Ocean Data Tools

https://oceandatatools.org

RVTEC 2024 Tutorial

David Pablo Cohn - david.cohn@openrvdas.org

OpenRVDAS

A modular platform for developing custom data acquisition systems to support vessels or vehicles.

OpenVDM

A flexible vessel-wide data managment system for organizing files from data acquistion systems

Sealog

A modular platform for building custom event-logging solutions to support vessels or vehicles.

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

OpenRVDAS

- **What is it?**
- What's special about it?
- What can it do?

Architecture that lets you snap together simple components to build a customized data acquisition system for your ship/station/chicken coop.

Intended function is to get data from sensors to file/database/network/graphics, with opportunity to process and/or mash it around into different formats on the way.

OpenRVDAS

- **What is it?**
- What's special about it?
- What can it do?

Core is made up of component ***readers***, ***transforms*** and ***writers*** that are combined to create **loggers**.

Additional servers make it easy to assemble, run and monitor collections of loggers and marshal the data they produce.

OpenRVDAS

- What is it?
- **What's special about it?**
- What can it do?

Open - so anyone who wants can look inside and mess with things that don't work for them.

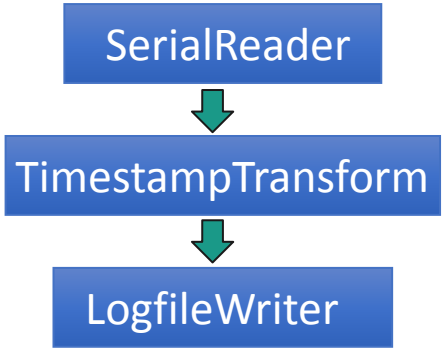
Modular - so easy to modify/extend/keep up to date.

Together, allow "snapping together" existing components to assemble loggers, creating new components as needed.

OpenRVDAS

- What is it?
- What's special about it?
- **What can it do?**

Log sensor data to file.

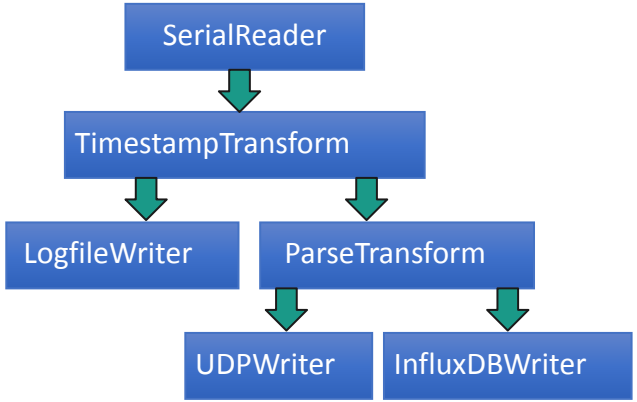



```
.9,1.04,M,,M,,*41
2014-08-01T00:00:00.931000Z $GPVTG,213.66,T,,M,9.4,N,
2014-08-01T00:00:00.931000Z $GPHDT,218.83,T*05
2014-08-01T00:00:00.931000Z $PSXN,20,1,0,0,0*3A
2014-08-01T00:00:00.931000Z $PSXN,22,0.29,0.83*39
2014-08-01T00:00:00.951000Z $PSXN,23,0.58,-1.09,218.8
2014-08-01T00:00:01.815000Z $GPZDA,000001.70,01,08,20
2014-08-01T00:00:01.815000Z $GPGGA,000001.70,2200.114
.9,1.08,M,,M,,*4A
2014-08-01T00:00:01.931000Z $GPVTG,214.31,T,,M,9.6,N,
2014-08-01T00:00:01.932000Z $GPHDT,218.65,T*0D
```

OpenRVDAS

- What is it?
- What's special about it?
- **What can it do?**

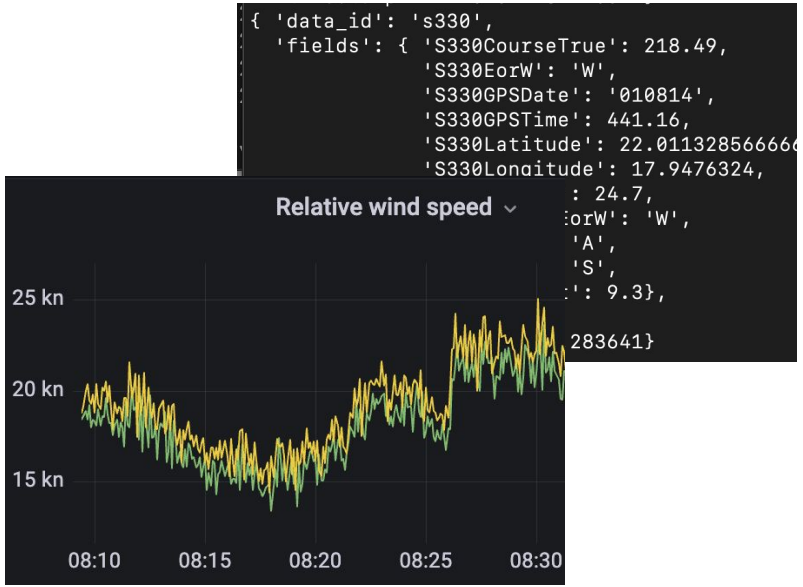
- Parse data into individual fields, send out over network and write to off-machine database.



OpenRVDAS

- What is it?
- What's special about it?
- **What can it do?**

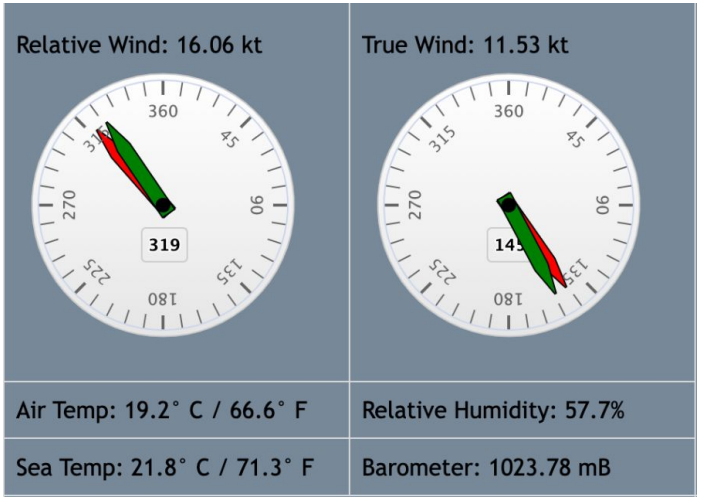
- Parse data into individual fields, send out over network and write to off-machine database.



OpenRVDAS

- What is it?
- What's special about it?
- **What can it do?**

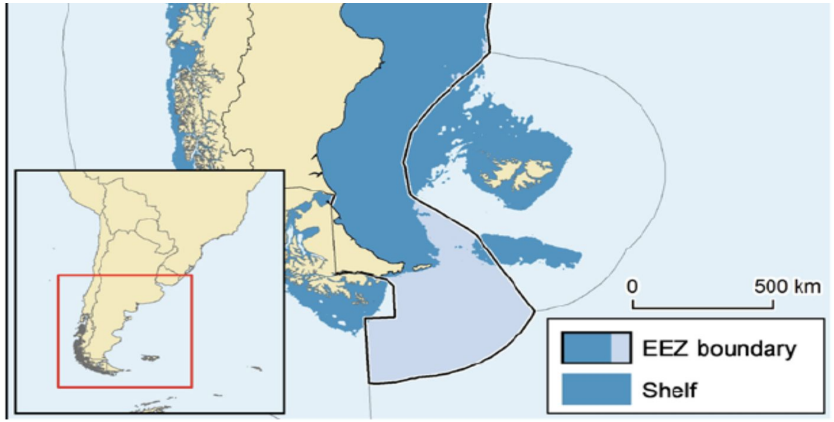
- Combine and numerically manipulate the fields to create derived data products like true winds or moving averages.



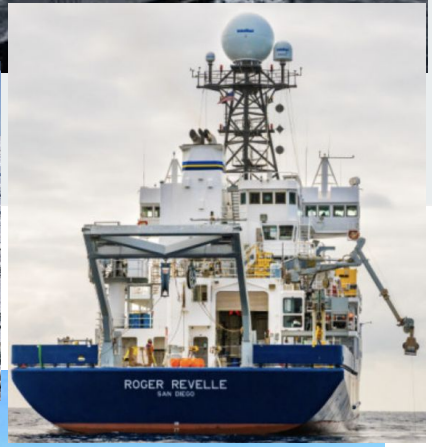
OpenRVDAS

- What is it?
- What's special about it?
- **What can it do?**

- Perform basic quality checks and raise alarms.
- Use raw or derived values to geofence or trigger other system state changes.



OpenRVDAS Installations



OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

Installation - basic download

- Clone repo from OceanDataTools on GitHub
- Allows using running individual loggers

```
% git clone https://github.com/OceanDataTools/openrvdas.git
```

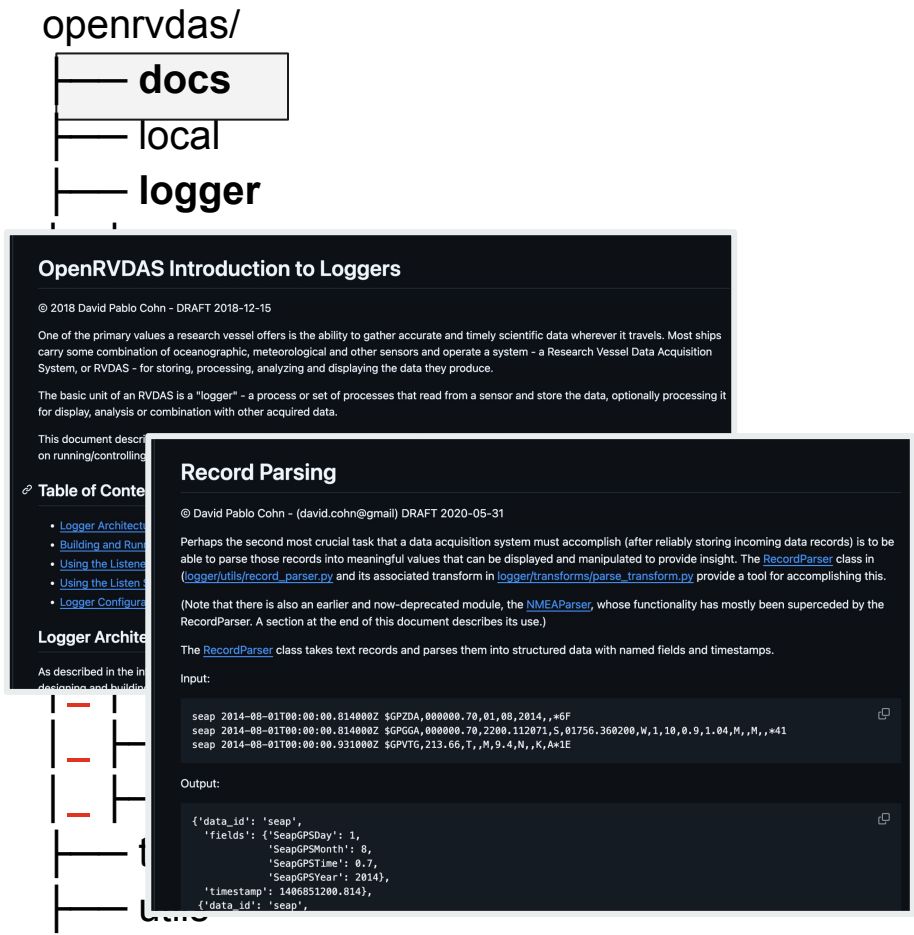
Code Orientation

```
openrvdas/
├── docs
├── local
├── logger
│   ├── listener
│   ├── readers
│   ├── transforms
│   └── writers
│   └── utils
├── server
│   ├── cached_data_server.py
│   ├── lmcmd.py
│   ├── logger_manager.py
│   ├── logger_runner.py
│   ├── logger_supervisor.py
├── test
├── utils
```

Code Orientation

docs directory contains

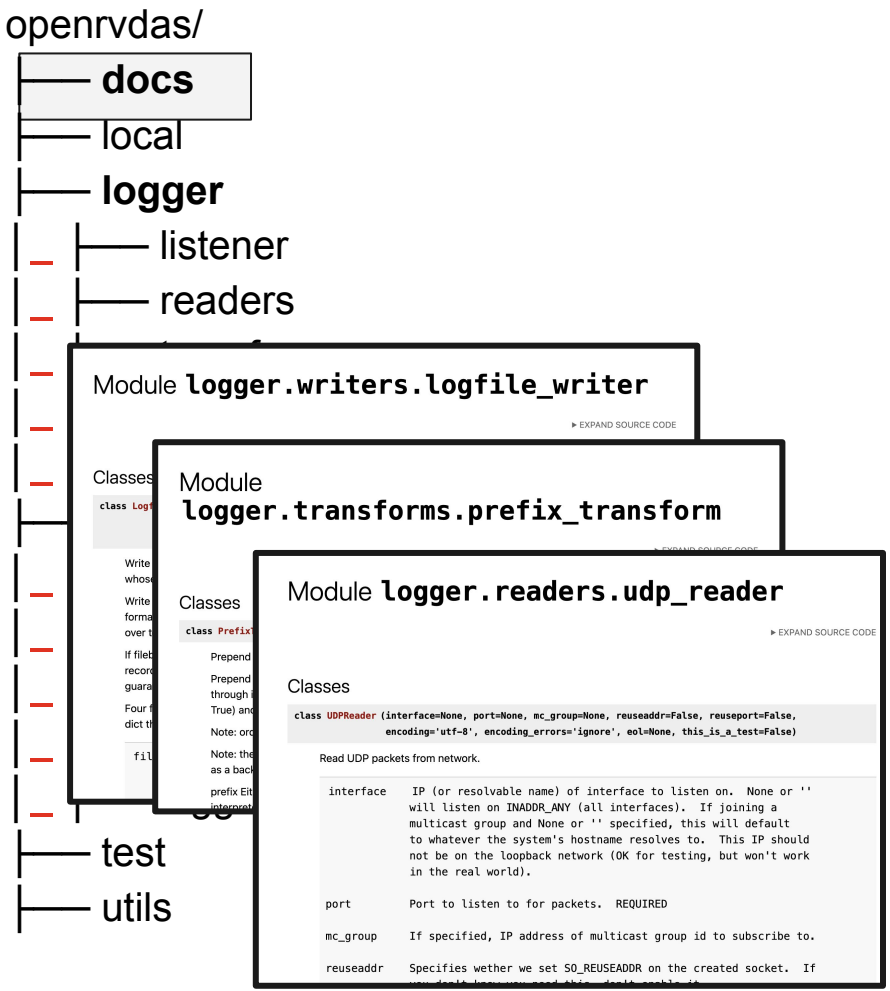
- **YAML-formatted documentation (link)**



Code Orientation

`docs` directory contains

- YAML-formatted documentation
- **HTML-formatted module docstrings (link)**

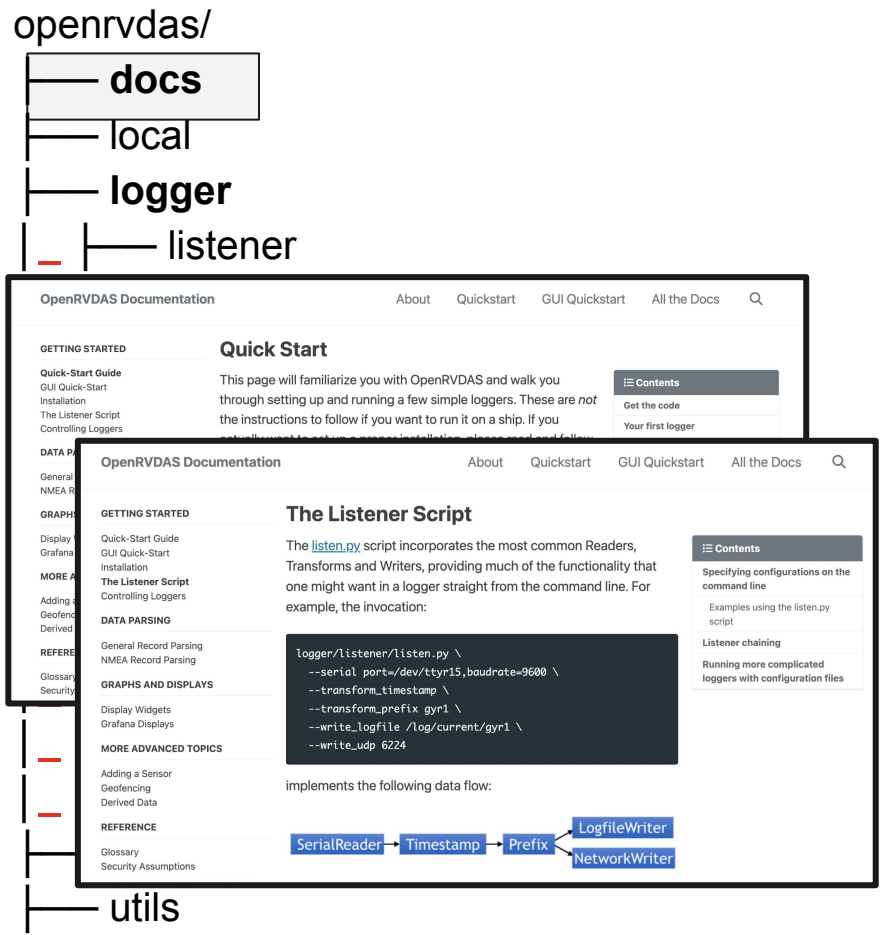


Code Orientation

`docs` directory contains

- YAML-formatted documentation
- HTML-formatted module docstrings

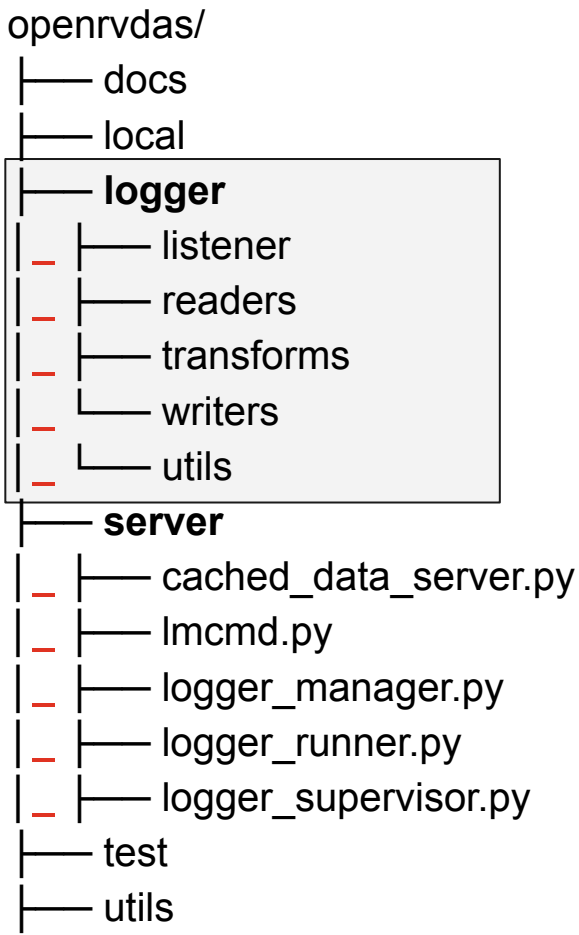
There is also online documentation at https://openrvdas.org



Code Orientation

`logger` directory contains

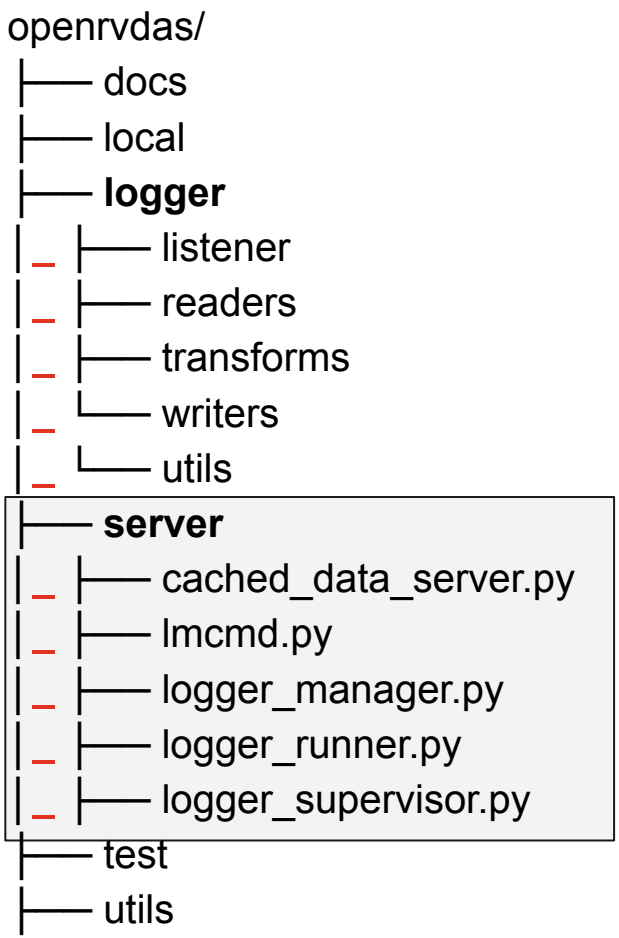
- components: **`readers`**, **`transforms`**, and **`writers`**
- **`listener`** that assembles and runs the components



Code Orientation

`server` directory contains

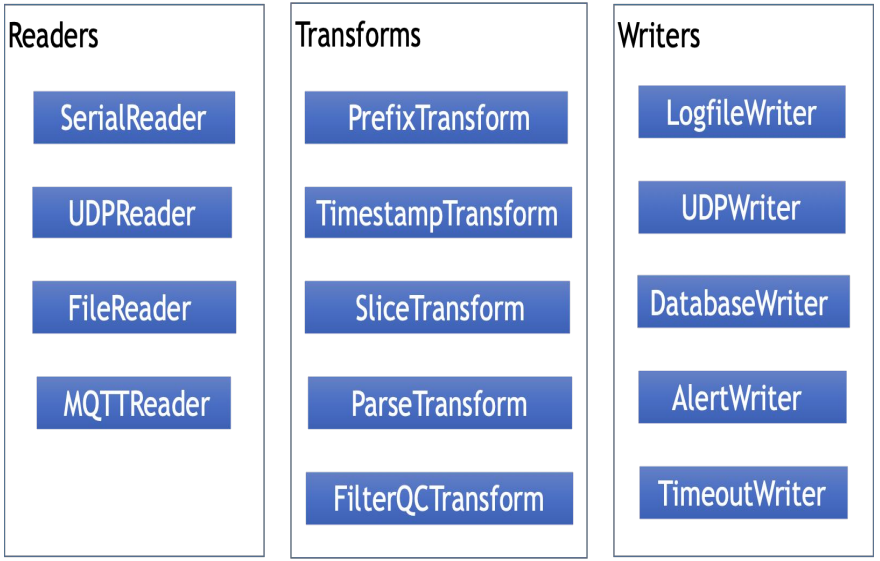
- Servers, surprisingly - scripts that monitor, communicate with and wrangle sets of loggers to make sure they do what you want them to.



Running Loggers

- **Components**
- Hardcoding
- The listen.py command line
- Logger configuration files

Loggers are composed of simple Python components.



Running Loggers

- Hardcoding together

```
reader = SerialReader(port='/dev/ttys1')
transform = TimestampTransform()
writer = LogfileWriter(filebase='/var/logs/knud')
while True:
  in_record = reader.read()
  out_record = transform.transform(in_record)
  writer.write(out_record)
```

Running Loggers

- The listen.py script - command line specification

```
listener.py --serial port=/dev/ttys1 \
     --transform_timestamp \
     --transform_prefix knud \
     --write_logfile /var/logs/knud \
     --write_udp 6221

listen.py --help
```

for all available options

Running Loggers

- The listen.py script with a logger configuration file

```
listener.py --config_file knud_net.yaml
```

```
readers:
  class: SerialReader
  kwargs:
    port: /dev/ttys1
    baudrate: 9600
transforms:
  class: TimestampTransform
writers:
  ...
```

Configuration Files

- YAML format
- Specify readers, transforms and writers
- Run using listen.py with `--config_file` argument

```
readers:
- class: SerialReader
  kwargs:
    port: /dev/ttys1
    baudrate: 9600
transforms:
- class: TimestampTransform
- class: PrefixTransform
  kwargs:
    prefix: knud
writers:
- class: UDPWriter
  kwargs:
    port: 6224

> listen.py \
  --config_file knud_net.yaml
```

Build Your Own...

- Any Python class having a **`read()`** method that returns some sort of Python object/record/string/number/DASRecord.

Reader

```
# Read a virtual Magic Eight Ball
class MagicEightBallReader():
  def __init__():
    ...

  # A blocking call
  def read():
    ...
    return result
```

Build Your Own...

- Any Python class having a **`transform(record)`** method that takes some sort of record as an argument and returns some sort of record (possible '**`None`**').

Transform

```
# Assuming our input is a string,
# return a scrambled version of it
class ScrambleStringTransform():
  def __init__():
    ...

  def transform(record):
    ...
    return scrambled_record
```

Build Your Own...

- Any Python class having a **`write(record)`** method that takes some sort of record as an argument and...maybe does something with it.

(OpenRVDAS doesn't actually care *what* you do with it, as long as you're happy.)

Writer

```
# Contact a skywriting agency and
# ask them to send a plane up to
# write the record in the sky in
# smoke
class SkyWriter():
  def __init__(skywriting_agency,
               aircraft_type):
    ...

  def write(record):
    ...
```

Configuration Files

- **`listen.py`** knows where to find most common components.
- Use **`module`** keyword to tell it where to find any others.
- **`kwargs`** specifies the component's keyword arguments.

```
# TeaLeafReader implementation
class TeaLeafReader():
  def __init__(tea_type='black',
               temp_in_c=100):
    ...
  def read():
    ...
    return result
```

Config file:

```
readers:
- class: TeaLeafReader
  module: local.tea_leaf_reader
  kwargs:
    tea_type: oolong
    temp_in_c: 95
```

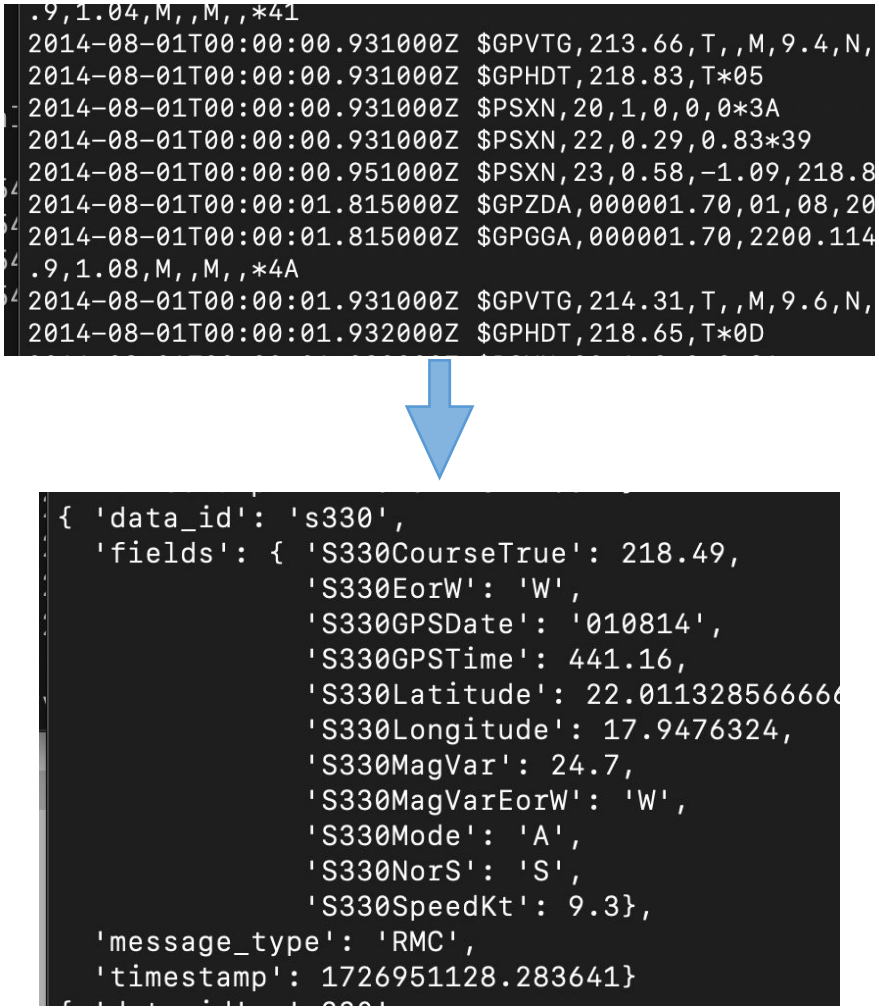
One Especially Important Transform

```
ParseTransform()
```

ParseTransform

- Convert raw ASCII into structured data that can be reformatted, manipulated and displayed

- Can return data as
  - dict of name:value pairs
  - JSON-encoded string
  - OpenRVDAS-specific **`DASRecord`** object



DASRecords

- Handy container for parsed data that includes information you might want.
  - timestamp
  - named value pair dict
  - metadata from parser
- Has methods for comparing, extractring, converting to/from JSON

```
# Print temp if record has changed
record = DASRecord(my_json_str)

if record.data_id == 'rtmp' and
   not record == old_record:
  print(f'{record.timestamp} '
        f'{record.fields["Temp"]'})

  logfile.write(record.as_json())

  old_record = record
```

Parsing Data

- **RecordParser** - takes in strings, returns structured data
- Typical format is "<data_id> <timestamp> <string of values>"

```
grv1 v014-08-01T00:00:00.462000Z 01:022470 00
grv1 2014-08-01T00:00:01.460000Z 01:024628 00
grv1 2014-08-01T00:00:02.459000Z 01:026391 00
grv1 2014-08-01T00:00:03.461000Z 01:026275 00
grv1 2014-08-01T00:00:04.462000Z 01:025053 00
```

Parsing Data

- Define record format, tell how to parse string of values

```
>>> parser = RecordParser(
                record_format='{data_id:w} {timestamp:ti} {field_string}',
                field_patterns=['{:d}:{GravityValue:d} {GravityError:d}'])
```

```
grv1 v014-08-01T00:00:00.462000Z 01:022470 00
grv1 2014-08-01T00:00:01.460000Z 01:024628 00
grv1 2014-08-01T00:00:02.459000Z 01:026391 00
grv1 2014-08-01T00:00:03.461000Z 01:026275 00
grv1 2014-08-01T00:00:04.462000Z 01:025053 00
```

Parsing Data

- **RecordParser** - takes in strings, returns structured data

```
>>> parser = RecordParser(
                record_format='{data_id:w} {timestamp:ti} {field_string}',
                field_patterns=['{:d}:{GravityValue:d} {GravityError:d}'])

>>> parser.parse_record('grv1 2017-11-10T01:00:06.572Z 01:024557 00')
   { 'data_id': grv1
     'timestamp': 1510275606.572,
     'fields':{
       'GravityValue': 24557,
       'GravityError': 0
       }
   }
```

Parsing Data

- **RecordParser** - takes in strings, returns structured data

```
>>> transform = ParseTransform(
                record_format='{data_id:w} {timestamp:ti} {field_string}',
                field_patterns=['{:d}:{GravityValue:d} {GravityError:d}'])

>>> transform.transform('grv1 2017-11-10T01:00:06.572Z 01:024557 00')
   { 'data_id': grv1
     'timestamp': 1510275606.572,
     'fields':{
       'GravityValue': 24557,
       'GravityError': 0
       }
   }
```

Parsing Data

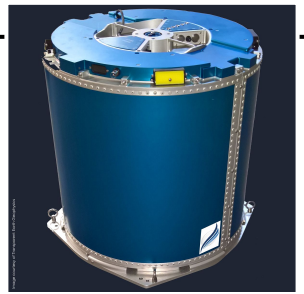
- **ParseTransform** - using stored device definitions

```
>>> transform = ParseTransform(
                definition_path='test/NBP1406/devices/nbp_devices.yaml')

>>> transform.transform('grv1 2017-11-10T01:00:06.572Z 01:024557 00')
   {
     'data_id': grv1
     'timestamp': 1510275606.572,
     'fields':{
       'GravityValue': 24557,
       'GravityError': 0
       }
    }
```

Parsing Data

- **Devices** and **Device types**

Device type: some category of instrument, e.g. a Seapath 330 or BGM-3 gravimeter.



Device: a specific instance of some device type, e.g. the BGM-3, serial number #BA-BGM3-001055, installed at station 367.5 of your ship.

Parsing Data

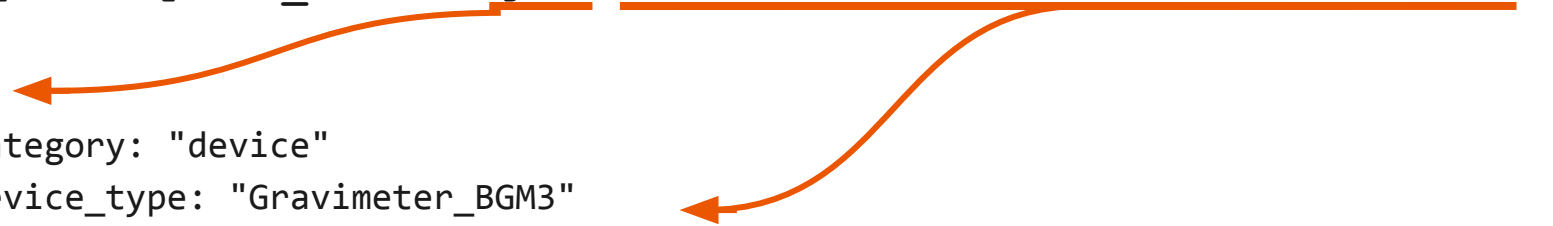
- **ParseTransform - device types**

```
Gravimeter_BGM3:
  category: "device_type"
  description: "Bell Aerospace BGM-3"
  format: "{CounterUnits:d}:{GravityValue:d} {GravityError:d}"
  fields:
    CounterUnits:
      description: "apparently a constant 01"
    GravityValue:
      units: "Flit Count"
      description: "mgal = flit count x 4.994072552 + bias"
    GravityError:
       description: "unknown semantics"
```

Parsing Data

- **ParseTransform** - devices

```
>>> parser.parse_record('grv1 2017-11-10T01:00:06.572Z 01:024557 00')

grv1:
   category: "device"
   device_type: "Gravimeter_BGM3"
   serial_number: "BA-BGM3-5003155"
   description: "Aft bulkhead 7, station 76.63; serves on /dev/ttys05"
   fields:
     GravityValue: "Grv1Value"
     GravityError: "Grv1Error"
```



Parsing Data

- **ParseTransform** - more complex devices

```
Seapath330:
  format:
    GGA: "${:2l}GGA,{GPSTime:f},{Latitude:nlat},{NorS:w},{Longitude:nlat},{E
    HDT: "${:2l}HDT,{HeadingTrue:f},T*{CheckSum:x}"
    VTG: "${:2l}VTG,{CourseTrue:of},T,{CourseMag:of},M,{SpeedKt:of},N,
    ZDA: "${:2l}ZDA,{GPSTime:f},{GPSDay:d},{GPSMonth:d},{GPSYear:d},{LocalHo
    PSXN20: "$PSXN,20,{HorizQual:d},{HeightQual:d},{HeadingQual:d},{RollPitc
    PSXN22: "$PSXN,22,{GyroCal:f},{GyroOffset:f}*{CheckSum:x}"
    PSXN23: "$PSXN,23,{Roll:f},{Pitch:f},{HeadingTrue:f},{Heave:f}*{CheckSum
```

Parsing Data

- **ParseTransform** - parsing formats

Under the hood, uses Python `parse` module

- recognizes all standard parse formats: f=float, x=hex, etc.
- has been extended in logger/utils/record_parser_formats.py to recognize many others: of=optional float, nlat=NMEA-format lat/lon, etc.

```
GGA: "${:2l}GGA,{GPSTime:f},{Latitude:nlat},{NorS:w},{Longitude:nlat},{EorW
VTG: "${:2l}VTG,{CourseTrue:of},T,{CourseMag:of},M,{SpeedKt:of},N,
```

Parsing Data

- Python `parse` module has many built-in data types

```
field_patterns=['{:d}:{GravityValue:d} {GravityError:d}']

l - Letters (ASCII)
w - Letters, numbers and underscore
W - Not letters, numbers and underscore
s - Whitespace
S - Non-whitespace
d - Digits (effectively integer numbers)
D - Non-digit
g - General number format (either d, f or e)
ti - ISO 8601 format date/time e.g. 1972-01-20T10:21:36Z
...
```

Parsing Data

- OpenRVDAS allows adding more parse formats

```
og - optional generalized number - also handles '#VALUE!' as None
ow - optional sequence of letters, numbers, underscores
nlat - NMEA-formatted latitude or longitude, converted to decimal degrees
nlat_dir - NMEA-formatted latitude or longitude along with hemisphere

...
```

Extra formats defined in `logger/utils/record_parser_formats.py`

Parsing Data

- **RegexParseTransform** - new contribution from CSIRO

Very similar to `ParseTransform` but, as it says on the tin, matches are specified by regex rather than `parse` format.

```
GGA: \$(?P<TalkerID>\w{2})GGA,\s*(?P<GPSTime>\-?\d*\.?\d*),\s*(?P<Latitude>\-?
HDT: \$(?P<TalkerID>\w{2})HDT,\s*(?P<HeadingTrue>\-?\d*\.?\d*),\s*T\*(?P<Check
VTG: \$(?P<TalkerID>\w{2})VTG,\s*(?P<CourseTrue>\-?\d*\.?\d*),\s*T,\s*(?P<Cour
```

Parsing Data - Metadata

- **Can tell `ParseTransform()` to compile and include metadata in DASRecords**

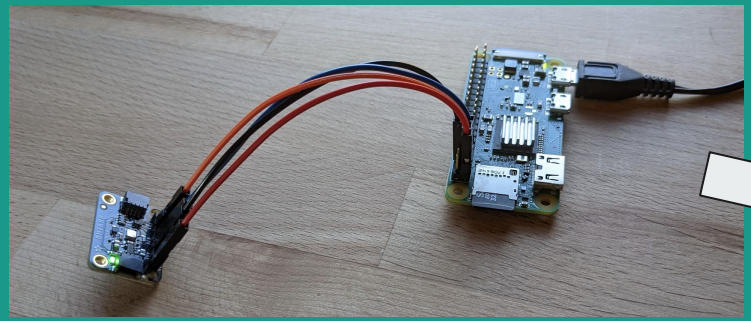
```
{'metadata': {'fields': {
  'TSG1Conductivity': {
    'description': 'Conductivity',
    'device': 'tsg1',
    'device_type': 'TSG_SBE45',
    'device_type_field': 'Conductivity',
    'units': 's/m'},
  'TSG1Salinity': {
    'description': 'Salinity',
    'device': 'tsg1',
    'device_type': 'TSG_SBE45',
    'device_type_field': 'Salinity',
    ...
```

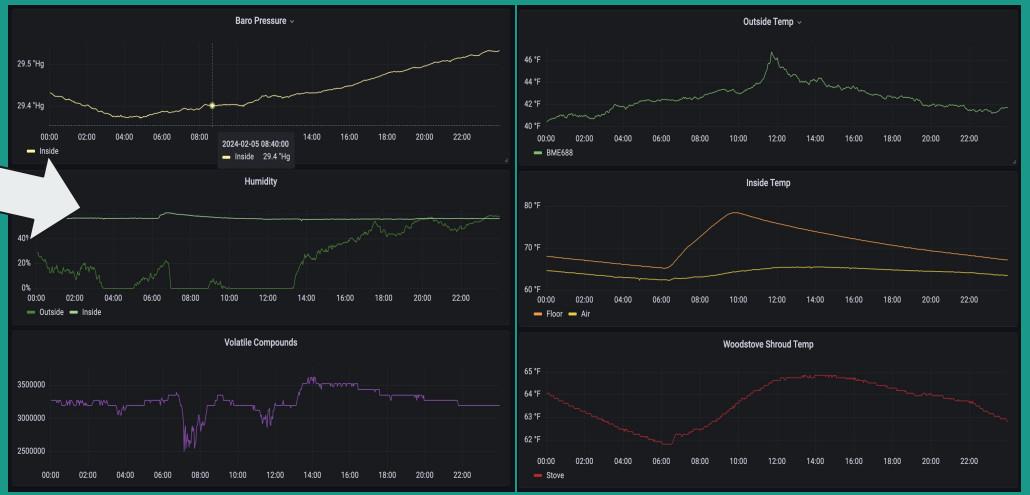
Parsing Data

- Once you've got the parsed numerical/text values from records, you can do all sorts of fun things with them.
- We'll talk about this soon.
- Full documentation at https://www.oceandatatools.org/openrvdas-docs/parsing/

That's all you need - if...

- ...you're running a small number of loggers.
- ...you never need to turn them off/on or change which ones are doing what.





OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

If you want to...

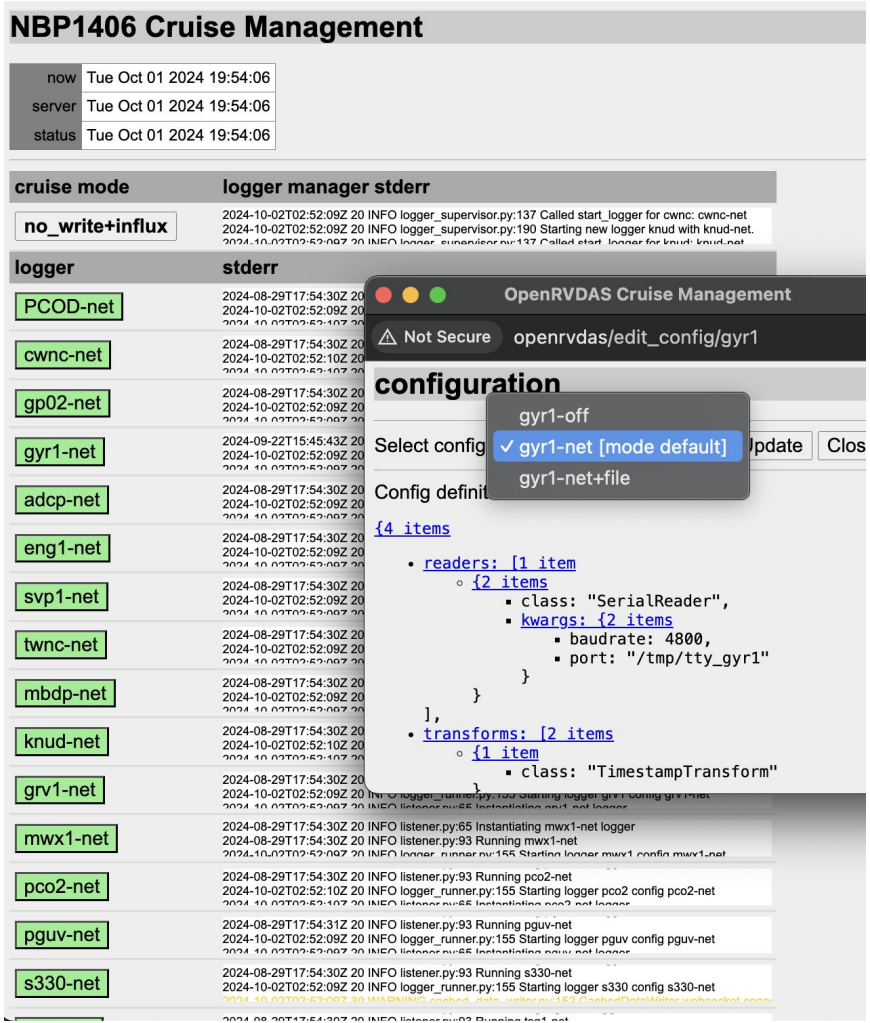
- ...run and monitor the status of many loggers
- ...change what they're doing based on whether you're in port, an EEZ, underway, running winches
- ...graphically monitor and change modes via web interface

Then you probably want the full installation.

```
OPENRVDAS_REPO=raw.githubusercontent.com/oceandatatools/openrvdas
BRANCH=master
curl -O -L https://$OPENRVDAS_REPO/$BRANCH/utils/install_openrvdas.sh
chmod +x install_openrvdas.sh
sudo ./install_openrvdas.sh
```

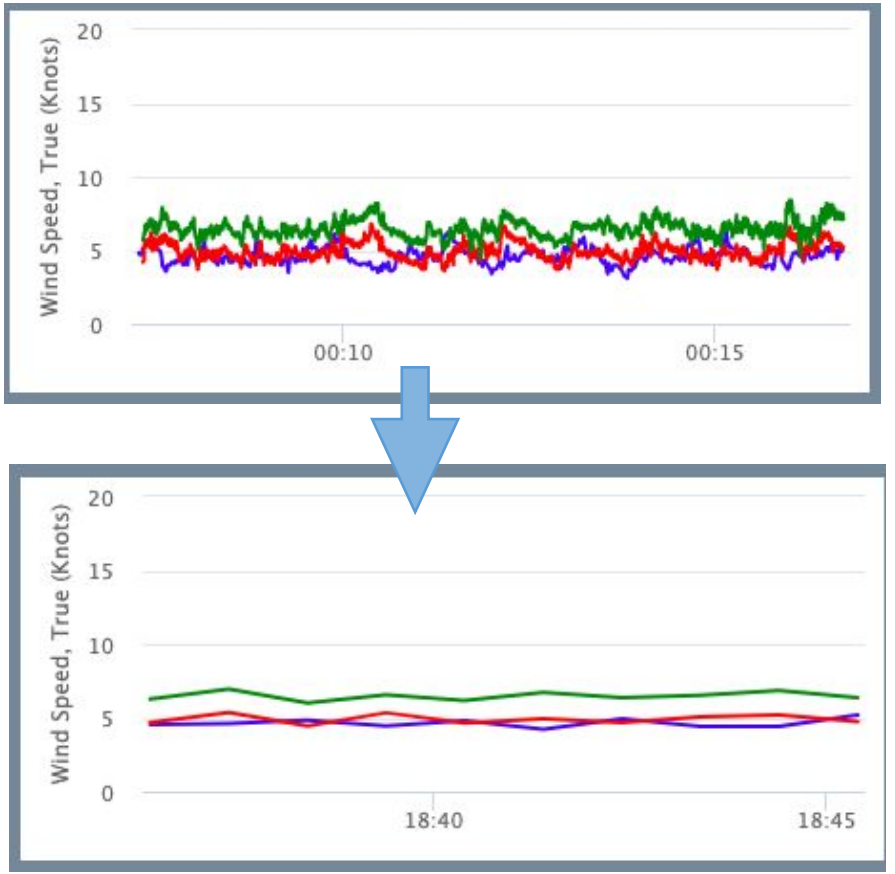
What it gets you

- **Web interface for controlling loggers.**
- Cached Data Server for integrating/manipulating multiple data sources.
- Database-backed persistent state management.



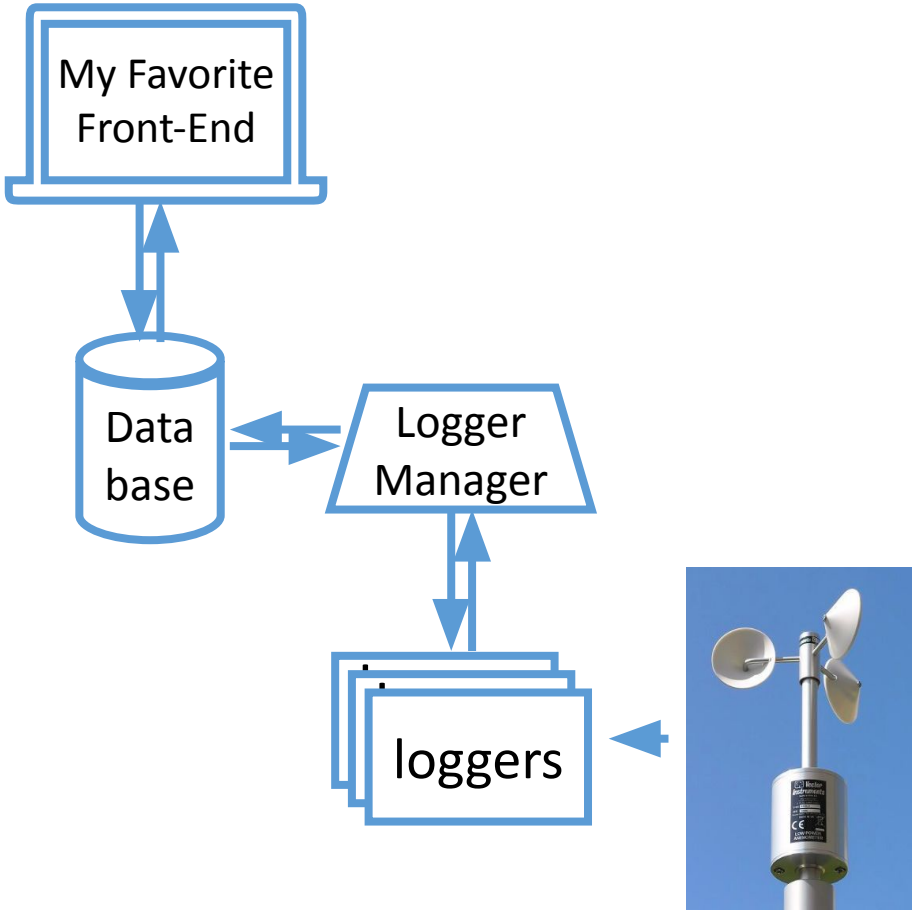
What it gets you

- Web interface for controlling loggers.
- **Cached Data Server for integrating/manipulating multiple data sources.**
- Database-backed persistent state management.



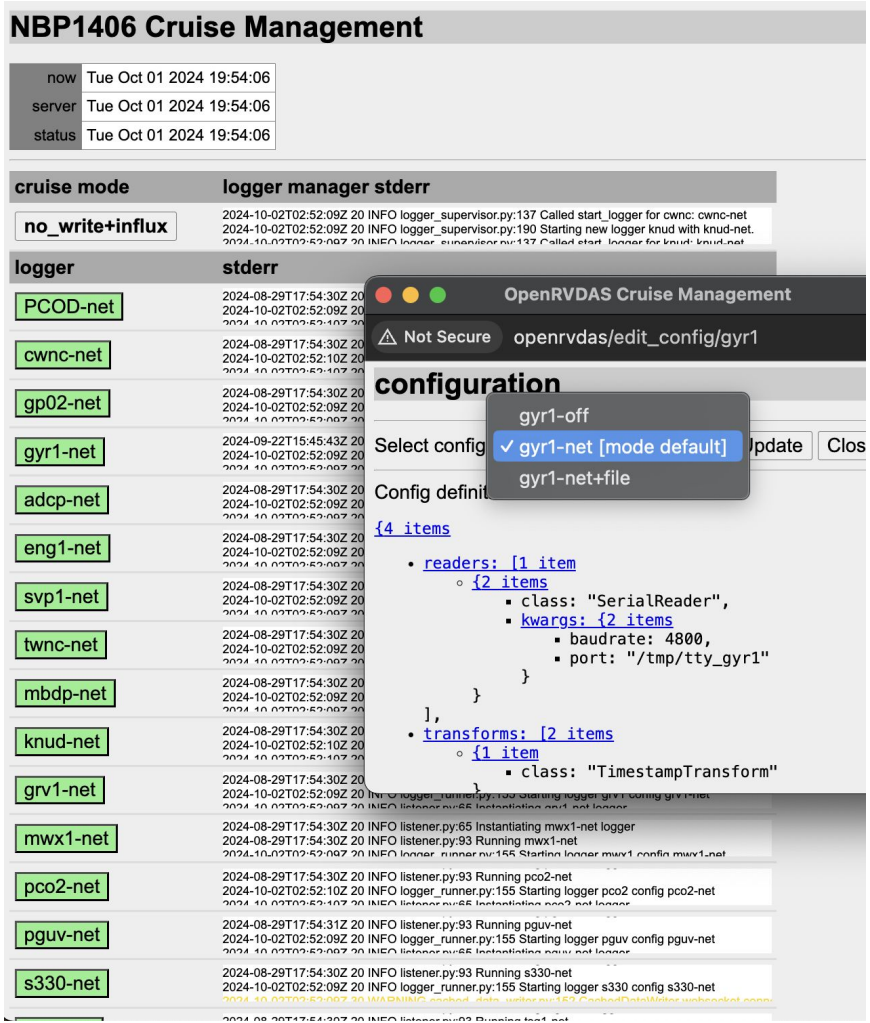
What it gets you

- Web interface for controlling loggers.
- Cached Data Server for integrating/manipulating multiple data sources.
- **Database-backed persistent state management.**



Controlling loggers

- **Default web interface.**
- Command line interface.
- RESTful API so you can roll your own.



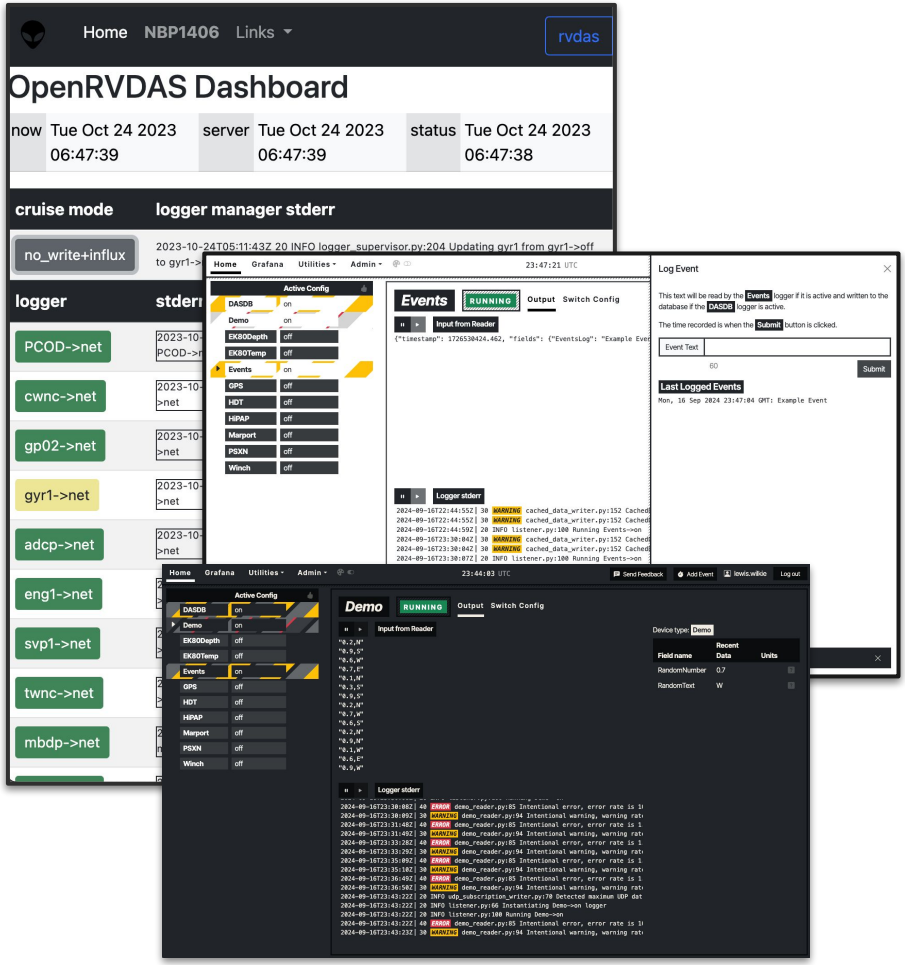
Controlling loggers

- Default web interface.
- **Command line interface.**
- RESTful API so you can roll your own.

```
openrvdas> server/logger_manager.py

command? load_configuration
NBP1406_cruise.yaml
command? get_modes
Available Modes: off, monitor, log, log+db

command? set_active_mode underway
command? get_loggers
Loggers: PCOD, cwnc, gp02, gyr1, adcp, eng
svp1, twnc, mbdp, knud, grv1, mwx1, pco2,
pguv, s330, tsg1, rtmp, hdas, tsg2, seap,
true_wind, subsample

command? get_logger_configs s330
Configs for s330: s330->off, s330->net,
s330->file/net, s330->file/net/db

command? set_active_logger_config s330
s330->off
command? quit
```

Controlling loggers

- Default web interface.
- Command line interface.
- **RESTful API so you can roll your own.**
  - Django or SQLite



A Peek Behind the Scenes

- Installation script sets up files in `/etc/supervisor/conf.d` to run a bunch of servers:

```
rvdas@openrvdas:/opt/openrvdas$ supervisorctl status
logger_manager           RUNNING   pid 2728950, uptime 116 days, 7:28:34
cached_data_server       RUNNING   pid 2728944, uptime 116 days, 7:28:34
django:nginx             RUNNING   pid 3079419, uptime 81 days, 2:10:23
django:uwsgi             RUNNING   pid 3079420, uptime 81 days, 2:10:23
simulate:simulate_nbp    RUNNING   pid 2728953, uptime 116 days, 7:28:34
```

Setting up a Cruise

- Build a cruise configuration
  - **configurations**
  - loggers
  - modes

```
seap-net:
  readers:
  - class: SerialReader
    kwargs:
      baudrate: 9600
      port: /tmp/tty_seap
  transforms:
  - class: TimestampTransform
  - class: PrefixTransform
    kwargs:
      prefix: seap
  writers:
  - class: UDPWriter
    kwargs:
      port: 6224
```

Setting up a Cruise

- Build a cruise configuration
  - **configurations**
  - loggers
  - modes

```
seap-file+net:
  readers:
  - class: SerialReader
    kwargs:
      baudrate: 9600
      port: /tmp/tty_seap
  transforms:
  - class: TimestampTransform
  - class: PrefixTransform
    kwargs:
      prefix: seap
  writers:
  - class: LogfileWriter
    kwargs:
      filebase: /var/data/raw/seap
  - class: UDPWriter
    kwargs:
      port: 6224
```

Setting up a Cruise

- Build a cruise configuration
  - **configurations**
  - loggers
  - modes

```
seap-off:{}
```

Setting up a Cruise

- Build a cruise configuration
  - **configurations**
  - loggers
  - modes

```
configs:
  seap-off:
    ...
  seap-net:
    ...
  seap-file+net:
    ...

  knud-off:
    ...
  knud-net:
    ...
  knud-file+net:
    ...

  rtmp-off:
    ...
```

Setting up a Cruise

- Build a cruise configuration
  - configurations
  - **loggers**
  - modes

```
loggers:
  seap:
    configs:
    - seap-off
    - seap-net
    - seap-net+file
  knud:
    configs:
    - knud-off
    - knud-net
    - knud-net+file
  rtmp:
    configs:
    - rtmp-off
    - rtmp-net
    - rtmp-net+file
```

Setting up a Cruise

- Build a cruise configuration
  - configurations
  - loggers
  - **modes**

```
modes:
  'off':
    seap: seap-off
    knud: knud-off
    rtmp: rtmp-off
    ...
  port:
    seap: seap-net
    knud: knud-off
    rtmp: rtmp-net
    ...
  eez:
    seap: seap-net
    knud: knud-net
    rtmp: rtmp-net
    ...
  underway:
    seap: seap-net+file
    knud: knud-net+file
    rtmp: rtmp-net+file
    ...
```

Setting up a Cruise - Pain Points

- Full cruise configuration files can be mind-numbingly long
- Creating/Editing/Modifying them can be error prone

We'll talk later about some tools and strategies that help

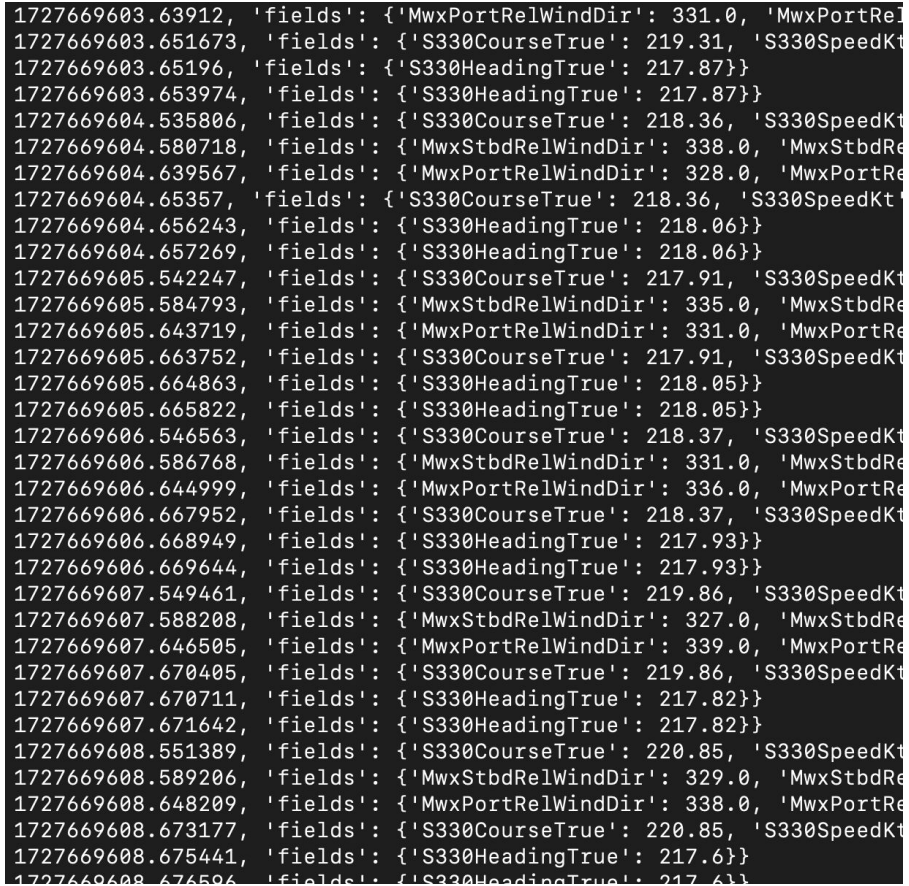
But first, how to do some of the fun and powerful stuff!

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

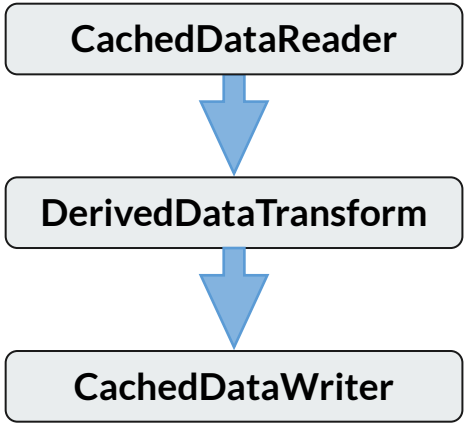
Cached Data Server

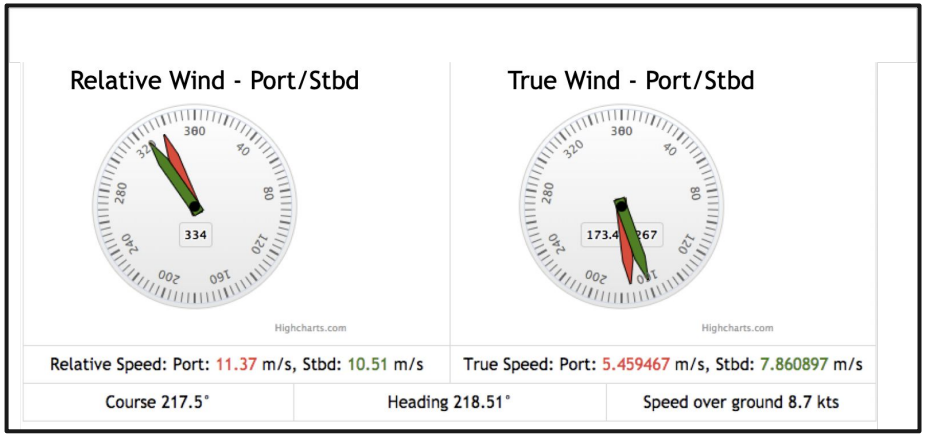
- An OpenRVDAS-specific pub-sub server in the **`servers/`** subdirectory.
- Installed and run as part of standard installation.
- Used to manage status messages, but can also use for sensor data.



Cached Data Server

- Loggers can write to it via **`CachedDataWriter`**
- Loggers can read from it via **`CachedDataReader`**
- Use for derived values
  - **read** from server
  - **compute**
  - **write** results back to server or send elsewhere





CachedDataReader

- Connects to CDS via websocket.
- Subscribes to values of interest, returns them when new values show up.

```
- class: CachedDataReader
  kwargs:
    data_server: localhost:8766
    subscription:
      fields:
        S330CourseTrue:
          seconds: 0
        S330HeadingTrue:
          seconds: 0
        S330SpeedKt:
          seconds: 0
        MwxRelWindDir:
          seconds: 0
        MwxRelWindSpeed:
          seconds: 0
```

CachedDataReader

- Connects to CDS via websocket,
- Subscribes to values of interest, returns them when new values show up.

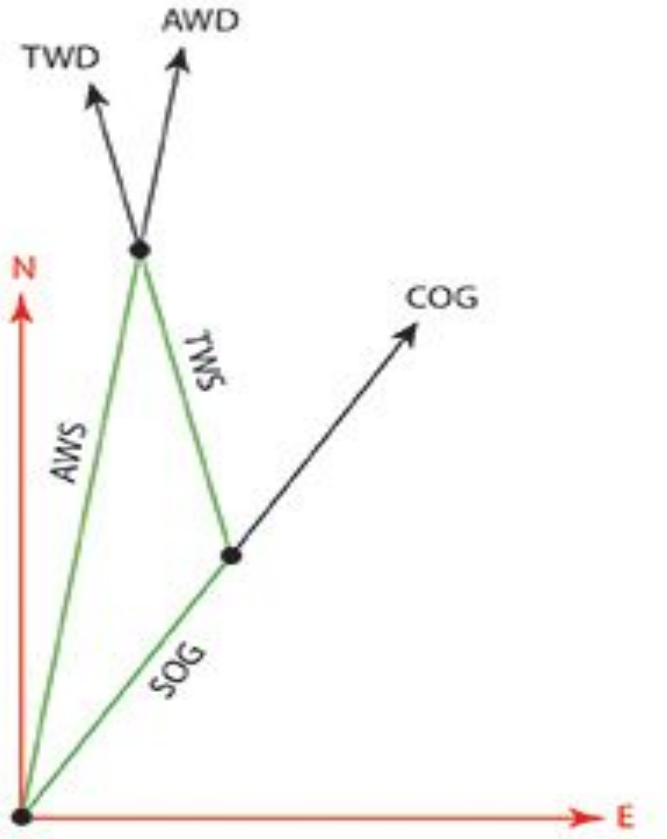
```
- class: CachedDataReader
  kwargs:
    data_server: localhost:8766
    subscription:
      fields:
        S330CourseTrue:
          seconds: 0
```

Acceptable values for 'seconds':
0 - provide only new values that arrive after subscription
-1 - provide the most recent value, and then all future new ones
num - provide num seconds of back data, then all future new ones

If 'seconds' is missing, use '0' as the default.

True Winds

- Depends on
  - heading
  - course over ground
  - speed over ground
  - relative wind speed
  - relative wind dir



True Winds

- Depends on
  - heading
  - course over ground
  - speed over ground
  - relative wind speed
  - relative wind dir

```
true_wind-on:
  readers:
  - class: CachedDataReader
    kwargs:
      data_server: localhost:8766
      subscription:
        fields:
          S330CourseTrue:
            seconds: 0
          S330HeadingTrue:
            seconds: 0
          S330SpeedKt:
            seconds: 0
          MwxRelWindDir:
            seconds: 0
          MwxRelWindSpeed:
            seconds: 0
  transforms:
  - class: TrueWindsTransform
    kwargs:
      ...
```

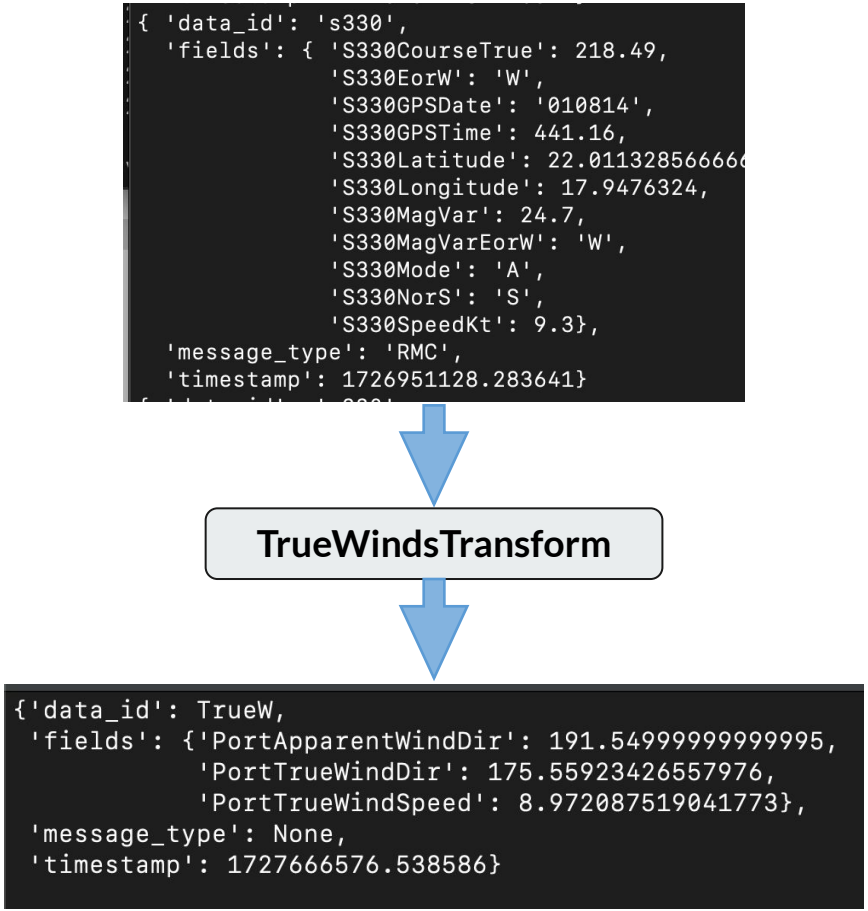
True Winds

- Depends on
  - heading
  - course over ground
  - speed over ground
  - relative wind speed
  - relative wind dir
- Send records to TrueWindsTransform
- Transform calls routines in `logger/utils/truewinds`

```
class: TrueWindsTransform
kwargs:
  heading_field: Gyr1HeadingTrue
  course_field: S330CourseTrue
  speed_field: S330SpeedKt
  wind_speed_field: MwxRelWindSpeed
  wind_dir_field: MwxRelWindDir

  convert_speed_factor: 0.5144

  true_speed_name: TrueWindSpeed
  true_dir_name: TrueWindDir
  apparent_dir_name: ApparentWindDir

  update_on_fields:
  - MwxRelWindDir
  max_field_age:
    S330CourseTrue: 15
    S330HeadingTrue: 15
    S330SpeedKt: 15
    MwxRelWindDir: 15
    MwxRelWindSpeed: 15
```

True Winds

- **`TrueWindsTransform`** takes **`DASRecords`** and looks for the values it needs in them.
- Outputs `None` if it doesn't have all the values it needs.
- Outputs a **`DASRecord`** if it *does* find all the values it needs.
- Caches values for next time.



Snapshots

- Much of the power of the architecture comes from the open-ended definition of `transforms` and `writers`.
- You pass a record to a **`transform`** and it gives you a record (possibly `'None'`) back.

```
readers
- class: CachedDataReader
  kwargs:
    ...

transforms:
- class: InterpolationTransform
  kwargs:
    ...

writers:
- class: CachedDataWriter
  kwargs:
    ...
```

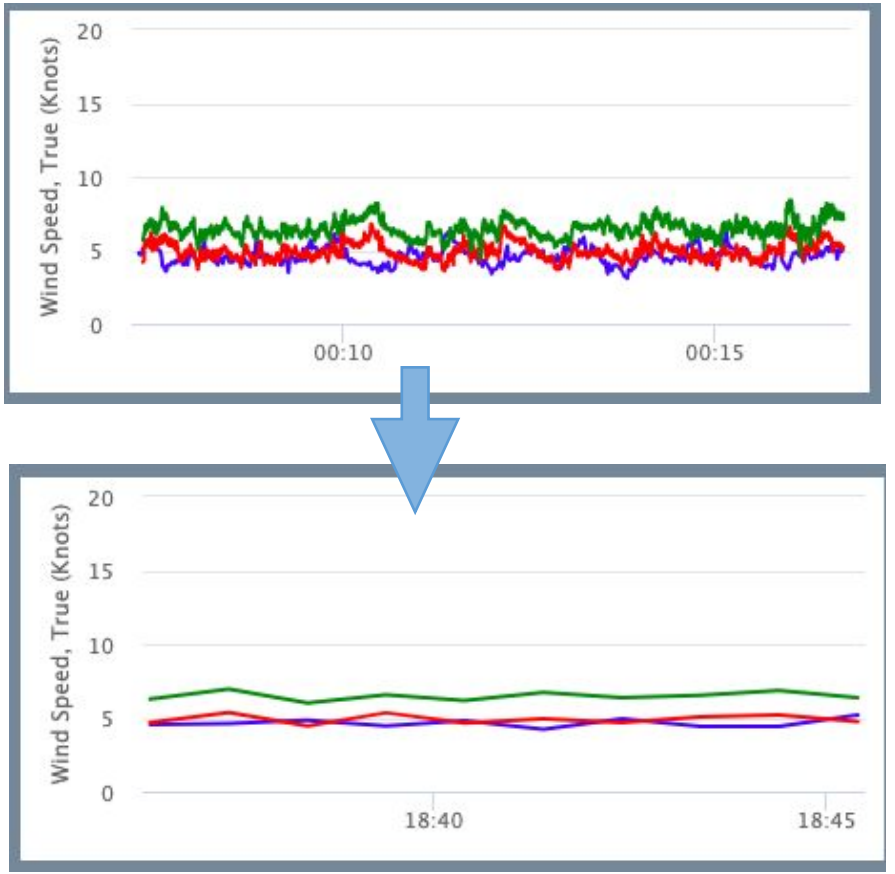
Snapshots

- Use this to aggregate values until ready to produce an output.
- E.g. when computing running averages.

```
- class: InterpolationTransform
  module:
   logger.transforms.interpolation_t
  kwargs:
    interval: 30
    window: 30
    field_spec:
      AvgRTMPTemp:
        source: RTMPTemp
        algorithm:
          type: boxcar_average
          window: 30
      AvgTrueWindDir:
        source: TrueWindDir
        algorithm:
          type: polar_average
          window: 30
    ....
```

Snapshots

- Use this to aggregate values until ready to produce an output.
- E.g. when computing running averages.



Quality Control using the CDS

```
readers:
- class: CachedDataReader
  kwargs:
    subscription:
      fields:
       TWNCTension: {seconds: 0}, TWNCPayout: {seconds: 0}
transforms:
- class: QCFilterTransform
  kwargs:
    bounds: TWNCTension:-150:10000,TWNCPayout:-60:175000
writers:
- class: AlertWriter
- class: LogfileWriter
  kwargs:
    filebase: /var/log/openrvdas/winch_errors
```

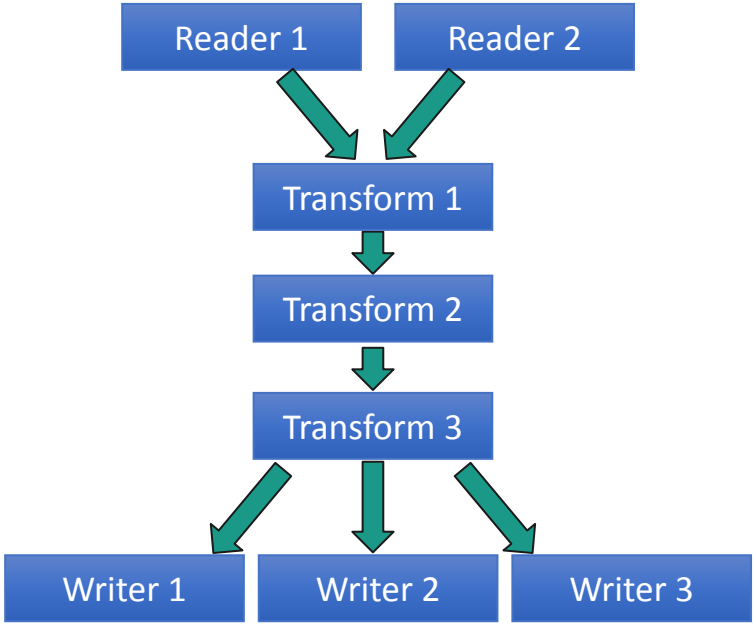
Quality Control using the CDS

```
readers:
- class: CachedDataReader
  kwargs:
    subscription:
      fields:
       Gyr1HeadingTrue: {seconds: 0}
writers:
- class: TimeoutWriter
  kwargs:
    timeout: 60
    message: No Gyro data received for 60 seconds
    resume message: Gyro data has resumed
    writer:
    - class: LogfileWriter
      kwargs:
        filebase: /var/log/openrvdas/winch_errors
```

A writer that takes other writers as an argument?!?

ComposedWriters

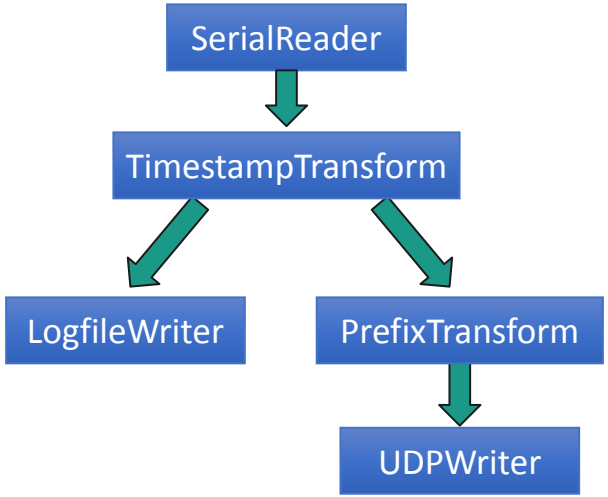
- Basic architecture is an "hourglass"
  - Readers in parallel
  - Transforms in series
  - Writers in parallel
- Means that all data go through same set of transforms.



ComposedWriters

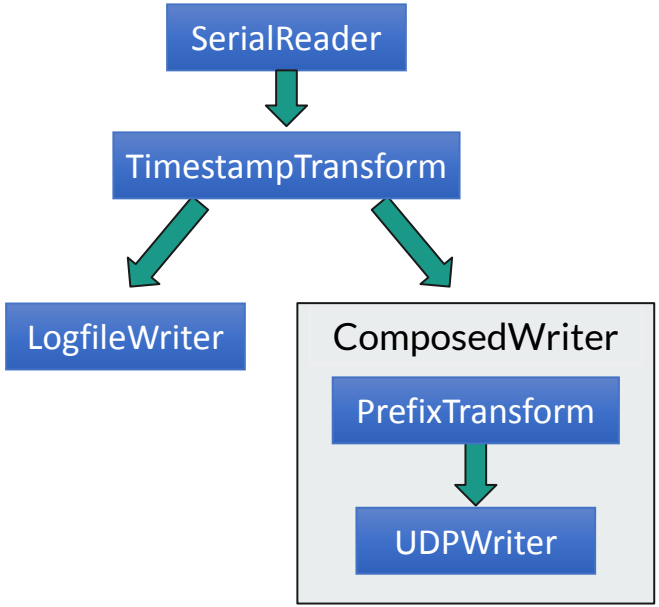
- But sometimes you want to do two incompatible transforms on the same data.

- E.g. save raw to file, but add instrument prefix to data sent out over network.



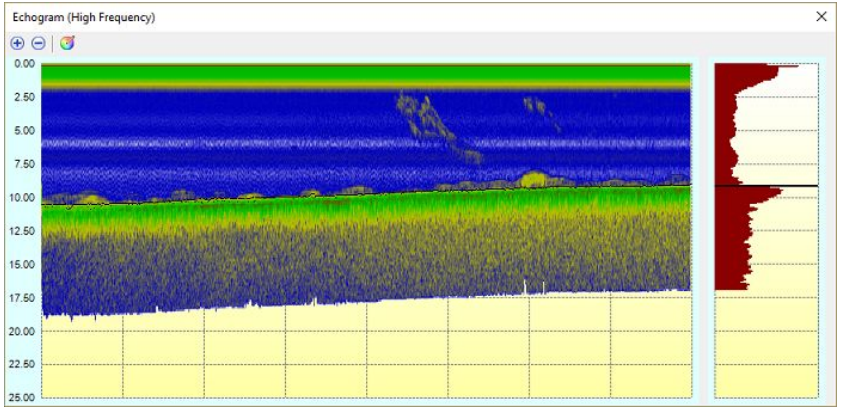
ComposedWriters

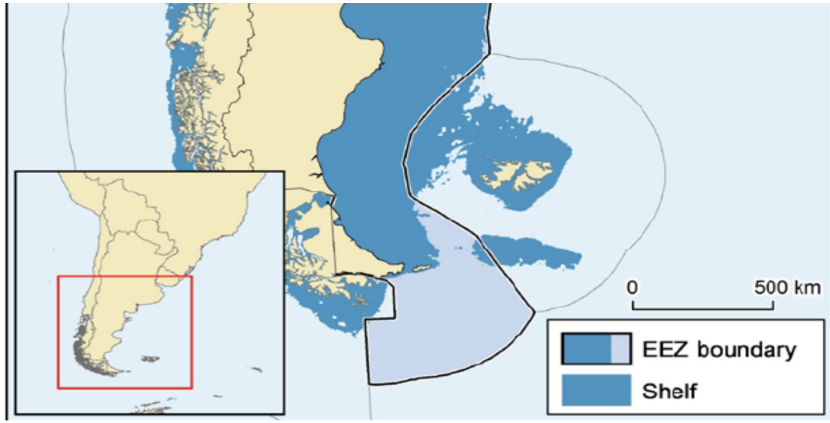
- Allow you to package up
  - one or more transforms applied to
  - one or more specific writers.



Geofencing

- USAP can't record data inside Argentine EEZ unless and Argentinian observer is on board.
- Need to manually switch from `no-write` to `write` when crossing EEZ boundary.
- Can we do it automatically?





Geofencing

- Two new modules:
  - **GeofenceTransform**
  - LoggerManagerWriter

```
- class: GeofenceTransform
  module:
    logger.transforms.geofence_transfor
  kwargs:
    latitude_field_name: s330Latitude
    longitude_field_name: s330Longitude
    boundary_file_name: /tmp/eez.gml
    leaving_boundary_message:
      set_active_mode write
    entering_boundary_message:
      set_active_mode no_write
```

Geofencing

- Two new modules:
  - `GeofenceTransform`
  - **`LoggerManagerWriter`**

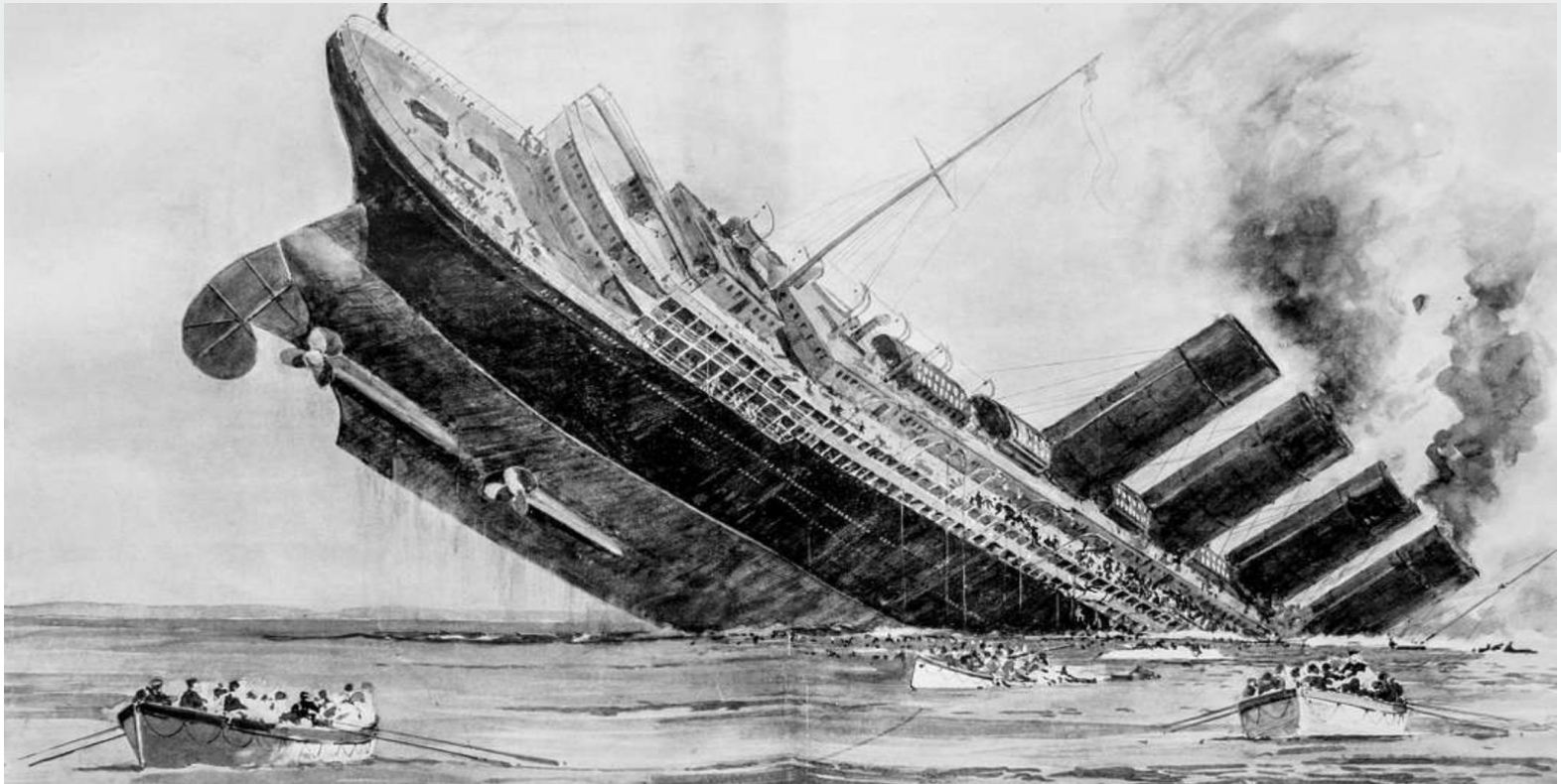
```
- class: GeofenceTransform
  module:
    logger.transforms.geofence_transfor
  kwargs:
    latitude_field_name: s330Latitude
    longitude_field_name: s330Longitude
    boundary_file_name: /tmp/eez.gml
    leaving_boundary_message:
      set_active_mode write
    entering_boundary_message:
      set_active_mode no_write

 writers:
 - class: LoggerManagerWriter
   module:
     logger.writers.logger_manager_writ
   kwargs:
     database: django
     allowed_prefixes:
     - 'set_active_mode '
```

Fun and Games with LoggerManagerWriter

- Can use to change any system state based on data values

```
transforms:
- class: QCFilterTransform
  kwargs:
    bounds: 'RTMPTemp:-10:40'
    message: 'set_active_logger_config rtmp rtmp-off'
writers:
- class: LoggerManagerWriter
  kwargs:
  ...
```

Okay, enough fun - where are the pain points?



OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

Pain Points

- Creating cruise configuration files.
- Creating device definitions.

Pain Points

- **Creating cruise configuration files.**
- Creating device definitions.

Cruise configuration files contain:

- Definition of configurations.
- Which config is associated with which logger.
- Which config of which logger is associated with each cruise mode.

Pain Points

- **Creating cruise configuration files.**

- Creating device definitions.

- Each logger may have 2-4 configurations.
- Each configuration may be 10-40 lines long.
- A typical ship may have 10-40 sensors.

Typical cruise configuration can be thousands of lines long.

Painful and error-prone to create and maintain manually.

Pain Points

- **Creating cruise configuration files.**
- Creating device definitions.

When you add a new logger you need to modify three sections:

`logger` - add the name, and names of each config associated with it.

`configs` - definitions of the named configs themselves.

`modes` - which config should be active for which logger in which mode.

Cruise Creation Tools

- Use YAML macros to reduce duplication

```
gnss->file: &gnss_base
   readers: {class: UDPReader, kwargs: {port: 9117}}
   transforms:
     - class: SplitTransform
       module: logger.transforms.split_transform
     - class: TimestampTransform
   writers:
     - class: LogfileWriter
       kwargs:
         filebase: data/raw/gnss
```

Cruise Creation Tools

- Use YAML macros to reduce duplication

```
gnss->file+net:
    <<: *gnss_base
    writers:
    - class: LogfileWriter
      kwargs:
        filebase: data/raw/gnss
    - class: UDPWriter
      kwargs:
        port: 6224
```

Cruise Creation Tools

- Python scripts such as **local/usap/create_cruise.py**

```
local/usap/create_cruise.py \
    test/NBP1406/NBP1406_port_defs.yaml \
    > test/NBP1406/NBP1406_cruise.yaml
```

(See **test/NBP1406/NBP1406_port_defs.yaml** for port definitions)

Cruise Creation Tools

- Or combination: script + macros, as Florida does:

  - Template **`logger_config.template`** in repo **`OceanDataTools/FIO-ODT-Configs`**

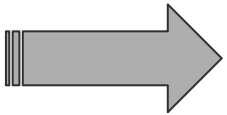
  - Filled out by **`build_openrvdas_config.sh`** script

Cruise Creation Tools

- Jinja-based templating, courtesy of Ella Pietraroia @ CSIRO

Code in **utils/jinja_config_creator**.

```
loggers:
  {% for device in devices %}
  {{ device }}:
    configs:
    - {{ device }}->off
    - {{ device }}->file/net
    - {{ device }}->file/net/db
  {% endfor %}
```



```
loggers:
  PCOD:
    configs:
    - PCOD->off
    - PCOD->net
    - PCOD->net/file
  cwnc:
    configs:
    - cwnc->off
    - cwnc->net
    - cwnc->net/file
  gp02:
    configs:
    - gp02->off
    - gp02->net
    - gp02->net/file
  gyr1:
    configs:
    - gyr1->off
```

CORIOLIX

- Full shipboard and ship-to-shore datapresence system developed at OSU by Chris Romsos, Jasmine Nahorniak et al.

  - Uses OpenRVDAS for core data collection.
  - Wraps a lot of handy cruise management tools around it.
  - E.g.: device database management
    - all devices and feeds managed in database.
    - update device in db using GUI
    - scripts propagate that update into a new cruise configuration script.

Pain Points

- Creating cruise configuration files.
- **Creating device definitions.**

Reading from device generally isn't a problem, so long as it communicates via

- UDP
- serial port
- MQTT
- Modbus
- TCP
- websockets

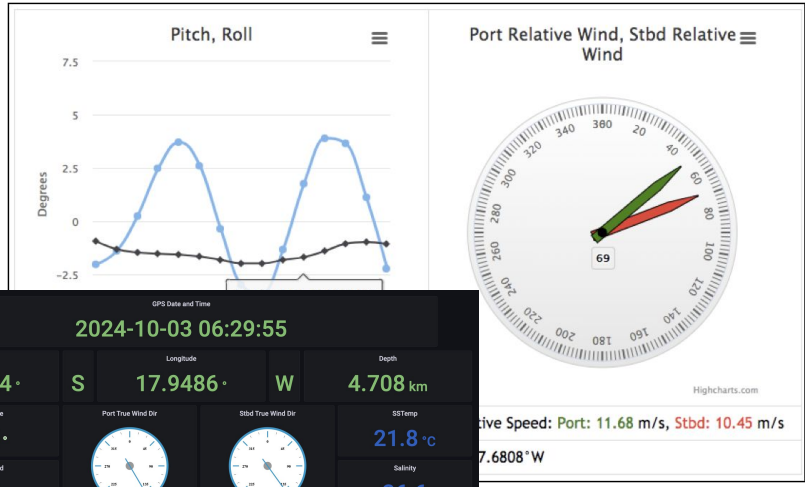
The challenge is often *parsing* the raw records that come from the device.

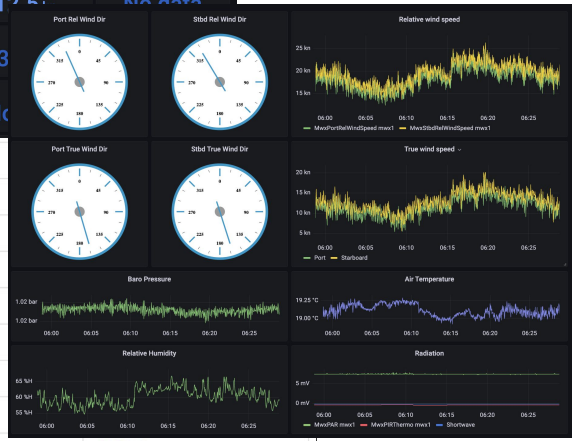
OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

Okay, a *little* more fun: Displaying Data

- Native display widgets + Highcharts
- InfluxDB + Grafana



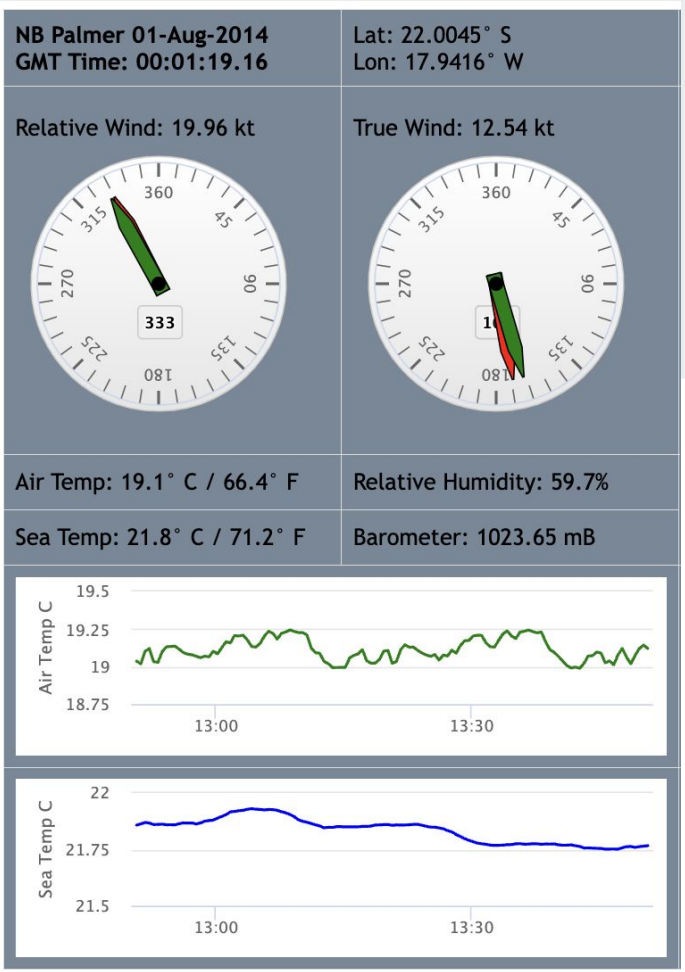


Port Wind Dir	339° M		
Stbd Wind Dir	334° M		
Lat	22.0099° S	Lon	
Heading	217.93° T	SOG	
Pitch	-1.3°	Roll	
RTemp	21.7942° C	Air Temp	
Eng Voltage	12.26V	Eng Case Temp	
Aq Room Flow	507.5	Seis Air Pressure	
Helo Deck Flow	565.7	PIR Res	
Hydro Lab Flow	233.8	PIR Out	
Freezer 1	-11.5° C	Freezer 2	-7.5° C

Display Widgets

- Original OpenRVDAS display method

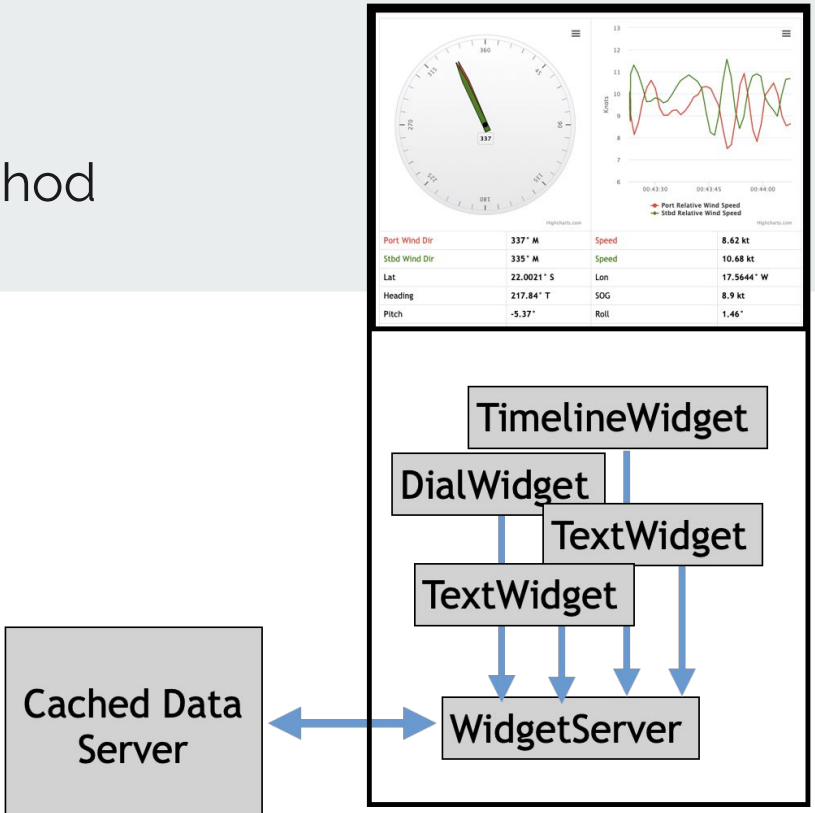
- Based on native JavaScript + Highcharts (or open source D3).
- Allows embedding displays in any web page.
- Note: Highcharts is proprietary commercial product, free to use for universities and non-profits.



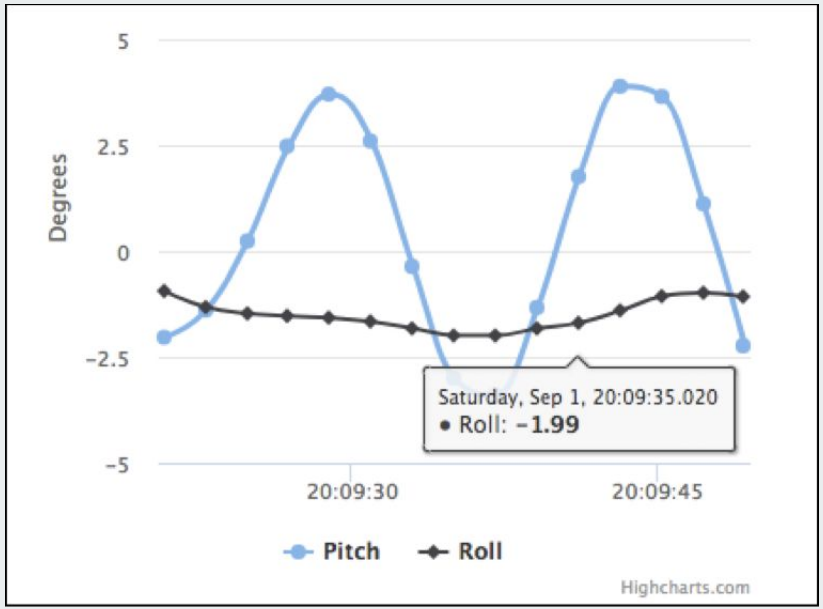
Display Widgets

- Original OpenRVDAS display method

- On page: define a div container where you want widget.
- In JavaScript
  - define widget and its fields and parameters.
  - pass all widgets to a WidgetServer.
- API makes creation/integration of new widget types easy.



Display Widgets




```
<div id="pitch-roll"></div>

<script type="text/javascript">
  var line_fields = {
    S330Pitch: {name: "Pitch",
                seconds: 30},
    S330Roll: {name: "Roll",
               seconds: 30}
  };

  var pr_widget = new
    TimelineWidget('pitch-roll',
                   line_fields,
                   'Degrees'));

  var widget_server = new
    WidgetServer([pr_widget],
                 'localhost:8766');
  widget_server.serve();
</script>
```

InfluxDB/Grafana

- Now preferred display route:
  - open source
  - large community of users and maintainers (who aren't us!).
- Two separate packages:
  - **InfluxDB** - time series database we write to.
  - **Grafana** - analytics, monitoring and visualization system.

InfluxDB/Grafana

- Getting data into InfluxDB works just the way you'd think...

```
netreader-on+influx:
    readers:
    - class: UDPReader
      kwargs:
        port: 6224

    transforms:
    - class: ParseTransform
      kwargs:
        definition_path: test/NBP1406/d

    writers:
    - class: CachedDataWriter
      kwargs:
        data server: localhost:8766
    - class: InfluxDBWriter
      kwargs:
        bucket_name: openrvdas
```

Installing InfluxDB/Grafana

- Script: **`utils/install_influxdb.sh`**

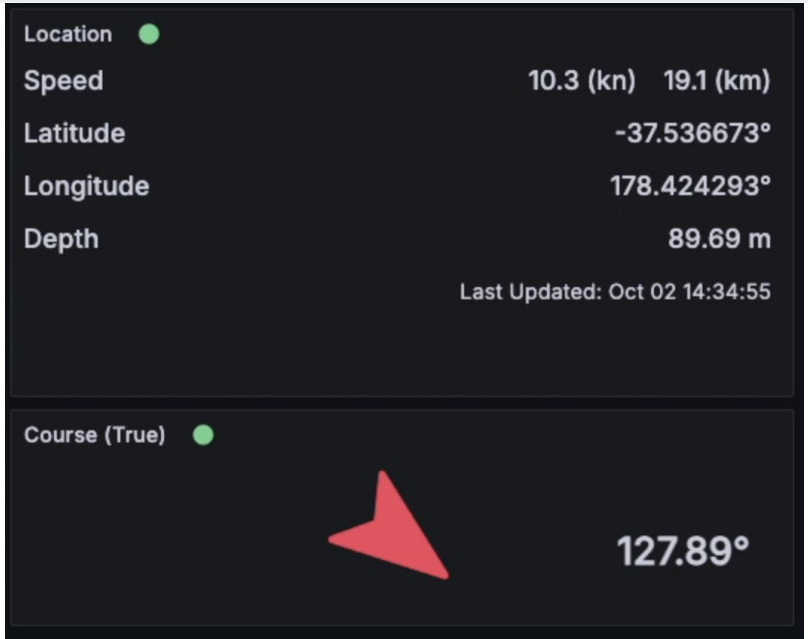
- Should be run as the user who will be running OpenRVDAS (e.g. `'rvdas'`).

```
(venv) rvdas@openrvdas:/opt/openrvdas$ utils/install_influxdb.sh
Recognizing OS type as Ubuntu
Reading pre-saved defaults from .install_influxdb_preferences
##########################################################
InfluxDB/Grafana/Telegraf configuration script
##########################################################
Path to openrvdas directory? (/opt)
```

Hot off the presses: CDS⇒Grafana displays

Another NIWA contribution: get Grafana to use CDS as datasource

- Traditionally, OpenRVDAS writes to InfluxDB, Grafana reads from InfluxDB.
- For time-critical displays, can have Grafana directly use CDS as source.
- Source and instructions in contrib repo



Other fun: InfluxDB for Quality Control

InfluxDB *Tasks* can make queries, produce conditional outputs

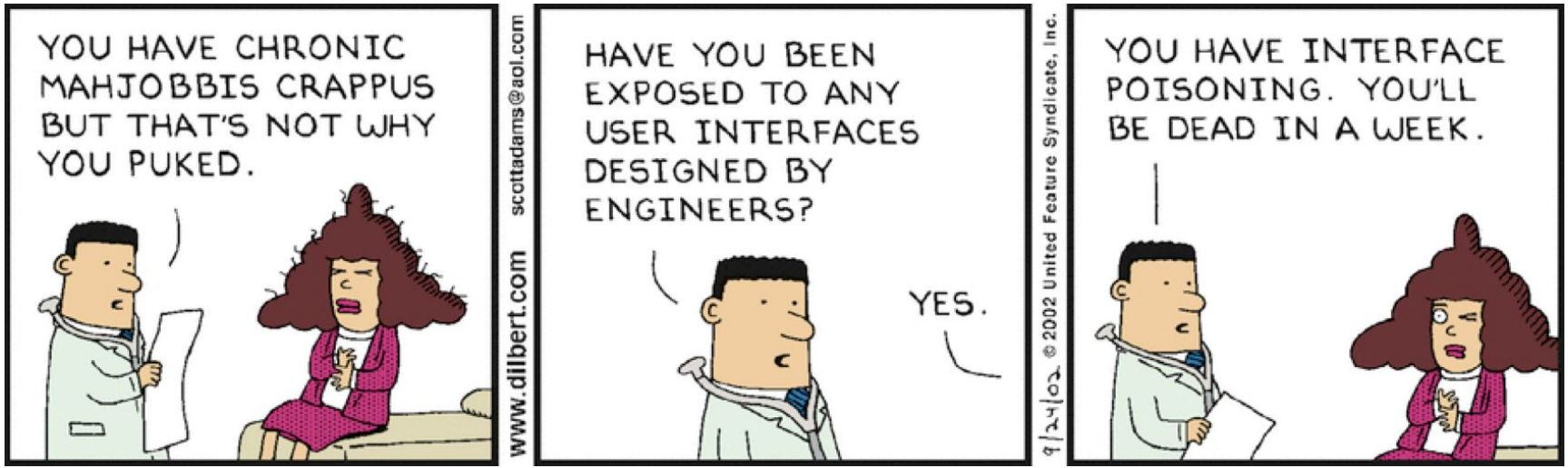
```
from(bucket: "openrvdas")
   |> range(start: -5m)    |> filter(
       fn: (r) => r["_measurement"] == "gyro_furuno_heading" or r["_measurement"] == "mru_trimble_bx992", )
   |> filter( fn: (r) => r["_field"] == "Gyro_HeadingTrue" or r["_field"] == "Trimble_BX992_HeadingTrue",)
   |> aggregateWindow(every: 2s, fn: last, createEmpty: true)
   |> map(
     fn: (r) => ({
       _time: r._time, _field: r._field,
       _measurement: "qa_alerts",
       _value: if not exists r._value then
                 -1
               else if r._value >= 0 and r._value < 360
                 1
               else
                 0,
       sensor: r.sensor, }),    )
   |> to(bucket: "qa_flags")
```

Other storage and display paths

- **`TimescaledbWriter`**, contributed by Lewis Wilke (NIWA) in `github.com/OceanDataTools/openrvdas_contrib`
- **`RedisWriter`**
- **`PostgreWriter`** and **`CORIOLIXWriter`**, contributed by Jasmine Nahorniak (OSU) in `github.com/OceanDataTools/openrvdas_rcrv`

Other interfaces

- RESTful API, SQLite API make it easy to build your own interface for controlling and monitoring loggers.



Other interfaces

- RESTful API, SQLite API make it easy to build your own interface for controlling and monitoring loggers.
- Kevin Pedigo (***USAP***) SQLite-based GUI at **`https://github.com/OceanDataTools/sqlite_gui`**
- Lewis Wilkie (***NIWA***) RESTful GUIs

Django administration

Site administration

AUTH TOKEN		
Tokens	+ Add	Change

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Permissions	+ Add	Change
Users	+ Add	Change

DJANGO_GUI		
Cruises	+ Add	Change
Last updates	+ Add	Change
Log messages	+ Add	Change
Logger config states	+ Add	Change
Logger configs	+ Add	Change
Loggers	+ Add	Change
Modes	+ Add	Change

Recent actions

My actions

openrvdas
User

Shortcuts to API functions

Create Permissions

/create-permissions/

This endpoint creates django global UI permissions. Permissions can be assigned and edited via the django admin interface.

Create Permissions

Create Permission Groups

/create-permission-groups/

This endpoint creates django permission groups for admins and viewers with default global UI permissions. Groups can be assigned and edited via the django admin interface.

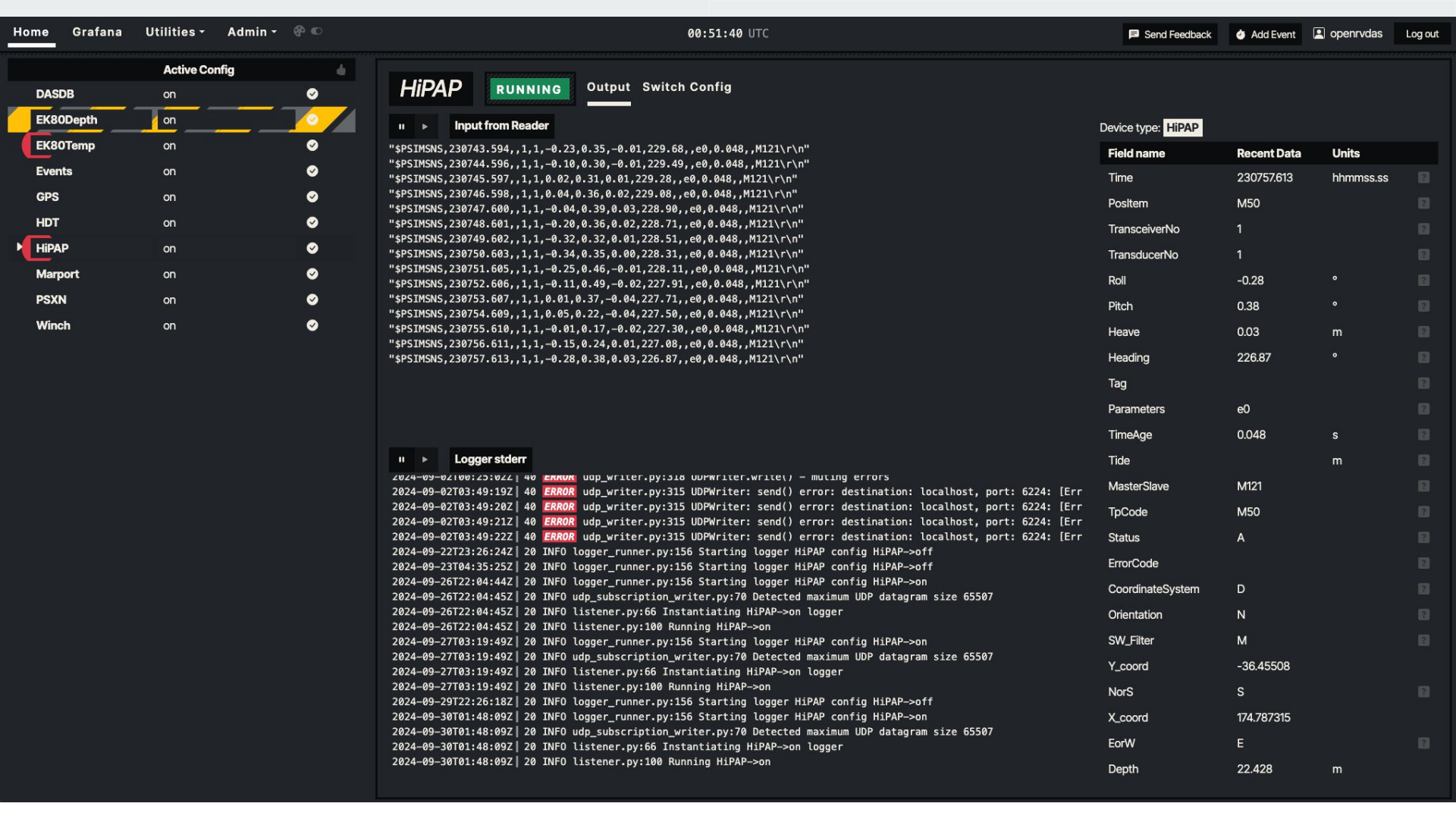
Create Permission Groups

Load Persistent Loggers

/load-persistent-loggers/

This endpoint reads logger **.yaml** files from **/local/logger_configs** on the OpenRVDAS server and creates persistent loggers which are not linked to a cruise file. Persistent Loggers can be edited in the UI without reloading a cruise or logger config file.

Load Persistent Loggers


Home
Grafana
Utilities
Admin
00:51:40 UTC
Send Feedback
Add Event
openrvdas
Log out
Active Config
DASDB on
EK80Depth on
EK80Temp on
Events on
GPS on
HDT on
HiPAP on
Marport on
PSXN on
Winch on
HiPAP
RUNNING
Output
Switch Config
Input from Reader
"$PSIMSNS,230743.594,,1,1,-0.23,0.35,-0.01,229.68,,e0,0.048,,M121\r\n"
"$PSIMSNS,230744.596,,1,1,-0.10,0.30,-0.01,229.49,,e0,0.048,,M121\r\n"
"$PSIMSNS,230745.597,,1,1,0.02,0.31,0.01,229.28,,e0,0.048,,M121\r\n"
"$PSIMSNS,230746.598,,1,1,0.04,0.36,0.02,229.08,,e0,0.048,,M121\r\n"
"$PSIMSNS,230747.600,,1,1,-0.04,0.39,0.03,228.90,,e0,0.048,,M121\r\n"
"$PSIMSNS,230748.601,,1,1,-0.20,0.36,0.02,228.71,,e0,0.048,,M121\r\n"
"$PSIMSNS,230749.602,,1,1,-0.32,0.32,0.01,228.51,,e0,0.048,,M121\r\n"
"$PSIMSNS,230750.603,,1,1,-0.34,0.35,0.00,228.31,,e0,0.048,,M121\r\n"
"$PSIMSNS,230751.605,,1,1,-0.25,0.46,-0.01,228.11,,e0,0.048,,M121\r\n"
"$PSIMSNS,230752.606,,1,1,-0.11,0.49,-0.02,227.91,,e0,0.048,,M121\r\n"
"$PSIMSNS,230753.607,,1,1,0.01,0.37,-0.04,227.71,,e0,0.048,,M121\r\n"
"$PSIMSNS,230754.609,,1,1,0.05,0.22,-0.04,227.50,,e0,0.048,,M121\r\n"
"$PSIMSNS,230755.610,,1,1,-0.01,0.17,-0.02,227.30,,e0,0.048,,M121\r\n"
"$PSIMSNS,230756.611,,1,1,-0.15,0.24,0.01,227.08,,e0,0.048,,M121\r\n"
"$PSIMSNS,230757.613,,1,1,-0.28,0.38,0.03,226.87,,e0,0.048,,M121\r\n"
Logger stderr
2024-09-02T03:49:19Z | 40 ERROR udp_writer.py:315 UDPWriter: send() error: destination: localhost, port: 6224: [Err
2024-09-02T03:49:20Z | 40 ERROR udp_writer.py:315 UDPWriter: send() error: destination: localhost, port: 6224: [Err
2024-09-02T03:49:21Z | 40 ERROR udp_writer.py:315 UDPWriter: send() error: destination: localhost, port: 6224: [Err
2024-09-02T03:49:22Z | 40 ERROR udp_writer.py:315 UDPWriter: send() error: destination: localhost, port: 6224: [Err
2024-09-22T23:26:24Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->off
2024-09-23T04:35:25Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->off
2024-09-26T22:04:44Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->on
2024-09-26T22:04:45Z | 20 INFO udp_subscription_writer.py:70 Detected maximum UDP datagram size 65507
2024-09-26T22:04:45Z | 20 INFO listener.py:66 Instantiating HiPAP->on logger
2024-09-26T22:04:45Z | 20 INFO listener.py:100 Running HiPAP->on
2024-09-27T03:19:49Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->on
2024-09-27T03:19:49Z | 20 INFO udp_subscription_writer.py:70 Detected maximum UDP datagram size 65507
2024-09-27T03:19:49Z | 20 INFO listener.py:66 Instantiating HiPAP->on logger
2024-09-27T03:19:49Z | 20 INFO listener.py:100 Running HiPAP->on
2024-09-29T22:26:18Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->off
2024-09-30T01:48:09Z | 20 INFO logger_runner.py:156 Starting logger HiPAP config HiPAP->on
2024-09-30T01:48:09Z | 20 INFO udp_subscription_writer.py:70 Detected maximum UDP datagram size 65507
2024-09-30T01:48:09Z | 20 INFO listener.py:66 Instantiating HiPAP->on logger
2024-09-30T01:48:09Z | 20 INFO listener.py:100 Running HiPAP->on
Device type: HiPAP
Field name
Recent Data
Units
Time 230757.613 hhmmss.ss
PosItem M50
TransceiverNo 1
TransducerNo 1
Roll -0.28 °
Pitch 0.38 °
Heave 0.03 m
Heading 226.87 °
Tag
Parameters e0
TimeAge 0.048 s
Tide m
MasterSlave M121
TpCode M50
Status A
ErrorCode
CoordinateSystem D
Orientation N
SW_Filter M
Y_coord -36.45508
NorS S
X_coord 174.787315
EorW E
Depth 22.428 m

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

Best Practices

- Logger design:
  - front line loggers should timestamp/save raw data and do little else
  - propagate simplest way can manage

```
readers:
  - class: SerialReader
    kwargs:
      port: /tmp/tty_rtmp
transforms:
  - class: TimestampTransform
writers:
  - class: LogfileWriter
    kwargs:
      filebase: /var/tmp/log/rtmp/raw/r
  - class: ComposedWriter
    kwargs:
      transforms:
      - class: PrefixTransform
        kwargs:
          prefix: rtmp
      writers:
      - class: UDPWriter
        kwargs:
            port: 6224
```

Best Practices

- Logger design:
  - front line loggers should timestamp/save raw data and do little else
  - propagate simplest way can manage
  - use second line "loggers" to parse and do more complicated processing.

```
readers:
  - class: UDPReader
    kwargs:
      port: 6224
transforms:
  - class: ParseTransform
    kwargs:
      metadata interval: 10
      definition_path: test/NBP1406/dev
writers:
  - class: CachedDataWriter
    kwargs:
      data server: localhost:8766
  - class: InfluxDBWriter
    kwargs:
      bucket_name: openrvdas
```

Best Practices

- Parsing
  - single parser handling all data is usually sufficient unless huge. data rates (e.g. winches)
  - simplifies cruise configurations.

```
readers:
  - class: UDPReader
    kwargs:
      port: 6224
transforms:
  - class: ParseTransform
    kwargs:
      metadata interval: 10
      definition_path: test/NBP1406/dev
writers:
  - class: CachedDataWriter
    kwargs:
      data server: localhost:8766
  - class: InfluxDBWriter
    kwargs:
      bucket_name: openrvdas
```

Best Practices

- Partition functionality
  - One machine running loggers, one running InfluxDB/Grafana, etc.
  - Relay between using UDP or CDS - UDP very lightweight and seems reliable enough.

Best Practices

- Partition functionality
  - Consider having one machine running frontline loggers, another running second line.

Best Practices

- Code organization
  - Create your own repo for ship/institution-specific code.
  - Check out into /opt/
  - Symlink into /opt/openrvdas/local

```
/opt/openrvdas_usap/
    \_devices/
```

```
/opt/openrvdas/local
```

OpenRVDAS

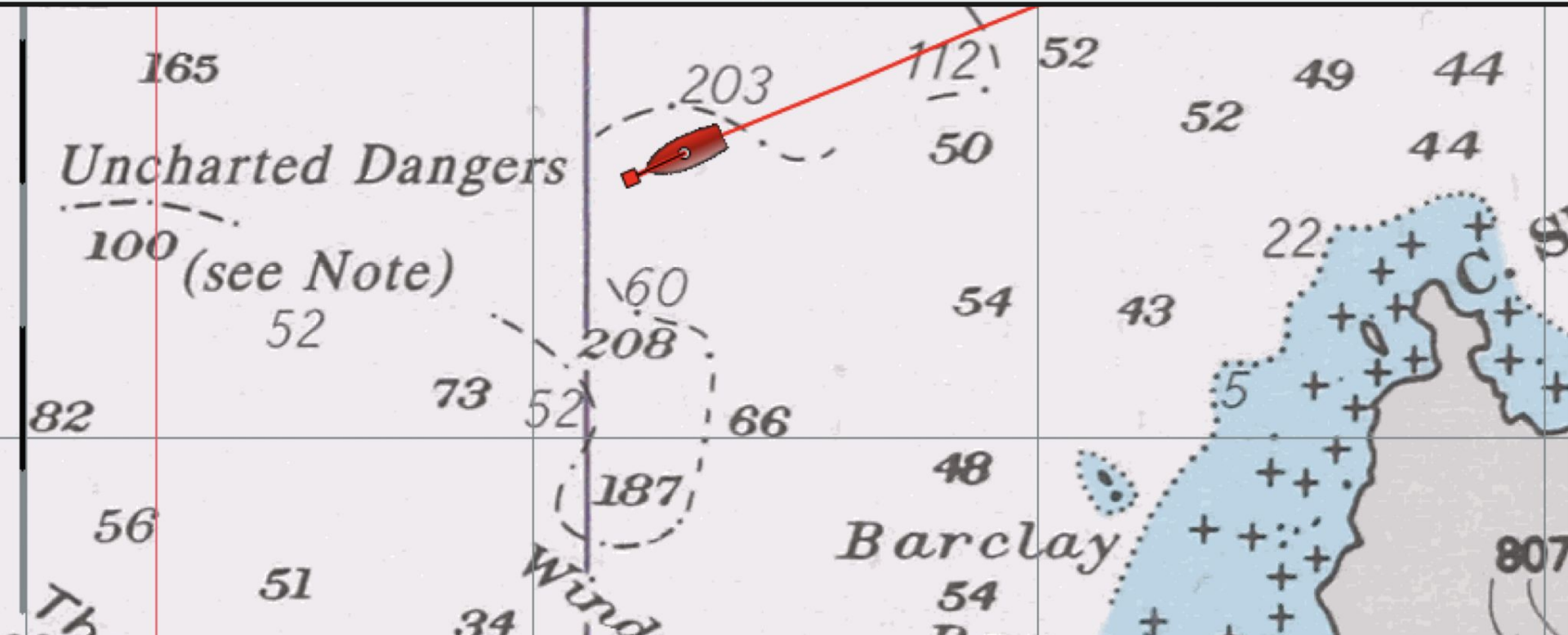
- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Whole system overview - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Pain points - cruise configurations/device definitions
- Displaying data - native/InfluxDB+Grafana
- Best practices
- Contributing
- Where to from here?

Contributing to OpenRVDAS

Because sharing is caring! ❤️

- Bug reports/feature requests: https://github.com/OceanDataTools/openrvdas/issues
- New code: https://github.com/OceanDataTools/openrvdas_contrib

Where to from here?



Where to from here?

- The easy parts

- New, improved modules
  - SealogWriter/SealogReader
  - ValueFilterTransform
- Expanded documentation
  - Video tutorials
  - Cookbook

Where to from here?

- The harder parts

- Simplified logger/cruise configuration creation and maintenance.
  - Jinja has been a good start
  - Graphical tools?
- Solution for long-term project support

Long Term Ocean Data Tools Support

- Current support comes from individual contracts
- Improvements and ongoing maintenance are largely volunteer (*Thank you, NIWA, CSIRO and USAP!*)
- Does it make sense to get affiliated with or absorbed by an institution?