developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Jan 25, 2022

HOME

Documentation Status

This package provides the ability to generate SKA Unique Identifiers (IDs) for differentiating entities that may be created and archived during telescope operation. Such entities include Schedule Bock Definitions, Data Products, etc.

CHAPTER

ONE

FEATURES

- Scan IDs
  - Generates one scan ID per request.
  - Scan ID is unique to a single SKA UID service instance, beyond execution of the service instance, for a maximum of 2^48 scans.
  - *Scenario*: OET requests a scan ID
    - *Given* skuid service is running
    - *When* I request a scan ID
    - *Then* The scan ID returned as an integer
    - *And* The numerical value of the scan ID is greater than 0 and less or equal to 2^48
  - *Scenario*: OET requests a second scan ID
    - *Given* I have already requested a scan ID
    - *When* I request a scan ID
    - *Then* The numerical value of the latest scan ID is one more than the previous scan ID
- Other supported entity IDs
  - Generates one entity ID on request, with entity type specified in the request.
  - Supports global uniqueness of entity IDs by encoding in the ID the identity of the unique SKA UID service instance as well as the type of entity, the date and time and a sequence number.
  - Allows addition of an entity type to be supported.
  - Persists supported entity types beyond execution of the SKA UID service instance.
  - *Scenario*: OET requests an ID for a Scheduling Block Definition
    - *Given* Entity type Scheduling Block Definition is supported by the SKA UID service instance
    - *When* I request an ID for a Scheduling Block Definition entity
    - *Then* The ID returned is a string of the *format type-generatorID-datetime-localSeq*
- Transaction IDs
  - Transaction ID from the service
    - Generates one transaction ID per request.
    - The ID is retrieved from the Skuid service
  - Local Transaction ID

  * Generates one transaction ID per request.
  * The ID is not retrieved from the skuid service and may have duplicates

- Sample usage:

```
import os

from ska_ser_skuid.client import SkuidClient

def get_transaction_id():
    if "SKUID_URL" in os.environ and os.environ["SKUID_URL"]:
        client = SkuidClient(os.environ["SKUID_URL"])
        return client.fetch_transaction_id()
    return SkuidClient.get_local_transaction_id()

transaction_id = get_transaction_id()
```

Should the skuid service not respond for any reason, a locally generated transaction ID will be returned.

```
client = SkuidClient("non_existing_url")
transaction_id = client.fetch_transaction_id()
print(transaction_id)
# txn-local-20200921-516590971
```

CHAPTER

TWO

REQUIREMENTS

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

CHAPTER

THREE

INSTALLATION TO MAKE USE OF THE CLIENT

Install from Nexus PYPI

```
> pip install --extra-index-url=https://artefact.skao.int/repository/pypi-all/simple ska-
↪ser-skuid
```

Use the client

```
> from ska_ser_skuid.client import SkuidClient
> client = SkuidClient("<URL TO SKUID SERVICE>")
> scan_id = client.fetch_scan_id()
> skuid = client.fetch_skuid('sbd')
```

CHAPTER

FOUR

INSTALLATION TO RUN A LOCAL SERVER

Install skuid client package as above.

```
> pip install --extra-index-url=https://artefact.skao.int/repository/pypi-all/simple ska-
↪ser-skuid
```

Install the server requirements.

```
> poetry install
```

CHAPTER

FIVE

TESTING

- Put tests into the `tests` folder
- Use PyTest as the testing framework
  - Reference: PyTest introduction
- Running tests:
  - Install `tox`
  - Server tests
    - `falcon` and `fasteners` is installed and the server as well as client functionality is tested
    - `tox -e server`
  - Client tests
    - Only installs the requirements for the client (not `falcon` or `fasteners`) and tests the client functionality only
    - `tox -e client`
- Running the test creates the `build/reports/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 `tox -e lint`
- Code analysis should only raise document related warnings (i.e. `#FIXME` comments) before merging the code

CHAPTER

SEVEN

RUN DEVELOPMENT SERVER

Ensure that gunicorn has been installed

```
> gunicorn ska_ser_skuid.server.http:app
```

Browse to http://127.0.0.1:8000/skuid

To get a new ID, browse to http://127.0.0.1:8000/skuid/ska_id/<entity_type>

- E.g http://127.0.0.1:8000/skuid/ska_id/dp

CHAPTER

EIGHT

BUILD AND RUN AS A DOCKER CONTAINER

- Build the image

```
> docker build -t skuid_service .
```

- Run the container

```
> docker run -p 8080:9870  skuid_service
```

- Browse to http://127.0.0.1:8080/

CHAPTER

NINE

SAMPLE REQUESTS

```
> curl http://127.0.0.1:8080/skuid

Welcome to skuid

Please use /skuid/ska_id/{entity_type} to retrieve a ID

> curl http://127.0.0.1:8080/skuid/ska_scan_id
"{\"scan_id\": 2, \"generator_id\": \"t0001\"}"

> curl http://127.0.0.1:8080/skuid/ska_id/sbd
"{\"ska_uid\": \"sbd-t0001-20200914-00009\", \"generator_id\": \"t0001\"}"

> curl http://127.0.0.1:8080/skuid/ska_transaction_id
"{\"transaction_id\": \"txn-t0001-20200914-123456789\", \"generator_id\": \"t0001\"}"

> curl http://127.0.0.1:8080/skuid/ska_id/eb
"{\"ska_uid\": \"eb-t0001-20210426-00012\", \"generator_id\": \"t0001\"}"
```

CHAPTER

TEN

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`:
  - Ensure `tox` has been installed

```
$ tox -e docs
```

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

CHAPTER

ELEVEN

SKA UNIQUE IDENTIFIER FORMAT DOCUMENTATION

This section describes the SKA Unique Identifier (SKA UID) format. The format of the scan ID differs from that of other types of entities like Scheduling Block Definition, Data Product, etc.

11.1 Scan IDs

String of format *localSeq* where

- *localSeq* - locally generated sequence: 15 digit string representation of a natural number (48 bits), with leading zeros.

An example of a scan ID: `000000000001234`

11.2 IDs of other entity types

String of format *type-generatorID-datetime-localSeq* where

- *type* - type of entity (object) the ID is for (SBD, SBI, etc.). Refer to class *EntityTypes*.
- *generatorID* - uniquely identifies the particular SKA ID generator service instance that issued the SKA UID. This can also be used to separate IDs that are generated for different purposes such as normal operations, testing, engineering, AIV.
- *datetime* : ISO8601 format date (in UT time zone), in the standard, basic, format plus some precision of time - this can help with localising a debugging window if required. Note we are assuming a 1 day precision for the moment;
- *localSeq* - locally generated sequence: 5 digit string representation of a natural number, with leading zeros.

A possible example SKA UID for a Scheduling Block Instance generated at the Mid telescope could be: `sbi-M001-20191031-01234`

This example assumes a 1 day date precision and 5 digits for the sequence

CHAPTER

TWELVE

SKA UID FORMAT

A description of the SKA Unique ID format can be found here.

- *SKA Unique Identifier Format Documentation*

Todo:

- Insert todo's here

12.1 Package-name documentation

This section describes requirements and guidelines.

12.1.1 Subtitle

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Public API Documentation

Functions

Classes

CHAPTER

THIRTEEN

PROJECT-NAME DOCUMENTATION

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