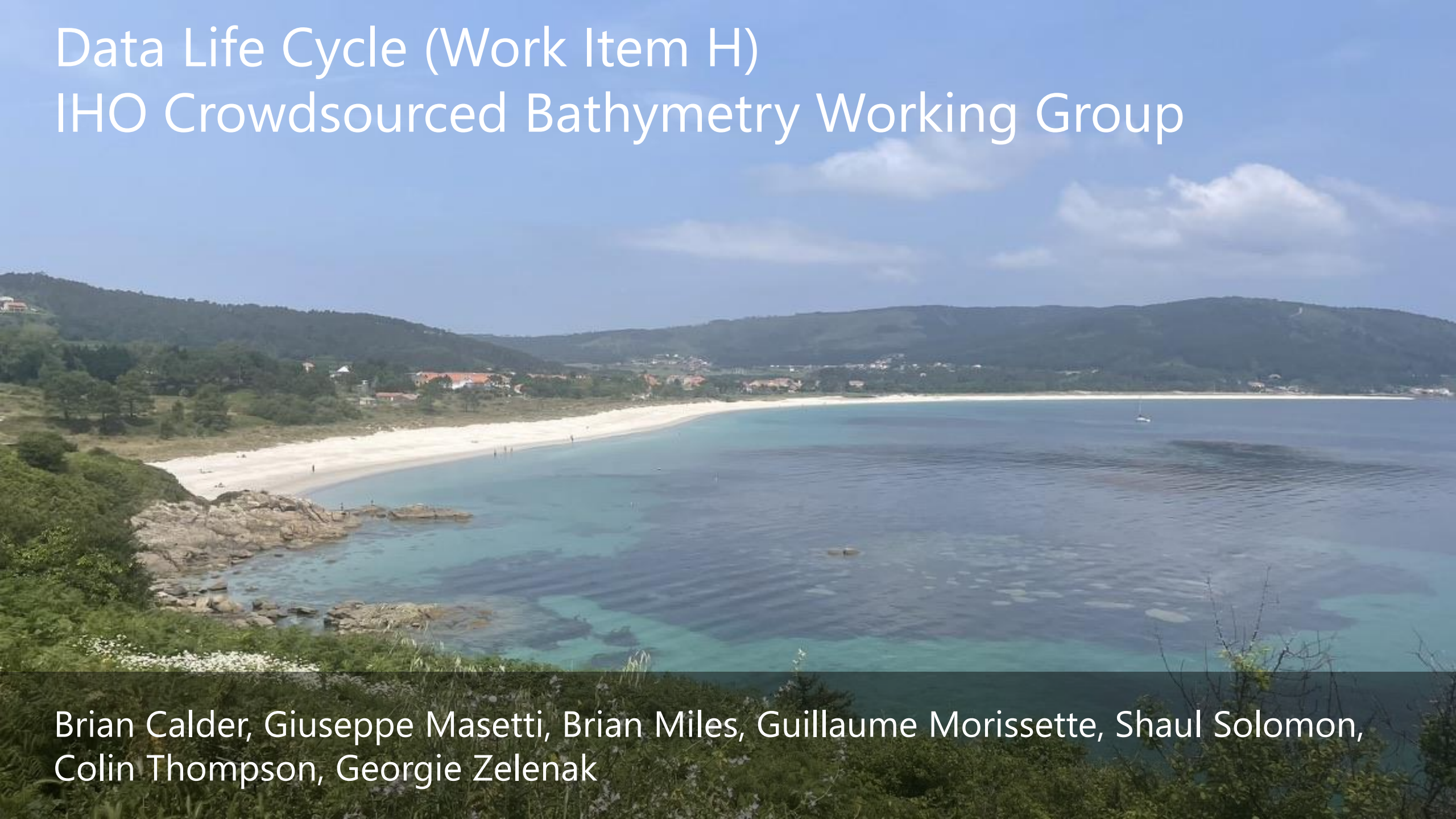
Data Life Cycle (Work Item H)
IHO Crowdsourced Bathymetry Working Group



Brian Calder, Giuseppe Masetti, Brian Miles, Guillaume Morissette, Shaul Solomon, Colin Thompson, Georgie Zelenak

“Clarify all aspects of the CSB data cycle and capture known issues, requirements and suggested enhancements.”

1. Capture Processing
2. Metadata
3. Data Format Extension
4. Data Access
5. Use Cases
6. Availability

1. Data Access
2. Metadata
3. Capture Processing
4. Data Format Extension
5. Use Cases
6. Availability

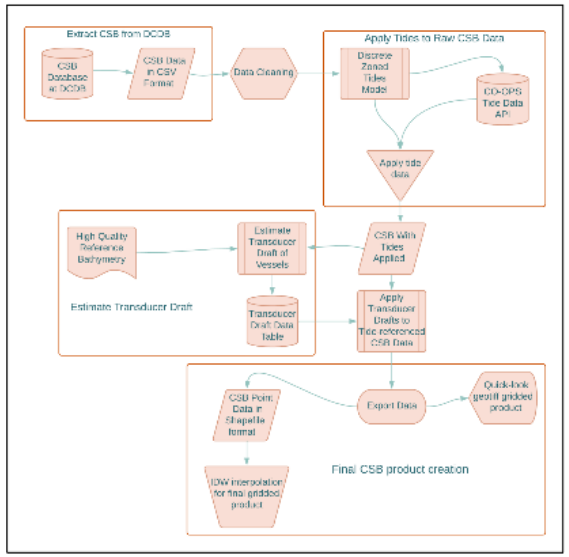


Figure 2: General workflow diagram for CSB processing script

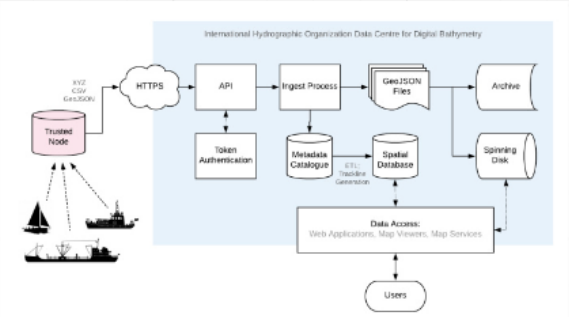
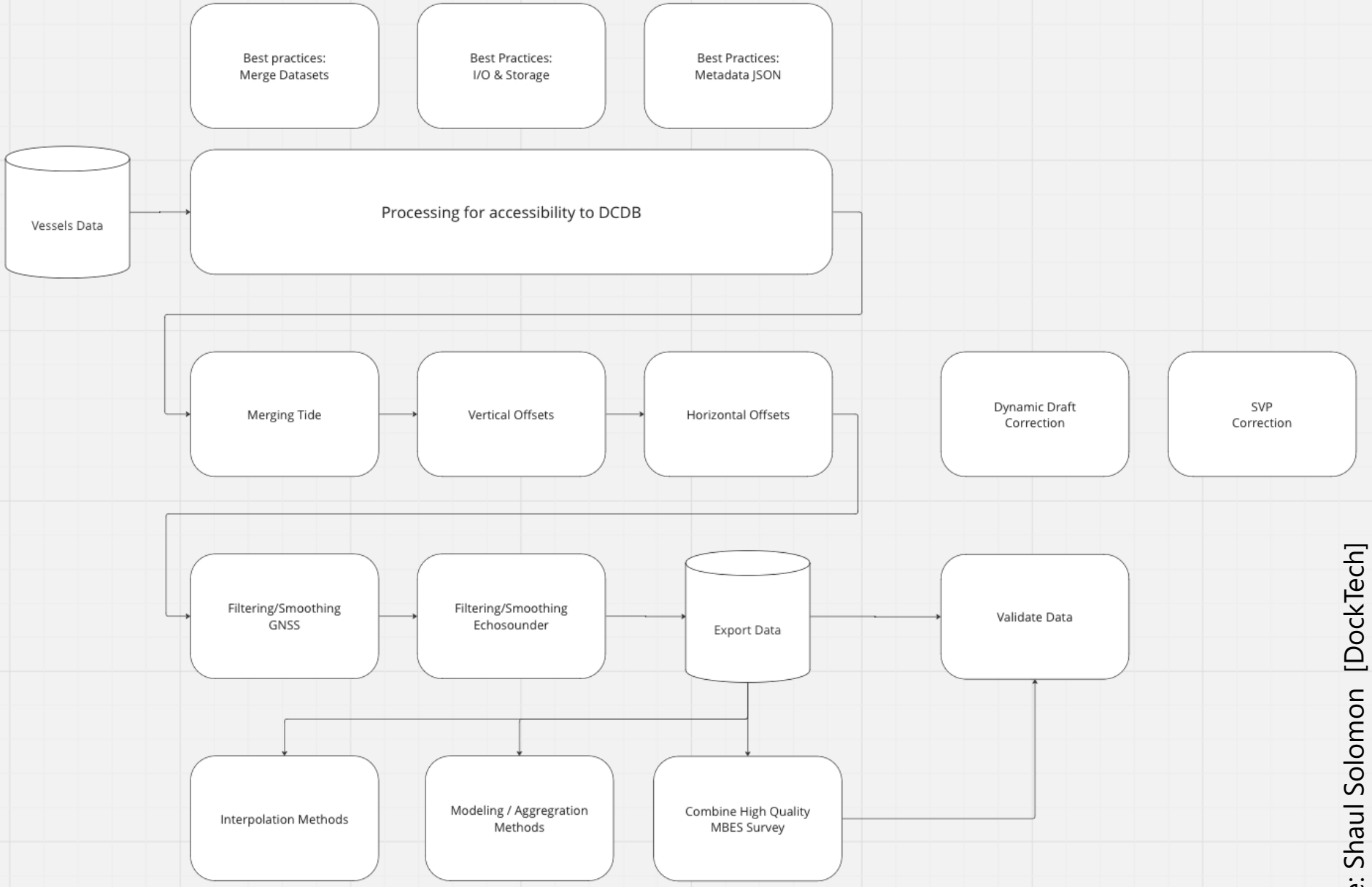


Figure 3. A schematic of the flow of CSB data from the Trusted Node, to the IHO DCDB, and to the public.



Source: Shaul Solomon [DockTech]

Metadata

Review the current metadata structure for CSB data for completeness, *encoding* methods, *validation mechanisms*, and support for end-user database mapping (e.g., for groups of collectors and other aggregations), and recommend modifications.

noaa_b12_v3_0_0_suggested.json — NOAA (git: main)


```
{
  "type": "FeatureCollection",
  "crs": {
    "horizontal": {
      "type": "EPSG",
      "value": 8252
    },
    "vertical": "Transducer"
  },
  "properties": {
    "providerContactPoint": {
      "orgName": "Example Cruises Inc",
      "email": "support@example.com",
      "logger": "Rose Point ECS",
      "loggerVersion": "1.0"
    },
    "convention": "GeoJSON CSB 3.0",
    "dataLicense": "CC0 1.0",
    "platform": {
      "uniqueID": "EXAMPLE-f8c469f8-df38-11e5-b86d-9a79f06e9478",
      "type": "Ship",
      "name": "USCGC Healy",
      "length": 128,
      "IDType": "MMSI",
      "IDNumber": 908338000,
      "sensors": [...],
      "correctors": {
        "positionReferencePoint": "GNSS",
        "soundSpeedDocumented": true,
        "positionOffsetsDocumented": true,
        "dataProcessed": true,
        "motionOffsetsApplied": true,
        "draftApplied": true
      }
    },
    "algorithms": [
      {
        "name": "deduplicate",
        "Params": ""
      }
    ]
  },
  "lineage": [
    {
      "type": "CRSTransformation",
      "timestamp": "2021-11-22T16:10:09.346821",
      "detail": {
        "origin": "EPSG:4326",
        "destination": "EPSG:8252",
        "method": "GeoTrans"
      }
    },
    {
      "type": "TimeInterpolation",
      "timestamp": "2021-11-22T16:10:09.346821",
```


Line: 1 JSON Tab Size: 4

b12_v3_1_0_example.json — IHO (git: main)


```
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "EPSG:4326"
    }
  },
  "properties": {
    "trustedNode": {
      "providerOrganizationName": "Sea-ID",
      "providerEmail": "support@sea-id.org",
      "uniqueVesselID": "SEAID-e8c469f8-df38-11e5-b86d-9a79f06e9478",
      "convention": "GeoJSON CSB 3.1",
      "dataLicense": "CC0 1.0",
      "providerLogger": "Rose Point ECS",
      "providerLoggerVersion": "1.0",
      "navigationCRS": "EPSG:4326",
      "verticalReferenceOfDepth": "Transducer",
      "vesselPositionReferencePoint": "GNSS"
    },
    "platform": {
      "uniqueID": "SEAID-e8c469f8-df38-11e5-b86d-9a79f06e9478",
      "type": "Private vessel",
      "name": "White Rose of Drachs",
      "length": 65,
      "IDType": "MMSI",
      "IDNumber": "369958000",
      "sensors": [...],
      "soundSpeedDocumented": true,
      "positionOffsetsDocumented": true,
      "dataProcessed": true,
      "contributorComments": "On 2022-03-08, at 20:30 UTC, the echo sounder lost bottom tracking
    },
    "processing": [
      {
        "type": "CRSChange",
        "timestamp": "2023-02-14T02:00:00.0000Z",
        "original": "EPSG:4326",
        "destination": "EPSG:8252",
        "method": "GeoTrans"
      },
      {
        "type": "VerticalReduction",
        "timestamp": "2023-02-14T03:00:00.0000Z",
        "reference": "ChartDatum",
        "datum": "CANNORTH2016v1HyVSEP_NAD83v6_CD",
        "method": "Predicted Waterlevel",
        "model": "CANNORTH2016v1HyVSEP_NAD83v6_CD"
      },
      {
        "type": "GNSS",
        "timestamp": "2023-02-14T04:00:00.0000Z",
        "algorithm": "RTKLib",
        "version": "1.2.0"
```


Line: 1 JSON Tab Size: 4

Source: Brian Miles [CCOM/JHC, CSBSchema Project]

b12_v3_1_0_example.json — IHO (git: main)


```
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "EPSG:4326"
    }
  },
  "properties": {
    "trustedNode": {
      "providerOrganizationName": "Sea-ID",
      "providerEmail": "support@sea-id.org",
      "uniqueVesselID": "SEAID-e8c469f8-df38-11e5-b86d-9a79f06e9478",
      "convention": "GeoJSON CSB 3.1",
      "dataLicense": "CC0 1.0",
      "providerLogger": "Rose Point ECS",
      "providerLoggerVersion": "1.0",
      "navigationCRS": "EPSG:4326",
      "verticalReferenceOfDepth": "Transducer",
      "vesselPositionReferencePoint": "GNSS"
    },
    "platform": {
      "uniqueID": "SEAID-e8c469f8-df38-11e5-b86d-9a79f06e9478",
      "type": "Private vessel",
      "name": "White Rose of Drachs",
      "length": 65,
      "IDType": "MMSI",
      "IDNumber": "369958000",
      "sensors": [...],
      "soundSpeedDocumented": true,
      "positionOffsetsDocumented": true,
      "dataProcessed": true,
      "contributorComments": "On 2022-03-08, at 20:30 UTC, the echo sounder lost bottom tracking
    },
    "processing": [
      {
        "type": "CRSChange",
        "timestamp": "2023-02-14T02:00:00.0000Z",
        "original": "EPSG:4326",
        "destination": "EPSG:8252",
        "method": "GeoTrans"
      },
      {
        "type": "VerticalReduction",
        "timestamp": "2023-02-14T03:00:00.0000Z",
        "reference": "ChartDatum",
        "datum": "CANNORTH2016v1HyVSEP_NAD83v6_CD",
        "method": "Predicted Waterlevel",
        "model": "CANNORTH2016v1HyVSEP_NAD83v6_CD"
      },
      {
        "type": "GNSS",
        "timestamp": "2023-02-14T04:00:00.0000Z",
        "algorithm": "RTKLib",
        "version": "1.2.0"
```


Line: 1 JSON Tab Size: 4

b12_v3_2_0-BETA_example.json — IHO (git: main)


```
{
  "type": "FeatureCollection",
  "crs": {
    "type": "name",
    "properties": {
      "name": "EPSG:4326"
    }
  },
  "properties": {
    "trustedNodePlatform": {
      "providerOrganizationName": "Sea-ID",
      "providerEmail": "support@sea-id.org",
      "uniqueVesselID": "SEAID-e8c469f8-df38-11e5-b86d-9a79f06e9478",
      "type": "Private vessel",
      "name": "White Rose of Drachs",
      "IDType": "MMSI",
      "IDNumber": "369958000"
    },
    "observationCollection": {
      "trustedNode": {
        "convention": "GeoJSON CSB 3.2",
        "dataLicense": "CC0 1.0",
        "providerLogger": "Rose Point ECS",
        "providerLoggerVersion": "1.0",
        "navigationCRS": "EPSG:4326",
        "verticalReferenceOfDepth": "Transducer",
        "vesselPositionReferencePoint": "GNSS"
      },
      "platform": {
        "length": 65,
        "sensorDescriptions": [...],
        "soundSpeedDocumented": true,
        "positionOffsetsDocumented": true,
        "dataProcessed": true,
        "contributorComments": "On 2022-03-08, at 20:30 UTC, the echo sounder lost bottom trackin
        },
      "processing": [
        {
          "type": "CRSChange",
          "timestamp": "2023-02-14T02:00:00.0000Z",
          "original": "EPSG:4326",
          "destination": "EPSG:8252",
          "method": "GeoTrans"
        },
        {
          "type": "VerticalReduction",
          "timestamp": "2023-02-14T03:00:00.0000Z",
          "reference": "ChartDatum",
          "datum": "CANNORTH2016v1HyVSEP_NAD83v6_CD",
          "method": "Predicted Waterlevel",
          "model": "CANNORTH2016v1HyVSEP_NAD83v6_CD"
        },
        {
          "type": "GNSS",
          "timestamp": "2023-02-14T04:00:00.0000Z",
```


Line: 1 JSON Tab Size: 4

Source: Brian Miles [CCOM/JHC, CSBSchema Project]

github.com

CCOMJHC / csbschema

Code | Issues 5 | Pull requests 1 | Discussions | Actions | Projects | Wiki | Security

Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

Dismiss

Filters | is:issue is:open | Labels 9 | Milestones 1 | New issue

5 Open | 5 Closed

Author | Label | Projects | Milestones | Assignee | Sort

- **Extra information required for VerticalReduction processing metadata** enhancement
  #19 opened 2 days ago by brian-r-calder
- **Non-mandatory elements cause validation errors** bug
  #18 opened 2 days ago by brian-r-calder
- **Missing non-required IDType breaks validation** bug
  #17 opened 2 days ago by brian-r-calder
- **Typo in GeoJSON CSB 3.1 (2023-08) schema** bug 1
  #15 opened 3 days ago by selimnairb
- **Associate CSB JSON documents with a particular schema version**
  #14 opened on Aug 21, 2023 by selimnairb

ProTip! Type g p on any issue or pull request to go back to the pull request listing page.

Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

© 2024 GitHub, Inc.

github.com

CCOMJHC / csbschema

Code | Issues 5 | Pull requests 1 | Discussions | Actions | Projects | Wiki | Security

Non-mandatory elements cause validation errors #18

Edit | New issue

Open **brian-r-calder** opened this issue 2 days ago · 0 comments

brian-r-calder commented 2 days ago Member

The properties/platform/{type,name,length} elements in 3.1.0 2023-08 are not mandatory in B.12, but they cause validation to fail if missing.

brian-r-calder added the bug label 2 days ago

Add a comment

Write | Preview

Add your comment here...

Markdown is supported | Paste, drop, or click to add files

Close issue | Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Assignees
No one—assign yourself

Labels
bug

Projects
None yet

Milestone
No milestone

Development
Create a branch
for this issue or link a pull request.

Notifications Customize
Unsubscribe
You're receiving notifications because you authored the thread.

1 participant

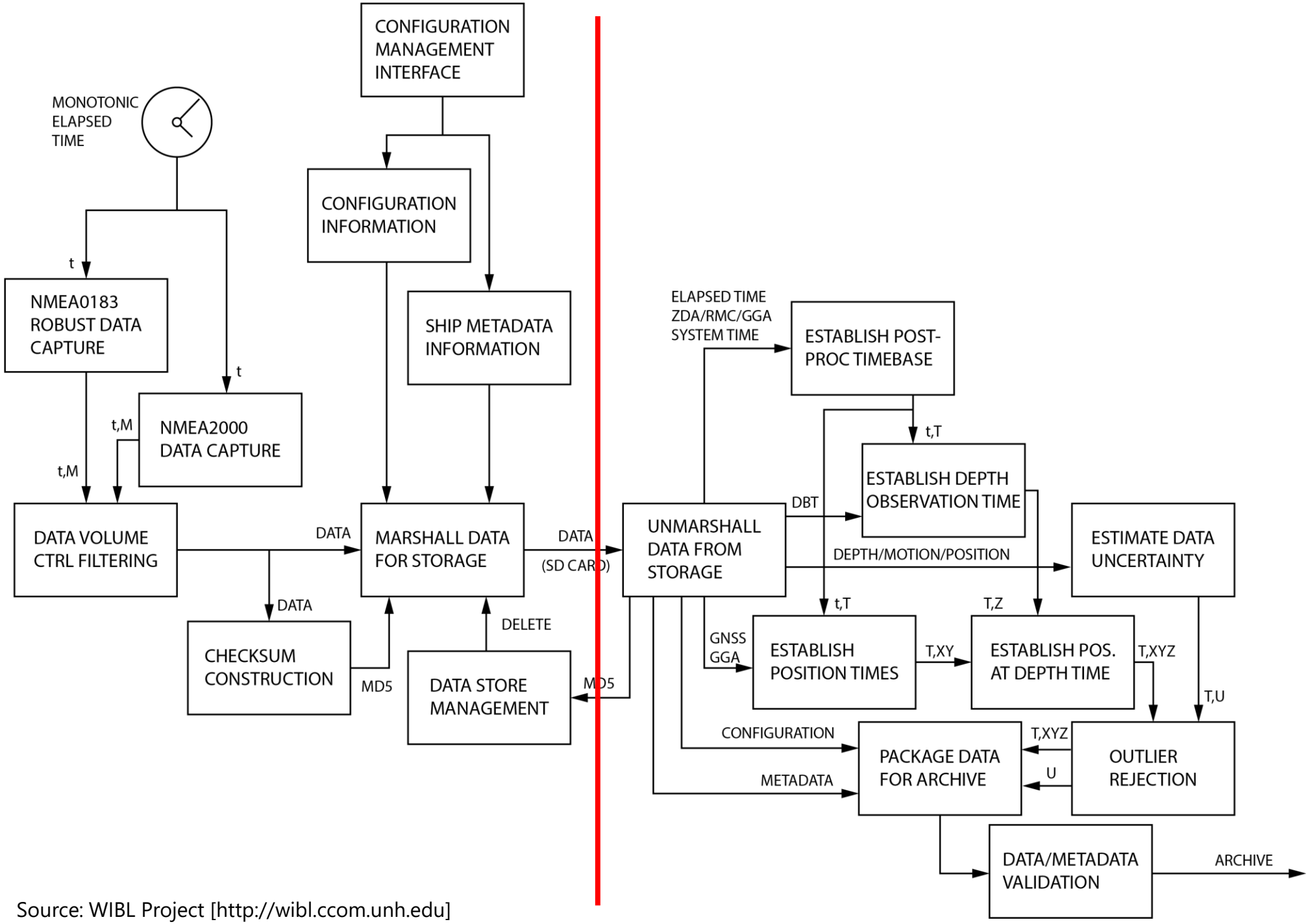
Lock conversation

Pin issue

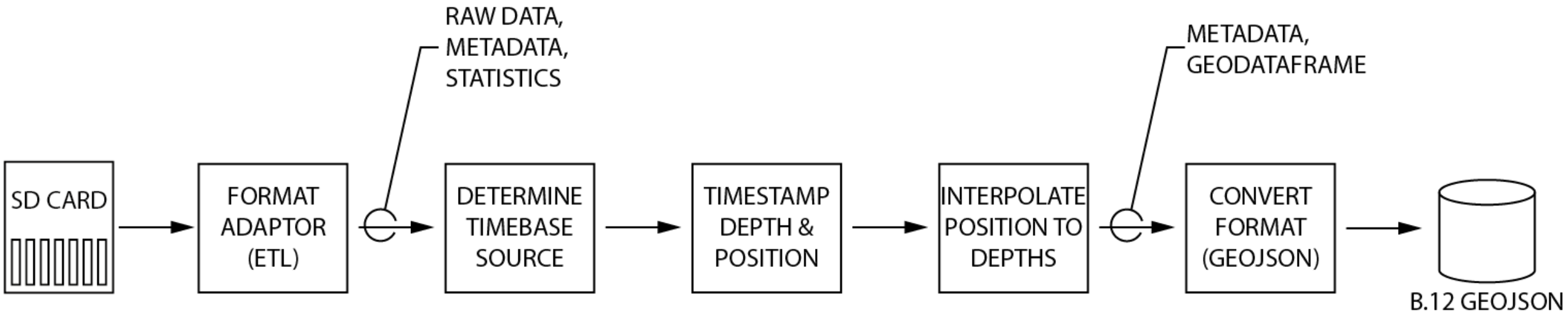
1. Collect suggested modifications in a draft specification
2. Test the implementation with example data
3. Iterate as required to beat out any deficiencies
4. Bring proposal to CSBWG/DCDB for adoption

Capture Processing

Review, recommend, and document (in conjunction with the available open source code development projects) the data flow of standard processing stages for data capture, including extensions for uncertainty estimation and associated metadata/data format extensions.



Source: WIBL Project [http://wibl.ccom.unh.edu]


RAW DATA,
METADATA,
STATISTICS
METADATA,
GEODATAFRAME
SD CARD
FORMAT
ADAPTOR
(ETL)
DETERMINE
TIMEBASE
SOURCE
TIMESTAMP
DEPTH &
POSITION
INTERPOLATE
POSITION TO
DEPTHS
CONVERT
FORMAT
(GEOJSON)
B.12 GEOJSON

OpenVBI

EXPLORER

- OPENVBI
  - build
  - openvbi
    - __pycache__
    - adaptors
      - __pycache__
      - __init__.py
      - dcdb.py
      - generic_ascii.py
      - teamsurv.py
      - ydvr.py
    - core
      - __pycache__
      - __init__.py
      - interpolation.py
      - metadata.py
      - observations.py
      - statistics.py
      - timebase.py
      - types.py
    - corrections
      - __pycache__
      - waterlevel
        - __pycache__
        - noaa
          - __pycache__
          - __init__.py
        - __init__.py
      - __init__.py
    - examples
      - basic_processing.py
      - dedup.py
      - metadata.py
      - prep.py
    - filters
      - __pycache__
- OUTLINE
- TIMELINE

basic_processing.py

openvbi > examples > basic_processing.py > ...

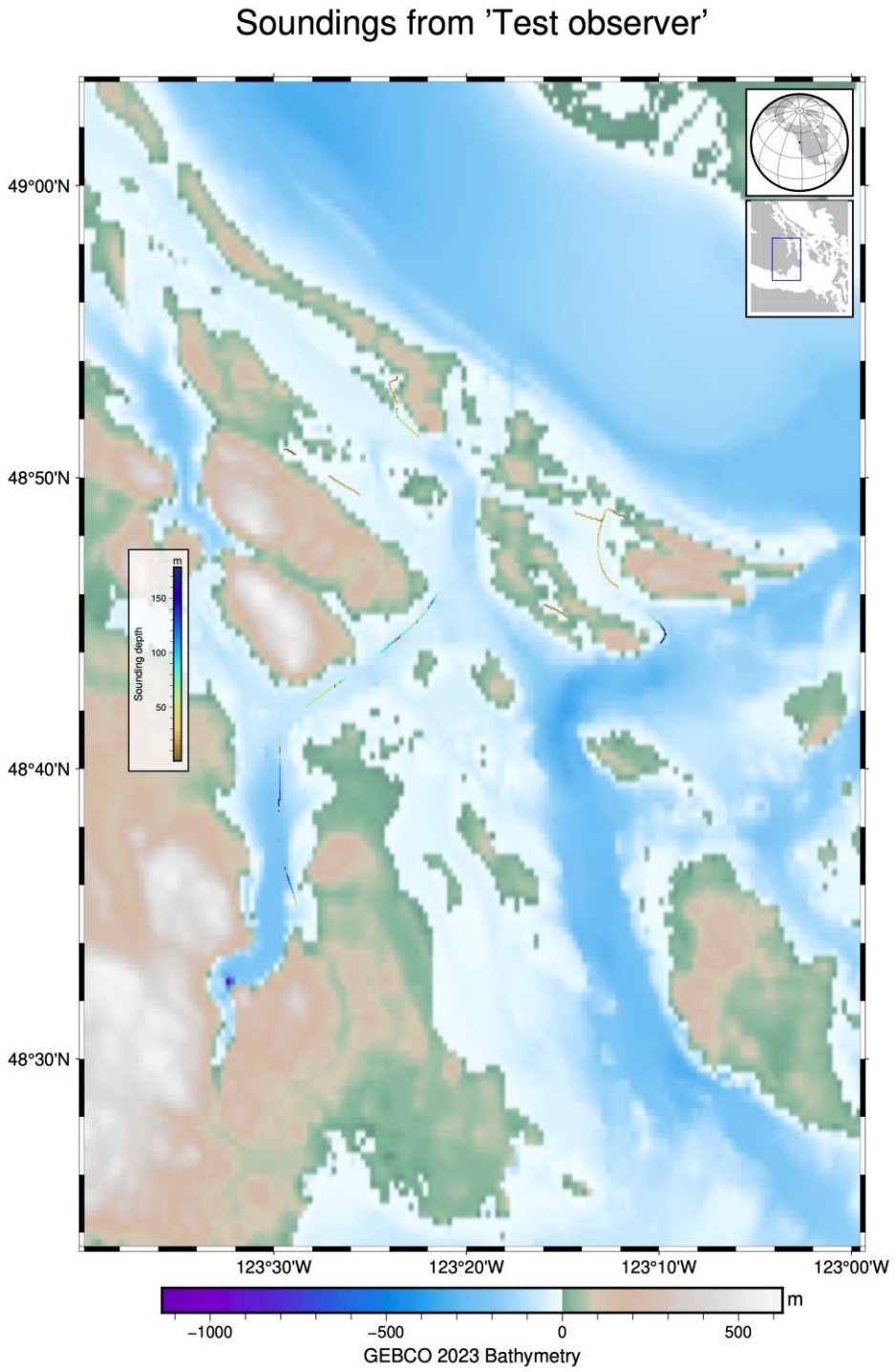
```python
import uuid
from openvbi.adaptors.ydvr import load_data
import openvbi.core.metadata as md
from openvbi.adaptors.dcdb import write_geojson, write_csv_json

# In order to fill out the metadata, we need a DCDB-style Trusted Node identification string,
# and some core data for the provider and e-mail.
dcdb_id = 'OPNVBI'
provider_id = 'OpenVBI'
provider_email = 'hello@openvbi.org'

# In practice, you'd want to have a fixed unique identifier for the particular logger that you're
# working on.  Since this isn't available in the file being loaded, you'd need to look up the
# filename (or other information) in a database to translate to the unique identifier.  For example
# purposes here, we just make up a random identifier
unique_id = dcdb_id + '-' + str(uuid.uuid4())

# Load from YachtDevices, and convert into a dataframe
data = load_data('/Users/brc/Projects-Extras/OpenVBI/ExampleData/00030095.DAT')

# Since there isn't a lot of information on the logger from the datafile, we need to populate the
# metadata separately.  We focus here on the mandatory metadata.  In practice, you would probably want
# to look up the unique identifier in a database of loggers, and pick out the appropriate metadata (or
# potentially a pre-formatted object).
data.meta = md.Metadata()
data.meta.setProviderID(provider_id, provider_email)
data.meta.setIdentifiers(unique_id, 'YDVR', '1.0')
data.meta.setReferencing(md.VerticalReference.TRANSDUCER, md.VerticalReferencePosition.GNSS)
data.meta.setVesselID(md.VesselIdentifier.MMSI, "000000000")

# YachtDevices is a NMEA2000 device, so we convert 'Depth' messages for depth information
data.generate_observations('Depth')

# Finally, write the output to a DCDB-compatible, IHO CSBWG B.12 GeoJSON file.  Note that the
# keyword parameters here are passed on to json.dump() directly, so you can control the output
# a little more directly.  Using "indent" here gives something that's more verbose but easier
# to read.
write_geojson(data, '00030095.json', indent=2)
write_csv_json(data, '00030095_copy', indent=2)
```

main | Sign in to Jira | No active issue | Bitbucket: Brian Calder: | 0 0 | 0 | Ln 12, Col 55 | Spaces: 4 | UTF-8 | LF | Python | 3.12.2 ('openvbi': conda)

Use Cases

Establish recommendations for one or more use cases of CSB, and associated guidelines for data products (e.g., interpretation of quality, known problems, required processing, etc.). Collaborate with other task groups preparing specialized products (e.g., for HO use).

Wireless
Inexpensive
Bathymetry
Logger



Source: Brian Miles [CCOM/JHC, WIBL Project]

Availability

Promote and contribute to an open source reference implementation of available algorithms for all the processing steps, preferably with a permissive license such as MIT or CC0.

Adapters

TeamSurv
YachtDevices
Generic ASCII
DCDB [I/O]

Core

Dataset Management
Time-based Interpolation
Timestamping/Depth Construction
Metadata Management
Packet & General Statistics
Uncertainty

Corrections

NOAA Waterlevels
CHS Waterlevels
Vertical Bias

Filters

Depth [Shoal/Deep]
Time [Early/Late]
Deduplication
Outliers

Examples

ELT → GeoJSON
Metadata Addition
Algorithmic Filtering

Source: OpenVBI Project [https://github.com/CCOMJHC/OpenVB

OpenVBI

EXPLORER

OPENVBI

- openvbi
  - core
    - __init__.py
    - interpolation.py
    - metadata.py
    - observations.py
    - statistics.py
    - timebase.py
    - types.py
    - corrections
      - __pycache__
      - waterlevel
        - __pycache__
        - noaa
          - __pycache__
          - __init__.py
        - __init__.py
      - __init__.py
    - examples
      - basic_processing.py
      - dedup.py
      - metadata.py
      - prep.py
    - filters
      - __pycache__
      - __init__.py
      - deduplicate.py
      - thresholding.py
      - timeslot.py
    - __init__.py
  - openvbi.egg-info
  - .gitignore
  - Dockerfile.build
  - LICENSE
  - pyproject.toml

OUTLINE

TIMELINE

thresholding.py

openvbi > filters > thresholding.py > shoaler_than > _metadata

```python
import openvbi.core.metadata as md
from openvbi.filters import Filter
from openvbi import version

# Remove any points that are shoaler than the threshold specified
class shoaler_than(Filter):
    def __init__(self, threshold: float) -> None:
        self._threshold = threshold
        super().__init__()

    def _execute(self, dataset: geopandas.GeoDataFrame) -> geopandas.GeoDataFrame:
        self.n_inputs = len(dataset)
        dataset = dataset[dataset['z'] > self._threshold]
        self.n_outputs = len(dataset)
        return dataset

    def _metadata(self, meta: md.Metadata) -> None:
        meta.addProcessingAction(md.ProcessingType.ALGORITHM, None,
            name='ShoalDepth Filter',
            source='OpenVBI',
            version=version(),
            comment=f'After filtering, total {self.n_outputs} points selected from {self.n_inputs}.')

# Remove any points that are deeper than the threshold specified
class deeper_than(Filter):
    def __init__(self, threshold: float) -> None:
        self._threshold = threshold
        super().__init__()

    def _execute(self, dataset: geopandas.GeoDataFrame) -> geopandas.GeoDataFrame:
        self.n_inputs = len(dataset)
        dataset = dataset[dataset['z'] < self._threshold]
        self.n_outputs = len(dataset)
        return dataset

    def _metadata(self, meta: md.Metadata) -> None:
        meta.addProcessingAction(md.ProcessingType.ALGORITHM, None,
            name='DeepDepth Filter',
            source='OpenVBI',
            version=version(),
            comment=f'After filtering, total {self.n_outputs} points selected from {self.n_inputs}.')
```

main | Sign in to Jira | No active issue | Bitbucket: Brian Calder: | 0 0 | 0 | Ln 46, Col 29 | Spaces: 4 | UTF-8 | LF | Python | 3.12.2 ('openvbi': conda)

CIDCO-Uncertainty

EXPLORER

CIDCO-UNCERT...
- doc
- src
  - datalogger_examples.py
  - georeference.py
  - test.py
  - uncertainty.py
- .gitignore
- LICENSE
- README.md

OUTLINE

TIMELINE

datalogger_examples.py

src > datalogger_examples.py > ...

```python
print("Hydroblock Uncertainty")
u = uncertainty.TotalPropagatedUncertainty()

tpu = u.compute(latitude,longitude,ellipsoidal_height,heading,pitch,roll,

print(tpu)
error_x=math.sqrt(tpu[0][0])
error_y=math.sqrt(tpu[1][1])
error_z=math.sqrt(tpu[2][2])

print(f"Approximate x error: {error_x:.3f} m")
print(f"Approximate y error: {error_y:.3f} m")
print(f"Approximate z error: {error_z:.3f} m")

print("--------------------------------------------------")

# ------------------------------------------
# Low-cost WIBL datalogger

#10m-ish error at equator for uncorrected GNSS readings
latitude_sigma = 0.0001
longitude_sigma = 0.0001
ellipsoidal_height_sigma = 1

#FIXME: what kind of IMU figures?
#IMU accuracy: the BNO055 datasheet gives +-1 degree on roll/pitch, and +
heading_sigma = math.radians(2/2)
pitch_sigma = math.radians(1/2)
roll_sigma = math.radians(1/2)

#FIXME: what kind of sonar ?
#Imagenex 852 datasheet advertises a 1mm range resolution...which is prob
depth_sigma = 0.001

#FIXME: those are purely fictional
#Offsets are unknown
offset_x_sigma = 1 #Length of the boat
offset_y_sigma = 1 #Half the width
offset_z_sigma = 1 #Twice the depth of the boat assuming transducer fixed


print("WIBL Uncertainty")
tpu = u.compute(latitude,longitude,ellipsoidal_height,heading,pitch,roll,

print(tpu)
error_x=math.sqrt(tpu[0][0])
error_y=math.sqrt(tpu[1][1])
error_z=math.sqrt(tpu[2][2])

print(f"Approximate x error: {error_x:.3f} m")
```

Preview README.md

```
# offset_z = 3 //meters

import georeference
g = georeference.Georeference()
result =
g.compute(latitude,longitude,ellipsoidal_height,heading,pitch,r
```

Uncertainty

Get the uncertainty associated with an XYZ position (in the ECEF frame) of a depth sounding (in the WGS84 frame)

```
# Example values
# latitude = math.radians(48.4525) //decimal degrees
# longitude = math.radians(-68.5232) //decimal degrees
# ellipsoidal_height = -28 //meters
# heading = math.radians(350)
# pitch = math.radians(0.5)
# roll = math.radians(1.5)
# depth = 5 //meters
# offset_x = 1 //meters
# offset_y = 2 //meters
# offset_z = 3 //meters
# latitude_sigma = 0.00001
# longitude_sigma = 0.0001
# ellipsoidal_height_sigma = 1
# heading_sigma = 1
# pitch_sigma = 1
# roll_sigma = 1
# depth_sigma = 0.5
# offset_x_sigma = 0.0000001
# offset_y_sigma = 0.0000001
# offset_z_sigma = 0.0000001

import uncertainty
u = uncertainty.TotalPropagatedUncertainty()
result =
u.compute(latitude,longitude,ellipsoidal_height,heading,pitch,r
```

main ⊗ 0 ⚠ 0 ⓘ 11 0

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.12.1 64-bit 11 Spell

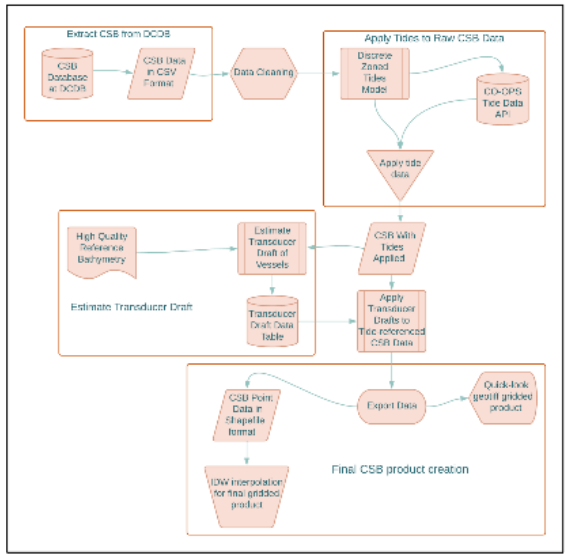


Figure 2: General workflow diagram for CSB processing script

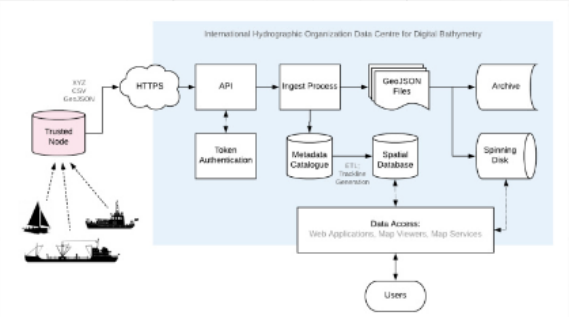
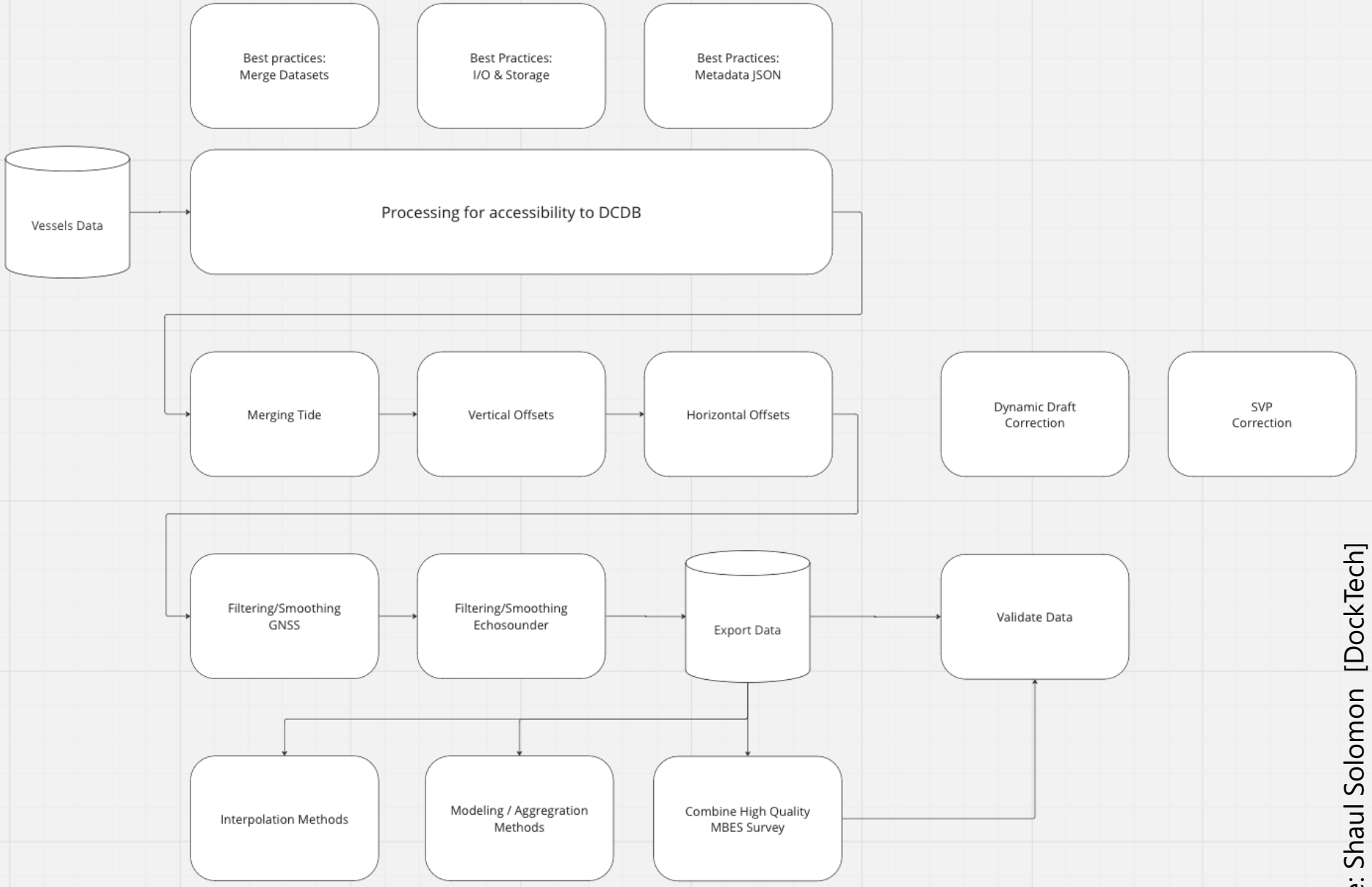


Figure 3. A schematic of the flow of CSB data from the Trusted Node, to the IHO DCDB, and to the public.



Source: Shaul Solomon [DockTech]

OpenVBI

EXPLORER

- OPENVBI
  - openvbi
    - core
      - __init__.py
      - interpolation.py
      - metadata.py
      - observations.py
      - statistics.py
      - timebase.py
      - types.py
      - corrections
        - __pycache__
        - waterlevel
          - __pycache__
          - noaa
            - __pycache__
            - __init__.py
          - __init__.py
        - __init__.py
      - examples
        - basic_processing.py
        - dedup.py
        - metadata.py
        - prep-instrumented.py
        - prep-simple.py
      - filters
        - __pycache__
        - __init__.py
        - deduplicate.py
        - thresholding.py
        - timeslot.py
      - __init__.py
    - openvbi.egg-info
    - .gitignore
    - Dockerfile.build
    - LICENSE
- OUTLINE
- TIMELINE

prep-simple.py

openvbi > examples > prep-simple.py > ...

```python
import copy
from openvbi.adaptors.ydvr import load_data
from openvbi.filters.thresholding import shoaler_than, deeper_than
from openvbi.filters.timeslot import before_time, after_time
from openvbi.corrections.waterlevel.noaa import SingleStation, ZoneTides
from openvbi.adaptors.dcdb import write_geojson

# Pull in data from YachtDevices raw binary file, and convert to depths assuming NMEA2000 data
data = load_data('00030095.DAT')
data.generate_observations('Depth')

# Calibrate acceptable depth and time windows for data (note that these are simply for
# demonstration purposes: this cuts off a lot of valid data!)
min_depth = data.depths['z'].min()
max_depth = data.depths['z'].max()
depth_range = max_depth - min_depth
shoal_threshold = min_depth + depth_range/3.0
deep_threshold = max_depth - depth_range/3.0

min_time = data.depths['t'].min() + 10.0*60.0 # Remove first ten minutes
max_time = data.depths['t'].max() - 10.0*60.0 # Remove last ten minutes

# Generate filters for shoal/deep depth, and early/late time
shoal = shoaler_than(shoal_threshold)
deep = deeper_than(deep_threshold)
early = before_time(min_time)
late = after_time(max_time)

# Filter for depth window, and time window
data = early.Execute(late.Execute(deep.Execute(shoal.Execute(data))))

# Correct for waterlevel using NOAA zoned tides and live API for waterlevels
zone_tide_wl = ZoneTides('tide_zone_polygons_new_WGS84_merge.shp')
zone_tide_wl.preload(data)
zone_tide_wl.correct(data)

# Generate B.12-format GeoJSON output
write_geojson(data, '00030095.json', indent=2)
```

main | Sign in to Jira | No active issue | Bitbucket: Brian Calder: | 0 0 | 0 | Ln 39, Col 1 | Spaces: 4 | UTF-8 | LF | Python | 3.12.2 ('openvbi': conda)

Upcoming Work

- Recommendations for DCDB data access
- Consensus on workflows for:
  - Developers (capture, conversion)
  - End-users (data discovery, understanding, re-use)
- OpenVBI code review & development
- Vertical bias correction in OpenVBI
- Uncertainty estimation in OpenVBI