advance the causes of open-source software and open science. To be eligible for authorship on scientific papers, contributors must:

1. Contribute to the development (including code, documentation, and examples) of icepyx. Substantial non-code contributions constitute eligibility for authorship.
2. Add themself to the *Contributors List*.
3. 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 lead author, 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 will 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.

> Disclaimer: These policies are not permanent or fixed and may change to accomodate community growth, best practices, and feedback.

Copyright notice: This document was inspired by the authorship guidelines provided by Fatiando a Terra and encourages potential co-authors to consider the resources provided by the NASA High Mountain Asia Team (HiMAT).

CHAPTER

NINE

ICEPYX DEVELOPMENT PLAN

This page provides a high-level overview of where icepyx is headed. The list does not claim to be all inclusive, nor is it exclusive. Rather, we aim to provide a set of broad objectives to be met over the course of months to years, given that they require substantial developer time. These goals will evolve with time as development proceeds and new insights are gained (which could mean we ultimately opt NOT to implement some of them). If you would like to propose changes to this development plan, please see *Modifying the Development Plan*

Items with a smaller scope are tracked as issues on our GitHub issue tracker. We invite you to join the active discussions happening there.

9.1 Enhancing User Interactivity and Visualization

The process of querying, obtaining, and working with ICESat-2's large datasets through a command line or similar interface poses challenges to many researchers who are uninterested in also becoming advanced software developers. However, downloading, storing, and backing up both raw and derived data are computationally and resource intensive, and frequent switching between tools for different steps in the research workflow are time intensive and difficult to reproduce. By simplifying the process of querying, subsetting, and visualizing data - both through relevant function methods and interactive tools like Jupyter widgets - icepyx aims to reduce or remove the need to download large, non-subsetted datasets, enable easy visualization throughout the data inquiry to analyzed data presentation steps, and provide a simple, community-based framework for reproducibility.

9.2 Improving Accessibility to Advanced Computing

Even as new resources and tools are developed to make computing easier, disciplinary researchers still face challenges finding time to maintain their computing environments while also meeting their research, teaching, and service objectives. Through integration of icepyx into the Pangeo ecosystem, only a handful of developers are able to manage the computational resources needed for working with ICESat-2 data, freeing researcher time and energy for exploring data. Further, the integration of Pangeo into existing advanced computing infrastructure (such as NASA's ADAPT) will enable researchers to take advantage of cloud computing without a significant need for re-tooling.

9.3 Open Science Example Use Cases

We are currently partnering with multiple researchers conducting investigations using ICESat-2 datasets. Their research, from data collection and analysis to publication, will be used to drive the development of icepyx functionality, ultimately providing software contributions and example workflows. Current collaborations focus on:

- impacts of blowing snow and low clouds on ICESat-2 measurements
- snow height in non-glaciated regions
- parameter assimilation into sea ice models

If you are or plan to work on a project using ICESat-2 datasets, we encourage you to use icepyx as a framework for finding and processing your data, from designing your analysis to writing code to analyze your data (if the analysis tools you need aren't already a part of icepyx, that is!) to generating publication figures. Please *contact us* if you have any questions or would like some guidance to get involved!

9.4 Data Analysis and Interaction

Multiple forms of traditional and advanced computational analysis are used by researchers to probe and analyze large datasets to answer challenging questions about the systems the data describes. These analysis techniques include filtering, application of corrections, trend detection, feature detection, statistics, and machine learning, among others. icepyx aims to easily integrate existing libraries that specialize in these types of analysis by providing easy ways to manipulate ICESat-2 data into the appropriate form required by each library and showcasing the use of these complex analyses to answer glaciological questions through easily-modifiable example workflows based on actual use cases.

9.5 Validation and Integration with Other Products

The complexity of multiple data access systems, many with different metadata formats and API access types, presents a challenge for finding and integrating diverse datasets to construct long time series. Many open-source resources provide tools for manipulating certain types of datasets, but few collate these resources into one computing environment (see *Improving Accessibility to Advanced Computing*) and provide wrappers and examples to easily conduct frequently performed analysis tasks. This portion of the development plan, driven by researcher use cases, will combine existing resources with new ones to improve researcher ability to easily compare diverse datasets across varying sensor types and spatial and temporal scales.

9.6 Modifying the Development Plan

Everyone is invited to review and propose new items for the Development Plan. icepyx is continually evolving and its direction is driven by your feedback and contributions.

Items listed in the Development Plan should be brief summaries of more detailed proposals. Each item listed should include:

1. Summary of proposed changes/additions
2. Motivation for the changes
3. Statement describing how the changes fit within the icepyx scope
4. More detailed plan for changes (e.g. implementation plan, examples (even if not implemented), potential issues)

Please submit your proposal as a GitHub issue, which will provide the developers and community members an opportunity to provide feedback on your suggestion. Once there is agreement on the proposal, submit a pull request to update the Development Plan, including a link to the discussion issue.

CHAPTER

TEN

CONTRIBUTOR COVENANT CODE OF CONDUCT

10.1 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.

10.2 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

10.3 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.

10.4 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.

10.5 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.

10.6 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:

10.6.1 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.

10.6.2 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.

10.6.3 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.

10.6.4 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.

10.7 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

ELEVEN

ICESAT-2 OPEN-SOURCE RESOURCES 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.

11.1 Resources Used in the Initial Development of icepyx

11.1.1 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.

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.

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

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.

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

11.2 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.

11.3 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
  - Enables data browsing on a map and selection of tracks and interactive data exploration for the higher level ICESat-2 datasets (i.e. ATL06+)

11.4 Ongoing Efforts

In addition to the ongoing development of icepyx itself, the ICESat-2 Cryosphere community continues to grow through a number of workshops and events.

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

COVID-19 forced the in-person event to be cancelled, but we're excited to extend the Hackweek model into a virtual space, ultimately making it more accessible by removing the need to travel. This year's event is scheduled to take place from 15-18 June 2020, with multiple instructional sessions taking place during the preceding week (8-12 June) to limit the daily duration and accomodate multiple time zones. Though only selected participants are able to tune in to the live tutorial sessions, the materials being taught are freely available in the ICESat-2 Hackweek GitHub Organization respositories. Watch here for updates on projects conducted during the hackweek, and feel free to check out the event's website.

CHAPTER

TWELVE

CONTACT US

The best way to contact us depends on what information you're looking for.

- Need help installing, running, or using *icepyx*? Add a new topic to ask for help on Discourse (after reviewing the documentation and existing topics, of course, to see if they answer your question!) or attend one of our regular virtual meetings (details below).
- Found a bug or have a feature request? Post an issue on GitHub!
- Have an idea you'd like to discuss? Start a conversation on Discourse or attend one of our regular virtual meetings (details 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!

12.1 Regular Meeting Schedule

Our team (developers, users, scientists, educators) meets regularly via Zoom to provide support, troubleshoot issues, and plan development. We meet on:

- the second Tuesday of the month at 4pm GMT (12pm Eastern, 9am Pacific)
- the fourth Monday of the month at 8pm GMT (4pm Eastern, 1pm Pacific)

Additional information about logging in to the meetings can be found on this Discourse post.

Absolutely NO previous software development experience is necessary to attend any meeting. Think of them more like coffee hour mixed with office hours than a conference call. We look forward to seeing you there!

CHAPTER

THIRTEEN

TRACKING ICEPYX USAGE

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)!

13.1 Projects and Organizations

Projects and organizations that use icepyx.

- NSIDC
- University of Washington e-Science institute
- ICESat-2 Cryospheric Hackweeks
- Colorado School of Mines Glaciology Laboratory

13.2 Publications and Presentations

ICESat-2 peer-reviewed research that utilizes icepyx and presentations that feature or explain icepyx

13.3 Downloads

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 user downloads and page views as shown below.

13.3.1 GitHub Traffic

Clones and views of the icepyx library directly on GitHub.

13.3.2 PyPI Downloads

Non-mirrored downloads of icepyx from the Python Package Index (e.g. using *pip install icepyx*).

Quick Install

BIBLIOGRAPHY

[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] 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.

[3] 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.

PYTHON MODULE INDEX

i

INDEX

O

P

Q

R

S

T

U

V