Tango GraphQL Documentation

Release 0.0.1

kits

May 05, 2022

Contents

A GraphQL implementation for Tango.

Contents:

CHAPTER 1

API Documentation

Contents:

1.1 AIOServer

A simple http backend for communicating with a TANGO control system

The idea is that each client establishes a websocket connection with this server (on /socket), and sets up a number of subscriptions to TANGO attributes. The server keeps track of changes to these attributes and sends events to the interested clients. The server uses Taurus for this, so polling, sharing listeners, etc is handled "under the hood".

There is also a GraphQL endpoint (/db) for querying the TANGO database.

1.2 Listener

1.3 Routes

1.4 Schema

Contents:

1.4.1 Attribute

1.4.2 Base

1.4.3 Device

1.4.4 Mutation

1.4.5 Query

1.4.6 Subscriptions

1.4.7 Tango

1.4.8 Types

Module containing the different types.

class tangogql.schema.types.**ScalarTypes**(**args*, ***kwargs*)

This class makes it possible to have input and output of different types.

The ScalarTypes represents a generic scalar value that could be: Int, String, Boolean, Float, List.

static coerce_type(*value*)

This method just return the input value.

Parameters **value** – Any

Returns Value (any)

static parse_literal(*node*)

This method is called when the value of type *ScalarTypes* is used as input.

Parameters **node** – value(any)

Returns Return an exception when it is not possible to parse the value to one of the scalar types.

Return type bool, str, int, float or Exception

static parse_value(*value*)

This method is called when an assignment is made.

Parameters **value** – value(any)

Returns value(any)

1.5 TangoDB

1.6 ttldict

TTL dictionary

Tricks / features:

- calling len() will remove expired keys
- __repr__() might show expired values, doesn't remove expired ones

class tangogql.ttldict.**TTLDict**(*default_ttl*, **args*, ***kwargs*)

Dictionary with TTL Extra args and kwargs are passed to initial .update() call

expire_at(*key*, *timestamp*)

Set the key expire timestamp

get_ttl(*key*, *now=None*)

Return remaining TTL for a key

is_expired(*key*, *now=None*, *remove=False*)

Check if key has expired

set_ttl(*key*, *ttl*, *now=None*)

Set TTL for the given key

CHAPTER 2

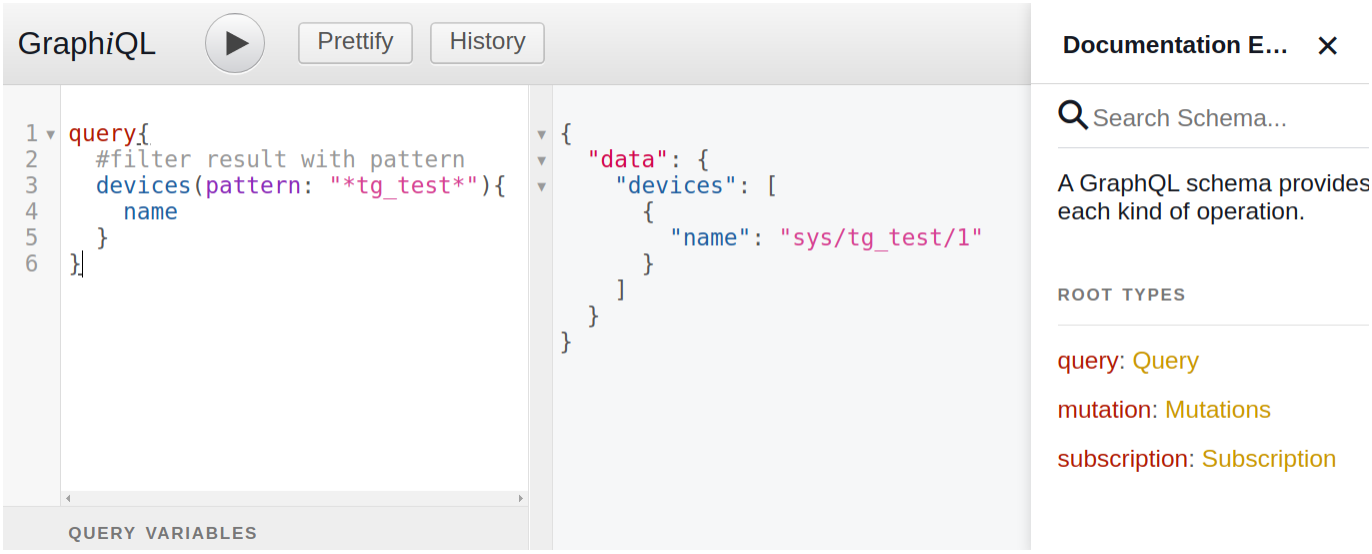
What is GraphiQL and how can be used

GraphiQL is deployed together with tangogql, it is a graphical interactive in-browser GraphQL IDE used to test GraphQL queries. For more info about it check:

Source code Docs for GraphiQL: https://graphiql-test.netlify.app/typedoc/

If you deployed the webjive suite with for example https://gitlab.com/MaxIV/webjive-develop GraphiQL url link should be acessible for you at: `` `http://localhost:5004/graphiql/` ``

To check the type of queries you can use on graphiql see: *Examples on query and mutation*



CHAPTER 3

Examples on query and mutation

3.1 Fetch information of devices

```
query{
    devices{
        name                # e.g. get the names of all devices
    }
}

query{
    devices(pattern: "*tg_test*"){            #filter result with pattern
        name
    }
}
```

3.2 Accessing attributes

```
query{
    devices(pattern: "sys/tg_test/1"){
        name,
        attributes {
            name,
            datatype,
            }
        }
    }

query{
    devices(pattern: "sys/tg_test/1"){
        name,
        attributes(pattern: "*scalar*") {
```

(continues on next page)

(continued from previous page)

```
                name,
                datatype,
                dataformat,
                label,
                unit,
                description,
                value,
                quality,
                timestamp
            }
            server{
            id,
            host
        }
    }
}
```

3.3 Deleting device property

```
mutation{
    deleteDeviceProperty(device:"sys/tg_test/1", name: "Hej"){
        ok,
        message
    }
}
```

3.4 Putting device property

```
mutation{
    putDeviceProperty(device:"sys/tg_test/1", name: "Hej", value: "test"){
        ok,
        message
    }
}
```

3.5 Deleting device property

```
mutation{
    deleteDeviceProperty(device:"sys/tg_test/1",name:"Hej"){
        ok,
        message
    }
}
```

3.6 Setting value for an attribute

```
mutation{
    SetAttributeValue(device:"sys/tg_test/1", name: "double_scalar",value: 2){
        ok,
        message
    }
}
```

3.7 Query all tango classes

```
query{
    classes(pattern:"*"){
        name
    }
}
```

3.8 Query all tango classes and corresponding devices

```
query{
    classes(pattern:"*"){
        name
        devices {
        name
        }
    }
}
```

CHAPTER 4

TangoGQL Logging

TangoGQL logging system uses a file called *logging.yaml* by default to configure the logging capabilites, this is an example of that file:

```
---
 version: 1
 disable_existing_loggers: False
 formatters:
   simple:
     format: "%(asctime)s - %(levelname)s - %(message)s"

 handlers:
   console:
     class: logging.StreamHandler
     level: DEBUG
     formatter: simple
     stream: ext://sys.stdout

   info_file_handler:
     class: logging.handlers.RotatingFileHandler
     level: INFO
     formatter: simple
     filename: /var/log/tangogql/info.log
     maxBytes: 10485760 # 10MB
     backupCount: 20
     encoding: utf8

   error_file_handler:
     class: logging.handlers.RotatingFileHandler
     level: ERROR
     formatter: simple
     filename: /var/log/tangogql/errors.log
     maxBytes: 10485760 # 10MB
     backupCount: 20
     encoding: utf8
```

(continues on next page)

(continued from previous page)

```
loggers:
  my_module:
    level: ERROR
    handlers: [console]
    propagate: no

root:
  level: DEBUG
  handlers: [console, info_file_handler, error_file_handler]
```

To change the format of the logging can simply change this line:

```
format: "1|%(asctime)s.%(msecs)03dZ|%(levelname)s|%(threadName)s|%(funcName)s|
↪%(filename)s#%(lineno)d|%(message)s"
```

CHAPTER 5

TangoGQL Features Toggle

TangoGQL has a function called features toggle capable of controling some features such as pub/sub. There is a file inside tangogql/ called tangogql.ini, the file looks like this:

```
# this configuration file is used to hold details of which features
# currently enabled in TangoGQL ( True = enabled False = disabled)

[feature_flags]
# Publish Subscribe is enable
publish_subscribe = True
```

Changing the *publish_subscribe = True* will enabled pub/sub on TangoGQL, in this case, TangoGQL will try to Subscribe to changeEvents on the device, if it fails it tries PeriodicEvents, and if that fails it falls back to polling

CHAPTER 6

TangoGQL Case Sensitive Convention

Tango Controls Framework uses ZMQ to manage events. ZMQ is not case sensitive so, it is necessary to define a convention to use upper case and lower case.

The Tango convention is to uses only lower case in Tango attribute name. But it accepts also the upper case. Also different tools, as POGO, permit to declare an attribute name with a different style of the lower case.

The situation can create conflicts when TangoGQL uses ZMQ to pass attribute names that are not case sensitive.

In order to avoid conflicts, TangoGQL transforms every attribute name in lower case. In this way, also if the attribute name doesn't follow the Tango Naming convention, the communication with TangoGQL proceed without problems.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index