OpenRVDAS Sikuliaq Notes

VM Details

CentOS 7 machine, built following instructions at https://github.com/davidpablocohn/openrvdas/blob/master/INSTALL.md

root password: CentOS
user: rvdas; password: rvdas
mysql root password: rvdas

Code repository: https://github.com/davidpablocohn/openrvdas (there is a sikuliaq branch)
Documentation: http://tinyurl.com/openrvdas-docs

Quick and Easy Run of the Code

Just to get started, you can mess around with the listen.py script, which you can think of as a specialized form of the 'cat' command: you specify inputs (readers), transformations and outputs (writers):

```
# Read a text file, parse it as NMEA data and write to stdout.
# Note 1: Sikuliaq date/time format differs from openrvdas default,
#         so need to override
# Note 2: A single "-" in place of a filename for --write_file means
#         write to stdout
logger/listener/listen.py \
    --time_format "%Y-%m-%dT%H:%M:%S.%fZ" \
    --file skq/sikuliaq.data \
    --transform_parse_nmea \
    --write_file -

# As before, but now also write to local SQL database
logger/listener/listen.py \
    --time_format "%Y-%m-%dT%H:%M:%S.%fZ" \
    --file skq/sikuliaq.data \
    --transform_parse_nmea \
    --database_password rvdas \
    --write_database rvdas@localhost:data \
    --write_file -
```

Try `logger/listener/listen.py --help`, and look at the documentation at Running OpenRVDAS Loggers for more information.

Different ways of using OpenRVDAS code

There are five(!) obvious ways of using the code, listed below in order of low-level to high-level. You will most likely want to use the latter ones; I'm listing the lower-level ways for pedagogical reasons.

1. Write logger using modules

It's fairly straightforward to manually instantiate and connect components to build a dedicated Python logger:

```
#!/usr/bin/env python3

from logger.readers.network_reader import NetworkReader
from logger.transforms.parse_nmea_transform import ParseNMEATransform
from logger.writers.database_writer import DatabaseWriter
from logger.writers.text_file_writer import TextFileWriter

network = ':54122'
time_format = '%Y-%m-%dT%H:%M:%S.%fZ'

reader = NetworkReader(network=network)
parser = ParseNMEATransform(time_format=time_format)

# Use defaults from database/settings.py
database_writer = DatabaseWriter()

# With no args, will write to stdout
text_file_writer = TextFileWriter()

while True:
  record = reader.read()
  if record:
    text_file_writer.write(record)

    # Database writer wants parsed records
    parsed_record = parser.transform(record)
    if parsed_record:
      database_writer.write(parsed_record)
```

(Note: if you run the above script, you can feed it data by running listen.py in another window: `logger/listener/listen.py --file skq/sikuliaq.data --write_network :54122`)

2. Use the listen.py script with command line arguments

The `listen.py` script incorporates the most common Readers, Transforms and Writers, providing much of the functionality that one might want in a logger straight from the command line. For example, the invocation:

```
# Writers operate in parallel; the --write_file option with "-" for
# an argument writes to stdout
logger/listener/listen.py \
    --network :54122 \
    --time_format '%Y-%m-%dT%H:%M:%S.%fZ' \
    --transform_parse_nmea \
    --database_password rvdas \
    --write_database rvdas@localhost:data \
    --write_file - \
    -v
```

implements the same dataflow as the previous Python code.

Note that the listen.py script has half a billion command line options and (as documented in `--help`) applies them in order. So, for example

```
logger/listener/listen.py \
    --file skq/sikuliaq.data \
    --transform_slice 2: \
    --transform_timestamp \
    --transform_prefix skq_data \
    --write_file -
```

will strip off the first two fields of each record, then prefix a timestamp to it, then the string '`skq_data`', while the invocation

```
logger/listener/listen.py \
    --file skq/sikuliaq.data \
    --transform_timestamp \
    --transform_prefix skq_data \
    --transform_slice 2: \
    --write_file -
```

will add the timestamp and '`skq_data`', prefix, then strip off those two newly-added columns.

3. Use the listen.py script with a JSON configuration file

The `listen.py` script can also read from a pre-assembled configuration file:

```
logger/listener/listen.py --config_file skq/gyro_1.json
```

where `gyro_1.json` consists of the JSON definition

```
{
    "name": "gyro_1->db",
    "readers": {
        "class": "NetworkReader",
        "kwargs": { "network": ":54122" }
    },
    "transforms": {
        "class": "ParseNMEATransform",
        "kwargs": { "time_format": "%Y-%m-%dT%H:%M:%S.%fZ" }
    },
    "writers": [
        { "class": "TextFileWriter" },
        {
          "class": "DatabaseWriter",
          "kwargs": {
            "user": "rvdas",
            "host": "localhost",
            "database": "data",
            "password": "rvdas"
          }
        }
      ]
}
```

(Run `listen.py --help` for a full list of the options that the script takes.)

4. Use the run_loggers.py script to run multiple loggers

The gyro_1.json file above encoded the configuration for one logger. We can also define a "cruise configuration" file which encodes configurations for multiple loggers and groups them into "modes": one set of configurations, e.g. when the ship is underway, another when it's in port.

The file skq/skq_cruise.json defines four modes:

- off - no loggers running
- file - all loggers running and saving data to their separate logfiles
- db - all loggers saving data to the SQL database
- file/db - all loggers saving to both logfiles and database

The cruise configuration can be read and used by the run_loggers.py script:

```
logger/utils/run_loggers.py \
    --config skq/skq_cruise.json \
    --mode file/db
```

It will start one subprocess for each logger configuration specified in the "file/db" mode, monitor it for health, and restart it if it dies for any reason.

(If started with the `--interactive` flag, `run_loggers.py` will listen to standard input and, if you enter the name of another mode, will kill off the running loggers and start loggers appropriate to the new mode.)

Use the Django web interface to interactively manage loggers

To keep things lightweight, we'll use Django's test server.

1. Determine the machine's ip address with '`ip addr`'; for the purposes of this tutorial, let's assume it's `10.0.0.113`.

2. Edit openrvdas/gui/settings.py, find the line where HOSTNAME is set and change it to

   ```
   HOSTNAME = '10.0.0.113'
   ```

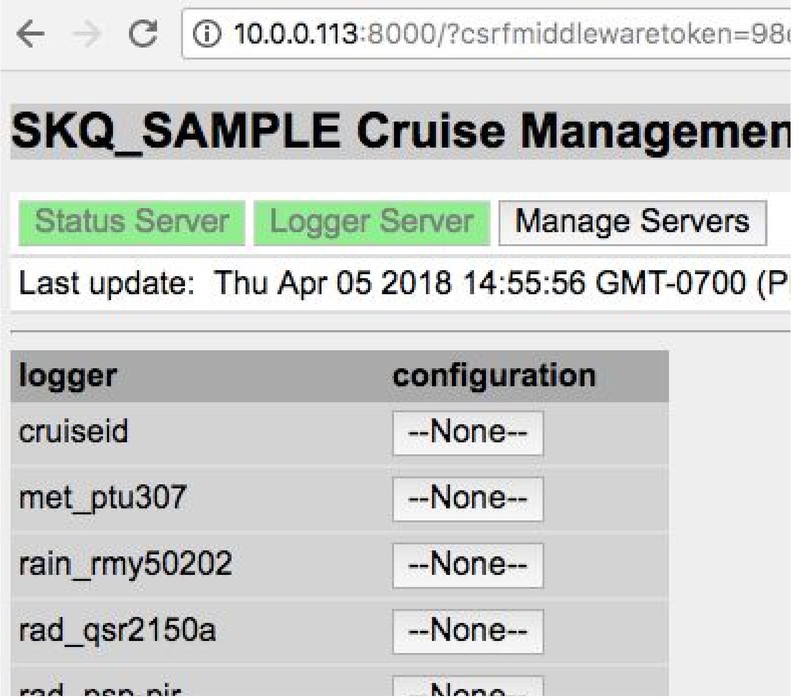
3. From the project directory (/home/rvdas/openrvdas), start the Django test server:

   ```
   python3 manage.py runserver 10.0.0.113:8000
   ```

4. In a separate terminal, also from the project directory, start the openrvdas servers:

   ```
   gui/run_servers.py
   ```

You should now be able to open a browser window to `http://10.0.0.113:8000` and see the web interface. Log into the interface as user rvdas (password rvdas), and the screen should look like this:

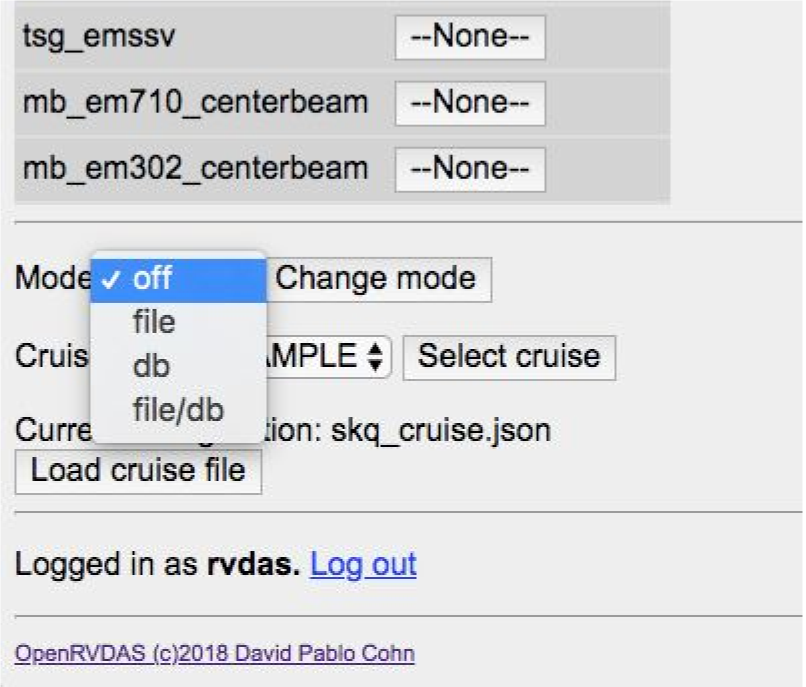


SKQ_SAMPLE Cruise Managemen

Last update: Thu Apr 05 2018 14:55:56 GMT-0700 (P

logger	configuration
cruiseid	--None--
met_ptu307	--None--
rain_rmy50202	--None--
rad_qsr2150a	--None--
rad_psp_pir	--None--

You can select any of the available configurations for a logger by clicking its configuration button. At the bottom of the page you can also select modes for the current cruise as well as switch between loaded cruise configurations:



tsg_emssv	--None--
mb_em710_centerbeam	--None--
mb_em302_centerbeam	--None--

Curre... ion: skq_cruise.json

Logged in as **rvdas**.

OpenRVDAS (c)2018 David Pablo Cohn

The supplied VM has three cruise configurations loaded:

- SKQ_SAMPLE - the configuration described in the previous section
- SKQ_6224 - a version of the previous configuration in which all loggers share UDP port 6224 to simplify testing (each logger configuration in this case includes a RegexFilterTransform to ensure that it only processes and stores UDP records matching the logger's name).[1]
- NBP1700 - minimal setup for a fictitious cruise involving serial port loggers. Running these loggers will require also setting up simulated serial ports using the `simulate_serial.py` script; see Running OpenRVDAS Loggers for details.

You can, of course, generate and load others if you wish.

[1] To exercise this configuration, start the loggers running, then in a separate window run

```
logger/listener/listen.py --file skq/sikuliaq.data --write_network :6224 -v
```