Marine Data Wrangling with OpenRVDAS and CORIOLIX

RVTEC 2025 Tutorial

https://oceandatatools.org
https://coriolix.ceoas.oregonstate.edu/

David Pablo Cohn - david.cohn@openrvdas.org (with Webb Pinner)
Chris Romsos - chris.romsos@oregonstate.edu

OpenRVDAS and CORIOLIX

CORIOLIX

- manages configuration files and metadata
- runs Logger Manager, database, visualizations and other processes
- coordinates real-time ship-to-shore datapresence of output

OpenRVDAS Logger Manager
reads configuration files, runs and monitors loggers.

OpenRVDAS loggers
read, process, distribute, store sensor data.

OpenRVDAS

- **Introduction** - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

OpenRVDAS

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

Part of suite of open source tools for data acquisition / management under the "Ocean Data Tools" collaboration.

- **OpenRVDAS** - modular platform for developing custom data acquisition systems.
- **OpenVDM** - flexible vessel-wide data management system for organizing files from data acquisition systems
- **Sealog** - modular platform for building custom event-logging solutions

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.

Main function is to get data

- from sensors
- to file/database/ network/displays

(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.

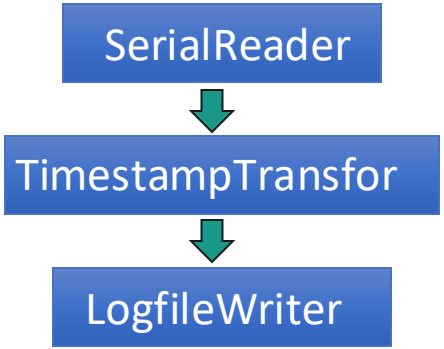
Modular - so easy to modify/extend/keep up to date.

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.

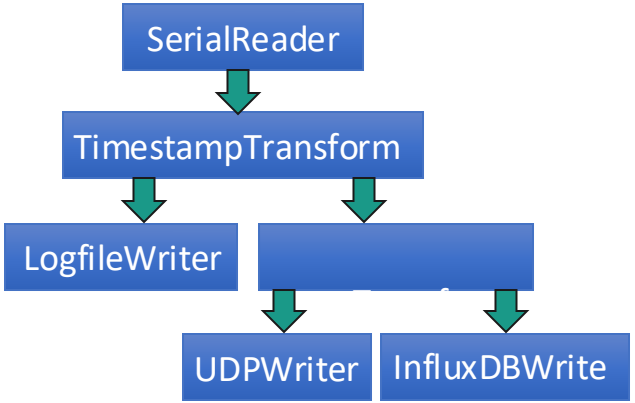



```
.9,1.04,M,,M,,*41
2014-08-01T00:00:00.931000Z $GPVTG,213.66,T,,M,9.4,N,
2014-08-01T00:00:00.931000Z $GPHDT,218.83,T*05
2014-08-01T00:00:00.931000Z $PSXN,20,1,0,0,0*3A
2014-08-01T00:00:00.931000Z $PSXN,22,0.29,0.83*39
2014-08-01T00:00:00.951000Z $PSXN,23,0.58,-1.09,218.8
2014-08-01T00:00:01.815000Z $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?**

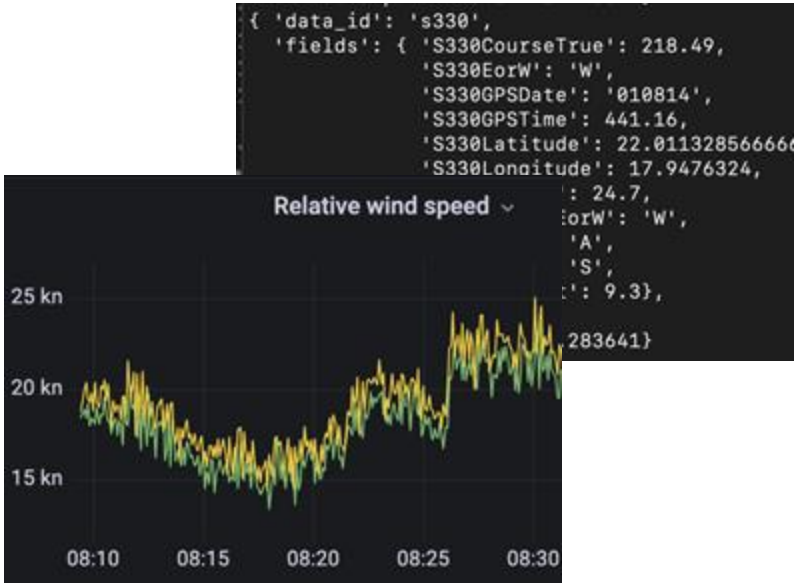
- Parse data into individual fields, send out over network and write to off-machine database.



OpenRVDAS

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

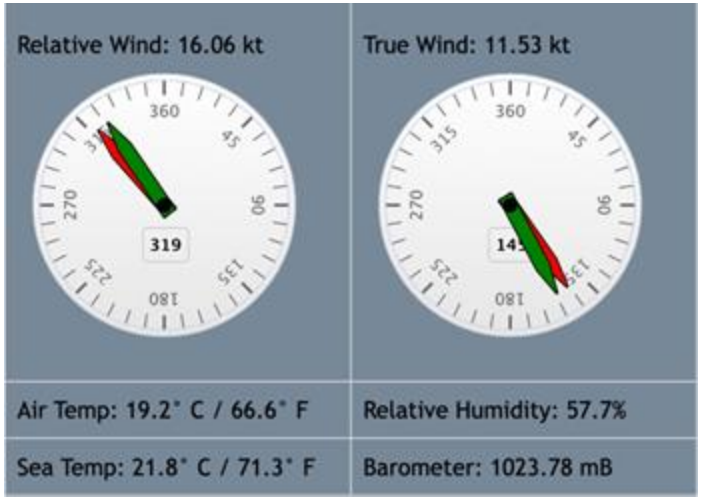
- Parse data into individual fields, send out over network and write to off-machine database.



OpenRVDAS

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

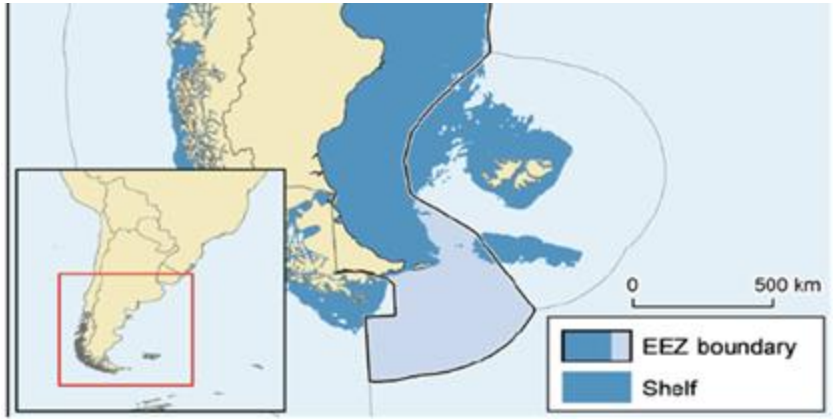
- Combine and numerically manipulate the fields to create derived data products like true winds or moving averages.



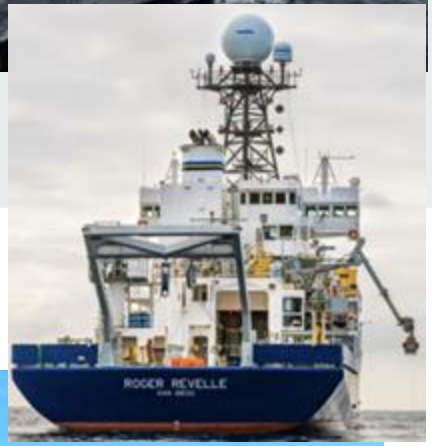
OpenRVDAS

- 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 log events, geofence or trigger other system state changes.



OpenRVDAS Installations



OpenRVDAS

- Introduction - what/why/where
- **Loggers 101** - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

Installation - basic download

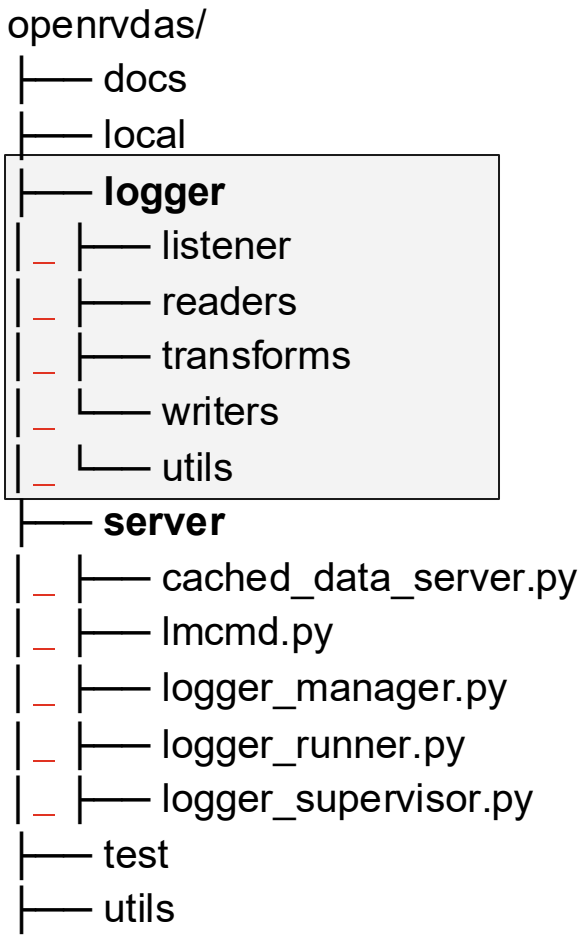
- Clone repo from OceanDataTools on GitHub
- Allows using running individual loggers

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

Code Orientation

`logger` directory contains

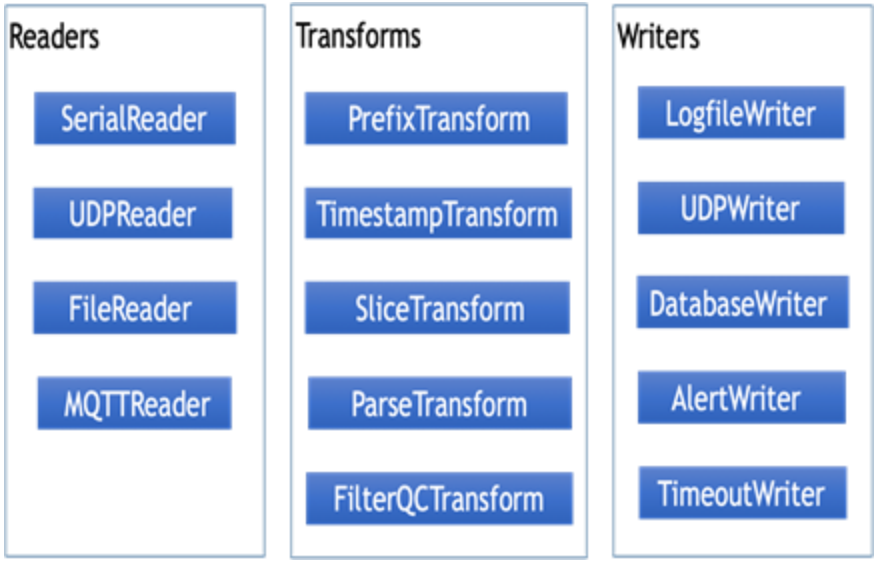
- components: **`readers`**, **`transforms`**, and **`writers`**
- **`listener`** that assembles and runs the components



Running Loggers

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

Loggers are composed of simple Python components.



Reader

```
serial_reader = SerialReader(port='/dev/ttys1')
in_record = serial_reader.read()

===> $HEHDT,219.55,T*15
```

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

Transform

```
timestamp = TimestampTransform()
timestamped_record = timestamp.transform(record)

$HEHDT,219.55,T*15   ===>
2025-09-03T00:00:02.462698Z $HEHDT,219.55,T*15
```

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

Writer

```
logfile = LogfileWriter(filebase='/var/logs/knud')
logfile.write(output_record)

2025-09-03T00:00:02.462698Z $HEHDT,219.55,T*15
 ===> ???
```

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

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

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

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

listen.py --help
```

to see all available options

Running Loggers

With a YAML-format logger configuration file

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

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

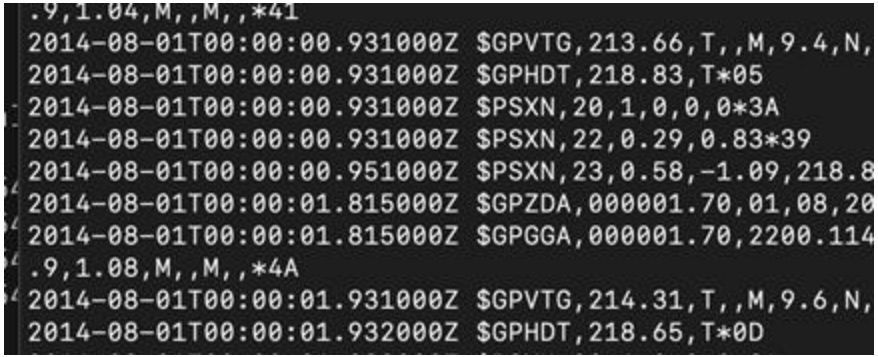
One Especially Important Transform

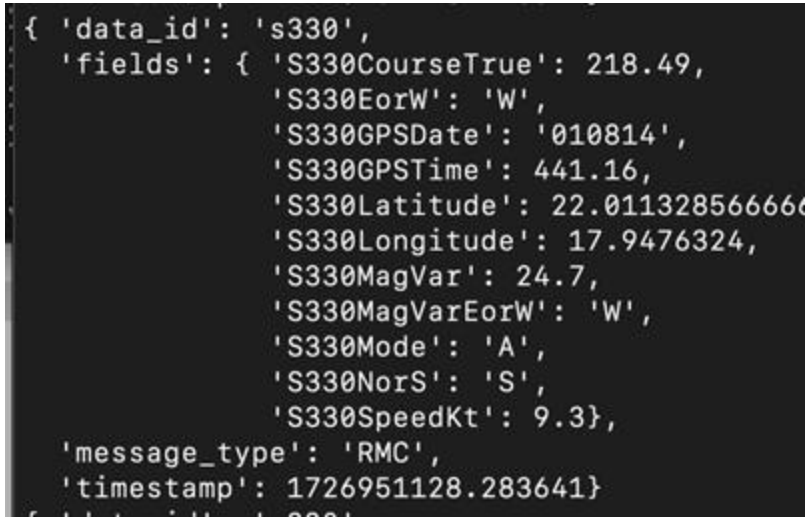
`ParseTransform()`

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





ParseTransform

- 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
       }
   }
```

ParseTransform

- Some sensors can emit wide range of outputs

```
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
```

ParseTransform

- Can use either inline or 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

- **`ParseTransform`** is built on Python **`parse`** module
- **`RegexParseTransform`** - similar, but matches are specified by *regex*.

```
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<Chec
VTG: \$(?P<TalkerID>\w{2})VTG,\s*(?P<CourseTrue>\-?\d*\.?\d*),\s*T,\s*(?P<Cou
```

Parsing Data

- Once you've got the parsed numerical/text values from records, you can do all sorts of fun things with them.

- Full documentation at https://www.oceandatatools.org/openrvdas-docs/parsing

Build Your Own...

- Use **`module`** keyword to tell **`listen.py`** where to your custom readers, writers and transforms.
- **`kwargs`** specifies the component's keyword arguments.

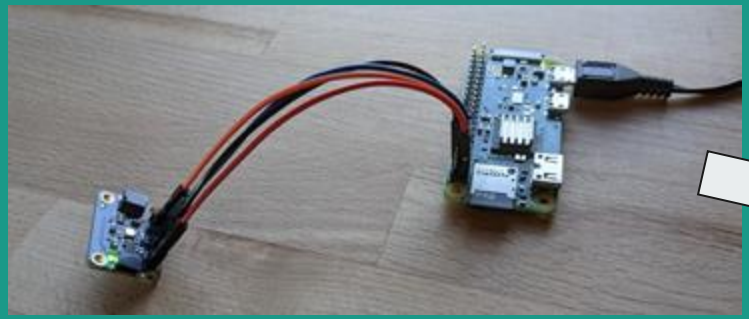
```
# TeaLeafReader implementation
class TeaLeafReader():
  def __init__(tea_type='black',
               temp_in_c=100):
    ...
  def read():
    ...
    return result
```

Config file:

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

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
- **Logger Manager** - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

If you want to...

- ...run and monitor the status of many loggers
- ...graphically monitor and change modes via web interface
- ...integrate event logging
- ...conditionally run different loggers in different situations (in port, an EEZ, underway, running winches)

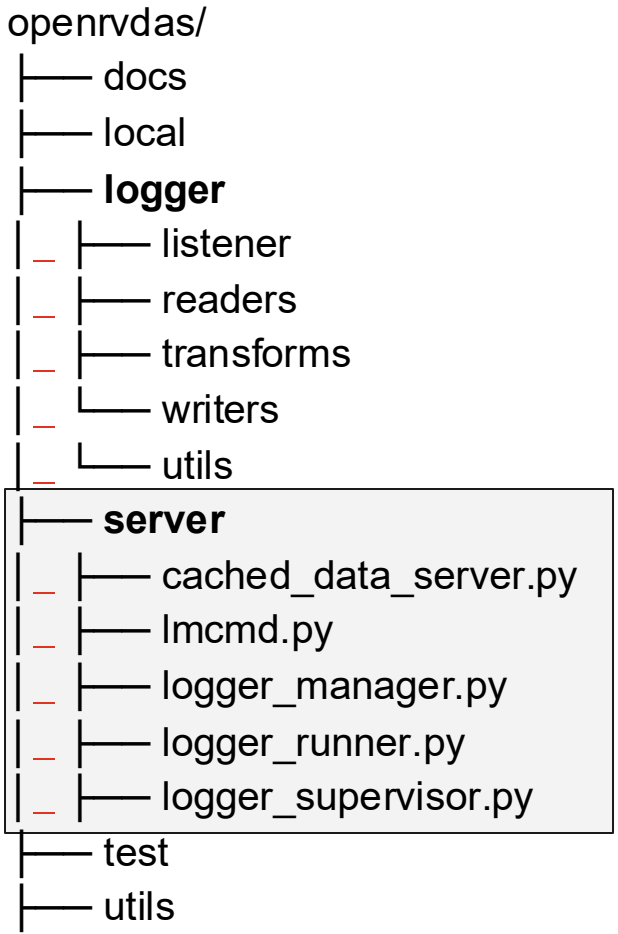
Then you probably want the full OpenRVDAS installation.

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

Code Orientation

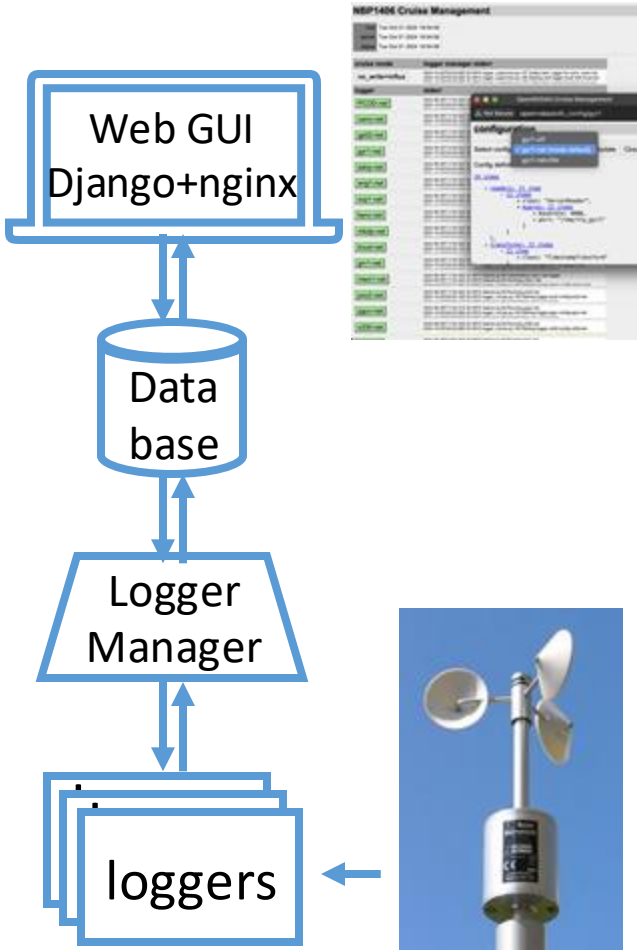
`server` directory contains

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



Server architecture

- **LoggerManager -** runs/monitors loggers
- **Database -** persistent record of current and desired states.
- **Django+nginx -** runs GUI, communicates with LoggerManager via database



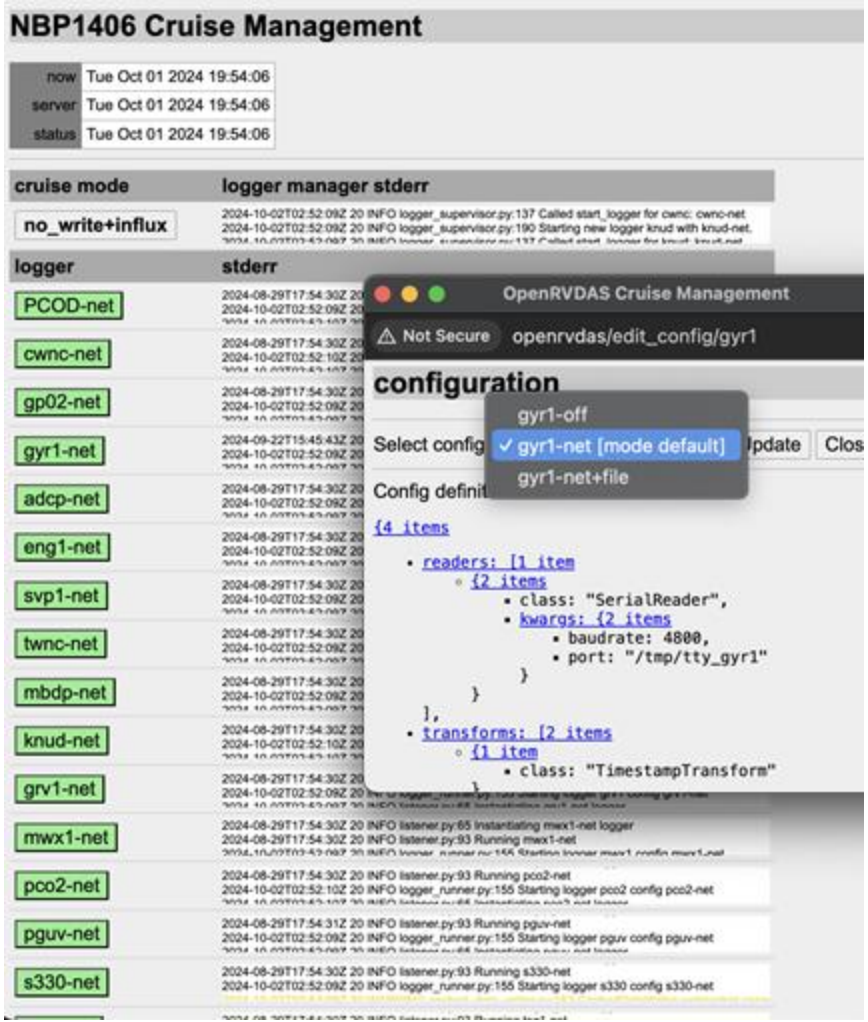
A Peek Behind the Scenes

- 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
```

What it gets you

- **Web interface for controlling loggers.**
- Cruise mode management
- Database-backed persistent state management.



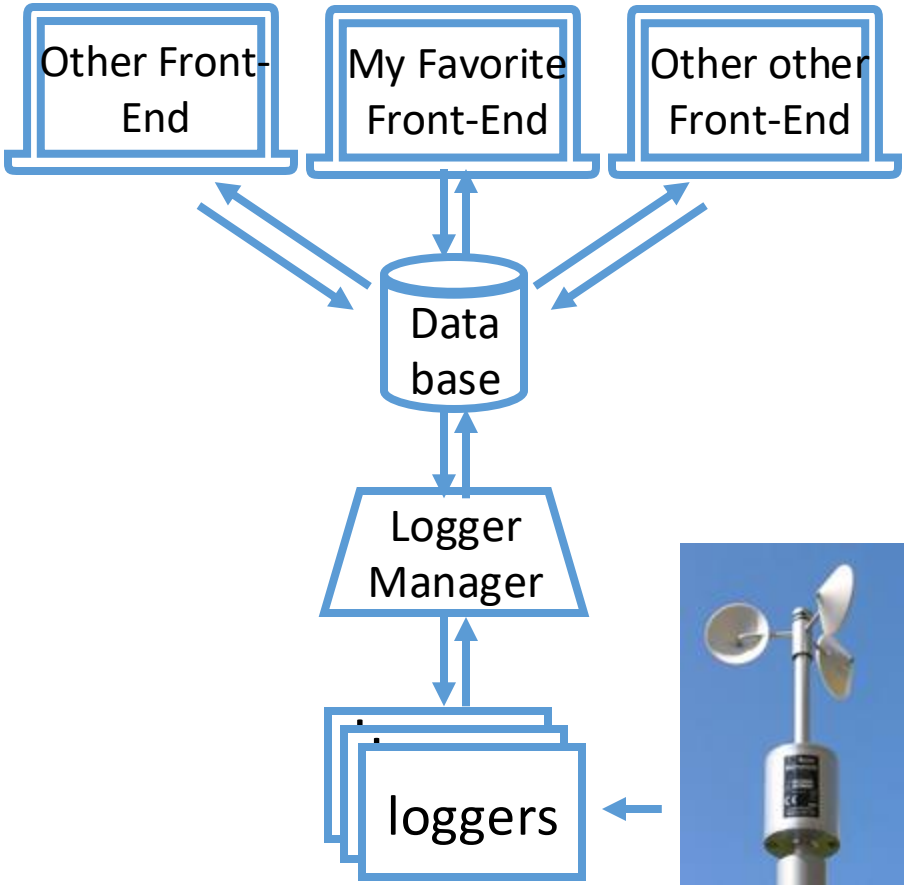
What it gets you

- Web interface for controlling loggers.
- **Cruise mode management**
- Database-backed persistent state management.

In different phases of a cruise, you may need different loggers in distinct configurations.

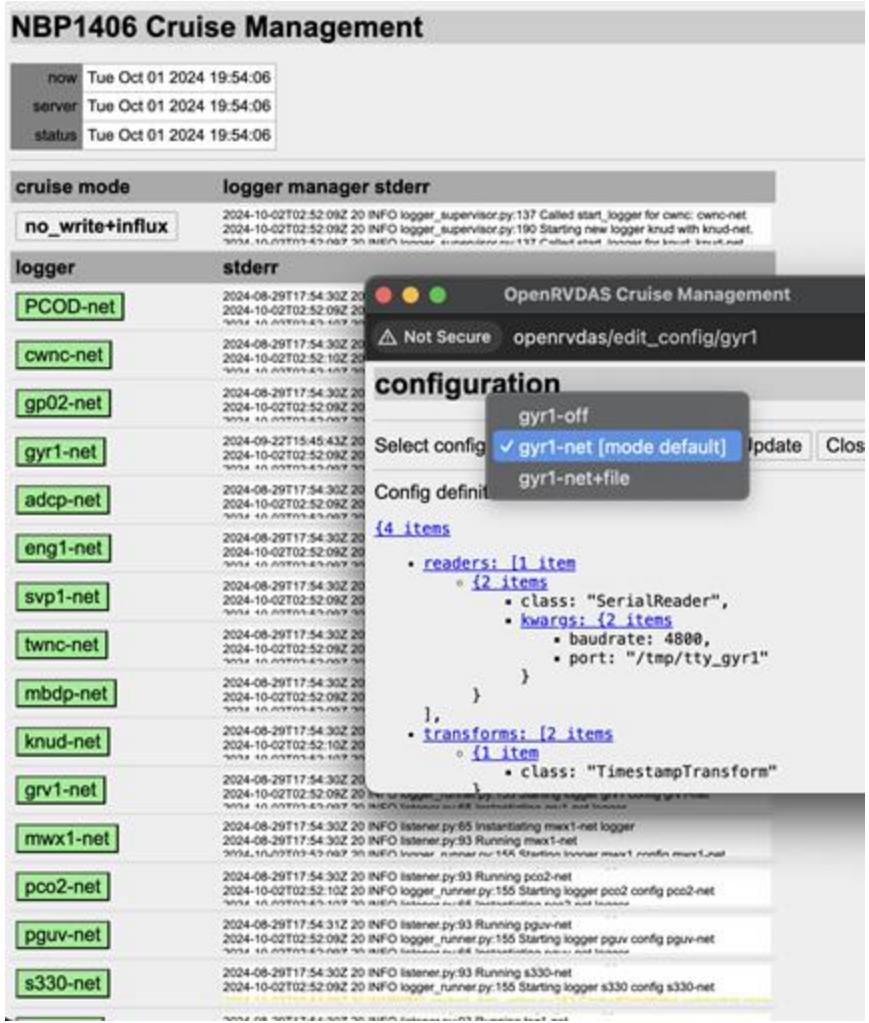
What it gets you

- Web interface for controlling loggers.
- Cruise mode management
- **Database-backed persistent state management.**



Controlling loggers

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



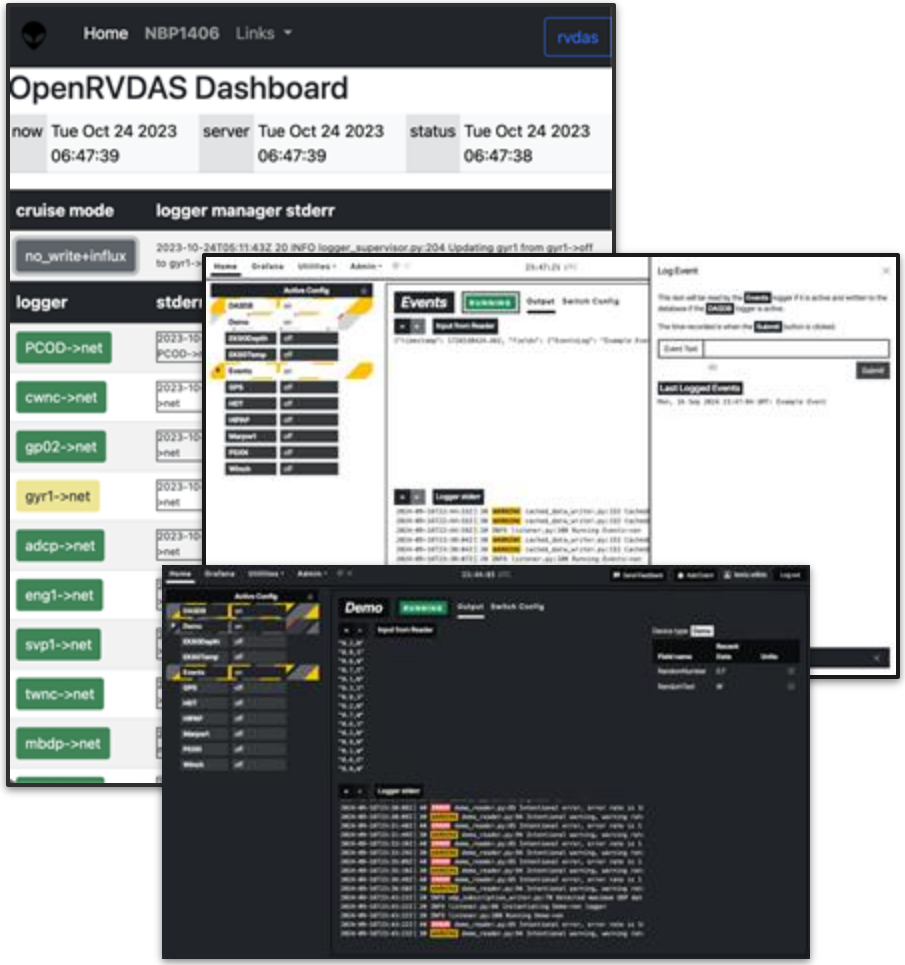
Controlling loggers

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

```
openrvdas> server/logger_manager.py

command? load_configuration NBP1406_cruise
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,
s330, tsg1, rtmp, hdas, tsg2, seap, true_w
subsample

command? get_logger_configs s330
Configs for s330: s330->off, s330->net, s3
>file/net, s330->file/net/db

command? set_active_logger_config s330 s33
>off
command? quit
```

Controlling loggers

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



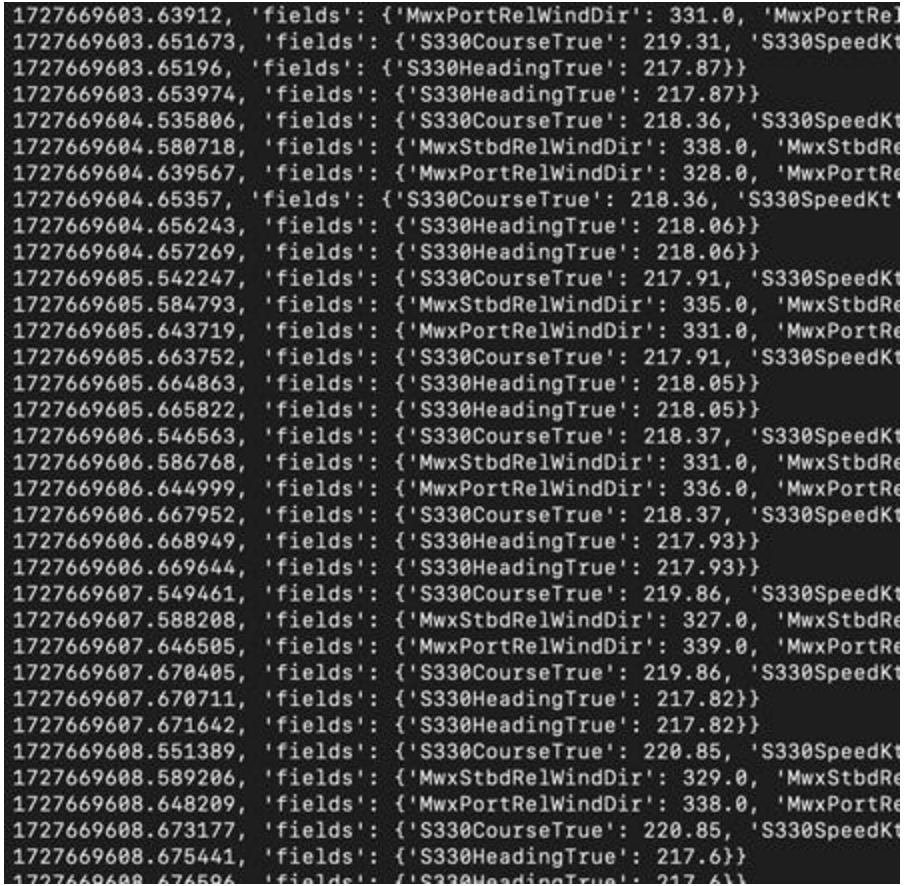
OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- **Cached Data Server** - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

Cached Data Server

- OpenRVDAS-specific pub-sub server in the **`servers/`** subdirectory.

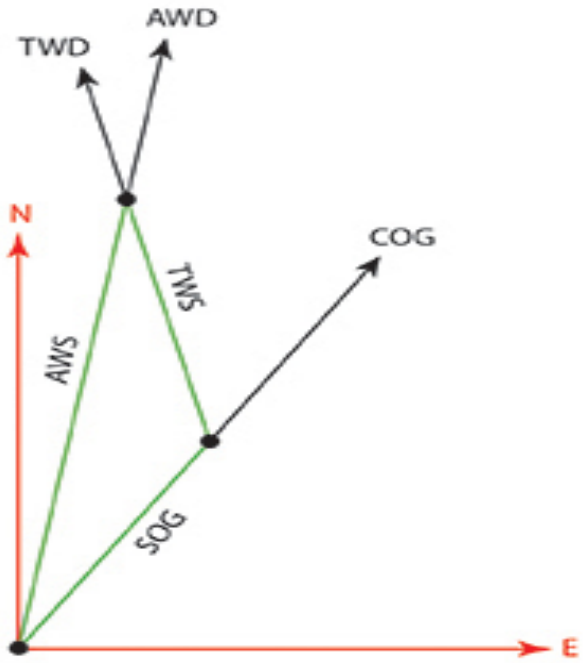
- Loggers can cache data for use by other loggers



True Winds

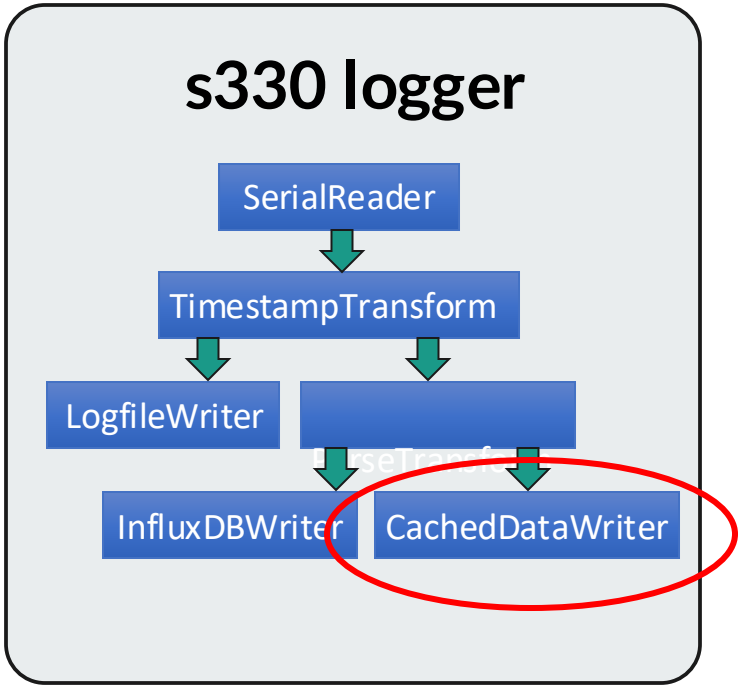
- Depends on
  - **compass**: heading
  - **GPS**: course and speed over ground
  - **WX**: relative wind speed, relative wind dir

- How to combine data from different (asynchronous) loggers?



Cached Data Server

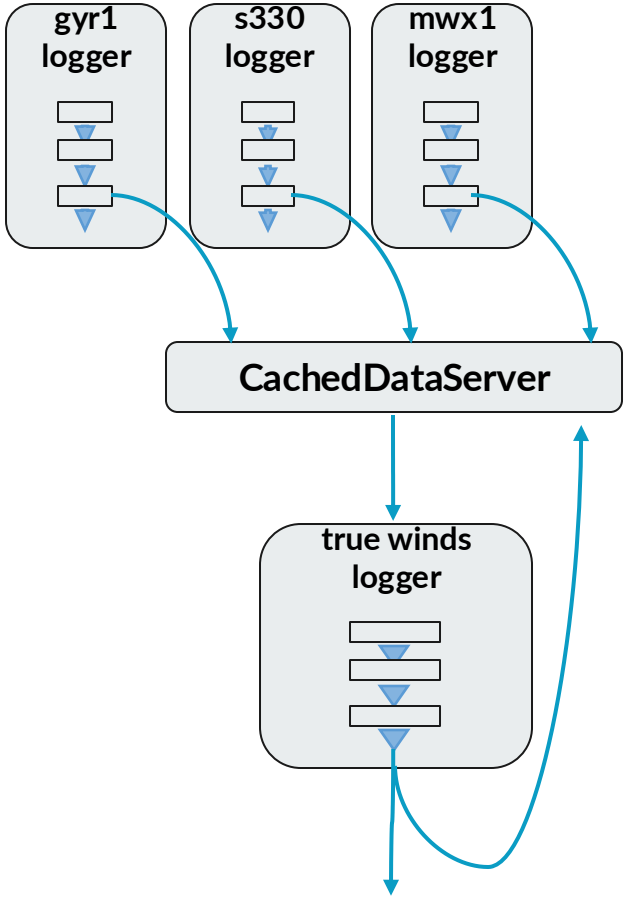
- Write to server via **`CachedDataWriter`**
- Read from it via **`CachedDataReader`**



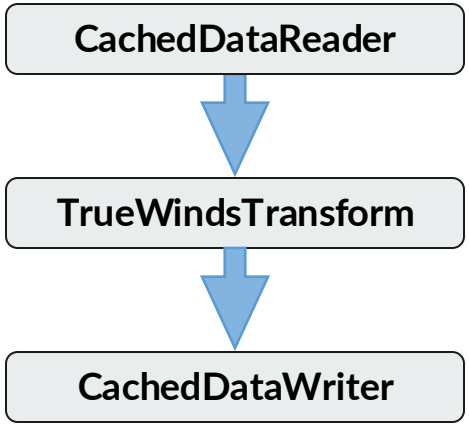
Cached Data Server

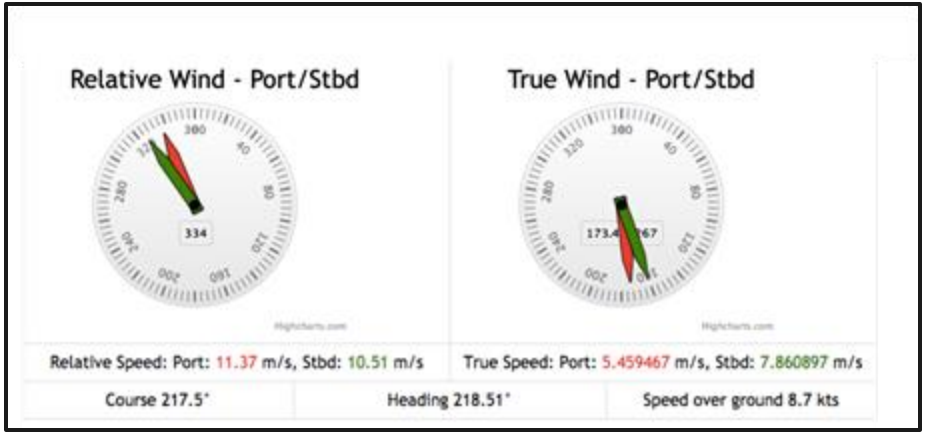
Derived data loggers can

- **Read** from Cached Data Server
- **Compute** derived data values
- **Write** results back to server and/or send elsewhere



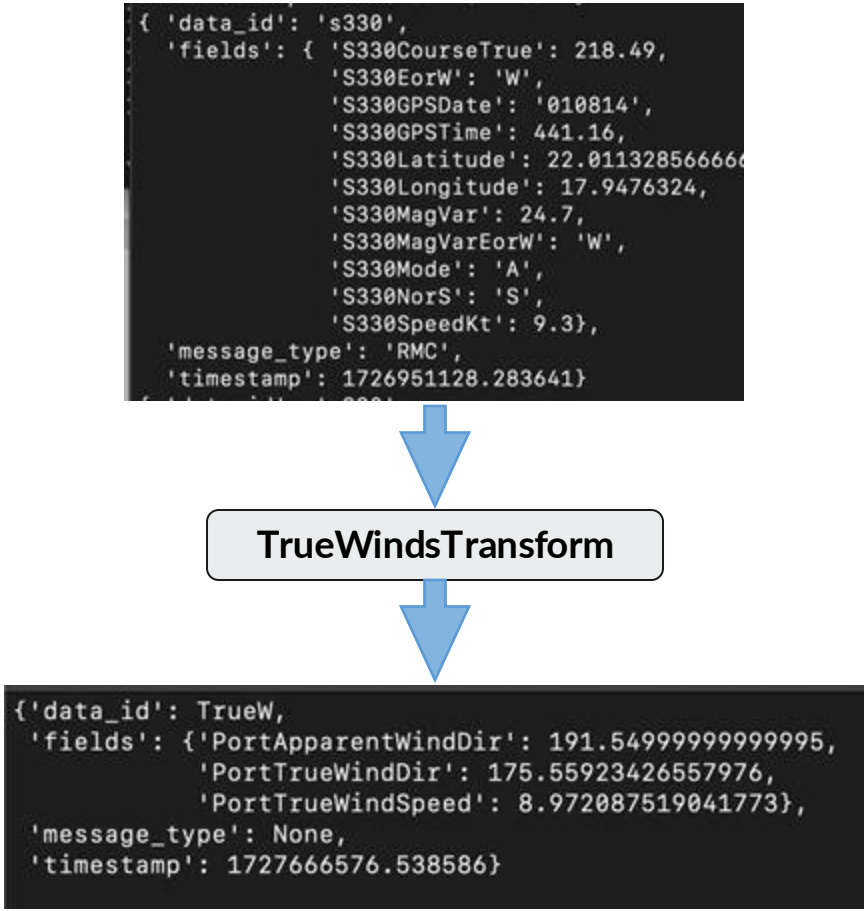
True Winds


CachedDataReader
TrueWindsTransform
CachedDataWriter


Relative Wind - Port/Stbd
True Wind - Port/Stbd
Relative Speed: Port: 11.37 m/s, Stbd: 10.51 m/s
True Speed: Port: 5.459467 m/s, Stbd: 7.860897 m/s
Course 217.5°
Heading 218.51°
Speed over ground 8.7 kts

True Winds

- **`TrueWindsTransform`** takes cached data and looks for the values it needs in it.
- 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.



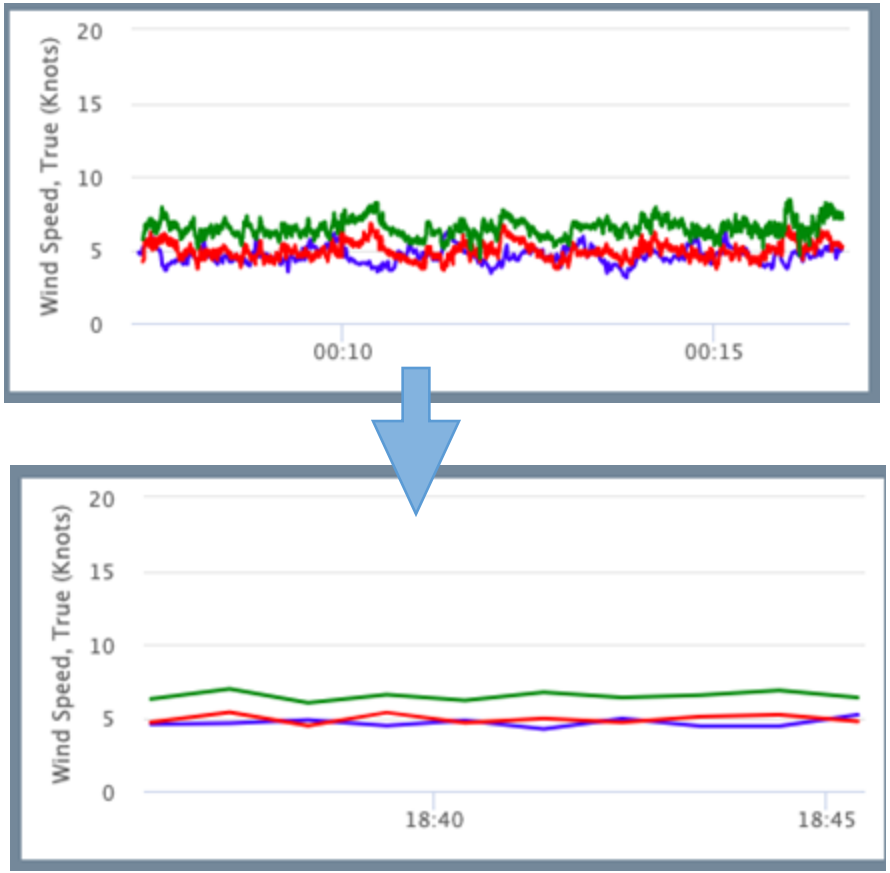
Snapshots

- 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:
    ...
```

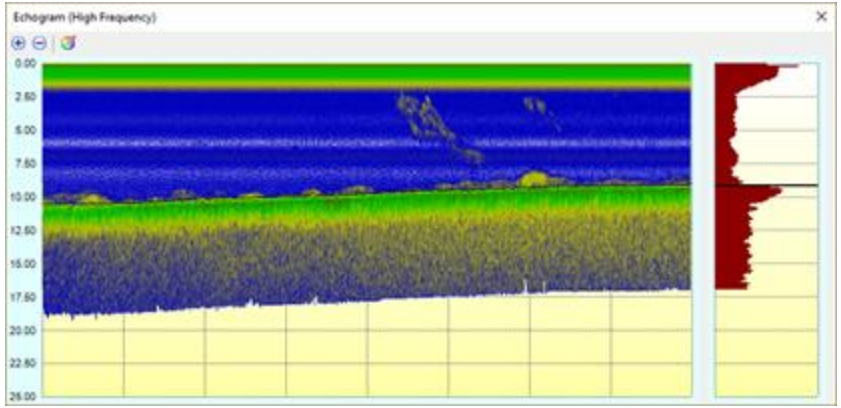
Snapshots

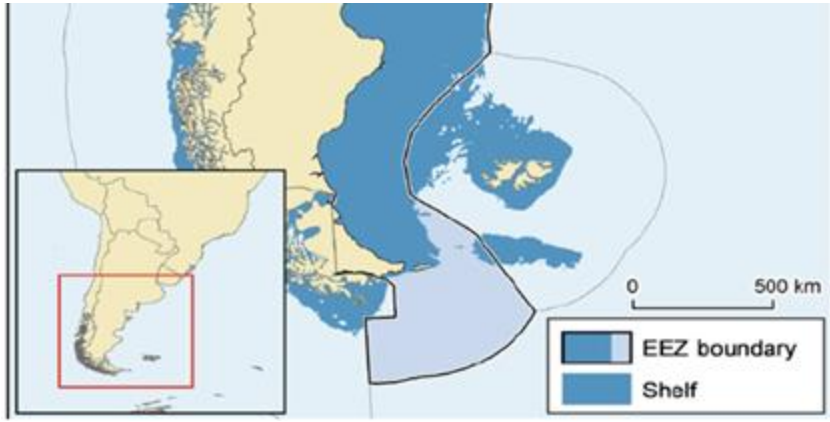
- Use this to aggregate values until ready to produce an output.
- E.g. when computing running averages.



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

- **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

- 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 '
```

Event Logging Integration

SealogWriter

- Logger conditions (such as values exceeded) can be sent to Sealog to be recorded as discrete events.

SealogReader

- Sealog events can be read into OpenRVDAS, either to record as part of data stream or (using `LoggerManagerWriter`) to trigger changes in OpenRVDAS running state.

Still new, so haven't yet plumbed possibilities...

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- **Beyond OpenRVDAS** - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

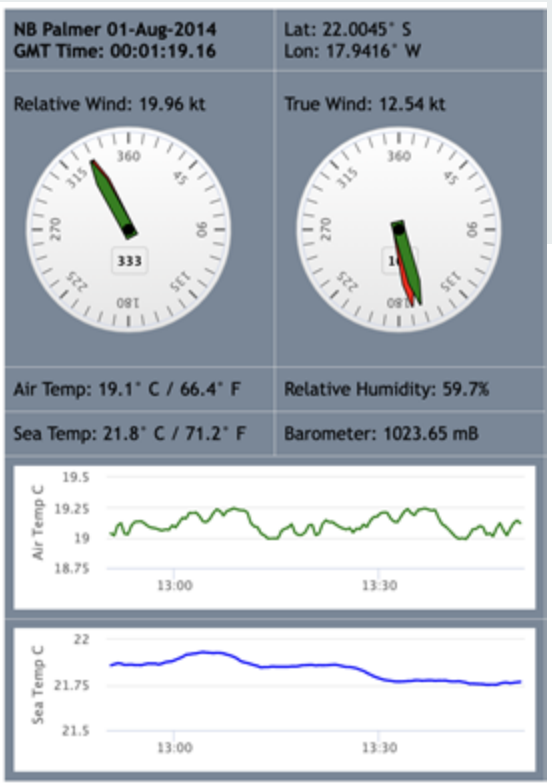
Getting data out of OpenRVDAS

- **`LogfileWriter`** - write data out to file for R2R, etc.
- **`UDPWriter/TCPWriter`** - send out on network
- **`CachedDataWriter`** - send to another OpenRVDAS installation on ship or elsewhere
- **`InfluxDBWriter`**
- **`TimescaledbWriter`**, contributed by Lewis Wilke (NIWA)
- **`RedisWriter`**
- **`PostgresWriter`** and **`CORIOLIXWriter`**, contributed by Jasmine Nahorniak (OSU)
- ... or roll your own!

Displaying OpenRVDAS data

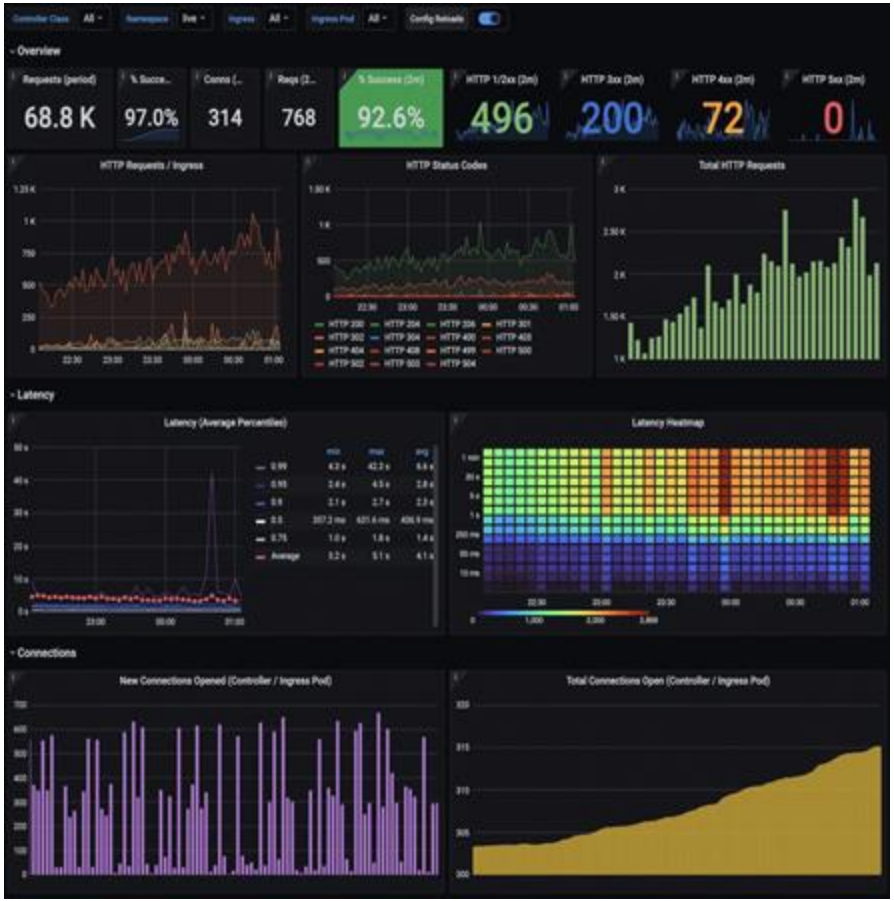
- Original OpenRVDAS display method

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



InfluxDB/Grafana

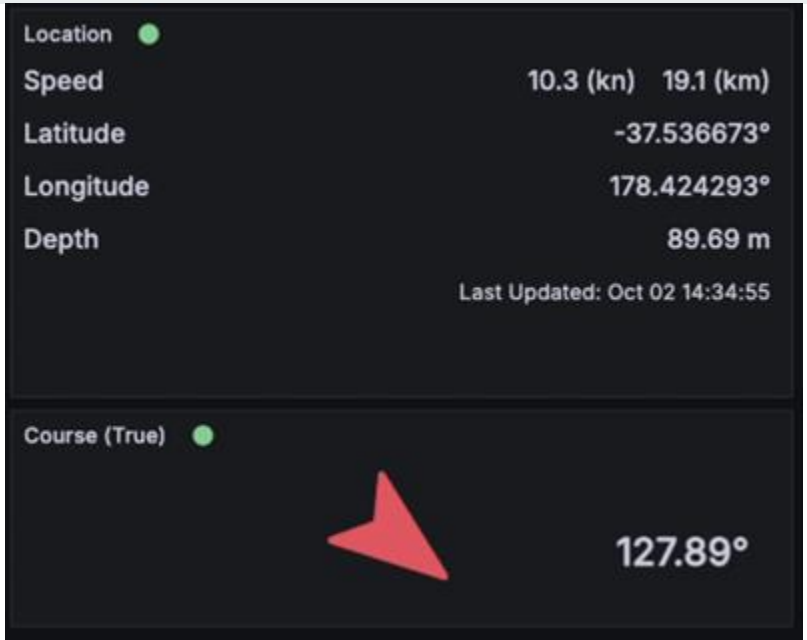
- Preferred display route:
  - open source
  - large community of users and maintainers (who aren't us!).
- Two separate packages:
  - **InfluxDB** - time series database we write to.
  - **Grafana** - analytics, monitoring and visualization system.



Cached Data Server ⇒ 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



Installing InfluxDB/Grafana

- Script: **`utils/install_influxdb.sh`**
- Should be run as the user who will be running OpenRVDAS (e.g. `'rvdas'`).

```
% utils/install_influxdb.sh
```

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:
        url: https://shore.marine.umsc.
        bucket_name: openrvdas
```

Grafana Widgets

Grafana Widgets

Like OpenRVDAS, Grafana lets you easily roll your own and contribute

- e.g. Webb Pinner's new integrated wind/compass widget
- https://github.com/OceanDataTools/grafana-compass-panel
- Samples: http://demo.openrvdas.org:3000

InfluxDB+Grafana 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")
```

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- **What *you* have to do** - cruise/device configurations
- Best practices
- Contributing
- Where to from here?

What *you* have to do

- Create cruise configuration files.
- Create device definitions.
- Keep everything updated as devices and needs change

Setting up a Cruise

Build a cruise configuration file

- **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 file

- **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 file

- **configurations**
- loggers
- modes

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

Setting up a Cruise

Build a cruise configuration file

- configurations
- **loggers**
- modes

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

Setting up a Cruise

Build a cruise configuration file

- 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

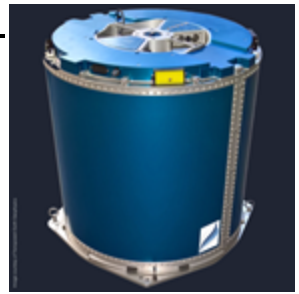
- New configuration templates simplify enormously
  - Old sample NBP1406 cruise: **1517 lines**
  - Templated sample NBP1406 cruise: **217 lines**

Creating device/device type 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
       }
    }
```

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.

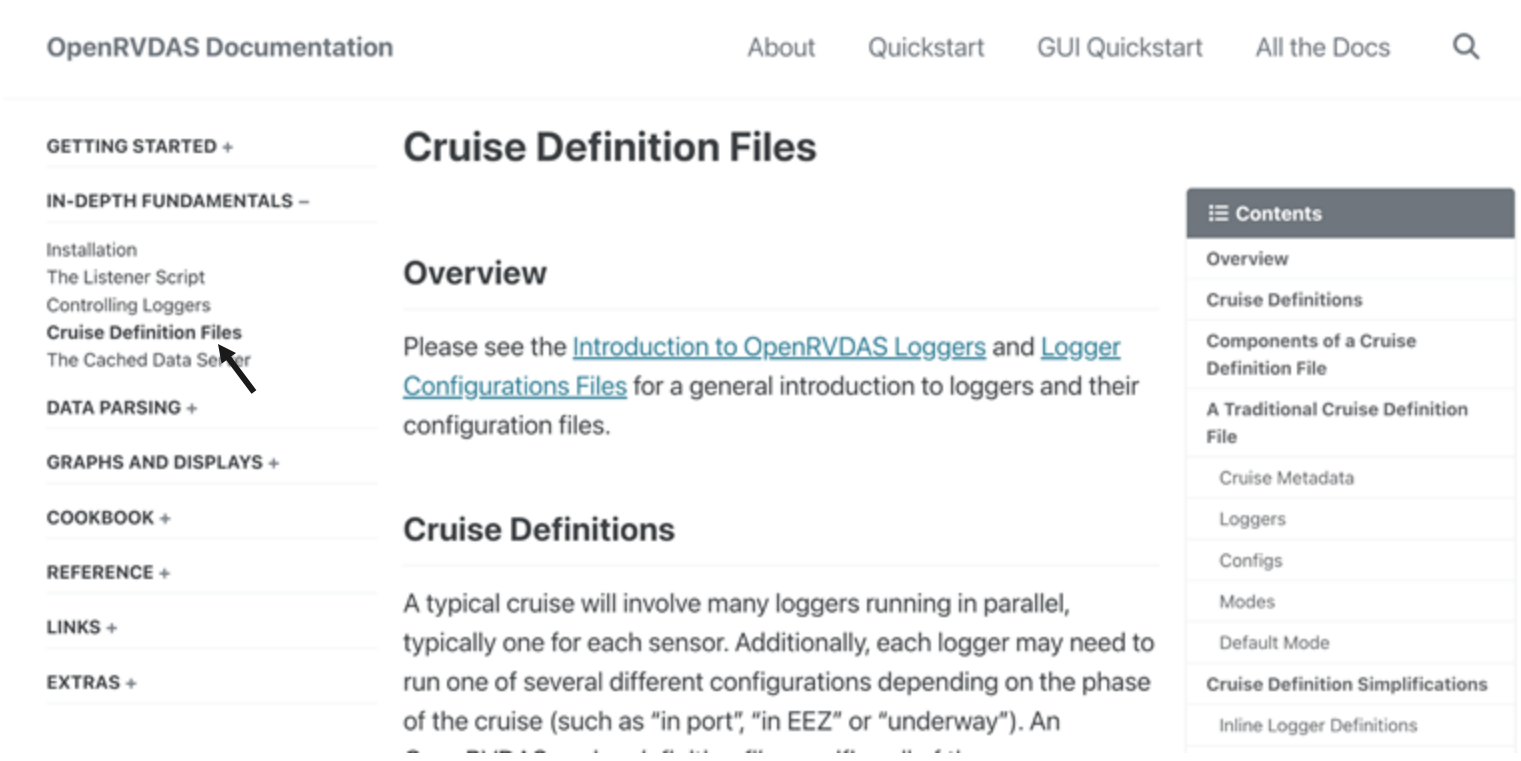
Device types

```
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"
```

Devices

```
>>> parser.transform('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"
```

All of this is documented at openrvdas.org



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.

OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- **Best practices**
- Contributing
- Where to from here?

Best Practices

- Set up a test harness for your data
- Partition your loggers
- Partition your machines
- Organize your code to play nice

Best Practices

- **Set up a test harness for your data**
- Partition your loggers
- Partition your machines
- Organize your code to play nice

Best Practices

- Set up a test harness for your data

Gather timestamped sample data for all your feeds:

```
2014-08-01T00:00:00.285000Z $INZDA,000000.17,01,08,2014,,*7E
2014-08-01T00:00:00.285000Z $INGGA,000000.16,2200.110899,S,01756.359432,W
2014-08-01T00:00:00.402000Z $INVTG,215.11,T,239.79,M,9.1,N,16.9,K,A*05
2014-08-01T00:00:00.522000Z $INRMC,000000.16,A,2200.110899,S,01756.359432
2014-08-01T00:00:00.522000Z $INHDT,218.26,T*1A
2014-08-01T00:00:00.522000Z $PSXN,20,1,0,0,0*3A
2014-08-01T00:00:00.522000Z $PSXN,22,0.03,-0.80*1F
2014-08-01T00:00:00.522000Z $PSXN,23,0.35,-1.74,218.26,0.58*13
...
```

Best Practices

- **Set up a test harness for your data**

```
logger/utils/simulate_data.py \
    --serial /tmp/tty_s330 \
    --filebase test/NBP1406/s330/raw/
```

```
$INGGA,000151.16,2200.332915,S,01756.552618,W,1,12,0.7,-3.34,M,4.67,M,,
$INVTG,218.54,T,243.22,M,9.6,N,17.7,K,A*02
$INRMC,000151.16,A,2200.332915,S,01756.552618,W,9.6,218.54,010814,24.7,
$INHDT,218.07,T*19
$PSXN,20,1,0,0,0*3A
$PSXN,22,0.04,-0.80*18
```

Best Practices

- Set up a test harness for your data

```
logger/utils/simulate_data.py \
  --config test/NBP1406/simulate_NBP1406.yaml
```

```
s330:
  class: Serial
  port: /tmp/tty_s330
  filebase: test/NBP1406/s330/raw/NBP1406_s330

gyr1:
  class: Serial
  port: /tmp/tty_gyr1
  filebase: test/NBP1406/gyr1/raw/NBP1406_gyr1
```

Best Practices

- Set up a test harness for your data
- **Partition your loggers**
- Partition your machines
- Organize your code to play nice

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.

See **`test/NBP1406`** for sample cruise configuration that follows best practices.

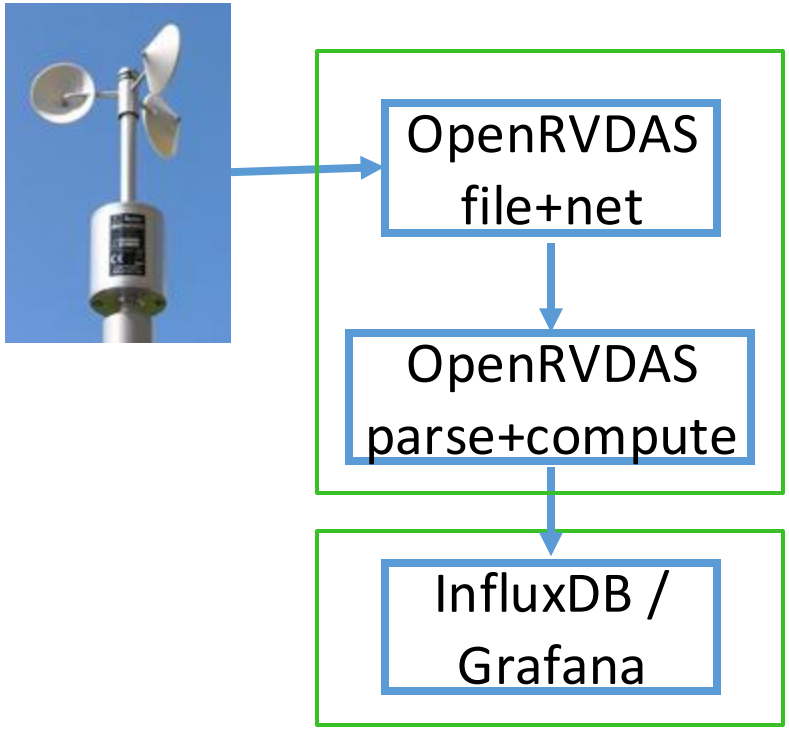
```
readers:
  - class: UDPReader
    kwargs:
      port: 6224
transforms:
  - class: ParseTransform
    kwargs:
      metadata_interval: 10
      definition_path: test/NBP1406/dev
writers:
  - class: CachedDataWriter
    kwargs:
      data_server: localhost:8766
  - class: InfluxDBWriter
    kwargs: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

Best Practices

- Set up a test harness for your data
- Partition your loggers
- **Partition your machines**
- Organize your code to play nice

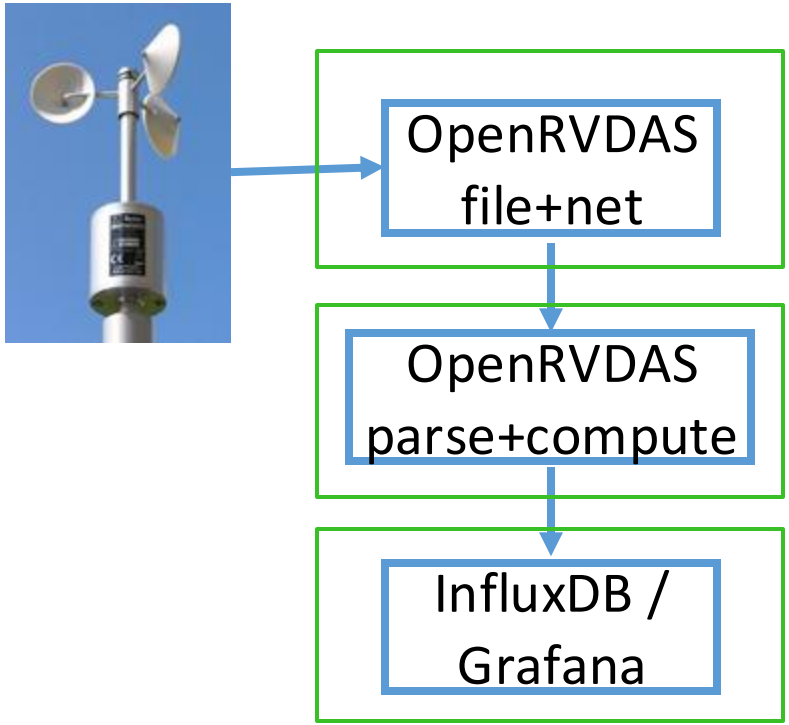
Best Practices

- Partition functionality
  - One machine running loggers
  - One machine 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.

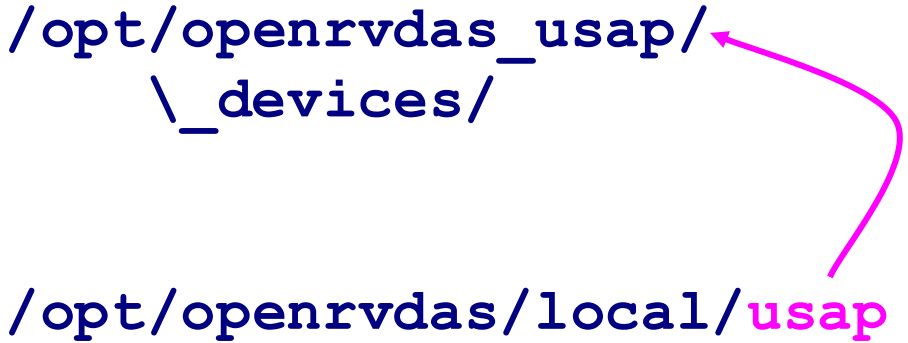


Best Practices

- Set up a test harness for your data
- Partition your loggers
- Partition your machines
- **Organize your code to play nice**

Best Practices

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



OpenRVDAS

- Introduction - what/why/where
- Loggers 101 - components/running/parsing
- Logger Manager - installation/controlling loggers
- Cached Data Server - fun and games with derived data
- Beyond OpenRVDAS - data distribution/storage/display
- What *you* have to do - cruise/device configurations
- 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?

- Expanded documentation
  - Video tutorials
  - Cookbook

- Solution for long-term project support
  - Current support comes from individual contracts
  - Improvements and ongoing maintenance are largely volunteer

Questions?