
Tango GraphQL Documentation

Release 0.0.1

kits

May 05, 2022

1	API Documentation	3
1.1	AIOServer	3
1.2	Listener	3
1.3	Routes	3
1.4	Schema	3
1.4.1	Attribute	4
1.4.2	Base	4
1.4.3	Device	4
1.4.4	Mutation	4
1.4.5	Query	4
1.4.6	Subscriptions	4
1.4.7	Tango	4
1.4.8	Types	4
1.5	TangoDB	4
1.6	ttldict	4
2	What is GraphQL and how can be used	5
3	Examples on query and mutation	7
3.1	Fetch information of devices	7
3.2	Accessing attributes	7
3.3	Deleting device property	8
3.4	Putting device property	8
3.5	Deleting device property	8
3.6	Setting value for an attribute	9
3.7	Query all tango classes	9
3.8	Query all tango classes and corresponding devices	9
4	TangoGQL Logging	11
5	TangoGQL Features Toggle	13
6	TangoGQL Case Sensitive Convention	15
7	Indices and tables	17

A GraphQL implementation for Tango.

Contents:

CHAPTER 1

API Documentation

Contents:

1.1 AIOServer

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

1.5 TangoDB

1.6 ttldict

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 taranta suite with for example <https://gitlab.com/tango-controls/web/taranta-suite> GraphiQL url link should be accessible for you at: `http://localhost:5004/graphiql/``

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

The screenshot displays the GraphiQL IDE interface. At the top, there's a header with the 'GraphiQL' logo, a play button, and 'Prettify' and 'History' buttons. The main area is split into two panes. The left pane shows a GraphQL query:

```
1 query{
2   #filter result with pattern
3   devices(pattern: "*tg_test*"){
4     name
5   }
6 }
```

 The right pane shows the corresponding JSON response:

```
{
  "data": {
    "devices": [
      {
        "name": "sys/tg_test/1"
      }
    ]
  }
}
```

 On the right side, there's a sidebar with a 'Documentation E...' link and a search bar labeled 'Search Schema...'. Below the search bar, it says 'A GraphQL schema provides each kind of operation.' Under the 'ROOT TYPES' section, it lists:
query: Query
mutation: Mutations
subscription: Subscription
At the bottom of the left pane, there's a section labeled 'QUERY VARIABLES'.

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 capabilities, 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"
```

TangoGQL Features Toggle

TangoGQL has a function called features toggle capable of controlling 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 enable 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

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 use 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`