pyRSKTools Documentation

Release 0.1.8

RBR Ltd

Aug 18, 2021

Contents:

1 Introduction **1**
- 1.1 What does version 0.y.z version mean? . . . 1

2 Table of Contents **3**
- 2.1 Installing . . . 3
- 2.2 Testing . . . 3
- 2.3 Using . . . 4

3 Indices and Tables **9**

CHAPTER 1

Introduction

pyRSKTools is a simple Python toolbox to open RSK SQLite files generated by RBR instruments. Its functionality is read-only. It is a partial port of the MATLAB-based RSKTools and is in initial stages of development.

pyRSKTools targets Python 3.

1.1 What does version 0.y.z version mean?

pyRSKTools is in initial development. Per SemVer, the API is liable to change drastically and suddenly. Additionally, data returned may be inaccurate, incomplete, or just plain incorrect. Think we're doing the wrong thing? Think we could do something better? Tell us: now is the best time to fix it.

CHAPTER 2

Table of Contents

2.1 Installing

PyPI should have a reasonably up-to-date version available:

```
$ pip3 install pyrsktools
```

If you want to install the latest development version from source, you can do that too:

```
$ git clone https://bitbucket.org/rbr/pyrsktools
$ cd pyrsktools
$ make install
$ # Or, if you prefer to do it yourself...
$ pip3 install -e .
```

2.2 Testing

A clean bill of health from both unit tests and the linter is required for a successful automated build.

2.2.1 Unit Tests

To test the project, set the `RSK` environment variable to point at an RSK file:

```
$ export RSK=~/Downloads/some_rsk.rsk
```

Then run the unit tests:

```
$ make test
```

2.2.2 Linting

To analyze the project for errors with Pyflakes, use the `lint` make target:

```
$ make lint
```

2.3 Using

2.3.1 The Essentials

Opening a Dataset

The module includes a function to open an RSK file:

```
>>> import pyrsktools # Import the library
>>> rsk = pyrsktools.open('some_rsk.rsk') # Load up an RSK
```

This returns an `RSK` object against which all other library operations are performed.

Once you're finished with the dataset, it should be closed:

```
>>> rsk.close()
```

`open` can also be used with the `with` statement:

```
>>> with pyrsktools.open('some_rsk.rsk') as rsk:
>>>     # Do something with the RSK. It will be automatically closed
>>>     # at the end of the block.
```

What's Inside?

Metadata

The RSK provides some basic metadata about itself:

```
>>> rsk.name # What was the filename of the RSK?
'080281_20150911_1112.rsk'
```

the instrument that recorded it:

```
>>> rsk.instrument # What instrument was used?
Instrument(serial=80281, model='RBRmaestro', firmware_version='1.2', firmware_
↪type=103)
>>> rsk.channels # What channels were present on the instrument?
OrderedDict([('conductivity_00', Channel(id=1, key='cond06', label='conductivity_00',␣
↪name='Conductivity', units='mS/cm', derived=False)), ('temperature_00',␣
↪Channel(id=2, key='temp09', label='temperature_00', name='Temperature', units='°C',␣
↪derived=False)), ('pressure_00', Channel(id=3, key='pres19', label='pressure_00',␣
↪name='Pressure', units='dbar', derived=False)), ('oxygensaturation_00',␣
↪Channel(id=4, key='doxy09', label='oxygensaturation_00', name='Dissolved O₂', units='
↪%', derived=False)), ('chlorophyll_00', Channel(id=5, key='fluo10', label=
↪'chlorophyll_00', name='Chlorophyll a', units='µg/l', derived=False)), ('cdom_00',␣
↪Channel(id=6, key='fluo11', label='cdom_00', name='CDOM', units='ppb',␣
↪derived=False)), ('turbidity_00', Channel(id=7, key='turb01', label='turbidity_00',␣
↪name='Turbidity', units='NTU', derived=False)), ('seapressure_00', Channel(id=8,␣
↪key='pres08', label='seapressure_00', name='Sea pressure', units='dbar',␣
↪derived=True)), ('depth_00', Channel(id=9, key='dpth01', label='depth_00', name=
↪'Depth', units='m', derived=True)), ('salinity_00', Channel(id=10, key='sal_00',␣
↪label='salinity_00', name='Salinity', units='PSU', derived=True))])
```

(continues on next page)

(continued from previous page)

and the deployment:

```
>>> rsk.deployment
Deployment(id=1, comment='', logger_status=None, logger_time_drift=0, download_
↪time=datetime.datetime(2015, 9, 11, 7, 12, 30, 905000, tzinfo=datetime.timezone.
↪utc), name='080281_20150911_1112.rsk', sample_size=6711588)
```

Samples

But you probably care most about the sample data. Samples can be accessed in two ways. They can always be accessed iteratively, via a generator:

```
>>> rsk.samples()
<generator object RSK.samples at 0x10741bf10>
>>> import itertools
>>> for sample in itertools.islice(rsk.samples(), 3):
...     sample
...
Sample(timestamp=datetime.datetime(2015, 8, 29, 8, 28, 9, 333000, tzinfo=datetime.
↪timezone.utc), conductivity_00=50.468727111816406, temperature_00=28.92376708984375,
↪ pressure_00=19.332664489746094, oxygensaturation_00=103.3949203491211, chlorophyll_
↪00=0.128173828125, cdom_00=0.0048828125, turbidity_00=1.0341796875, seapressure_
↪00=9.200664520263672, depth_00=9.144135475158691, salinity_00=30.400436401367188)
Sample(timestamp=datetime.datetime(2015, 8, 29, 8, 28, 9, 500000, tzinfo=datetime.
↪timezone.utc), conductivity_00=50.469181060791016, temperature_00=28.92388916015625,
↪ pressure_00=19.370471954345703, oxygensaturation_00=103.39202880859375,␣
↪chlorophyll_00=0.1822509765625, cdom_00=0.082763671875, turbidity_00=1.0419921875,␣
↪seapressure_00=9.238471984863281, depth_00=9.181711196899414, salinity_00=30.
↪40065574645996)
Sample(timestamp=datetime.datetime(2015, 8, 29, 8, 28, 9, 667000, tzinfo=datetime.
↪timezone.utc), conductivity_00=50.468833923339844, temperature_00=28.92388916015625,
↪ pressure_00=19.395383834838867, oxygensaturation_00=103.46443939208984,␣
↪chlorophyll_00=0.24127197265625, cdom_00=0.116455078125, turbidity_00=1.0439453125,␣
↪seapressure_00=9.263383865356445, depth_00=9.206469535827637, salinity_00=30.
↪400409698486328)
```

Or if NumPy is available, they can be retrieved into an array:

```
>>> rsk.npsamples()
array([ (datetime.datetime(2015, 8, 29, 8, 28, 9, 333000, tzinfo=datetime.timezone.
↪utc),   5.04687271e+01,  28.92376709,  19.33266449,  103.39492035,  0.12817383,  0.
↪00488281,  1.03417969,  9.20066452,  9.14413548,   3.04004364e+01),
       (datetime.datetime(2015, 8, 29, 8, 28, 9, 500000, tzinfo=datetime.timezone.
↪utc),   5.04691811e+01,  28.92388916,  19.37047195,  103.39202881,  0.18225098,  0.
↪08276367,  1.04199219,  9.23847198,  9.1817112 ,   3.04006557e+01),
       (datetime.datetime(2015, 8, 29, 8, 28, 9, 667000, tzinfo=datetime.timezone.
↪utc),   5.04688339e+01,  28.92388916,  19.39538383,  103.46443939,  0.24127197,  0.
↪11645508,  1.04394531,  9.26338387,  9.20646954,   3.04004097e+01),
       ...,
       (datetime.datetime(2015, 9, 11, 7, 11, 26, 833000, tzinfo=datetime.timezone.
↪utc),   5.70757780e-04,  21.03649902,  10.05975151,  105.14163208, -0.0090332 , -0.
↪18981934, -0.04785156, -0.07224846, -0.07180457,   1.03646256e-02),
       (datetime.datetime(2015, 9, 11, 7, 11, 27, tzinfo=datetime.timezone.utc),   9.
↪17379744e-04,  21.03649902,  10.06026268,  105.09892273, -0.01196289, -0.18041992, -
↪0.0390625 , -0.07173729, -0.07129654,   1.03617543e-02),
```

(continues on next page)

(continued from previous page)

```
       (datetime.datetime(2015, 9, 11, 7, 11, 27, 167000, tzinfo=datetime.timezone.
↪utc),  -3.77910328e-04,  21.03656006,  10.06090927,  105.13765717, -0.01171875, -0.
↪17102051, -0.02880859, -0.0710907 , -0.07065392,   0.00000000e+00)],
      dtype=[('timestamp', 'O'), ('conductivity_00', '<f8'), ('temperature_00', '<f8
↪'), ('pressure_00', '<f8'), ('oxygensaturation_00', '<f8'), ('chlorophyll_00', '<f8
↪'), ('cdom_00', '<f8'), ('turbidity_00', '<f8'), ('seapressure_00', '<f8'), ('depth_
↪00', '<f8'), ('salinity_00', '<f8')])
```

Data returned from both the `samples` and `npsamples` functions can be time-limited via the `start_time` and `end_time` named arguments:

```
>>> from datetime import datetime, timezone
>>> rsk.npsamples(start_time=datetime(2015, 9, 2, tzinfo=timezone.utc),
...               end_time=datetime(2015, 9, 3, tzinfo=timezone.utc))
array([ (datetime.datetime(2015, 9, 2, 0, 0, tzinfo=datetime.timezone.utc),  50.
↪78747177,  23.58898926,  97.16591644,  2.78220224,  0.44580078,  0.55810547, -0.
↪15380859,  87.03392029,  86.49918365,  34.35256195),
       (datetime.datetime(2015, 9, 2, 0, 0, 0, 167000, tzinfo=datetime.timezone.utc),␣
↪ 50.81333542,  23.61236572,  97.32424164,  2.77061081,  0.44104004,  0.54382324, -
↪0.15332031,  87.19224548,  86.65653992,  34.35404587),
       (datetime.datetime(2015, 9, 2, 0, 0, 0, 333000, tzinfo=datetime.timezone.utc),␣
↪ 50.84253311,  23.61846924,  97.51081848,  2.74815583,  0.42736816,  0.5559082 , -
↪0.15722656,  87.3788147 ,  86.84196472,  34.3714447 ),
       ...,
       (datetime.datetime(2015, 9, 2, 23, 59, 59, 500000, tzinfo=datetime.timezone.
↪utc),  47.95243073,  20.36456299,  113.82951355,  0.53426731,  0.95812988,  0.
↪65759277, -0.08203125,  103.69750977,  103.06039429,  34.69152832),
       (datetime.datetime(2015, 9, 2, 23, 59, 59, 667000, tzinfo=datetime.timezone.
↪utc),  47.94488144,  20.36938477,  113.88066864,  0.53906661,  0.96099854,  0.
↪65661621, -0.08105469,  103.74867249,  103.1112442 ,  34.681427  ),
       (datetime.datetime(2015, 9, 2, 23, 59, 59, 833000, tzinfo=datetime.timezone.
↪utc),  47.93851089,  20.36968994,  113.9302597 ,  0.53700191,  0.96630859,  0.
↪63171387, -0.06982422,  103.79826355,  103.16053009,  34.67597961)],
      dtype=[('timestamp', 'O'), ('conductivity_00', '<f8'), ('temperature_00', '<f8
↪'), ('pressure_00', '<f8'), ('oxygensaturation_00', '<f8'), ('chlorophyll_00', '<f8
↪'), ('cdom_00', '<f8'), ('turbidity_00', '<f8'), ('seapressure_00', '<f8'), ('depth_
↪00', '<f8'), ('salinity_00', '<f8')])
```

The values given are expected to be `datetime` objects.

Sample data is intended to be easily explorable through the use of named fields:

```
>>> # The average of the first 6,000 temperature values:
>>> temperatures = [row.temperature_00 for row in itertools.islice(rsk.samples(), 6_
↪000)]
>>> sum(temperatures) / len(temperatures)
26.74410189819336
>>> # All salinity values from the start of the dataset to a cutoff date:
>>> rsk.npsamples(end_time=datetime(2015, 8, 29, 13, 0, tzinfo=datetime.timezone.
↪utc))['salinity_00']
array([ 30.4004364 ,  30.40065575,  30.4004097 , ...,  32.32573318,
    32.32009506,  32.31411743])
```

Geographic

The iOS and Android apps collect GPS geodata, which is accessible via the `geodata` function:

```
>>> rsk.geodata()
<generator object RSK.geodata at 0x1245cdf48>
>>> import itertools
>>> for geo in itertools.islice(rsk.geodata(), 3):
...     geo
...
Geo(timestamp=datetime.datetime(2019, 9, 27, 16, 24, 45, 145000, tzinfo=datetime.
↪timezone.utc), latitude=48.66791163945951, longitude=-123.38619064505487,␣
↪accuracy=7.074523989379777, accuracyType='HoriPhone')
Geo(timestamp=datetime.datetime(2019, 9, 27, 16, 24, 46, 145000, tzinfo=datetime.
↪timezone.utc), latitude=48.667910759559035, longitude=-123.38618598100749,␣
↪accuracy=6.162333509464028, accuracyType='HoriPhone')
Geo(timestamp=datetime.datetime(2019, 9, 27, 16, 24, 47, 145000, tzinfo=datetime.
↪timezone.utc), latitude=48.66790415214656, longitude=-123.38618235385722,␣
↪accuracy=5.5269390333275465, accuracyType='HoriPhone')
```

Regions

If you deployed your instrument with cast detection enabled, you can easily work with the data from each profile or cast.

The `profiles` function provides a generator to access all profiles:

```
>>> rsk.profiles()
<generator object RSK._query_regions at 0x10741bf10>
>>> next(rsk.profiles()) # Grab the first profile.
Region(start_time=datetime.datetime(2015, 8, 29, 8, 28, 9, 333000, tzinfo=datetime.
↪timezone.utc), end_time=datetime.datetime(2015, 8, 29, 8, 35, 14, 667000,␣
↪tzinfo=datetime.timezone.utc), label='', description=None)
```

And directional casts can be accessed by direction:

```
>>> next(rsk.casts(pyrsktools.Region.CAST_DOWN)) # Downcasts...
Region(start_time=datetime.datetime(2015, 8, 29, 8, 28, 9, 333000, tzinfo=datetime.
↪timezone.utc), end_time=datetime.datetime(2015, 8, 29, 8, 31, 45, 333000,␣
↪tzinfo=datetime.timezone.utc), label='', description=None)
>>> next(rsk.casts(pyrsktools.Region.CAST_UP)) # ...and upcasts.
Region(start_time=datetime.datetime(2015, 8, 29, 8, 31, 45, 333000, tzinfo=datetime.
↪timezone.utc), end_time=datetime.datetime(2015, 8, 29, 8, 35, 14, 667000,␣
↪tzinfo=datetime.timezone.utc), label='', description=None)
```

The `Region` object returned by both of these methods provides access to the limited range of samples pertinent to the region in time during which the profile or cast occurred:

```
>>> cast = next(rsk.casts(pyrsktools.Region.CAST_UP))
>>> cast.samples()
<generator object RSK.samples at 0x10741bf10>
>>> cast.npsamples()
array([ (datetime.datetime(2015, 8, 29, 8, 31, 45, 333000, tzinfo=datetime.timezone.
↪utc),  47.28219604,  19.76168823,  115.52977753,   1.56193031e-02,  0.48699951,  0.
↪51086426, -0.17822266,  105.39778137,  104.75022125,  34.63968658),
       (datetime.datetime(2015, 8, 29, 8, 31, 45, 500000, tzinfo=datetime.timezone.
↪utc),  47.18008041,  19.68182373,  115.39983368,   7.59091694e-03,  0.49487305,  0.
↪515625  , -0.17773438,  105.26783752,  104.62107849,  34.62173462),
       (datetime.datetime(2015, 8, 29, 8, 31, 45, 667000, tzinfo=datetime.timezone.
↪utc),  47.1333313 ,  19.57336426,  115.24497223,   1.43164247e-02,  0.48565674,  0.
↪52685547, -0.18701172,  105.11297607,  104.46716309,  34.67309189),
```

(continues on next page)

(continued from previous page)

```
       ...,
       (datetime.datetime(2015, 8, 29, 8, 35, 14, 167000, tzinfo=datetime.timezone.
↪utc),  50.45388412,  28.91912842,   11.95853233,   1.04415199e+02,  0.35913086,  0.
↪32739258, -0.12060547,    1.82653236,    1.81531024,  30.39524841),
       (datetime.datetime(2015, 8, 29, 8, 35, 14, 333000, tzinfo=datetime.timezone.
↪utc),  50.45434189,  28.91921997,   11.91053104,   1.04440277e+02,  0.37084961,  0.
↪29492188, -0.11181641,    1.77853107,    1.76760387,  30.39551353),
       (datetime.datetime(2015, 8, 29, 8, 35, 14, 500000, tzinfo=datetime.timezone.
↪utc),  50.45206451,  28.91882324,   11.87156296,   1.04411316e+02,  0.36724854,  0.
↪30310059, -0.11621094,    1.73956299,    1.72887516,  30.39424133)],
      dtype=[('timestamp', 'O'), ('conductivity_00', '<f8'), ('temperature_00', '<f8
↪'), ('pressure_00', '<f8'), ('oxygensaturation_00', '<f8'), ('chlorophyll_00', '<f8
↪'), ('cdom_00', '<f8'), ('turbidity_00', '<f8'), ('seapressure_00', '<f8'), ('depth_
↪00', '<f8'), ('salinity_00', '<f8')])
```

2.3.2 Useful Things

Plotting

All of these examples depend on NumPy and Matplotlib:

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
```

and are run against the first upcast found in our dataset:

```
>>> samples = next(rsk.casts(pyrsktools.Region.CAST_UP)).npsamples()
```

Time-Series

```
>>> plt.title('Time-Series Example')
>>> plt.xlabel('Time')
>>> plt.ylabel(rsk.channels['temperature_00'].label())
>>> plt.plot(samples['timestamp'], samples['temperature_00'])
>>> plt.savefig('timeseries.svg')
```

Depth Plot

```
>>> plt.title('Depth Plot Example')
>>> plt.xlabel(rsk.channels['salinity_00'].label())
>>> plt.ylabel(rsk.channels['depth_00'].label())
>>> plt.gca().invert_yaxis()
>>> plt.plot(samples['salinity_00'], samples['depth_00'])
>>> plt.savefig('depthplot.svg')
```

CHAPTER 3

Indices and Tables

- genindex
- search