developer.skatelescope.org Documentation

Release 0.1.1

Marco Bartolini

Feb 25, 2022

HOME

Documentation Status

Simple package to both calculate derived data and amend measurement sets with other data. In general the input format can be another measurement set or a JSON formatted file.

CHAPTER

ONE

REQUIREMENTS

The system used for development needs to have Python 3 and `pip` installed. Also, the CASA Measures package needs to be up-to-date. This can be pulled from Westerbork Radio Telescope by the following steps:

```
> cd /var/lib/casacore/data
> cd wget ftp://ftp.astron.nl/outgoing/Measures/WSRT_Measures.ztar
> cd mv WSRT_Measures.ztar WSRT_Measures.tar.gz
> cd gunzip WSRT_Measures.tar.gz
> cd tar -xvf WSRT_Measures.tar
```

CHAPTER

TWO

INSTALL A VIRTUAL ENVIRONMENT

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.

CHAPTER

THREE

INSTALL THE PACKAGE

The package can be installed from a local checkout of this repository by:

```
> python3 -m pip install --extra-index-url==https://artefact.skao.int/repository/pypi-
↪internal/simple -e .
```

Or directly from the artefact repository by

```
> python3 -m pip install --extra-index-url==https://artefact.skao.int/repository/pypi-
↪internal/simple ska-sdp-msadd
```

CHAPTER

FOUR

RUNNING THE TESTS

The current test suite can be ran by

```
> make test
```

CHAPTER

FIVE

ADDING TESTS

- 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

CHAPTER

SIX

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 ska_python_skeleton`. All pertaining options reside under the `.pylintrc` file.
- Code analysis should only raise document related warnings (i.e. `#FIXME` comments) before merging the code

CHAPTER

SEVEN

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`

CHAPTER

EIGHT

DEVELOPMENT

8.1 PyCharm

As this project uses a `src` folder structure, under *Preferences* > *Project Structure*, the `src` folder needs to be marked as "Sources". That will allow the interpreter to be aware of the package from folders like `tests` that are outside of `src`. When adding Run/Debug configurations, make sure "Add content roots to PYTHONPATH" and "Add source roots to PYTHONPATH" are checked.

Todo:

- add the antenna table to the measurement set
- perhaps add arbitrary measurement set tables based on the contents of a JSON file
- add the information about the JSON table format

CHAPTER

NINE

MSADD - MEASUREMENT SET ADD DERIVED DATA

Simple tool to add derived products to existing measurement sets. Basic functionality is to add UVW in a catalog frame to a measurement set - based on the DELAY_DIR direction and the antenna positions. But it is envisioned that other tables will be added.

9.1 UVW Calculation and Addition

The scheme to perform this is based upon the CASACORE package and utilises the python-casacore wrappers to the measures functionality of those classes. There are other member functions of this package that do not utilise this functionality. .. Automatic API Documentation section. Generating the API from the docstrings. Modify / change the directory structure as you like

9.1.1 The Applications

msadd-uvw

The first application developed with this package is `msadd-uvw` it can be run from the command line. Options can be examined by `msadd-uvw -h`

Essentially this takes a direction, a set of antenna positions and updates an output measurement set with the UVW calculated by the utilities in the package.

The following options are supported:

- `-i, --input`: MSv2 input model file used for reference antenna positions, the `locations` option should really be used. (default: None)
- `-o, --output`: Output MSv2 - this is the table which needs updating. (default: None)
- `-l, --locations` : JSON input antenna-locations file - this can be a local location or a URL (default: None)
- `-d, --delay`: Delay Centre Over-ride – You should set this. Otherwise the tools will inspect the output measurement set for the FIELD table. The units are (hour, degree). The format is anything an Astropy SkyCoord will take so "00 42 30 +41 12 00" or "00:42.5 +41:12" are fine
- `f, --frame`: Reference frame for the delay direction (ICRS)
- `s, --swap`: Default order of baseline calculation is Antenna2 - Antenna1 - you can swap this if you want.

9.1.2 The Module Contents

The Antenna Table

A simple set of classes and functions to read and construct an antenna table in memory from either a reference measurement set of a JSON file. THe JSON file can be stored remotely and accessed via URL

class ska.sdp.msadd.antenna.**AntennaTable**(*input_table_name: Optional[str] = None*, *json_file_location=None*)

simple class that updates the antenna table using either a JSON object or another table from a reference MS The same class can deal with multiple reference options - in order of preference: It first checks if local MS reference table is given. Then it checks whether a remote file location is given. Finally it checks for a local json file.

get_antenna_pos(*index*)

Returns the XYZ antenna position as held in the internal representation of the antenna table

Parameters **index** (*int*) – The antenna index as listed in the relevant row of the MAIN table

Return position The XYZ position geocentric of the antenna

Return type [x, y, z]

load_table_from_json(*data*)

Create an internal array of antenna dictionaries using the JSON format as input

Parameters **data** (*dict*) – Dictionary structure as generated by json.loads - the assumption is that this remains in order

load_table_from_ms(*input_table_name*)

If the input table name exists - set the input table load the values into a dictionary

Parameters **input_table_name** (*str*) – The actual ANTENNA table in a measurement set

The UVW Table

These methods are used to calculate UVW based upon the contents of a telescope model (antenna locations, Time, look direction). The contents of which are obtained from the AntennaTable and a reference measurement set.

class ska.sdp.msadd.uvw.**UpdateUvwColumn**(*input_table_name=None*, *json_file_location=None*, *output_table_name=None*, *swap_baselines=False*, *delay_ra: typing.Optional[<MagicMock id='140025722400464'>] = None*, *delay_dec: typing.Optional[<MagicMock id='140025722400464'>] = None*)

A simple class that updates the UVW column in the main table - derived from ANTENNA locations in the ANTENNA table

calculate_uvw_for_row(*row*)

Calculate the UVW for the row of internal representation of the table. Note this does not update the row in the measurement set. The geocentric UVW of each antenna is calculated separately using its position - this is probably better than doing it to some reference position.

Parameters **row** (*int*) – The row to calculate

Return uvw The UVW in J2000 epoch

Return type [u,v,w]

get_antenna1_from_row(*row*)

Get the antenna index of antenna 1 for the given row

Parameters **row** (*int*) – row index

get_antenna2_from_row(*row*)
Get the antenna index of antenna 2 for the given row

Parameters **row** (*int*) – row index

get_antenna_position_from_index(*index*)
Get the antenna position from the internal representation of the antenna table

Parameters **index** (*int*) – index

get_delay_dir_from_table(*row=0*)
Pull the delay direction from the reference measurement set

Parameters **row** (*int*) – row to take it from (0)

Returns a list of form [ra,dec] in radian measure

Return type [target_ra, target_dec]

get_time_from_row(*row*)
Get the time from the internal representation of the main table row

Parameters **row** (*int*) – row index

Return time

Return type astropy.Time

load_columns_from_ms(*reference_table_name*)
If the input table name exists - set the input table load the values into a dictionary

load_from_output_table()
If the input table name exists - set the input table

update_uvw_for_all_rows()
Calculate the UVW for the all the rows of internal representation of the table. This does update the measurement set on disk

The Calculation Utilities

The actual work in calculating the UVW is performed by these functions. There are some utilities that perform these tasks from first principles and others that use casacore measures - as accessed via the python wrappers

ska.sdp.msadd.utils.**get_antenna_uvw**(*xyposA: <MagicMock id='140025722185488'>*, *epoch: <MagicMock id='140025722434896'>*, *ra: <MagicMock id='140025723585808'>*, *decl: <MagicMock id='140025723585808'>*, *position_frame='itrf'*, *epoch_frame='J2000'*, *swap_baselines=False*) → <MagicMock id='140025722198544'>
Return the geocentric uvw vector for the the given position at the given epoch.

Note: This actually just calls get_uvw_J2000 with the geocentre as the other antenna and this antenna as the reference position.

Note: This is the per antenna calculation that you probably want

Parameters

- **xyposA** (*list*) – position of the antenna (vector geocentric XYZ in m)
- **epoch** (*astropy.Time*) – Time for which to calculate the UVW
- **epoch_frame** (*str*) – Whether you want a catalog (J2000) frome or epoch of date (default: 'J2000')

- **position_frame** (*str*) – The frame of the input positions (generally WGS84 or ITRF) we are using casacore for this and these are the two frames supported
- **ra** (*astropy.Angle*) – Right Ascension (J2000)
- **decl** (*astropy.Angle*) – Declination (J2000)
- **swap_baselines** (*Bool*) – (xyposB - xyposA) is assumed - if the reverse is required set this to True

Returns The geocentric UVW baseline

Return type numpy.array [u,v,w]

ska.sdp.msadd.utils.**get_r**(*h*, *delta*) → <MagicMock id='140025722361232'>

Return the matrix that will project XYZ into UVW. This is the matrix that is typically used to get the transformation from a geocentric position to a UVW at the epoch of date. See A.R. Thompson, J.M. Moran, and G.W. Swenson Jr., Interferometry and Synthesis in Radio Astronomy (equation 4.3 in the Third Edition)

Parameters

- **h** (*double*) – – the Greenwich hour angle in radian
- **delta** (*double*) – – the declination in radian

Returns A numpy array of the matrix R

Return type numpy.array

ska.sdp.msadd.utils.**get_uvw**(*xyposA: list*, *xyposB: list*, *epoch: <MagicMock id='140025722434896'>*, *ra: <MagicMock id='140025723585808'>*, *decl: <MagicMock id='140025723585808'>*, *swap_baselines=False*) → <MagicMock id='140025722854096'>

Return the baseline vector for the 2 given positions in epoch of date

Parameters

- **xyposA** (*list*) – position of antenna 1 (vector geocentric XYZ in m)
- **xyposB** (*list*) – position of antenna 2 (vector geocentric XYZ in m)
- **epoch** (*astropy.Time*) – Time for which to calculate the UVW
- **ra** (*astropy.Angle*) – Right Ascension (J2000)
- **decl** (*astropy.Angle*) – Declination (J2000)
- **swap_baselines** (*Bool*) – (xyposB - xyposA) is assumed - if the reverse is required set this to True

Returns The uvw baseline at Epoch of Date

Return type numpy.array

ska.sdp.msadd.utils.**get_uvw_J2000**(*xyposA: <MagicMock id='140025722478032'>*, *xyposB: <MagicMock id='140025722600912'>*, *refpos: <MagicMock id='140025722142992'>*, *epoch: <MagicMock id='140025722434896'>*, *ra: <MagicMock id='140025723585808'>*, *decl: <MagicMock id='140025723585808'>*, *position_frame='itrf'*, *swap_baselines=False*) → <MagicMock id='140025722164240'>

Return the baseline vector for the 2 given positions at the J2000 Epoch. This ensures that any image formed by the Fourier transform of a UVW cube in this frame will itself be in the ICRS frame.

Note: I believe that the ICRS and J2000 are aligned <at> J2000 - but caveat emptor until I've sorted that out.

Note: This assumes you have a reference position at which all the sidereal angles are calculated but it is more likely that you want a "per antenna" calculation. So be sure you want this before you use it.

Parameters

- **xyposA** (*list*) – position of antenna 1 (vector geocentric XYZ in m)
- **xyposB** (*list*) – position of antenna 2 (vector geocentric XYZ in m)
- **epoch** (*astropy.Time*) – Time for which to calculate the UVW
- **ra** (*astropy.Angle*) – Right Ascension (J2000)
- **decl** (*astropy.Angle*) – Declination (J2000)
- **position_frame** (*str*) – The frame of reference for the positions ITRF is default - WGS84 is also supported
- **swap_baselines** (*Bool*) – (xyposB - xyposA) is assumed - if the reverse is required set this to True

Returns The uvw baseline at J2000

Return type numpy.array [u,v,w]

ska.sdp.msadd.utils.**time_to_quantity**(*in_time*)

I have found that for some reason I get either a string or a byte array as returned isot format from astropy.time There seems to be no reason to it - but it does seem to be a function of the value. Some dates are always byte arrays and some are always strings - this helper function just tests what it going on tries to return a valid casacore.quanta

Parameters **in_time** (*astropy.Time*) – astropy Time object

Returns a casacore.quantity of the same value

CHAPTER

TEN

MSADD - MEASUREMENT SET ADD DERIVED DATA

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

PYTHON MODULE INDEX

S

INDEX