Shape-Link Documentation

Release 0.1.1

Paul Müller

Feb 05, 2021

CONTENTS:

This is Shape-Link, a Python library and comman-line utility for interfacing with Shape-In, the acquisition software for RT-DC. This is the documentation of Shape-Link version 0.1.1.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

To install Shape-Link, use one of the following methods:

- **from PyPI:** `pip install shapelink`
- **from sources:** `pip install .`

1.2 Citing shapelink

If you use shapelink in a scientific publication, please cite it with:

> Philipp Rosendahl, Paul Müller (2021), Shape-Link version X.X.X: Python library for interfacing with Shape-In [Software]. Available at https://github.com/ZELLMECHANIK-DRESDEN/shapelink.

CHAPTER

TWO

COMMAND-LINE INTERFACE

Shape-Link comes with a command-line-interface (CLI) for running plugins.

2.1 shape-link

```
shape-link [OPTIONS] COMMAND [ARGS]...
```

2.1.1 run-plugin

Run a Shape-Link plugin file

Example usages:

```
# run a plugin
shape-link run-plugin plugins/slp_rolling_mean.py
# run a plugin with a simulator thread (for plugin testing)
shape-link run-plugin -w data.rtdc -f image,deform slp_rolling_mean.py
```

```
shape-link run-plugin [OPTIONS] PATH
```

Options

-w, --with-simulator <with_simulator>
Run the Shape-In simulator in the background using the RT-DC dataset specified (used for testing).

-f, --features <features>
Comma-separated list of features to send by the Shape-In simulator; Defaults to all innate features. A list of valid feature names can be found in the dclab docs (Advanced Usage -> Notation).

Arguments

PATH
Required argument

2.1.2 run-simulator

Run the Shape-In simulator using data from an RT-DC dataset file

Example usage:

```
shape-link run-simulator --features image,deform /path/to/data.rtdc
```

```
shape-link run-simulator [OPTIONS] PATH
```

Options

-f, --features <features>
Comma-separated list of features to send by the Shape-In simulator; Defaults to all innate features. A list of valid feature names can be found in the dclab docs (Advanced Usage -> Notation).

Arguments

PATH
Required argument

CHAPTER

THREE

WRITING PLUG-INS

A Shape-Link plug-in is a Python script with a class derived from *shapelink.ShapeLinkPlugin* and some additional meta data. Let's have a look at `this example plugin` which prints the rolling mean of a few scalar features to stdout:

```
import shutil

import numpy as np

from shapelink import ShapeLinkPlugin

# We use the terminal width to make sure a line doesn't get cluttered
# with prints from a previous line.
TERMINAL_WIDTH = shutil.get_terminal_size((80, 20))[0]


class RollingMeansPlugin(ShapeLinkPlugin):
    """Displays a rolling mean of a few scalar features"""
    def __init__(self, *args, **kwargs):
        super(RollingMeansPlugin, self).__init__(*args, **kwargs)
        self.window_size = 100
        self.scalar_data = {}

    def after_register(self):
        print(" Preparing for transmission")
        for feat in self.registered_data_format.scalars:
            self.scalar_data[feat] = np.zeros(self.window_size) * np.nan

    def after_transmission(self):
        print("\n End of transmission\n")

    def handle_event(self, event_data):
        """Handle a new event"""
        window_index = event_data.id % self.window_size
        for ii, feat in enumerate(self.registered_data_format.scalars):
            self.scalar_data[feat][window_index] = event_data.scalars[ii]
        # print the first three features to stdout
        msgs = [" Rolling means: "]
        num_prints = min(3, len(self.registered_data_format.scalars))
        for ii in range(num_prints):
            feat = self.registered_data_format.scalars[ii]
            msgs.append("{}: {:.3g}".format(feat,
                                            np.mean(self.scalar_data[feat])))
        line = "  ".join(msgs)
        if len(line) < TERMINAL_WIDTH:
```

(continues on next page)

(continued from previous page)

```
            line += " " * (TERMINAL_WIDTH - len(line))
        print(line, end="\r", flush=True)

        return False


info = {
    "class": RollingMeansPlugin,
    "description": "Displays a rolling mean of a few scalar features",
    "name": "Rolling Means",
    "version": "0.1.0",
}
```

The main action happens in the *handle_event* function - your plugin **must** implement at least this function. The two functions *after_register* and *after_transmission* can be used to set things up (e.g. creation of an additional output file) or to tear things down (e.g. closing that file). Use the `__init__` function for defining additional class properties. The `info` dictionary is required so that the plugin can be run via the *Command-line interface*.

CHAPTER

FOUR

CODE REFERENCE

4.1 shapelink.shapelink_plugin

Receive data in real-time from a Shape-In instance via zmq

class shapelink.shapelink_plugin.**EventData**

class shapelink.shapelink_plugin.**ShapeLinkPlugin**(*bind_to='tcp://*:6666'*, *verbose=False*)

Shape-Link plug-in meta class

Parameters

- **bind_to** (*str*) – IP and port to bind to (where Shape-In runs)
- **verbose** (*bool*) – Set to *True* to see additional debugging information.

after_register()
Called after registration with Shape-In is complete

after_transmission()
Called after Shape-In ends data transmission

abstract handle_event(*event_data:* shapelink.shapelink_plugin.EventData) → bool
Abstract method to be overridden by plugins implementations

handle_messages()
Handle messages from Shape-In

Please don't override this function. Use *ShapeLinkPlugin.handle_event()* for your customized plugins.

4.2 shapelink.util

Utility functions

These functions replicate QDataStreams behavior in C++. In PySide2 QDataStream does not accept array type data.

In C++ an array is serialized by writing: 1) Uint32 number of elements 2) type array elements

more significant bytes are written first. (big-endian)

if numpy "to_bytes" is used the native little-endian format appears

shapelink.util.**qstream_read_array**(*stream:* *PySide2.QtCore.QDataStream*, *datatype:* *numpy.dtype*) → numpy.array
Read array data from a stream with a specified type

shapelink.util.**qstream_write_array**(*stream: PySide2.QtCore.QDataStream*, *array: numpy.array*) → int
Write array data to a stream with a specified type :param stream: :param array: :return:

4.3 shapelink.shapein_simulator

Simulate a Shape-In instance

The communication is based on a simple REQ REP pattern all methods return when the transmission was acknowledged by the peer.

class shapelink.shapein_simulator.**ShapeInSimulator**(*destination='tcp://localhost:6666'*, *verbose=False*)

register_parameters(*scalar_hdf5_names=None*, *vector_hdf5_names=None*, *image_hdf5_names=None*, *settings_names=None*, *settings_values=None*)
Register parameters that are sent to other processes

send_end_of_transmission()
Send end of transmission packet

send_event(*event_id: int*, *scalar_values: numpy.array*, *vector_values: List[numpy.array]*, *image_values: List[numpy.array]*) → bool
Send a single event to the other process

shapelink.shapein_simulator.**start_simulator**(*path*, *features=None*, *verbose=1*)
Run a Shape-In simulator using data from an RT-DC dataset

4.4 shapelink.msg_def

Definitions for message ids (numeric)

shapelink.msg_def.**MSG_ID_EOT = -10**
end of transmission

shapelink.msg_def.**MSG_ID_EOT_ACK = -11**
end of transmission acknowledged

shapelink.msg_def.**MSG_ID_REGISTER = -1**
registration

shapelink.msg_def.**MSG_ID_REGISTER_ACK = -2**
registration acknowledged

CHAPTER

FIVE

CHANGELOG

List of changes in-between Shape-Link releases.

5.1 version 0.1.1

- docs: add section for writing plugins

5.2 version 0.1.0

- feat: rudimentary command-line interface for running plugins

5.3 version 0.0.1

- initial release

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

E

H

M

P

Q

R

S