# Vigilent GraphQL queries and mutations

## Keywords used in the automation script

### Keywords

|  |  |  |
| --- | --- | --- |
| **#** | **gqlQuery /gqlMutation** | **gqlQuery/gqlMutation** |
| 1 | configWriteMutation | gqlMutation |
| 2 | pointWriteMutation | gqlMutation |
| 3 | setSetPointLimits | gqlMutation |
| 4 | setGroupPropertymutation | gqlMutation |
| 5 | testEventLogMutation | gqlMutation |
| 6 | setSFCMutation | gqlMutation |
| 7 | setBOPMutation | gqlMutation |
| 8 | releaseOverrideOfAllAHUsMutation | gqlMutation |
| 9 | groupNameOidMutation | gqlQuery |
| 10 | getAHUStatusInGroupQuery | gqlQuery |
| 11 | getCtrlStateValueQuery | gqlQuery |
| 12 | getRackSensorPointsOfGroupQuery | gqlQuery |
| 13 | getAlarmStatusQuery | gqlQuery |
| 14 | overrideGet | gqlQuery |
| 15 | getSpecificSensorPointDetails | gqlQuery |
| 16 | getGroupsAHUProperties | gqlQuery |
| 17 | setGrpProp | gqlMutation |
| 18 | AHUMismatchInGroup | gqlQuery |
| 19 | getAHUStateofAhuInGroup | gqlQuery |
|  |  |  |

## GraphQL Mutations:

**configWriteMutation** – This mutation is used to set the system properties in Tools->Configs->DASHM

e.g. Tools->Configs->DASHM-> Property Name – NumGuardUnits & value - 5

mutation configWrite {

configSet(requests: [{module: "DASHM", name: "NumGuardUnits", value: "5"}]) {

index

reason

}

}

**pointWriteMutation** – This mutation is used to set the temperature of Rack Bottom or Rack Top with oid.

mutation pointWrite {

pointWrite(requests: [{oid: 2220, value: 66.78}]) {

index

reason

}

}

**setSetPointLimits** – This mutation is used to set high limit temperature and low limit temperature for the Rack Top with oid.

mutation targetSetPoints {

targetSet(requests: [{oid: 2228, value: 80.8, target: LIMIT\_HIGH, origin: "MANUAL"}, {oid: 2228, value: 60.8, target: LIMIT\_LOW, origin: "MANUAL"}]) {

index

reason

}

}

**setGroupPropertymutation** – This mutation is used to set group property value based on property name with group oid.

mutation setGrpProp {

propertyWrite(requests: [{oid: 598, name: "AllowNumExceedencesGuard", int: 8}]) {

index

reason

}

}

**testEventLogMutation** – This mutation is used to set custom message for event log.

mutation TestEventLog {

eventLog(requests: [{source: "Ideavat AuQA", message: "message"}]) {

index

reason

}

}

**setSFCMutation** – This mutation is used to set SFC value for an AHU with SFC oid.

mutation targetSetSFC {

targetSet(requests: [{oid: 676, value: 88, unit: percent100, target: CONTROL, origin: "MANUAL", priority: 70}]) {

index

reason

}

}

**setBOPMutation** – This mutation is used to set BOP value for an AHU with BOP oid.

mutation targetSetBOP {

targetSet(requests: [{oid: 8440, value: 1, target: CONTROL, origin: "MANUAL", priority: 70}]) {

index

reason

}

}

**releaseOverrideOfAllAHUsMutation** – This mutation is used to release all AHUs.

mutation targetClearOverride {

targetClear(requests: [{oid: 8440, target: CONTROL, origin: "MANUAL"}, {oid: 8456, target: CONTROL, origin: "MANUAL"}, {oid: 8472, target: CONTROL, origin: "MANUAL"}, {oid: 8488, target: CONTROL, origin: "MANUAL"}, {oid: 8504, target: CONTROL, origin: "MANUAL"}, {oid: 8520, target: CONTROL, origin: "MANUAL"}, {oid: 8536, target: CONTROL, origin: "MANUAL"}, {oid: 8552, target: CONTROL, origin: "MANUAL"}, {oid: 676, target: CONTROL, origin: "MANUAL"}, {oid: 767, target: CONTROL, origin: "MANUAL"}, {oid: 858, target: CONTROL, origin: "MANUAL"}, {oid: 949, target: CONTROL, origin: "MANUAL"}, {oid: 1040, target: CONTROL, origin: "MANUAL"}, {oid: 1131, target: CONTROL, origin: "MANUAL"}, {oid: 1222, target: CONTROL, origin: "MANUAL"}, {oid: 1313, target: CONTROL, origin: "MANUAL"}]) {

index

reason

}

}

## GraphQL Queries:

**groupNameOidMutation** – This query is used to fetch the group oid based on group name.

query getGroupOid {

site {

groups: children (selector: {type: Group, name: "General-test"}) {

oid

}

}

}

**getAHUStatusInGroupQuery** - This query is used to fetch the all AHUs and their BOP & SFC oid’s based on group name.

query getAHUStatusInGroup {

site {

groups: children(selector: {type: Group, name: "General-test"}) {

oid

type

displayName

ahus: children(selector: {type: AHU}) {

oid

type

displayName

name

controls: search (selector: {target: CONTROL}, pruneDepth: false) {

oid

type

displayName

name

status: targetStatus(target: CONTROL) {

origin

}

}

}

}

}

}

**getCtrlStateValueQuery** - This query is used to fetch the group’s control state value based on group name.

query getCtrlStateValue {

site {

groups: children(selector: {type: Group, name: "General-test"}) {

children(selector: {type: GroupStatus, name: "Group Status"}) {

children(selector: {type: State, name: "CtrlState"}) {

name

pointCurrent {

value

}

}

}

}

}

}

**getRackSensorPointsOfGroupQuery** - This query is used to fetch the rack sensor points of a group based on group name.

query rackSensorPoints {

site {

groups: children(selector: {type: Group, name: "General-test"}) @skip(if: false) {

oid

name

racks: children(selector: {type: Rack}) {

oid

displayName

points: children {

oid

name

type

pointCurrent {

value

}

}

}

}

}

}

**getAlarmStatusQuery** -

query alarmStatus {

alarms(selector: {subjectName: "Imputes-test", type: GroupDeadSensor}) {

type

severity

status

}

}

**overrideGet** -

query overrideGet {

site {

groups: children(selector: {type: Group, name: " General-test"}) {

oid

name

ahus: children(selector: {type: AHU}) {

oid

name

controls: search(selector: {target: CONTROL}) {

oid

type

name

point: pointCurrent(unit: percent100) {

value

}

targetStatus(target: CONTROL, unit: percent100) {

requests {

status

origin

priority

value

unit

}

}

}

}

}

}

}

**getSpecificSensorPointDetails**

query getSpecificSensorPointDetails {

site {

name

groups: children(selector: {type: Group, name: "General-test"}) {

name

sensors: search(selector: {types: [RAT]}, maxResults: 9999) {

name

type

oid

pathName(details: true)

pointCurrent(units: [degF, kWe]) {

unit

value

tstamp

}

}

}

}

}

**MinOn**

**getGroupsAHUProperties**

query getGroupsAHUProperties {

site {

groups: children(selector: {type: Group, name: "General-test"}) {

name

ahus: children(selector: {type: AHU}) {

name

CoolSource: propString(name: "CoolSource")

DesignCapacity: prop(name: "DesignCapacity") {

value: float

unit

}

DesignCop: propFloat(name: "DesignCop")

}

}

}

} ​​

**setGrpProp**

mutation setGrpProp {

propertyWrite(requests: [{oid: 35876, name: "FanCtrlMax", float: 0.5}]) {

index

reason

}

}

**Alarm Queries**

**Query to fetch the Ahu in mismatch state in the group**

**AHUMismatchInGroup**

query AHUMismatchInGroup {

site {

groups: children(selector: {type: Group, name: "${group\_name}"}) {

nameahus: children(selector: {type: AHU}) {

name

pointCurrent: children(selector: {name: "SyncFaultStatus"}) {

type

name

pathName

SyncFaultStatus: pointCurrent {

value

}

}

}

}

}

}

**Example** –

query AHUMismatchInGroup {

site {

groups: children(selector: {type: Group, name: "NoBindings"}) {

nameahus: children(selector: {type: AHU}) {

name

pointCurrent: children(selector: {name: "SyncFaultStatus"}) {

type

name

pathName

SyncFaultStatus: pointCurrent {

value

}

}

}

}

}

}

**getAHUStateofAhuInGroup**

query getAHUStateofAhuInGroup { site { groups: children(selector: {type: Group, oid: ${group\_oid}}) { ahus: children(selector: {type: AHU}) { AHUState: prop(name: "AhuState") { name string }}}}}

**Example** -

query getAHUStateofAhuInGroup { site { groups: children(selector: {type: Group, oid: ${group\_oid}}) { ahus: children(selector: {type: AHU}) { AHUState: prop(name: "AhuState") { name string }}}}}