developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Nov 10, 2022

HOME

Documentation Status

There is likely to be a wider implementation of a more capable telescope model - however for the purposes of quick SDP development this is fastAPI based server - which will deliver JSON formatted products - backed by a JSON formatted telescope model.

I have decided to isolate the server, the model and the model maintainer into three objects. The server will be this repository. The physical form of the model will be a JSON structure, The creation and maintenance of the JSON structure is provided by a third product.

CHAPTER

ONE

REQUIREMENTS

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

CHAPTER

TWO

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.

CHAPTER

THREE

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

CHAPTER

FOUR

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

FIVE

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

SIX

DEVELOPMENT

6.1 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:

- Insert todo's here

CHAPTER

SEVEN

TMLITE SERVER

This server uses fastAPI to serve the contents of the `prototype_model.json` file via a web interface.

7.1 Starting the Server

I would suggest that new users check out the documentation for fastAPI as this will clearly demonstrate how this is set up.

The simplest way to launch the current server is via `docker-compose`. Running the following: `docker-compose up --build`

This will execute the following docker-compose which will start a container running the server and exposing port 80 of the container:

```
version: '2'
services:
    tmlite:
        build:
        context: .
        dockerfile: Dockerfile
        container_name: ska-sdp-tmlite-server
        volumes:
            - "/var/run/docker.sock:/var/run/docker.sock"
        hostname: localhost
        expose:
            - "80"
        ports:
            - "80:80/tcp"
        command: ["uvicorn", "ska.tmlite.main:app", "--host", "0.0.0.0", "--port", "80"]
```

7.2 Accessing The Server

One of the advantages of fastAPI is that it self documents. So once you have the server running simply connect to it:

```
> docker-compose up --build
.....
> ska-sdp-tmlite-server | INFO:     Uvicorn running on http://0.0.0.0:80 (Press CTRL+C␣
↪to quit)
```

Assuming you kept the same port exposure open your browser at `http://localhost:80/docs`

For example the request to obtain the full model is:

```
>curl -X 'GET' 'http://localhost/model/current' -H 'accept: application/json'
```

And this will return the full model

7.3 The Model Structure

On server construction there are two models created a "default" model and a "current" model. The default model should not be altered but the current one can be changed in part. Also new models can be added and altered at will. WHen the server shuts down all alterations are lost. THis scheme is not for the long term storage of models - but the short term access of them.

7.4 Storage backends

The initial Telescope Model data is loaded from one of the supported storage backends. The storage to be used is selected by setting the `STORAGE_BACKEND` environment variable to the name of one of the supported storage backends.

Currently only a GitLab storage backend is supported,

7.4.1 `gitlab` backend

The `gitlab` storage backend loads a Telescope Model file from the SKA SDP TMLite data repository repository. Please read its documentation, as it explains how data is organised and presented. Note that like this project, the TMLite data repository is also currently designed to work in a read-only fashion from the standpoint of this TMLite server.

A set of environment variables control this process, all of which must be prefixed with `STORAGE_BACKEND__`:

- `CLONE_DIRECTORY` is the local directory holding the clone of the repository, defaults to `<OS-temp-dir>/tmplite_gitlab_repository`.
- `PRIVATE_TOKEN`, `JOB_TOKEN` and `OAUTH_TOKEN`, if defined, are used for authentication against GitLab.
- `MODEL_PATH` is the file to be loaded from the repository clone as the Telescope Model, it defaults to `prototype_model.json`.
- `REPOSITORY_REF` indicates the git reference (SHA, branch, tag) to retrieve when cloning the repository. If not given, the default repository branch is used.

If the `CLONE_DIRECTORY` doesn't exist then a clone of the repository is created in that location. If it exists no further action occurs. Note that this implies that this backend can be used to point to an existing directory/file in the local filesystem containing a valid Telescope Model, even if that directory is not a git repository.

At the moment, and as mentioned earlier, all editions are ephemeral: once the server shuts down they are all lost.

7.5 Example API tasks

Probably the simplest way to access this is via the python `requests` module - but of course for the GET methods you can even use a browser if you want.

7.5.1 Getting the full default model

A simple python api:

```
> import request
> url = "http://localhost/model/default/"
> response = requests.get(url)
```

7.5.2 Getting the current model

Is as simple as changing the URL:

```
> url = "http://localhost/model/current/"
```

But there are a number of methods coded up for you to get subsets of the model and even change things for example:

```
> url = "http://localhost/current/ska1_low/update_antennas"
> mccs = {"station_ids": ['0','1','2','3']}
> response = requests.post(url,json=mccs)
```

This will update the current model to only include antennas from the list.

7.6 Extending the Model and Server

This is a minimal `hello world` implementation - You should have enough information to extend the server to respond more smartly to requests and return slices through the model for example.

CHAPTER

EIGHT

PUBLIC API

The FastAPI is documented internally at `http://localhost:80/docs` once the server is running. The functions and classes currently implemented in the model are:

8.1 Models

class ska.tmlite.models.**Station**(*, *interface: str*, *station_name: str*, *diameter: str*, *properties: Optional[MCCSProperties] = None*, *location: Optional[TelModelLocation] = None*, *fixed_delays: List[TelModelFixedDelays] = []*, *niao: float*, *long: Optional[str] = None*, *lat: Optional[str] = None*, *x: Optional[float] = None*, *y: Optional[float] = None*, *z: Optional[float] = None*)

The schema for the SKA stations:

Parameters

- **name** – (str) an identifier for the station
- **dish_diameter** – (str) dish/stations size in meters
- **long** – (str) longitude [optional]
- **lat** – (str) latitude [optional]
- **x** – (float) Geocentric x
- **y** – (float) Geocentric y
- **z** – (float) Geocentric z

class ska.tmlite.models.**LayOut**(*, *description: Optional[str] = None*, *reference: Optional[str] = None*, *comment: Optional[str] = None*, *revision: Optional[str] = None*, *telescope: str*, *coordinates: Optional[str] = None*, *units: Optional[str] = None*, *receptors: List[Station] = []*)

The Schema for station layout

Parameters

- **description** – str identifier
- **reference** – str where did this come from (provinence?)
- **comment** – str
- **revision** – str
- **telescope** – str telescope name (dup)

- **coordinates** – str Coordinate frame (ITRF)
- **units** – str (meters etc
- **antennas** – Set[Station] = [] an array of Station

class ska.tmlite.models.**RFI**(*, *description: str*, *freq_start: str*, *freq_stop: str*)

RFI Schema

Parameters

- **description** – str identifier (what source etc)
- **freq_start** – str (Hz)
- **freq_stop** – str (Hz)

class ska.tmlite.models.**SKA1Mid**(*, *layout: LayOut*, *static_rfi_mask: Optional[List[RFI]] = None*)

Container item for the model for SKA1 Mid

Parameters

- **layout** – LayOut
- **static_rfi_mask** – Set[RFI] = []

class ska.tmlite.models.**SKA1Low**(*, *layout: LayOut*, *static_rfi_mask: Optional[List[RFI]] = None*)

Container item for the model for SKA1 Low

Parameters

- **layout** – LayOut
- **static_rfi_mask** – Set[RFI] = []

class ska.tmlite.models.**Instruments**(*, *ska1_low: Optional[SKA1Low] = None*, *ska1_mid: Optional[SKA1Mid] = None*)

Container item for the Instrumets

Parameters

- **ska1_low** – SKA1Low
- **ska1_mid** – SKA1Mid

class ska.tmlite.models.**TModelLite**(*, *instrument: Instruments*)

Container item for the whole model

Parameters
instrument – Instruments

class ska.tmlite.models.**MCCSGeoJSON**(*, *type: str*, *name: str*, *crs: MCCSCrs*, *features: List[MCCSFeatures]*)

Container item for the GeoJSON file generated by MCCS. Ideally we would not have to add this. But it has been deemed necessary to demonstrate the reading in of this file format.

Currently,the schema is:

Parameters

- **type** – str
- **name** – str
- **crs** – MCCSCrs

- **features** – List[MCCSFeatures]

class ska.tmlite.models.**MCCSCrs**(*, *type: str*, *properties: MCCSProperties*)

The CRS items in the MCCS GeoJSON file

Parameters

- **type** – str
- **properties** – MCCSProperties

class ska.tmlite.models.**MCCSFeatures**(*, *type: str*, *properties: MCCSProperties*, *geometry: MCCSGeometry*)

The Feature carries the station information in the MCCSGeoJSON model

Parameters

- **type** – str
- **properties** – MCCSProperties
- **geometry** – MCCSGeometry

class ska.tmlite.models.**MCCSGeometry**(*, *type: str*, *coordinates: List[float]*)

The MCCS Geometry object - this is what actually contains the location

Parameters

- **type** – str
- **coordinates** – List[float]

class ska.tmlite.models.**MCCSProperties**(*, *name: str*, *nof_antennas: Optional[int] = None*, *antenna_type: Optional[str] = None*, *tpms: Optional[Set[int]] = None*, *station_num: Optional[int] = None*)

The Properties Object - contains a description of the station

Parameters

- **name** – str
- **nof_antennas** – Optional[int] = None
- **antenna_type** – Optional[str] = None
- **tpms** – Optional[Set[int]] = None
- **station_num** – Optional[int] = None

8.2 Server functions

async ska.tmlite.server.main.**add_model**(*item_id: str*, *item: Union[TModelLite, MCCSGeoJSON]*)

PUT method that takes a full model to add - will replace If the model is in MCCSGeoJSON format we will convert it to TModelLite model :param model: TModelLite or MCCSGeoJSON in json format :return: the model:

async ska.tmlite.server.main.**get_instrument**(*item_id: str*, *instrument_id: str*)

GET method for the given instrument

Parameters
item_id – str

Returns
model

async ska.tmlite.server.main.**get_model**(*item_id: str*)

GET method for the given model

Parameters
item_id – str the label for the model that is to be returned

Returns
model

async ska.tmlite.server.main.**get_layout**(*item_id: str*, *instrument: str*)

GET method for the given model - perform a conversion from any other model

Parameters
item_id – str

Returns
model

async ska.tmlite.server.main.**get_static_rfi_mask**(*item_id: str*, *instrument_id: str*)

GET method to return the rfi mask

Returns
the JSON representation of the RFI mask

ska.tmlite.server.main.**read_root**()

async ska.tmlite.server.main.**update_model_antennas**(*item_id: str*, *instrument_id: str*, *item: Mccs*)

Update the layout of the current model to match the input This actually takes the current station_id's and uses them to add the antennas that match the IDs from the default model into the current model.

async ska.tmlite.server.main.**update_model_layout_from_file**(*item_id: str*, *instrument_id: str*, *item: Union[TModelLite, MCCSGeoJSON]*)

Updates the layout using an input model (either TModelLite or MCCSGeoJSON

async ska.tmlite.server.main.**update_model_layout_from_storage**(*item_id: str*, *instrument_id: str*, *file_id: str*)

Updates the layout using the contents of a file in the backend storage

ska.tmlite.server.main.**update_model_layout**(*item_id: str*, *instrument_id: str*, *item: dict*)

Updates the current model layout with the contents of the dictionary

Parameters

- **item_id** – str - the label of the model to be updated
- **instrument_id** – str the instrument [ska1_low | ska1_mod]
- **item** – dict JSON representation of the station position

8.3 Client

CHAPTER

NINE

TMLITE DOCUMENTATION

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

- *TMLite Server*

INDEX