Echotools

Release 0.2.0

Mar 25, 2023

CONTENTS:

CHAPTER ONE

INTRODUCTION

This document contains information about the data processing software provided by the echotools library (*echotools-libs*) python modules. The library uses the *dbdreader* python module to decode slocum glider files. The slocum binaries, for decoding glider files, do not provide sufficient precision when extracting information from the DBD files. The *dbdreader* provides full precision float values to extract the embedded echograms stored in the least significant digits of the echometrics data.

The echotools library will be able to decode three types of echometrics data. The echometrics modes are:

- *metrics*: Echometrics; no embedded echogram.
- *combo*: Echometrics and a low resolution embedded echogram.
- *egram*: The echometrics data is replaced by a medium resolution echogram.

CHAPTER TWO

INSTALLATION

2.1 GUTILS

For use within the GUTILS python module for processing glider data into the IOOS Glider DAC, three items are required:

- *dbdreader* python module
- *processEchogram.py* python script
- *teledyne.py* python class utilized by *processEchogram.py*

At some point, *echotools* will be released to the public allowing for easier integration of echotools processing code as a standalone module.

The *dbdreader* package is available from github.com/smerckel/dbdreader.

The *GUTILS* package is available from github.com/SECOORA/GUTILS.

The *echotools* package is kept in a **private** repository that resides at github.com/phishdoc/Glider-echo-tools. Access can be requested from John Horne (*jhorne@uw.edu*).

2.2 ECHOTOOLS

The echotools library has a two step installation process. The first step is using *pybuilder* to build the source tree. After the source tree is built, it can be installed using *pip*. NOTE: Be aware that the version number may be different than what is indicated below.

```
$ git clone git@github.com:phishdoc/Glider-echo-tools.git
$ cd Glider-echo-tools
$ cd echotools-libs
$ pyb
$ cd target/dest/echotools-0.2.0
$ python -m pip install .
```

This can be combined with *GUTILS* in a conda environment. Review and use the *gutils/rebuildGUTILS.sh* script found in the *echotools-libs* directory. This script builds a complete environment based on python 3.9.

CHAPTER THREE

DOCUMENTATION

3.1 Processing of Glider Echograms

Echograms may be processed using the information transmitted over iridium or post deployment. The echogram is embedded into the echometrics information stored in the Teledyne Webb TBD files. A companion SBD file is required for additional time and depth information. A cache file is typically needed to decode the binary structure of the tbd and sbd file using the `dbdreader` python module.

3.1.1 Output

The python program `processEchograms.py` can decode the glider files and produce three types of output:

- CSV with or without a header)
- netcdf4 as a combined segment or separated by type (sbd, tbd)
- image (png, jpg, pdf) files in one or more styles: binned, scatter or pcolormesh

Most information is typically saved to a file. Some formats can be requested to be sent to standard output (stdout). If using the `teledyne.py` `Glider()` class, the echogram data frame may be manipulated or used directly for other purposes.

Metadata

When saving the CSV to a file or stdout, three columns of information are provided.

- Column 1: Time GMT/UTC (seconds since 01-01-1970)
- Column 2: Depth (meters)
- Column 3: Sv (dB re 1 m2/m3)

NOTE: Unique keys are formed from the first two columns giving this dataset a 3D like structure.

3.1.2 processEchograms.py

For more information on the operation details of this program and library, please refer to *Programs and Modules*.

Syntax

```
usage: processEchograms.py [-h] [--inpDir INPDIR] [--inpFile INPFILE] [-t T]
                           [--tbdFile TBDFILE] [--sbdFile SBDFILE] [--cacheDir CACHEDIR]
                           [--dbd2asc DBD2ASC] [--csvOut CSVOUT] [--csvHeader] [--ncDir␣
↪NCDIR]
                           [--ncSeparate] [--imageOut IMAGEOUT] [--outDir OUTDIR] [--
↪debug]
                           [--echogramBins ECHOGRAMBINS] [--echogramRange ECHOGRAMRANGE]
                           [--plotType PLOTTYPE] [--binnedDepthLabels] [--title TITLE]
                           [--deploymentDir DEPLOYMENTDIR] [--template TEMPLATE]
                           [--templateDir TEMPLATEDIR] [--dacOverlay DACOVERLAY]
                           [--saveBits SAVEBITS]

Echogram processing: This reads Teledyne Web glider files. The glider files may include
echograms from an echosounder. This program can operate on a single file or a whole␣
↪deployment
of files.

optional arguments:
  -h, --help            show this help message and exit
  --inpDir INPDIR       full or relative path to glider input files
  --inpFile INPFILE     glider input file(s)
  -t T                  glider file type: sfmc, rt or delayed
  --tbdFile TBDFILE     full or relative path with filename to glider tbd binary input␣
↪file (DEPRECATED)
  --sbdFile SBDFILE     full or relative path with filename to glider sbd binary input␣
↪file (DEPRECATED)
  --cacheDir CACHEDIR   Directory with glider cache files; default current directory
  --dbd2asc DBD2ASC     full or relative path with filename to glider dbd2asc binary␣
↪(DEPRECATED)
  --csvOut CSVOUT       full or relative path with filename to write CSV output or
↪'stdout';
                        default None
  --csvHeader           (flag) include header with CSV output; default False
  --ncDir NCDIR         full or relative path with filename to write netCDF output;␣
↪default
                        None
  --ncSeparate          save tbd, sbd and sv data separately; default False
  --imageOut IMAGEOUT   filename to write image or stdout; default None
  --outDir OUTDIR       output directory for csv and plot files
  --debug               (flag) show extra debugging for this python script; default False
  --echogramBins ECHOGRAMBINS
                        Echogram bins; default 20
  --echogramRange ECHOGRAMRANGE
                        Echogram range; default -60.0 (meters) instrument facing up;␣
↪positive
                        values instrument facing down
```

(continues on next page)

(continued from previous page)

```
  --plotType PLOTTYPE   Use one or more of binned, scatter, pcolormesh or all; default␣
↪binned
  --binnedDepthLabels   (flag) Use the original depth bin labels instead of the adjusted␣
↪depth
                        labels for the time/depth binned plot; default False
  --title TITLE         optional figure title; default None
  --deploymentDir DEPLOYMENTDIR
                        full or relative path directory with json files
  --template TEMPLATE   full or relative path to metadata template
  --templateDir TEMPLATEDIR
                        full or relative path to metadata template directory
  --dacOverlay DACOVERLAY
                        full or relative path to an overlay configuration file
  --saveBits SAVEBITS   full or relative path to save the bit encoding of the echogram
```

3.1.3 Examples (command line)

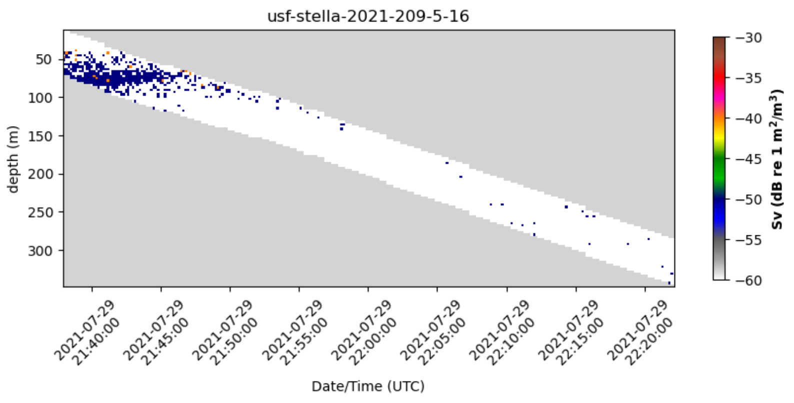
usf-stella-2021-209-5-16

Produce echogram plots of University of South Florida glider with instrument facing **downwards** with a range of 60.0 meters:

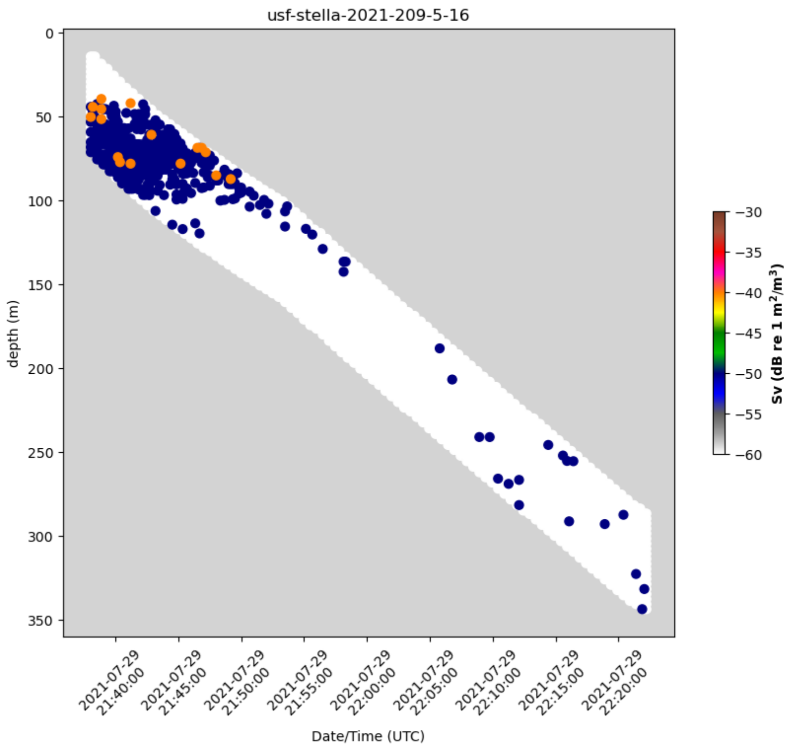
```
$ processEchograms.py --cacheDir examples/cache --inpDir examples/data \
    --echgramRange 60.0 \
    --inpFile usf-stella-2021-209-5-16 --imageOut default.png \
    --outDir examples/output --title "usf-stella-2021-209-5-16" -t sfmc \
    --plotType all \
    --vbsBins [-10,-20,-25,-30,-35,-45,-55] --dBLimits [-30,-60]
```

The should result in the creation of three figures.

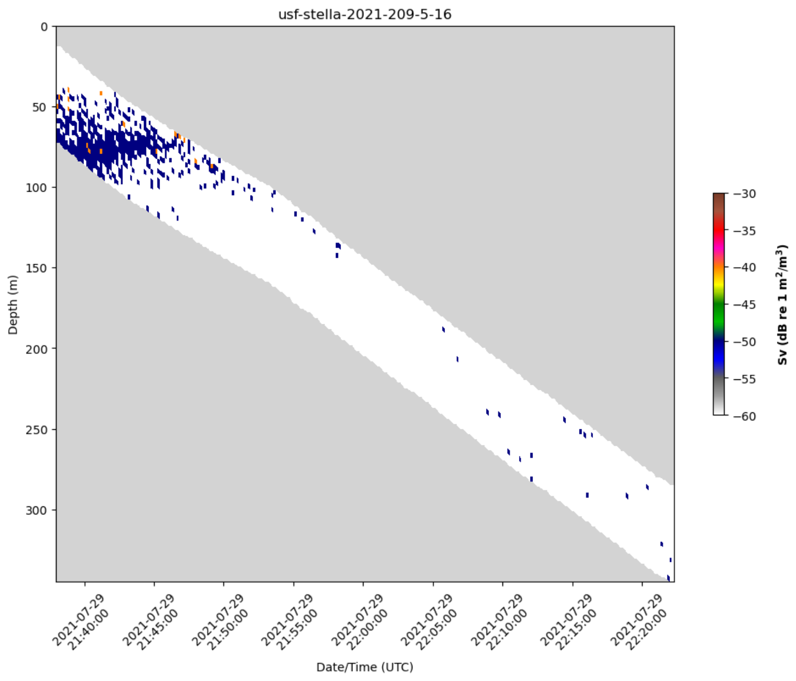
Using the binned plot type:



Using the scatter plot type:



Using the `pcolormesh` plot type:

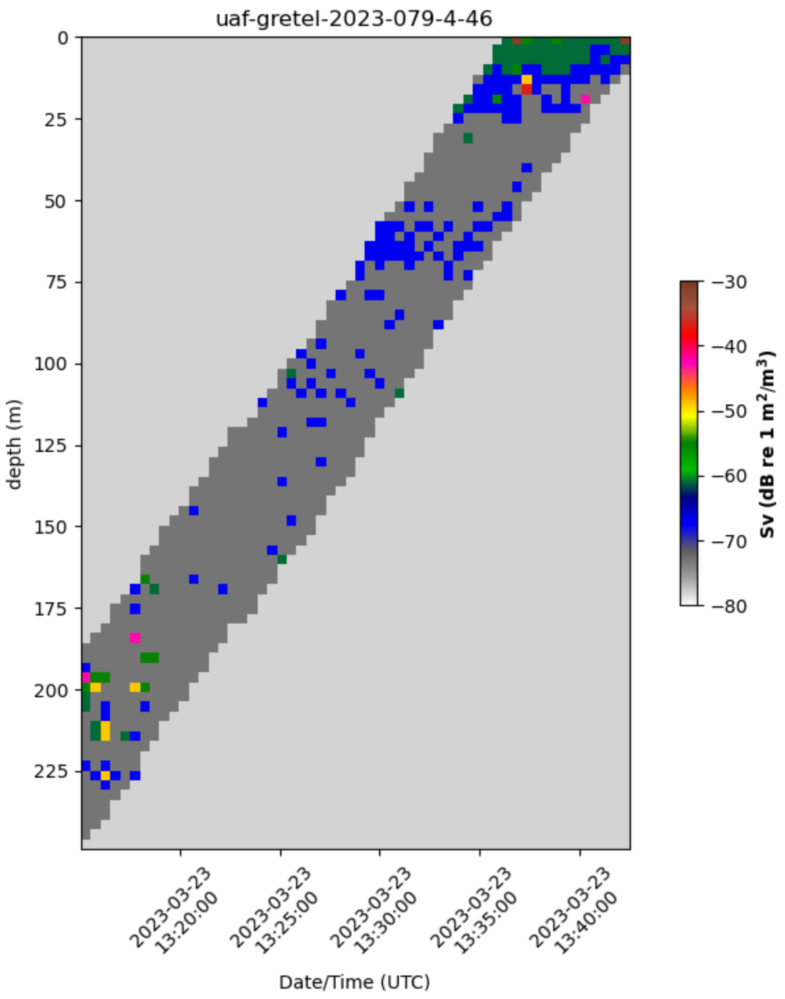


uaf-gretel-2022-019-8-0

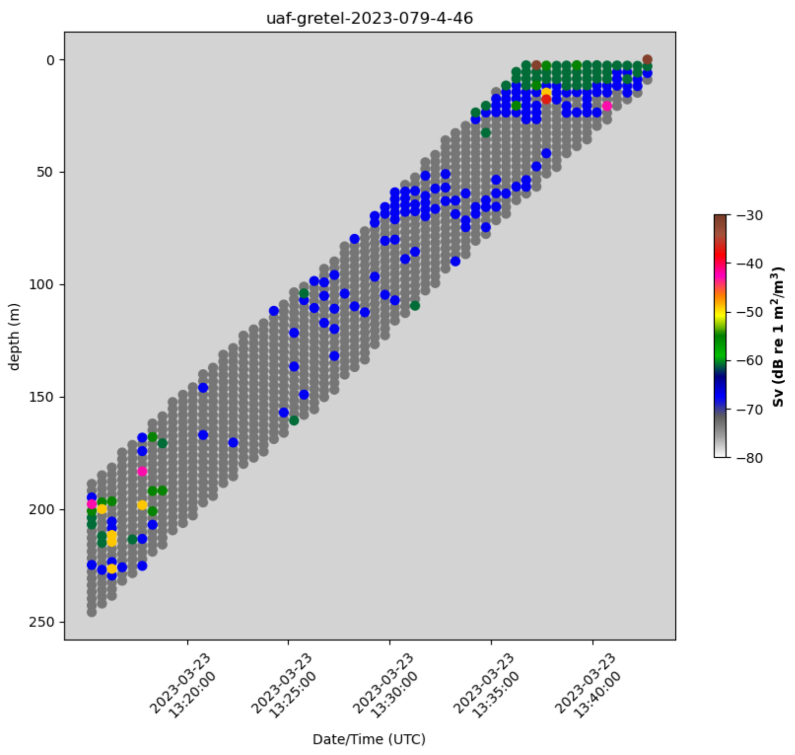
Produce echogram plots of University of Alaska Fairbanks glider with instrument facing **upwards** with a range of 60.0 meters (`echogramRange=-60.0`) and output to CSV:

```
$ processEchograms.py --cacheDir examples/cache --inpDir examples/data \
    --echgramRange -60.0 \
    --inpFile uaf-gretel-2023-079-4-46 --imageOut default.png \
    --outDir examples/output --title "uaf-gretel-2023-079-4-46" -t sfmc \
    --plotType all --vbsBins [-34,-40,-46,-52,-58,-64,-70] --dBLimits [-30,-80] \
    --csvOut uaf-gretel-2023-079-4-46.csv
```

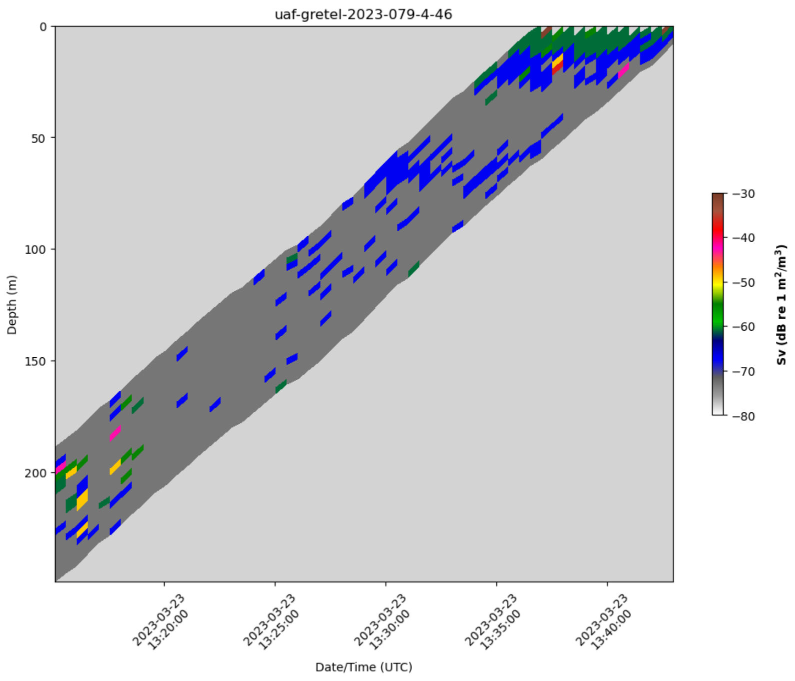
Using the `binned` plot type:



Using the `scatter` plot type:



Using the `pcolormesh` plot type:



The resulting output should look very similar to the standard output shown below.

Same example, but sending the CSV to stdout:

```
$ processEchograms.py --cacheDir examples/cache --inpDir examples/data \
    --echgramRange -60.0 \
```

(continues on next page)

(continued from previous page)

```
    --inpFile uaf-gretel-2023-079-4-46 --outDir examples/output \
    -t sfmc --vbsBins [-34,-40,-46,-52,-58,-64,-70] --dBLimits [-30,-80] \
    --csvOut stdout
```

Echogram data printed to standard output:

```
1679577318.457703, 245.765747, -73.000000
1679577318.457703, 242.765747, -73.000000
1679577318.457703, 239.765747, -73.000000
1679577318.457703, 236.765747, -73.000000
1679577318.457703, 233.765747, -73.000000
....
```

Information sent to standard output can be captured or piped into another process for upstream processing or storage by another program.

3.1.4 Examples (configuration files)

In these examples, the GUTILS configuration files are used with processEchograms.py to perform echogram processing.

Configuration file: deployment.json

The majority of this file provides metadata that is written into the netcdf file (if requested). The portion of the configurtion file that provides arguments to the echogram processor is in the echograms **subsection** of extra_kwargs:

```
    "extra_kwargs": {
        "echograms": {
            "enable_nc": true,
            "enable_ascii": true,
            "enable_image": false,
            "plot_type": "all",
            "color_bar": "ek80",
            "svb_thresholds": [-34,-40,-46,-52,-58,-64,-70],
            "svb_limits": [-30.0,-80.0],
            "echogram_bins": 20,
            "echogram_range": -60.0,
            "echogram_range_units": "meters"
        }
```

3.1.5 Using the Glider() python class

The Glider() python class can read glider files directly using the dbdreader python module. After setting some arguments in the class object, a call can be made to attempt to read the embedded echogram. If the function fails for any reason, an empty spreadsheet is returned. If an echogram is present, the data frame returned will be a spreadsheet of values as described in the *Metadata* section of the manual.

Manual method

Here is a snippet of code that converts the glider files, extracts the echograms, produces graphics, exports a CSV file and presents the data frame in numpy, pandas and xarray forms.

```
#!/usr/bin/env python

# This is an example program that converts
# an echogram to a scatter plot by setting
# Glider() class arguments manually.

import os, glob
import pandas as pd
import xarray as xr
from echotools.parsers.slocum import teledyne

if __name__ == '__main__':
    # Setup the glider
    glider = teledyne.Glider()

    # Debug control: set True to increase verbosity
    glider.debugFlag = False

    segment = 'uaf-gretel-2023-079-4-46'
    args = {
        'vbsBins' : [-34,-40,-46,-52,-58,-64,-70],
        'dBLimits' : [-30.0,-80.0],
        'echogramBins' : 20,
        'echogramRange' : -60.0,
        'inpDir': 'data',
        'inpFile': f'{segment}',
        'cacheDir': 'cache',
        'outDir': 'output',
        'imageOut' : f'default.png',
        'title' : f'{segment}',
        'plotType' : 'all',
        'csvHeader' : True,
        'csvOut' : f'{segment}.csv'
    }

    # Pass arguments to the glider class object
    glider.args = args

    # Find and read glider files
    file_glob = \
        os.path.join(args['inpDir'],
        "%s%s" % (segment, ".?[bB][dD]"))

    glider_files = glob.glob(file_glob)
    for glider_file in glider_files:
        glider.readDbd(
            inputFile = glider_file,
            cacheDir = args['cacheDir']
```

(continues on next page)

(continued from previous page)

```
        )

    # Attempt to extract echogram from glider data
    glider.readEchogram()

    # Create and export CSV file
    glider.createEchogramSpreadsheet()
    glider.handleSpreadsheet()

    # Create graphics
    glider.handleImage()

    # The echogram is stored in a numpy data object
    echogram_numpy = glider.data['spreadsheet']

    print("The echogram at this point is available in a numpy object")
    print("with the columns in the first row: [time, depth, Sv]")
    print(echogram_numpy[0,:])
    print()

    # Convert the echogram numpy object to pandas and set the time
    # column as an index.
    echogram_pandas = pd.DataFrame(data = echogram_numpy,
        columns = ['time', 'depth', 'Sv']).set_index('time')

    print("The first row of the pandas data frame:")
    print(echogram_pandas.iloc[0])
    print()

    # Convert the pandas data frame to xarray, add units
    # and convert time timestamp to datetime object
    echogram_xarray = echogram_pandas.to_xarray()
    echogram_xarray['time'].attrs['units'] = 'seconds since 1970-01-01'
    echogram_xarray['time'] = pd.to_datetime(echogram_xarray['time'], unit='s')
    echogram_xarray['depth'].attrs['units'] = 'meters'
    echogram_xarray['Sv'].attrs['units'] = 'dB re 1 m2/m3'

    print("The first row of the xarray data frame:")
    print(echogram_xarray.isel(time=0))
```

The output of this program should produce:

```
The echogram at this point is available in a numpy object
with the columns in the first row: [time, depth, Sv]
[ 1.67957732e+09  2.45765747e+02 -7.30000000e+01]

The first row of the pandas data frame:
depth    245.765747
Sv       -73.000000
Name: 1679577318.4577026, dtype: float64

The first row of the xarray data frame:
```

(continues on next page)

(continued from previous page)

```
<xarray.Dataset>
Dimensions:  ()
Coordinates:
    time     datetime64[ns] 2023-03-23T13:15:18.457702656
Data variables:
    depth    float64 245.8
    Sv       float64 -73.0
```

Configuration file method

This demonstrates code using the deployment.json files used by GUTILS. This program only provides access to the echogram numpy object.

```
#!/usr/bin/env python

# This is an example program that converts
# an echogram to a scatter plot by setting
# Glider() class arguments using the
# GUITLS style configuration files.

import os, glob
from echotools.parsers.slocum import teledyne

if __name__ == '__main__':
    # Setup the glider object
    glider = teledyne.Glider()

    # Debug control: set True to increase verbosity
    glider.debugFlag = False

    segment = 'uaf-gretel-2023-079-4-46'
    args = {
        'deploymentDir': 'config',
        'templateDir': 'config',
        'template': 'slocum_dac.json',
        'inpDir': 'data',
        'cacheDir': 'cache'
    }

    # Pass arguments to the glider class object
    glider.args = args

    glider.loadMetadata()
    glider.args = glider.updateArgumentsFromMetadata()

    # Pass arguments to the glider class object
    glider.args = args

    # Find and read glider files
    file_glob = \
        os.path.join(args['inpDir'],
```

(continues on next page)

(continued from previous page)

```
        "%s%s" % (segment, ".?[bB][dD]"))

    glider_files = glob.glob(file_glob)
    for glider_file in glider_files:
        glider.readDbd(
            inputFile = glider_file,
            cacheDir = args['cacheDir']
        )

    # Attempt to extract echogram from glider data
    glider.readEchogram()
    glider.createEchogramSpreadsheet()

    # The echogram is stored in a numpy data object
    print("The echogram at this point is available in a numpy object")
    print("with the columns in the first row: [time, depth, Sv]")
    print(glider.data['spreadsheet'][0,:])
```

3.2 Teledyne Webb file types

3.2.1 Flight data extensions

- dbd - full resolution data
- sbd - shortened flight data that is sent back via iridium
- mlg - message log files from the flight computer

3.2.2 Science data extensions

- ebd - full resolution data
- tbd - shortened science data that is sent back via iridium
- nlg - message log files from the science computer. We are running the echodroid in verbose more so all of the back and forth between the Science persistor, the Odroid and the mini are logged in this file.

Cache files

Cache files (cac) files help describe the binary structure of the tbd and sbd data files.

3.3 Auxillary programs

3.3.1 Originally developed programs

For the original toolset provided by Alex Silverman, see the `alex/original` directory. Within that directory run the `echoGen.sh` script with appropriate symbolic links to data and cache files.

NOTES:

- The Teledyne Webb binary `dba_sensor_filter` is required for this toolset.
- A symbolic link from `examples` needs to point to the directory with `data` and `cache` directory.
- A symbolic link from `bin` needs to point to the directory with the Teledyne Webb binaries.

Here is an example call to these scripts from the main shell program:

```
. echoGen.sh uaf-gretel-2022-019-6-0.tbd
```

The shell script performs the following steps:

- Decode (tmpSSV.txt)
- Reorder (tmpEcho.txt)
- Extract time/depth from tbd (tmpBar.txt)
- Create images from decoded temporary files
  - Raw image
  - Depth (bin) corrected image

This toolset of scripts formed the basis for creating the `processEchograms.py` script and `teledyne.py` python class/library.

3.4 History

A bit of history for the `echotools` library.

3.4.1 Version 1

Initial decoding and plotting code was provided by Alex Silverman.

3.4.2 Version 2

Initial echotools Glider() class developed around Alex's code and the python module `dbdreader`.

3.4.3 Version 3

The Glider() class has morphed into a hybrid cluster of code from four sources: Alex, Rob, a prior version of echotools written in python version 2 and `GUTILS`.

3.4.4 Version 4 (future)

Maybe rebuild the code stack from the bottom up using lessons learned and leveraging `pyarrow` and `parquet` for intermediate storage and I/O. Datasets should be easily converted to existing dataframes (numpy, pandas and xarray). There is a new lightweight REST service `xpublish` that can tuned to be very fast as compared to the current giants `ERDDAP` and `THREDDS` which require `java` to run at the expense of needing a machine with lots of RAM.

3.5 Teledyne linux binaries

The zip file is stored within the repository. Extract only the binaries needed to perform post processing after downloading the repository.

Source of these binaries require registration with Teledyne Marine Webb Reserches forum:

https://datahost.webbresearch.com/

NOTE: These may no longer be needed with the incorporation of the `dbdreader` python module into echotools.

3.6 Example data

3.6.1 USF: Stella

July Pt Sur cruise

- Mode: echo
- Instrument is pointed down (+60.0 meters)

tbd	cache	description
usf-stella-2021-209-5-16.tbd	d0f88888.cac	Day 5 Segment 16

3.6.2 UAF: Gretel

Deployed from Seward, AK: March 21, 2023

- Mode: combo
- Instrument is pointed up (-60.0 meters)
- VBS thresholds: `[-34,-40,-46,-52,-58,-64,-70]`
- DB limits: `[-30,-80]`

files	cache	description
uaf-gretel-2022-019-8-0.tbd	16708f40.cac	Day 4 Segment 46
uaf-gretel-2022-019-8-0.sbd	79884a79.cac	Day 4 Segment 46

3.7 Teledyne files

3.7.1 File extensions

Flight data:

- dbd: full resolution data
- sbd: shortened flight data that is sent back via iridium
- mlg: message log files from flight computer

Science data:

- ebd: full resolution data
- tbd: shortened science data that is sent back via iriduim
- nlg: message log files from the science computer

3.8 Programs and Modules

3.8.1 Programs

processEchograms.py

This script contains functions that decode an echogram from the echometrics data stream (if present). The resultant information can be:

- Rendered in plot a plot: binned, scatter or pcolormesh
- Saved to a CSV file
- Saved to a netCDF file
- Available as an xarray object
- Available as a pandas object

The output can be directed to a file or standard output (stdout). The netCDF may only be saved to a file. The netCDF variables may be saved as separate files for ease of aggregation.

Data written as CSV output is in three columns (comma separated values): Timestamp, Depth, Density. A metadata desciption is given below.

Metadata for the netCDF file is pulled from deployment configuration files: deployment.json and instruments.json; A general template file is also required for generic metadata.

Image output is controlled by the filename extension provided. A typical format is PNG. Use ".png" in the filename to produce a PNG formatted image.

Metadata

Here is the metadata description for each of the coloums of this dataset (ASCII, spreadsheet):

Column	Description
Timestamp	seconds since 01-01-1970 epoch; timezone GMT/UTC
Depth	depth (meters)
Density	Sv (dB)

Command line arguments

```
-h, --help            show this help message and exit
--inpDir INPDIR       full or relative path to directory with
                      glider data
--inpFile INPFILE     glider input file; one segment
--t T                 data type: sfmc, rt, delayed
--tbdFile TBDFILE     full or relative path with filename
                      to Teledyne glider tbd binary input file
--sbdFile SBDFILE     full or relative path with filename
                      to Teledyne glider sbd binary input file
--cacheDir CACHEDIR   Directory with glider cache files;
                      default current directory
--dbd2asc DBD2ASC     full or relative path with filename
                      to glider dbd2asc binary
--csvOut CSVOUT       full or relative path with filename
                      to write CSV output or 'stdout'; default None
--ncDir NCDIR         full or relative path of a directory for
                      writing netCDF file(s); default None
--ncFormat NCFORMAT   alternate time format for saving netCDF files
--csvHeader           (flag) include header with CSV output; default False
--imageOut IMAGEOUT   filename to write image or stdout; default None
--outDir OUTDIR       output directory for csv and plot files; default None
--debug               (flag) show extra debugging for this
                      python script; default False
--echogramBins ECHOSOUNDERRANGE
                      Number of bins in use by the echogram;
                      default 20
--echogramRange ECHOSOUNDERRANGE
                      Echogram range; default -60.0 (meters)
                      instrument facing up; positive values
                      instrument facing down
--title               optional figure title; default None
--plotType            One or more of binned, scatter, pcolormesh or
                      all; default binned
--binnedDepthLabels   (flag) Use the original depth bin
                      labels instead of the adjusted depth
                      labels for the time/depth binned plot;
                      default False
--deploymentDir       full or relative path to deployment and
                      instrument json files; default None
```

(continues on next page)

(continued from previous page)

```
--templateDir         full or relative path to glider
                      template directory; default None
--saveBits            output the echogram bits for
                      validation.
```

processEchograms.**printArgs**(*args*)

This is a convienence function for printing dictionary elements.

Parameters
args (dict(), required) –

Return type
Prints information to standard output.

processEchograms.**printAttributes**(*myObj*)

This is a convienence function for printing object attribute information for the supplied object. This function writes to standard output and does not print any objects with an underscore prefix. This was mainly used for debugging during development.

Parameters
myObj (any, required) –

Return type
Prints information to standard output.

processEchograms.**showHelp**(*parser*)

This prints the program description and arguments to standard output and exits.

processEchograms.**sussFileType**(*args*)

Suss out file types we are attempting to work with based on the output directory path.

3.8.2 Modules

teledyne

class echotools.parsers.slocum.teledyne.**Glider**(*tbdFile=None*, *sbdFile=None*, *cacheDir=None*, *dbd2asc=None*, *debugFlag=False*)

Bases: object

A container class for handling Teledyne Webb glider data.

Table 7-1

Glider : dbd mbd sbd mlg Science: ebd nbd tbd nlg

Initialize a glider object.

Parameters

- **tbdFile** (str) – Full or relative path with filename to glider tbd file.
- **sbdFile** (str) – Full or relative path with filename to glider sbd file.
- **cacheDir** (str) – Full or relative path to directory with glider (sensor) cache files.
- **dbd2asc** (str) – Full or realtive path with filename to Teledyne Webb binary dbd2asc. NOTE: System must be able to execute the binary.

- **debugFlag** (bool) – Flag for extra debugging information printed to standard output. Default: False

addAttributes(*ncDS*, *varName*)

This function adds variable attributes to the netCDF variable from the specified template file.

applyDeploymentGlobalMetadata(*ncDS*)

This function applies the deployment.json global metadata to the xarray Dataset object.

applyGlobalMetadata(*ncDS*)

Generically apply global metadata data to xarray Dataset object

applyTemplateGlobalMetadata(*ncDS*)

This function applies the {template}.json global metadata to the xarray Dataset object.

calculateMissionPlan()

Calculate mission plan parameters based on current arguments.

collectDbdVariables(*ncDS*)

This function collects dbd file variables.

collectEbdVariables(*ncDS*)

This function collects ebd file variables.

collectEchogram(*ncDS*)

This function collects the echogram and adds it to the xarray Dataset object.

collectSbdVariables(*ncDS*)

This function collects sbd file variables.

collectTbdVariables(*ncDS*)

This function collects tbd file variables.

createEchogramSpreadsheet()

This function reads GLIDER.data['echogram'] and places it in a spreadsheet format in GLIDER.data['spreadsheet']. The function is expecting at least two fields to have been read from the provided ebd/tbd file.

Notes for self.args

- args['debugFlag']: Boolean flag. If True, additional output is printed to standard output.
- args['plotType']: String. Available plot types are: binned, scatter and pcolormesh.

createFileInventory(*fileList*, *cache_dir*)

Create a slocum file inventory from a file list. This list must include file extensions.

dateFormat(*dttmObj=datetime.datetime(2023, 3, 25, 20, 9, 4, 387100)*, *fmt='%Y-%m-%dT%H:%M:%SZ'*)

Format a given datetime.datetime object into a string of the default format: "%Y-%m-%dT%H:%M:%SZ". If a datetime.datetime object is not provided, use the current time.

extractColumns(*source*, *columns=[]*, *ignoreNaNColumns=[]*, *asDict=False*)

This function extracts requested columns from the GLIDER.data[source] object. This function will also remove rows for specified columns that contain NaNs.

NOTE: This function replaces the need for dba_sensor_filter and subsequently ignoring output of NaNs.

Parameters

- **columns** (list) – Named columns to subset from GLIDER.data[source] object.

- **ignoreNaNColumns** (`list`) – One the data is collected by column, rows are eliminated in named columns where the values are NaN.
- **asDict** (`bool`) – This is a flag to change the return value as a python dict() object where the column names are the dictionary keys.

Returns
This returns a subset of data stored in GLIDER.data[source]. This will either be another numpy array or a python dictionary.

Return type
`numpy array or dict()`

`filterFiles`(*start_time=None*, *end_time=None*)

Filter a loaded list of slocum files by the start and end times. This requires use of a created or loaded file inventory.

Parameters

- **start_time** (`str`) – A string containing the start time.
- **end_time** (`str`) – A string containing the end time.

Returns
Returns a python list of matching files between the start and end times.

Return type
`list()`

`findTimeInterval`(*timeAxis*, *plotType*, *ax*, *nticks=10*)

For a given time axis and number of ticks, return an array of axisLocations and axisLabels.

timeAxis is an array of timestamps in seconds.

`findTimeVariable`(*parameterList*)

Determine the time variable for a given parameter list.

Parameters
parameterList (`list()`) – A python list of parameters.

Returns
m_present_time or *sci_m_present_time* if detected. Returns None if either is not found.

Return type
`str`

`getDepthPixel`(*reqDepth*, *minDepth*, *maxDepth*, *depthBinSize*)

For the image (plotting) routine, depths are placed in discrete pixels by the depth bin size. The first pixel has the depth range of minimum depth to minimum depth plus depth bin size.

Notes

For a depth bin size of 3.0 meters and a minimum depth of 2.0 meters. Bin zero (0) should be 2.0 to 5.0 meters, bin (1) will be from 5.0 meters to 8.0 meters and so forth.

Parameters

- **reqDepth** (`float`) – Requested depth (meters)
- **minDepth** (`int`) – Minimum depth bin
- **maxDepth** (`int`) – Maximum depth bin (not used)

- **depthBinSize** (`float`) – Actual depth size of each depth bin (meters)

Returns
Returns the depth bin adjusted to the minimum depth bin

Return type
`int`

getFullFilenamesFromFileInventory()

Using the loaded file inventory, return full file paths.

getScale(*sensorName*)

Check the template for a scaling factor prior to assigning the final data.

groupFiles(*fileList*)

Take a list of filenames and group them by name without thier extension.

Parameters
array (`list()`) – A python list of filenames with extensions. The case of the file extension is ignored.

Returns
A python dictionary of file groups.

Return type
`dict()`

handleImage()

This function handles writing out a graphical image. By default, the image rendering uses descrete pixels that have depth and time bins. If –useScatterPlot is set, a scatter plot is produced instead of a time/depth binned (raster image/imshow) plot. The raster plot coordinates are the depth and time bins which requires redefining x and y labels on the fly.

Notes for self.args

- args['imageOut']: May be a full or relative path with filename or *stdout*.
- args['debugFlag']: Boolean flag. If True, additional output is printed to standard output.
- args['useScatterPlot']: Boolean flag. If True, a scatter plot is produced instead of the depth/time binned plot.

handleSpreadsheet()

This function handles writing out the decoded tbd data in CSV format. Either a filename is provided or the spreadsheet is sent to 'stdout'. If the csvHeader flag is set to True, a header is also provided.

Notes for self.args

- args['debugFlag']: Boolean flag. If True, additional output is printed to standard output.
- args['csvOut']: May be a full or relative path with filename or *stdout*.
- args['csvHeader']: Boolean flag. If True, a header is included with CSV output to file or standard out.

loadFileInventory(*fname*)

This loads an existing file inventory of slocum files.

Inventory file structure:
Start, End, File, Cache Z_CACHE:.... Z_PATH0000:....

loadMetadata()

This function generically loads deployment and other metadata required for processing glider files into uniform netCDF files.

`nearest`(*array*, *val*)

Find the nearest value in a sorted numpy array and return the index for the nearest value.

Parameters

- **`array`** (`numpy`) – A value sorted numpy array that is to be searched.
- **`val`** (`value`) – The value to search for the closest matching element in the provided numpy array.

Returns
The index of the closest matching element.

Return type
`int`

`obtainFillValue`(*varName*)

This function obtains a _FillValue to use for the specified variable. If a fill value is not configured, the default is -9999.9.

`obtainVariableName`(*sensorName*)

This function obtains a variable name from a given glider sensor name. There is sometimes some mapping. This requires use of the template file and checking 'rename_from' elements.

`readDbd`(***kwargs*)

This function reads any DBD glider file using the dbdreader python library.

Parameters
`**kwargs` – See below

Returns
xarray Dataset or python dictionary

Return type
xarray Dataset() or dict()

Keyword Arguments

- *inputFile* (`string`) – relative or full path to glifer DBD file
- *cacheDir* (`string`) – relative or full path to glider cache directory. This overrides the default class object value.
- *returnDict* (`boolean`) – when set True, the data returned from this function is a python dictionary. Default: **False**

`readEchogram`()

This function reads the glider tbd file and extracts the embedded echogram from the echometrics data. The *dbd2asc* file cannot be used since it truncates the least significant bits in which the echogram is embedded.

NOTE: This function *readEchogram* should only be used for extracting encoded echogram information embedded in the echometrics data. All other glider files may be read using *dbd2asc*. We highly recommend using the python *dbdreader* module.

This function automatically determines if the glider was in "echo" or "combo" mode. Prior knowledge of the operational mode is not necessary.

TODO: Allow decoding of "egram" mode.

`readSbd`()

This function reads a glider sbd file. This also reads the corresponding cache file for additional metadata.

`readTbd()`

This function reads a glider tbd file using the Teledyne Webb linux binary dbd2asc. This also reads the corresponding cache file for additional metadata.

`readTemplateUnlimitedDims()`

This function reads {template}.json and allocates unlimited dimensions for the xarray Dataset object.

`saveFileInventory`(*invFile*, *sort_by_time=False*)

This saves a file inventory of slocum files.

Inventory file structure:
Start, End, File, Cache Z_CACHE:. . . . Z_PATH0000:. . . .

`stopToDebug()`

Generic function to stop python in its debugger. When stopped by this function, it is necessary to go up one level in the execution stack to get to the exact location of the breakpoint. Use the *up* command to go up one level in the execution stack.

`updateArgumentsFromMetadata()`

Adjust run time arguments based on metadata read by the glider class. This will only adjust arguments if self.deployment is set.

`writeNetCDF()`

Write out an IOOS DAC compatable netCDF file.

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX