developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Jun 21, 2023

REPOSITORY

This project is developing the Observatory Science Operations Data Archive prototype for the Square Kilometre Array.

CHAPTER

ONE

MEMORYREPOSITORY MODULE (SKA_DB_ODA.INFRASTRUCTURE.MEMORY.REPOSITORY)

This module contains implementation of the AbstractRepository class, using memory. Useful for development and tests as it doee not require a heavyweight database implementation.

class ska_db_oda.infrastructure.memory.repository.**MemoryBridge**(*entity_dict: Dict[U, Dict[int, T]]*)

Implementation of the Repository bridge which persists entities in memory.

Entities will be stored in a nested dict: *{<entity_id>: {<version>: <entity>}}*

create(*entity: T*) → None

Implementation of the RepositoryBridge method.

See *create()* docstring for details

query(*qry_params: QueryParams*) → List[U]

Queries the latest version of the entity based on QueryParams class from the ODA and returns the corresponding entity ID

Returns an empty list if no entities in the repository match the parameters.

Raises

- **ValueError** – if the qry_params are not supported
- **OSError** – if an error occurs while querying the entity

read(*entity_id: U*) → None

Implementation of the RepositoryBridge method.

See *read()* docstring for details

update(*entity: T*) → T

Implementation of the RepositoryBridge method.

See *update()* docstring for details

CHAPTER

TWO

ABSTRACTREPOSITORY MODULE (SKA_DB_ODA.DOMAIN.REPOSITORY)

This module contains the AbstractRepository base class and the entity specific types.

class ska_db_oda.domain.repository.**AbstractRepository**(*bridge:* RepositoryBridge*[T, U]*)

Generic repository that defines the interface for users to add and retrieve entities from the ODA. The implementation is passed at runtime (eg postgres, filesystem) and metadata updates are handled via the mixin class.

It is expected to be typed to SBDefinitionRepository, SBInstanceRepository, etc.

add(*entity: T*) → None

Stores the entity in the ODA.

The entity passed to this method will have its metadata validated and updated.

Raises

- **ValueError** – if the validation of the sbd or its metadata fails
- **OSError** – if an error occurs while persisting the SBD

get(*entity_id: U*) → T

Retrieves the latest version of the entity with the given id from the ODA.

Raises

- **KeyError** – if the sbd_id is not found in the repository
- **OSError** – if an error occurs while retrieving the SBD

query(*qry_param: QueryParams*) → List[U]

Queries the latest version of the entity based on QueryParams class from the ODA and returns the corresponding entity ID

Returns an empty list if no entities in the repository match the parameters.

Raises

- **ValueError** – if the qry_params are not supported
- **OSError** – if an error occurs while querying the entity

class ska_db_oda.domain.repository.**ExecutionBlockRepository**(*bridge:* RepositoryBridge*[T, U]*)

Abstraction over persistent storage of ExecutionBlocks

class ska_db_oda.domain.repository.**ProjectRepository**(*bridge:* RepositoryBridge*[T, U]*)

Abstraction over persistent storage of Projects

class ska_db_oda.domain.repository.**RepositoryBridge**

This class is the implementor of the Bridge pattern which decouples the persistence method from the Repository abstraction.

It is designed to be used as a composition within a repository and offers CRUD type methods.

abstract create(*entity: T*) → None

Stores a new, versioned entity in the repository

Raises

- **ValueError** – if the validation of the entity or its metadata fails
- **OSError** – if an error occurs while persisting the entity

abstract query(*qry_params: QueryParams*) → List[U]

Queries the latest version of the entity based on QueryParams class from the ODA and returns the corresponding entity ID

Returns an empty list if no entities in the repository match the parameters.

Raises

- **ValueError** – if the qry_params are not supported
- **OSError** – if an error occurs while querying the entity

abstract read(*entity_id: U*) → T

Retrieves the latest version of the entity with the given id from the ODA.

Raises

- **KeyError** – if the sbd_id is not found in the repository
- **OSError** – if an error occurs while retrieving the SBD

abstract update(*entity: T*) → None

Updates version 1 of the entity with the given entity ID in the repository, or creates version 1 if it doesn't already exist.

Raises

- **ValueError** – if the validation of the entity or its metadata fails
- **OSError** – if an error occurs while persisting the entity

class ska_db_oda.domain.repository.**SBDefinitionRepository**(*bridge:* RepositoryBridge*[T, U]*)

Abstraction over persistent storage of SBDefinitions

class ska_db_oda.domain.repository.**SBInstanceRepository**(*bridge:* RepositoryBridge*[T, U]*)

Abstraction over persistent storage of SBInstances

CHAPTER

THREE

FILESYSTEMREPOSITORY MODULE (SKA_DB_ODA.INFRASTRUCTURE.FILESYSTEM.REPOSITORY)

This module contains implementations of the AbstractRepository class, using the filesystem as the data store.

class ska_db_oda.infrastructure.filesystem.repository.**FilesystemBridge**(*filesystem_mapping: FilesystemMapping*, *base_working_dir: str | PathLike = PosixPath('/var/lib/oda')*)

Implementation of the Repository bridge which persists entities to a filesystem.

Entities will be stored under the following filesystem structure: */<base_working_dir>/<entity_type_dir/<entity_id>/<version>.jso*
For example, by default version 1 of an SBDefinition with sbd_id sbi-mvp01-20200325-00001 will be stored at:
/var/lib/oda/sbd/sbi-mvp01-20200325-00001/1.json

create(*entity: T*) → None

Implementation of the RepositoryBridge method.

To mimic the real database, entities are added to a list of pending transactions and only written to the filesystem when the unit of work is committed.

See *create()* docstring for details

query(*qry_params: QueryParams*) → List[U]

Queries the latest version of the entity based on QueryParams class from the ODA and returns the corresponding entity ID

Returns an empty list if no entities in the repository match the parameters.

Raises

- **ValueError** – if the qry_params are not supported
- **OSError** – if an error occurs while querying the entity

read(*entity_id: U*) → T

Gets the latest version of the entity with the given entity_id.

As this method will always be accessed in the context of a UnitOfWork, the pending transactions also need to be checked for a version to return. (Similar to with a database implementation where an entity that was added to a transaction but not committed would still be accessible inside the transaction.)

update(*entity: T*) → None

Implementation of the RepositoryBridge method.

To mimic the real database, entities are added to a list of pending transactions and only written to the filesystem when the unit of work is committed.

See *update()* docstring for details

class ska_db_oda.infrastructure.filesystem.repository.**QueryFilterFactory**

Factory class that returns a list of Python functions equivalent to a user query. Each function processes an entity, returning True if the entity passes the query test.

static filter_between_dates(*query: DateQuery*)

Returns a function that returns True if a date is between a given range.

static match_editor(*query: UserQuery*)

Returns a function that returns True if a document editor matches a (sub)string.

CHAPTER

FOUR

POSTGRESREPOSITORY MODULE (SKA_DB_ODA.INFRASTRUCTURE.POSTGRES.REPOSITORY)

This module contains implementations of the AbstractRepository class, using Postgres as the data store.

class ska_db_oda.infrastructure.postgres.repository.**PostgresBridge**(*postgres_mapping: PostgresMapping*, *connection: psycopg.Connection*)

Implementation of the Repository bridge which persists entities in a PostgreSQL instance.

create(*entity: T*) → None

Implementation of the RepositoryBridge method.

See *create()* docstring for details

query(*qry_params: QueryParams*) → List[U]

Queries the latest version of the entity based on QueryParams class from the ODA and returns the corresponding entity ID

Returns an empty list if no entities in the repository match the parameters.

Raises

- **ValueError** – if the qry_params are not supported
- **OSError** – if an error occurs while querying the entity

read(*entity_id: U*) → T

Implementation of the RepositoryBridge method.

See *read()* docstring for details

update(*entity: T*) → None

Implementation of the RepositoryBridge method.

See *update()* docstring for details

CHAPTER

FIVE

RESTREPOSITORY MODULE (SKA_DB_ODA.CLIENT.REPOSITORY)

This module contains the bridge implementation for Repository class which connects to an ska-db-oda deployment over the network.

class ska_db_oda.client.repository.**RESTBridge**(*rest_mapping: RESTMapping*, *base_rest_uri: str*)

Implementation of the Repository bridge which connects to the ODA API.

create(*entity: T*)

Adds the entity to a pending transaction, ready to be committed as part of the UoW

Note unlike other implementations, this method does not update the metadata, as this is done inside the ODA service after the API is called

query(*qry_params: QueryParams*) → List[U]

Queries the ODA API with the parameters

See `query()` docstring for details

read(*entity_id: U*) → T

Gets the latest version of the entity with the given entity_id.

As this method will always be accessed in the context of a UnitOfWork, the pending transactions also need to be checked for a version to return. (Similar to with a database implementation where an entity that was added to a transaction but not committed would still be accessible inside the transaction.)

update(*entity: T*) → None

Adds the entity to a pending transaction, ready to be committed as part of the UoW.

As the update method edits in the existing entity, if an entity with the same ID is passed twice to this method, it will overwrite the previous version 1 in the pending transactions.

CHAPTER

SIX

MEMORYUNITOFWORK MODULE (SKA_DB_ODA.UNIT_OF_WORK.MEMORYUNITOFWORK)

This module contains the implementation of the AbstractUnitOfWork Class.

class ska_db_oda.unit_of_work.memoryunitofwork.**MemoryUnitOfWork**(*session: MemorySession | None = None*)

A lightweight non-persistent implementation of the AbstractUnitOfWork that can store and retrieve SchedulingBlock objects.

Commits or rolls back a series of database transactions as an atomic operation. Changes between commits are tracked via the _transactions variable.

commit() → None

Implementation of the AbstractUnitOfWork method.

See *commit()* docstring for details

rollback() → None

Implementation of the AbstractUnitOfWork method.

See *rollback()* docstring for details

CHAPTER

SEVEN

ABSTRACTUNITOFWORK MODULE (SKA_DB_ODA.UNIT_OF_WORK.ABSTRACTUNITOFWORK)

This module contains the UnitOfWork Abstract Base Class.

All UnitOfWork implementations to conform to this interface.

class ska_db_oda.unit_of_work.abstractunitofwork.**AbstractUnitOfWork**

Provides the interface to store or retrieve a group of Scheduling Block Objects.

Commits or rollsback a series of database transactions as an atomic operation

abstract commit() → None

Commits the Unit of Work.

Raises

ValueError – if the validation of committed sbds or the metadata fails

abstract rollback() → None

Initiates the rollback of this Unit of Work.

If no commit is carried out or an error is raised, the unit of work is rolled back to a safe state

CHAPTER

EIGHT

FILESYSTEMUNITOFWORK MODULE (SKA_DB_ODA.UNIT_OF_WORK.FILESYSTEMUNITOFWORK)

FilesystemUnitOfWork adds unit of work transaction support to the FilesystemRepository.

class ska_db_oda.unit_of_work.filesystemunitofwork.**FilesystemUnitOfWork**(*base_working_dir: str | PathLike = PosixPath('/var/lib/oda')*)

Implementation of the AbstractUnitOfWork which persists the data in the filesystem

commit() → None

Implementation of the AbstractUnitOfWork method.

See *commit()* docstring for details

rollback() → None

Implementation of the AbstractUnitOfWork method.

See *rollback()* docstring for details

CHAPTER

NINE

POSTGRESUNITOFWORK MODULE (SKA_DB_ODA.UNIT_OF_WORK.POSTGRESUNITOFWORK)

class ska_db_oda.unit_of_work.postgresunitofwork.**PostgresUnitOfWork**(*connection_pool: psycopg_pool.ConnectionPool | None = None*)

A PostgreSQL implementation of the UoW which persists data in an instance of PostgreSQL specified in the initialisation config

commit() → None

Implementation of the AbstractUnitOfWork method.

See *commit()* docstring for details

rollback() → None

Implementation of the AbstractUnitOfWork method.

See *rollback()* docstring for details

CHAPTER

TEN

RESTUNITOFWORK MODULE (SKA_DB_ODA.UNIT_OF_WORK.RESTUNITOFWORK)

This module contains the implementation of the AbstractUnitOfWork Class which connects to a remote ODA API.

class ska_db_oda.unit_of_work.restunitofwork.**RESTUnitOfWork**(*rest_uri: str | None = None*)

Implementation of the AbstractUnitOfWork which connects with the ska-db-oda API over the network

commit() → None

Implementation of the AbstractUnitOfWork method.

See *commit()* docstring for details

rollback() → None

Implementation of the AbstractUnitOfWork method.

See *rollback()* docstring for details

CHAPTER

ELEVEN

REPOSITORY_CLASS_DIAGRAM

CHAPTER

TWELVE

UNIT_OF_WORK_CLASS_DIAGRAM

CHAPTER

THIRTEEN

OVERVIEW

CHAPTER

FOURTEEN

ODA KUBERNETES DEPLOYMENT

14.1 Deploying ODA using Helm charts

Deploying the ODA umbrella Helm chart will deploy the ODA REST client, an instance of Postgres and pgadmin.

To install the umbrella chart, navigate to the root directory of the repository and run

```
$ make k8s-install-chart
```

To uninstall the chart

```
$ make k8s-uninstall-chart
```

Inspect the deployment state with

```
$ make k8s-watch
```

14.2 Backend Storage

Note: These instruction assume you are using the standard SKA Minikube installation that is configured and installed via the ska-cicd-deploy-minikube project.

There are different implementations of the ODA - `memory`, `filesystem` and `postgres` - which can be imported and used as required.

The ODA REST API can be configured to use any of these at runtime, using the Helm values.

14.2.1 Filesystem

To configure a filesystem backend, with a Kubernetes persistent volume which provides persistence that survives Kubernetes redeployments and pod restarts, set the following values for the Helm chart:

```
rest:
  ...
  backend:
    type: filesystem
    filesystem:
      # true to mount persistent volume, false for non-persistent storage
```

(continues on next page)

(continued from previous page)

```
    use_pv: true
    # path on Kubernetes host to use for entity storage
    pv_hostpath: /mnt/ska-db-oda-persistent-storage
  ...
```

For a default installation with no Helm value overrides, access the PersistentVolume on the minikube node as follows:

```
$ # SSH to minikube cluster
$ minikube ssh
$ # navigate to default ODA storage directory
$ cd /mnt/ska-db-oda-persistent-storage/
```

Files can also be stored outside minikube by making `pv_hostpath` match the `MOUNT_FROM` and `MOUNT_TO` values set when rebuilding minikube with the ska-cicd-deploy-minikube project. For example, if minikube is rebuilt with

```
$ make MOUNT_FROM=$HOME/oda MOUNT_TO=$HOME/oda all
```

and `pv_hostpath` is set to match `$MOUNT_TO`, entities will be stored directly on your local filesystem in the `$HOME/oda` directory. See the bottom of this page for a full example.

14.2.2 Postgres

To use postgres as a backend, a running instance of PostgreSQL must be available, and the Helm values set as below:

```
rest:
  ...
  backend:
    type: postgres
    postgres:
      host:
      port: 5432
      db:
        name: postgres
        table:
          sbd: tab_oda_sbd
  ...
```

If using the local postgres deployed as part of the umbrella chart, the host will be set at deploy time in the Makefile.

There are also relevant values of the postgresql dependency chart (See the 'PostgresQL deployment' section below for more details). By default, the `make k8s-install-chart` target will set the postgres values to those required to connect to the PostgreSQL instance also deployed by the chart.

14.3 Enabling ingress for local testing

The ODA REST server can be exposed to allow local testing. This is achieved by setting ingress.enabled to true when deploying the chart. For example,

Note: The command below to set ingress is different than the usual K8S_CHART_PARAMS="–set ska-db-oda.ingress.enabled=true" as there are Postgres parameters also set in the Makefile which should not be overwritten.

```
$ # from the ODA project directory, install the ODA including the custom values
$ make k8s-install-chart

$ # capture the minikube IP address in an environment variable
$ export MINIKUBE_IP=`minikube ip`

$ # capture the ODA deployment namespace in an environment variable
$ export ODA_NAMESPACE=`make k8s-vars | grep 'Selected Namespace' | awk '{ print $3 }'`

$ # construct full ODA endpoint URL
$ export ODA_ENDPOINT=http://$MINIKUBE_IP/$ODA_NAMESPACE/api/v1/sbds

$ # Try an invalid ODA request. The error message shows the ODA is accessible and␣
↪working.
$ curl $ODA_ENDPOINT
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method is not allowed for the requested URL.</p>
```

14.4 Example: ODA deployment using a local directory for backend storage

Caution: This will delete any existing Minikube deployment!

In this example, the user `tango` wants to deploy the ODA so that the ODA stores and retrieves SBs from the local directory `/home/tango/oda`. We'll set an environment variable to hold this location.

```
$ export ODA_DIR=$HOME/oda
```

Minikube needs to be deployed with a persistent volume that makes $ODA_DIR available inside the Kubernetes cluster. This is achieved by redeploying Kubernetes using the ska-cicd-deploy-minikube project. Checkout the project and (re)deploy Minikube like so:

```
$ # checkout the ska-cicd-deploy-minikube project
$ git clone --recursive https://gitlab.com/ska-telescope/sdi/ska-cicd-deploy-minikube
$ cd ska-cicd-deploy-minikube

$ # redeploy Minikube. Caution! This will delete any existing deployment!
$ make MOUNT_FROM=$ODA_DIR MOUNT_TO=$ODA_DIR clean all
```

The ODA chart can now be installed. For a local installation, it can be useful to expose the ODA ingress so that the ODA deployment can be exercised from outside Minikube, i.e., from your host machine. We also want to configure the ODA to use the directory exposed at `$ODA_DIR`. These aspects are configured by setting the relevant Helm chart values. These values could be set individually using `K8S_CHART_PARAMS="--set parameter1=foo --set parameter2=bar"` etc., but as there are several values to set we will define them in a setting file (overrides.yaml) to be included when deploying the ODA.

```
$ # navigate to the directory containing the ska-db-oda project
$ cd path/to/ska-db-oda

$ # inspect contents of our Heml chart overrides. This example enables ODA
$ # ingress and configures the backend to to use the persistent volume
$ # exposed at $ODA_DIR. Create this file if required.
$ cat overrides.yaml
rest:
  ingress:
    enabled: true
  backend:
    type: filesystem
    filesystem:
      use_pv: true
      pv_hostpath: /home/tango/oda   <-- replace with the value of $ODA_DIR

$ # install the ODA including the custom values
$ make K8S_CHART_PARAMS="--values overrides.yaml" k8s-install-chart
```

The state of the deployment can be inspected with `make k8s-watch`. The output for a successful deployment should look similar to below:

```
$ make k8s-watch

NAME                          READY   STATUS    RESTARTS   AGE
pod/ska-db-oda-rest-test-0    1/1     Running   0          24s

NAME                            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
service/ska-db-oda-rest-test    ClusterIP   10.98.75.197   <none>        5000/TCP   24s

NAME                                    READY   AGE
statefulset.apps/ska-db-oda-rest-test   1/1     24s

NAME                                                        CAPACITY   ACCESS MODES␣
↪RECLAIM POLICY   STATUS      CLAIM                                                 ␣
↪STORAGECLASS   REASON   AGE
persistentvolume/pvc-b44332d8-e8b8-472b-a407-2080c850dee0   1Gi        RWO           ␣
↪Delete           Bound       ska-db-oda/ska-db-oda-persistent-volume-claim-test  ␣
↪standard               24s
persistentvolume/ska-db-oda-persistent-volume-test          1Gi        RWO           ␣
↪Delete           Available                                                         ␣
↪standard               24s

NAME                                                          STATUS   VOLUME        ␣
↪                          CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/ska-db-oda-persistent-volume-claim-test   Bound    pvc-b44332d8-
↪e8b8-472b-a407-2080c850dee0   1Gi        RWO            standard       24s
```

(continues on next page)

(continued from previous page)

```
NAME                                            CLASS    HOSTS   ADDRESS   PORTS   AGE
ingress.networking.k8s.io/ska-db-oda-rest-test  <none>   *                 80      24s
```

SBs uploaded to the ODA will be stored in `$ODA_DIR`. Any SB JSON files stored in `$ODA_DIR` directory will be retrievable via the ODA REST API, e.g.,

```
$ # get the ODA endpoint, which is a combination of Minikube IP address and
$ # deployment namespace
$ MINIKUBE_IP=`minikube ip`
$ ODA_NAMESPACE=`make k8s-vars | grep 'Selected Namespace' | awk '{ print $3 }'`
$ ODA_ENDPOINT=http://$MINIKUBE_IP/$ODA_NAMESPACE/api/v1/sbds

$ # get the SBD ID from the SB used for unit tests. We'll upload the SB to this URL.
$ SBD_ID=`grep sbd_id tests/unit/testfile_sample_low_sb.json \
        | awk '{ gsub("\"", ""); gsub(",", ""); print $2 }'`

$ # upload the unit test SB to the ODA
$ curl -iX PUT -H "Content-Type: application/json"  \
       -d @tests/unit/testfile_sample_low_sb.json   \
       $ODA_ENDPOINT/$SBD_ID
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Date: Mon, 21 Feb 2022 11:24:25 GMT
Content-Type: application/json
Content-Length: 76
Connection: keep-alive

{"message":"Created. A new SB definition with UID sbi-mvp01-20200325-00001."}

$ # list contents of $ODA_DIR. The uploaded SB should be stored there.
$ ls $ODA_DIR
sbi-mvp01-20200325-00001.json
```

14.5 PostgresQL deployment

By default, the ska-db-oda chart with the command `make k8s-install-chart` will install both the ODA client and ODA PostgresQL DB. The installation of the DB is based on the Bitnami Helm chart.

The parameters that can be set for the DB deployment are:

```
ADMIN_POSTGRES_PASSWORD        secretpassword ## Password for the "postgres" admin user
ENABLE_POSTGRES                true ## Enable or not postgresql
```

The DB will be installed only if the variable `ENABLE_POSTGRES` is `true`. The service type is set to be LoadBalancer.

This means that it is possible to reach the database directly at the ip address of the result of this command `kubectl get svc -n ska-db-oda | grep LoadBalancer | awk '{print $4}'`.

Note: If using Minikube, the LoadBalancer needs to be exposed via *minikube tunnel* See here for more details.

There are other parameters that can be changed only with the values file of the chart. They are the following:

```
postgresql:
commonLabels:
   app: ska-db-oda

enabled: true

image:
   debug: true

auth:
   postgresPassword: "secretpassword"

primary:
   service:
      type: LoadBalancer

   initdb:
      scriptsConfigMap: ska-db-oda-initdb
      user: "postgres"

   persistence:
      enabled: true
      ## @param primary.persistence.mountPath The path the volume will be mounted at
      ## Note: useful when using custom PostgreSQL images
      ##
      mountPath: /bitnami/postgresql
      ## @param primary.persistence.storageClass PVC Storage Class for PostgreSQL␣
↪Primary data volume
      ## If defined, storageClassName: <storageClass>
      ## If set to "-", storageClassName: "", which disables dynamic provisioning
      ## If undefined (the default) or set to null, no storageClassName spec is
      ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
      ##   GKE, AWS & OpenStack)
      ##
      storageClass: "nfss1"
      ## @param primary.persistence.accessModes PVC Access Mode for PostgreSQL volume
      ##
      accessModes:
      - ReadWriteMany
      ## @param primary.persistence.size PVC Storage Request for PostgreSQL volume
      ##
      size: 12Gi
```

More parameters can be found at the bitnami helm chart for PostgreSQL.

CHAPTER

FIFTEEN

REST API

15.1 REST API

The ODA REST API supports resources for SBDefinitions, SBInstances, ExecutionBlocks and Projects.

Each resource supports a GET and PUT method as documented below.

The is also a root end point which will support PUT for multiple resources as an atomic operation.

The endpoints, with the accepted requests and expected responses are documented below:

`GET /`

Returns a home page for the ODA API

Returns a basic HTML page for the ODA API with links to documents

Status Codes

- 200 OK – OK

`PUT /`

Store multiple entities

Stores the entities as an atomic operation.

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request, validation of against OpenAPI spec failed
- 422 Unprocessable Entity – Unprocessable Entity, semantic error in request eg mismatched IDs
- 500 Internal Server Error – Internal Server Error

`GET /sbds`

Get SBDefinition IDs filter by the query parameter

Retrieves the IDs of the SBDefinitions which match the query parameters. Currently only a single query field is permitted, eg user or created_on - extra parameters passed to the request will be ignored, with an order of precedence of user > created_on > modified_on. Also requests without parameters will return an error rather than returning all IDs. This behaviour will change in the future

Query Parameters

- **user** (*string*) –
- **user_match_type** (*string*) –

- **created_before** (*string*) –
- **created_after** (*string*) –
- **last_modified_before** (*string*) –
- **last_modified_after** (*string*) –

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 500 Internal Server Error – Internal Server Error

`GET /sbds/{sbd_id}`

Get SBDefiniton by identifier

Retrieves the SBDefinition with the given identifier from the underlying data store, if available

Parameters

- **sbd_id** (*string*) –

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found
- 500 Internal Server Error – Internal Server Error

`PUT /sbds/{sbd_id}`

Store SBDefinition by identifier

Stores the SBDefinition with the given identifier in the underlying data store. If the identifier does not exist in the data store, a new version 1 will be created. If a version does exist, an update will be performed. In this case, the metadata in the recieved SBDefinition should match the existing version - the user of this API should not edit metadata before sending.

Parameters

- **sbd_id** (*string*) –

Status Codes

- 200 OK – OK, the SBDefinition was created or updated
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 422 Unprocessable Entity – Unprocessable Entity, semantic error in request eg mismatched IDs
- 500 Internal Server Error – Internal Server Error

`GET /sbis`

Get SBInstance IDs filter by the query parameter

Retrieves the IDs of the SBInstances which match the query parameters. Currently only a single query field is permitted, eg user or created_on - extra parameters passed to the request will be ignored, with an order of precedence of user > created_on > modified_on. Also requests without parameters will return an error rather than returning all IDs. This behaviour will change in the future

Query Parameters

- **user** (*string*) –

- **user_match_type** (*string*) –
- **created_before** (*string*) –
- **created_after** (*string*) –
- **last_modified_before** (*string*) –
- **last_modified_after** (*string*) –

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 500 Internal Server Error – Internal Server Error

`GET /sbis/{sbi_id}`

Get SBInstance by identifier

Retrieves the SBInstance with the given identifier from the underlying data store, if available

Parameters

- **sbi_id** (*string*) –

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found
- 500 Internal Server Error – Internal Server Error

`PUT /sbis/{sbi_id}`

Store SBInstance by identifier

Stores the SBInstance with the given identifier in the underlying data store. If the identifier does not exist in the data store, a new version 1 will be created. If a version does exist, an update will be performed. In this case, the metadata in the recieved SBInstance should match the existing version - the user of this API should not edit metadata before sending.

Parameters

- **sbi_id** (*string*) –

Status Codes

- 200 OK – OK, the SBInstance was created or updated
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 422 Unprocessable Entity – Unprocessable Entity, semantic error in request eg mismatched IDs
- 500 Internal Server Error – Internal Server Error

`GET /ebs`

Get ExecutionBlock IDs filter by the query parameter

Retrieves the IDs of the ExecutionBlocks which match the query parameters. Currently only a single query field is permitted, eg user or created_on - extra parameters passed to the request will be ignored, with an order of precedence of user > created_on > modified_on. Also requests without parameters will return an error rather than returning all IDs. This behaviour will change in the future

Query Parameters

- **user** (*string*) –
- **user_match_type** (*string*) –
- **created_before** (*string*) –
- **created_after** (*string*) –
- **last_modified_before** (*string*) –
- **last_modified_after** (*string*) –

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 500 Internal Server Error – Internal Server Error

`GET /ebs/{eb_id}`

Get ExecutionBlock by identifier

Retrieves the ExecutionBlock with the given identifier from the underlying data store, if available

Parameters

- **eb_id** (*string*) –

Status Codes

- 200 OK – OK
- 404 Not Found – Not Found
- 500 Internal Server Error – Internal Server Error

`PUT /ebs/{eb_id}`

Store ExecutionBlock by identifier

Stores the ExecutionBlock with the given identifier in the underlying data store. If the identifier does not exist in the data store, a new version 1 will be created. If a version does exist, an update will be performed. In this case, the metadata in the recieved ExecutionBlock should match the existing version - the user of this API should not edit metadata before sending.

Parameters

- **eb_id** (*string*) –

Status Codes

- 200 OK – OK, the ExecutionBlock was created or updated
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 422 Unprocessable Entity – Unprocessable Entity, semantic error in request eg mismatched IDs
- 500 Internal Server Error – Internal Server Error

`POST /ebs/{eb_id}/request_response`

Add a record to the Execution Block

Adds the record of the function called on the telescope and its response to the Execution Block with the given eb_id. The purpose of this resource as opposed to the generic update of a whole entity is that user of the scripting functions can record these functions and the responses without having to know the internals of the ODA or the Execution Block data structure.

Parameters

- **eb_id** (*string*) –

Status Codes

- 200 OK – The record was sucessfully added to the ExecutionBlock
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 404 Not Found – Not Found
- 500 Internal Server Error – Internal Server Error

`POST /ebs/create`

Create an ExecutionBlock with a generated eb_id

Creates an 'empty' ExecutionBlock, ie one without any commands and responses, with an eb_id generated from SKUID and persists it in the ODA.

Status Codes

- 200 OK – OK
- 500 Internal Server Error – Internal Server Error

`GET /prjs`

Get Project IDs filter by the query parameter

Retrieves the IDs of the Projects which match the query parameters. Currently only a single query field is permitted, eg user or created_on - extra parameters passed to the request will be ignored, with an order of precedence of user > created_on > modified_on. Also requests without parameters will return an error rather than returning all IDs. This behaviour will change in the future

Query Parameters

- **user** (*string*) –
- **user_match_type** (*string*) –
- **created_before** (*string*) –
- **created_after** (*string*) –
- **last_modified_before** (*string*) –
- **last_modified_after** (*string*) –

Status Codes

- 200 OK – OK
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 500 Internal Server Error – Internal Server Error

`GET /prjs/{prj_id}`

Get Project by identifier

Retrieves the Project with the given identifier from the underlying data store, if available

Parameters

- **prj_id** (*string*) –

Status Codes

- 200 OK – OK

- 404 Not Found – Not Found
- 500 Internal Server Error – Internal Server Error

`PUT /prjs/{prj_id}`

Store Project by identifier

Stores the Project with the given identifier in the underlying data store. If the identifier does not exist in the data store, a new version 1 will be created. If a version does exist, an update will be performed. In this case, the metadata in the recieved Project should match the existing version - the user of this API should not edit metadata before sending.

Parameters

- **prj_id** (*string*) –

Status Codes

- 200 OK – OK, the ExecutionBlock was created or updated
- 400 Bad Request – Bad Request, eg validation of against OpenAPI spec failed
- 422 Unprocessable Entity – Unprocessable Entity, semantic error in request eg mismatched IDs
- 500 Internal Server Error – Internal Server Error

15.2 REST API URLs

The Resource url changes depending on deployment method used for the Flask REST server. GET Curl calls for single SBDs will be used in these examples.

15.2.1 Deploying locally

If deploying locally (with `make rest` or `python src/ska_db_oda/rest/rest.py`) **the port is required** :

Hostname:port

- localhost:**5000**

Resource url

- /api/v1/sbds/<sbd_id>

Curl call

- curl -iX GET -H -d "localhost:5000/api/v1/sbds/sbd-mvp01-20200325-00001"

15.2.2 Deploying via Kubernetes

If deploying via Kubernetes and Helm with `make install-chart` **The port must be ommitted**, and `ska-db-oda` (kubernetes namespace) must be prepended to the resource url. Deploying on kuberetes allows a choice of hostnames to use in the urls for curl calls:

Hostnames

- localhost
- kubernetes.docker.internal

- <your machine name>

Resource url

- **ska-db-oda**/api/v1/sbds/<sbd_id>

Curl calls

- curl -iX GET -H -d "http://localhost/ska-db-oda/api/v1/sbds/sbd-mvp01-20200325-00001"
- curl -iX GET -H -d "http://kubernetes.docker.internal/ska-db-oda/api/v1/sbds/sbi-mvp01-20200325-00001"

CHAPTER

SIXTEEN

ODA COMMAND LINE INTERFACE (CLI)

The ODA Command Line Interface package provides the user with a simple interface for accessing and querying different entities stored in the ODA.

Currently supported entity types (followed by the CLI abbreviation) are:

- Scheduling Block Definitions (sbds)
- Scheduling Block Instances (sbis)
- Execution Blocks (ebs)
- Projects (prjs)

Or run *oda –help* for a list of available entity types.

16.1 Configuration

ODA CLI is installed as part of the ska-db-oda package:

```
$ pip install ska-db-oda --extra-index-url=https://artefact.skao.int/repository/pypi-
↪internal/simple
```

The CLI assumes ODA server is running at the address specified by ODA_URI environment variable. Set the variable, for example when deployed locally with k8s:

```
$ export ODA_URI=http://<minikube_ip>/<kube_namespace>/api/v1
```

16.2 Commands

ODA CLI command	Parameters	Description
get	entity_id	Get entity by ID, first specifying entity type. Example: oda sbis get sbi-mvp01-20200325-00001
query		Query ODA for entity IDs. The query can be one of the folowing:
	user	User query: Specify a name of the creator of the entity, set starts_with to True if only wanting to match the beginning of the user (False by default).
	starts_with	
	created_before	Date created query: Specify a start and/or end date for when the entity was first created
	created_after	
	last_modified_before	Date modified query: Specify a start and/or end date for when the entity was last modified.
	last_modified_after	

CHAPTER

SEVENTEEN

ODA EXECUTION BLOCK API CLIENT

The ODA offers a custom API for Execution Blocks, so they can be created and updated during telescope operation without operators needing to know the internal ODA details. See the REST API documentation for more.

Generally, it is expected that the *create_eb* function will be called at the start of a session, and function calls where the request/response should be stored in the ExecutionBlock should be decorated with *@capture_request_response*. See docstrings below for more details.

This client is intended to be imported when sending commands to the telescope, for example during a Jupyter notebook session or SB driven observing from the OET.

It provides functions to interact with the ODA EB API.

ska_db_oda.client.ebclient.**capture_request_response**(*fn: Callable*) → Callable

A decorator function which will record requests and responses sent to the telescope in an Execution Block within the ODA. It will send individual request_response objects to the ODA /ebs/<eb_id>/request_response API over HTTP, containing the decorated function name, the arguments and the return value, as well as timestamps.

Important: the function assumes two environment variables are set:

- ODA_URI: the location of a running instance of the ODA, eg https://k8s.stfc.skao.int/staging-ska-db-oda/api/v1/
- EB_ID: the identifier of the ExecutionBlock to update. The create_eb function from this module should have already been called during execution, which will set this variable.

The decorator is designed such that it does not block execution of commands if there is a problem with the ODA connection, or the environment variables are not set. Instead, a warning message is logged and execution allowed to continue. Also, any errors raised by the decorated function will not be changed, they will just be recorded in the ODA and reraised.

The standard OSO Scripting functions are decorated using this function, so will automatically record request/responses. To record other function calls in an Execution Block, either use this decorator in your source code:

```
from ska_db_oda.client.ebclient import capture_request_response

@capture_request_response
def my_function_to_record(args):
    ...
```

or use at runtime when calling the function:

```
from ska_db_oda.client.ebclient import capture_request_response

capture_request_response(my_function_to_record)(args)
```

ska_db_oda.client.ebclient.**create_eb**() → str

Calls the ODA /ebs/create API to create an 'empty' ExecutionBlock in the ODA, ready to be updated with request/responses from telescope operation.

This function will also set the EB_ID environment variable to the value of the eb_id returned from the create request, so it can be used by subsequent calls to capture_request_response during the same session.

Important: the ODA_URI environment variables must be set to a running instance of the ODA, eg https://k8s.stfc.skao.int/staging-ska-db-oda/api/v1/

Returns

The eb_id generated from SKUID that is persisted in the ODA

Raises

- **KeyError** – if the ODA_URI variable is not set
- **ConnectionError** – if the ODA requests raises an error or returns a status code other than 200

CHAPTER

EIGHTEEN

CONNECTING TO THE POSTGRESQL DATABASE INSTANCE

These instructions concern the PostgreSQL instance deployed to STFC cloud. For a deployment of a PostgreSQL instance to a local Kubernetes cluster, see the Kubernetes section.

18.1 Connect using PGAdmin link

Use the below link to connect with the PostgreSQL database instance via PGadmin:

https://k8s.stfc.skao.int/staging-ska-db-oda/pgadmin4

Once you are able to see the Pgadmin screen, login using the admin credentials given in the project's CI/CD variables. Below query can be used to get data from query editor:

```
$ SELECT table_name
  FROM information_schema.tables
  WHERE table_schema='public';
```

18.2 Connect using PSQL terminal window

The PostgreSQL database instance can also be connected using the psql terminal window. You can use below command once you are on the terminal. The password is available in the projects CI/CD variables.

The Postgres host can be found from the CI/CD pipeline job `info-dev-environment`

```
$ PGPASSWORD=******** psql -U postgres -d postgres -h <POSTGRES HOST>
```

Once you are connected using psql terminal, there are a few basic commands:

```
$ \du -- to check which user you are connected to
$ \dt -- to get a list of avilable tables
$ select info from tab_oda_sbd; --- to get json column from SBD table
$ \q to quit
```

18.3 Schema created for ODA

An initial schema has been defined for the ODA. Below database tables are identified so far :-

tab_oda_sbd → A table to store SBDs

tab_oda_sbi → A table to store SBIs

tab_oda_prj → A table to store Projects

tab_oda_obs_prj → A table to store Observation Programs

tab_oda_exe_blk → A table to store Execution Blocks

Once you login to the Pgadmin link you should be able to see the above list of tables under DB–>Schema–>Tables. Alternatively, you can also use below command on the query tool to fetch these tables:

```
$ SELECT table_name
  FROM information_schema.tables
  WHERE table_schema='public'
  AND table_type='BASE TABLE';
```

Job created for database:

https://gitlab.com/ska-telescope/db/ska-db-oda/-/jobs/2667353723

If there are any changes to the schema, the above job needs to be stopped and re-deployed. With every redeployment the IP address mentioned in the PGAdmin link as well as connection string for psql would change. We should make a note of it.

18.4 Useful links for PostgreSQL

Below are a few useful links in order to get used to PostgreSQL and PGadmin:

https://www.postgresql.org/

https://www.pgadmin.org/

CHAPTER

NINETEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

HTTP ROUTING TABLE

/

/ebs

/prjs

/sbds

/sbis

INDEX

P

Q

R

S

U