developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

May 18, 2020

Contents

1 Installation **3**
- 1.1 Dependencies . . . 3
- 1.2 This package . . . 4

2 Receiver **5**
- 2.1 Configuration options . . . 5
- 2.2 Running . . . 6
- 2.3 Running Multiple Receivers . . . 6

3 Sender **7**
- 3.1 Configuration options . . . 7

4 API documentation **9**
- 4.1 Packetisers . . . 9
- 4.2 Transmitters . . . 9
- 4.3 Payloads . . . 9
- 4.4 Receivers . . . 9
- 4.5 Others . . . 10

5 CBF-SDP Interface Emulator - Quick Start **11**
- 5.1 Running the Simple Send-Receive Pair . . . 11
- 5.2 Install . . . 11
- 5.3 Testing . . . 12
- 5.4 Code analysis . . . 12
- 5.5 Writing documentation . . . 12

Python Module Index **15**

Index **17**

These are all the packages, functions and scripts that form part of the project.

CHAPTER 1

Installation

1.1 Dependencies

Optionally, first create a virtual environment.

This package uses the OSKAR MS libraries for doing Measurement Set reading and writing. The OSKAR project contains numerous other functionality though which can be more expensive to build and install. To install **only** this part of the C++ library follow these instructions:

```
# Install dependencies and build tool
apt-get -y update
apt-get -y install cmake libblas-dev liblapack-dev casacore-dev

# Get a copy of OSKAR
git clone https://github.com/OxfordSKA/OSKAR.git
mkdir OSKAR/oskar/ms/release
cd OSKAR/oskar/ms/release

# Add -DCMAKE_INSTALL_PREFIX=$VIRTUAL_ENV if installing into a virtual
# environment
cmake -DCASACORE_LIB_DIR=/usr/lib/x86_64-linux-gnu ..

# Build and install
make -j4
make install

# Got back to the root directory where you started
cd ../../../..
```

Our package does not use the C++ OSKAR MS library directly but its python bindings. However, the OSKAR python bindings package does not support out of the box to install **only** the MS part of the bindings. Thus, the following steps will be required:

```
# Get a copy of this package
git clone https://gitlab.com/ska-telescope/cbf-sdp-emulator

# Use this package's copy of setup.py to build OSKAR's python bindings
cd OSKAR/python
mv ../../cbf-sdp-simulator/3rdParty/setup_oskar_ms.py ./setup.py
python3 ./setup.py build
python3 ./setup.py install
cd ../../
```

1.2 This package

This is a standard setuptools-based program so the usual installation methods should work:

```
# Go into the top-level directory of this repository
cd cbf-sdp-emulator

# Using "pip" will automatically install any dependencies from PyPI
pip install .

# Use pip in editable mode if you are actively changing the code
pip install -e .
```

CHAPTER 2

Receiver

An `emu-recv` program should be available after installing the package. This program receives packets. It is an extensible package that is built to be agnostic to the actual mode of reception.

2.1 Configuration options

Configuration options can be given through a configuration file or through the command-line. See `emu-recv -h` for details. An example configuration file is given in 'example.conf'.

The following configuration categories/names are supported for the SPEAD2 receivers - other receivers can be added that are not SPEAD2 compliant. But currently as of this initial version we are only supplying the SPEAD2 UDP receiver. The API for adding more receivers in this framework is supplied:

- `reception`: these are configuration options applied when reading the input Measurement Set.
- `bind_hostname`: The IP address or hostname of the interface to which to bind for reception.
- `receiver_port_start`: The initial port number to which to bind.
- `num_ports`: [optional] The number of ports to which to bind. This can also be calculated from the 'channels_per_stream' option and the number of channels in the data-model.
- `consumer`: [spead2_mswriter] Which class to attach to the receiver. The default class is a measurement set writer. But others can be added.
- `datamodel`: This is important. We have no interface with the telescope model (TM) and therefore have to obtain all the observation metadata from somewhere. We have decided to use the Measurement Set as the basis for this. So you should supply a measurement set that contains the same metadata as that which is being sent. it does not have to be the same file - but the output measurement set parameters and the UVW will be taken from this file. We open and close this file quickly so there should be no issue with multiple open files. Providing the receiver is started before the sender.
- `transmission`: these are options that generally apply to the transmission method and are mostly ignored by the receiver except:

- `channels_per_stream`: The number of channels for which data will be sent in a single stream. This is used in the case where multiple ports are required with multiple channels per port. You dont actually need this - but without it you have to set num_ports appropriately

2.2 Running

The application is really simple to run. It is installed as an entrypoint when you install the package and will run as simply as: `emu_recv -c <configuration_file>` Or you can specify the various options on the command line. A typical configuratio file is supplied and looks like this:

```
[transmission]

method = spead2_transmitters
target_host = 127.0.0.1
target_port_start = 41000
channels_per_stream = 1
rate = 247000

[reader]

[payload]

method = icd

[reception]

method = spead2_receivers
receiver_port_start = 41000
consumer = spead2_mswriter
datamodel = tests/data/gleam-model.ms
outputfilename = tests/data/recv-vis.ms
```

Note by default we use a spead2 transmitter and receiver and a `icd` payload. We have designed this package to be extensible and if you want to add different transmitters and receivers you should be able to

In practical terms it makes sense to start the receiver(s) before the transmitter so they are waiting for data. But you do not have to - the protocols and consumers are flexible enough to be started when the data-stream is already running.

2.3 Running Multiple Receivers

In many places we have made design decisions that are common to the SKA-SA systems. In the case of multiple consumers we are simply using UDP multicast. THis means that multiple consumers can access the same transmitted stream if they bind to the same multicast IP address and ports. We have examples of this operations in the example configurations.

This multiple-receiver operation is required when displays and monitoring is required to sample the data in transit - before it becomes a measurement set.

CHAPTER 3

Sender

An emu-send program should be available after installing the package. This program takes a Measurement Set and transmits it over the network using the preferred transmission method.

3.1 Configuration options

Configuration options can be given through a configuration file or through the command-line. See emu-send -h for details.

The following configuration categories/names are supported:

- reader: these are configuration options applied when reading the input Measurement Set.
- start_chan: the first channel for which data is read. Channels before this one are skipped. If start_chan is bigger than the actual number of channels in the input MS an error is raised.
- num_chan: number of channels for which data is read. If num_chan + start_chan are bigger than the actual number of channels in the input MS then num_chan is adjusted.
- num_repeats: number of times a single set of visibilities should be sent after being read, defaults to 1. Bigger values will send the same data over and over, which is less realistic but imposes less stress on the file-system.
- transmission: these are options that apply to the transmission method.
- method: the transmission method to use, defaults to spead2.
- target_host: the host where data will be sent to.
- target_port_start: the first port where data will be sent to.
- channels_per_stream: number of channels for which data will be sent in a single stream.
- max_packet_size: the maximum size of packets to build, used by spead2.
- rate: the maximum send data rate, in bytes/s. Used by spead2, defaults to 1 GB/s.

CHAPTER 4

API documentation

This section describes requirements and guidelines.

4.1 Packetisers

We begin with the packetiser we have written as a default `emulator` this is a pretty simple package that uses the transmitter and payload classes defined in the configuration to send data.

At the moment we have an assumption that the ICD payload is being used. but minor changes to the packetise method would remove that requirement. Very minimal work is needed to replicate this with another payload.

The actual transmission protocol is abstracted into the `transmitters` and this is currently defaulting to SPEAD2 and UDP. But as this is almost completely abstracted should be easy to change.

4.2 Transmitters

The transmitters are envisaged to be at least as diverse as UDP, IBV and perhaps ROCE we have only implemented the UDP transmitter. But extensions should be trivial

4.3 Payloads

4.4 Receivers

UDP Protocol Multi-stream SPEAD2 receiver

class `cbf_sdp.receivers.spead2_receivers.`**`receiver`**(*config*, *tm*, *loop*)

SPEAD2 receiver

This class uses the spead2 library to receive a multiple number of streams, each using a single UDP reader. As heaps are received they are given to a single consumer.

run()
    Receive all heaps, passing them to the consumer

4.5 Others

CHAPTER 5

CBF-SDP Interface Emulator - Quick Start

Ok so you don't want to read all the documentation, or just want to get something running straight away. Open in the quickstart directory and you will find some simple configuration files.

There are quickstart examples for the following situations:

1. A simple send and receive pair for a small number of channels on a single stream. The basic and simplest scheme - this will not expected to scale beyond a few hundred channels.
2. A simple send and receive pair for a larger number of channels using multiple streams but a single output file. This employs a multi-threaded asynchronous receive and should scale. Although the performance may be limited by disk performance - both sending and receiving.
3. A send and receive using a multicast to send to two separate receivers simultaneously one writing the data to disk and the other not. To mimic the functionality of multiple consumers.
4. A send and receive using multiple senders to many receivers using multicast each writing their own separate measurement sets. This may be required as the the array continues to grow and if the PSI or early releases employ more than one CBF-node.

Each experiment is in its own directory - example data sets are included in the tests/data directory. You should not need to install anything other than this package to get them to work.

5.1 Running the Simple Send-Receive Pair

The system used for development needs to have Python 3 and `pip` installed.

5.2 Install

Always use a virtual environment. Pipenv is now Python's officially recommended method, but we are not using it for installing requirements when building on the CI Pipeline. You are encouraged to use your preferred environment isolation (i.e. `pip`, `conda` or `pipenv` while developing locally.

For working with `Pipenv`, follow these steps at the project root:

First, ensure that `~/.local/bin` is in your `PATH` with:

```
> echo $PATH
```

In case `~/.local/bin` is not part of your `PATH` variable, under Linux add it with:

```
> export PATH=~/.local/bin:$PATH
```

or the equivalent in your particular OS.

Then proceed to install pipenv and the required environment packages:

```
> pip install pipenv # if you don't have pipenv already installed on your system
> pipenv install
> pipenv shell
```

You will now be inside a pipenv shell with your virtual environment ready.

Use `exit` to exit the pipenv environment.

5.3 Testing

- Put tests into the `tests` folder
- Use PyTest as the testing framework
  - Reference: PyTest introduction
- Run tests with `python setup.py test`
  - Configure PyTest in `setup.py` and `setup.cfg`
- Running the test creates the `htmlcov` folder
  - Inside this folder a rundown of the issues found will be accessible using the `index.html` file
- All the tests should pass before merging the code

5.4 Code analysis

- Use Pylint as the code analysis framework
- By default it uses the PEP8 style guide
- Use the provided `code-analysis.sh` script in order to run the code analysis in the `module` and `tests`
- Code analysis should be run by calling `pylint cbf_sdp`. All pertaining options reside under the `.pylintrc` file.
- Code analysis should only raise document related warnings (i.e. `#FIXME` comments) before merging the code

5.5 Writing documentation

- The documentation generator for this project is derived from SKA's SKA Developer Portal repository

- The documentation can be edited under `./docs/src`
- If you want to include only your README.md file, create a symbolic link inside the `./docs/src` directory if the existing one does not work:

```
$ cd docs/src
$ ln -s ../../README.md README.md
```

- In order to build the documentation for this specific project, execute the following under `./docs`:

```
$ make html
```

- The documentation can then be consulted by opening the file `./docs/build/html/index.html`

Python Module Index

.

../cbf_sdp/transmitters/spead2_transmitters.py, 6

c

cbf_sdp.receivers.spead2_receivers, 9

Index

Symbols

`../cbf_sdp/transmitters/spead2_transmitters.py` (*module*), 6

C

`cbf_sdp.receivers.spead2_receivers` (*module*), 9

R

`receiver` (*class in cbf_sdp.receivers.spead2_receivers*), 9

`run()` (*cbf_sdp.receivers.spead2_receivers.receiver method*), 10