QPath® Python SDK User Guide

Version: 2.4.2023

© QuantumPath® 2021 | All rights reserved

Index

Installation 7
Installing QPath® Python SDK 7
Update QPath® Python SDK 7
Config file 8
QPath® SDK quickstart 9
Import SDK 9
Create new qSOA workspace 9
Get environments 10
Get active environment 11
Set active environment 12
Check security service status 13
Basic user Authentication 14
User Authentication with Base64 password 16
User Authentication with SHA-256 password 18
Login session 20
Login status 21
Version of ConnectionPoint service 22
QuantumPath® account licence 23
Solution 25
Quantum Solution List 25
Quantum Solutions 26
Quantum Solution Name 27
Device 29
Quantum Device List 29
Quantum Devices 30
Quantum Device Name 32
Flow 34
Quantum Flow List 34
Quantum Flows 35
Quantum Flow Name 37
Execution 38

© QuantumPath® 2021 | All rights reserved

Run Quantum Application 38
Quantum Execution Response 40
Run Quantum Application Synchronous 43
Execution Response Representation 45
Asset 47
Asset Catalog 47
Asset Information 49
Create Asset 51
Create Asset Flow 59
Publish Flow 63
Update Asset 64
Asset Management Result 71
Create Asset and Get Result 73
Create Asset Flow and Get Result 77
Update Asset and Get Result 80
Delete Asset 83
Execution Historic 85
Get Quantum Execution Historic 85
Get Quantum Execution Historic Result 88
Circuit Gates Creation 91
Available gates 91
Create Circuit Gates 91
Get Circuit Body 94
Get Number of Qubits 94
Get Default Qubit State 95
Get Qubit States 95
Set Default Qubit States 96
Initialize Qubit States 97
Flexible ways to create circuits 97
Basic gates 99
Hadamard gate 99
Pauli X gate 99
Pauli Y gate 100

© QuantumPath® 2021 | All rights reserved

Pauli Z gate 101
Swap gate 101
Basic gates (Control) 102
Control Hadamard gate 102
Control X gate 103
Toffoli gate 103
Prepared gates 104
Square root of Z, S gate 104
Adjoint square root Z gate 105
Square root of X gate 105
Adjoint square root X gate 106
Square root of Y gate 107
Adjoint square root Y gate 107
Four root of Z, T gate 108
Adjoint four root Z gate 109
Four root of X gate 109
Adjoint four root X gate 110
Four root of Y gate 111
Adjoint four root Y gate 111
Rotation Gates 112
Phase gate 112
Rotation X gate 113
Rotation Y gate 113
Rotation Z gate 114
Utilities 115
Measure 115
Control 115
Add Created Gate 116
Multi Controlled Gate 116
Barrier 117
Begin repeat 117
End repeat 118
Annealing Circuit Creation 119

© QuantumPath® 2021 | All rights reserved

Create Annealing Circuit 119
Parameter 120
Create Circuit Parameter 120
Add Parameter 121
Auxiliary Data 121
Create Circuit Auxiliary Data 121
Add Auxiliary Data 122
Class 123
Create Circuit Class 123
Add Class 125
Variable 126
Create Circuit Variable 126
Add Variable 127
Rule 127
Create Rule Expression 127
Create Rule 133
Add Rule 135
Circuit Flow Creation 136
Create Circuit Flow 136
Start Node 137
Init Node 137
Circuit Node 138
Repeat Node 138
End Node 139
Comment Node 139
Link Nodes 140
Summary: Object types 141
Summary: Methods 154
Summary: Custom Exceptions 175

© QuantumPath® 2021 | All rights reserved

What is QPath® Python SDK?

QPath® Python SDK is a software that allows working with quantum computers to create solutions and flows, for their execution in the main vendors. This is through the use of functions that communicate with the QPath® qSOA® architecture API. Streamlining the use of the tool and the possibility of using Jupyter Notebooks as support for managing and obtaining results from quantum circuits.

© QuantumPath® 2021 | All rights reserved

Installation

QPath® SDK is available for installation from the PyPI package repository using pip.

Installing QPath® Python SDK

With pip, the SDK can be installed using the following command:

```
pip install QuantumPathQSOAPySDK
```

After this, it is recommended to restart the kernel of the Notebook (if it is used) to make sure that the package has been recognized.

Update QPath® Python SDK

To update the package to the latest version available in the PIP repository, it is done using the following command:

```
pip install QuantumPathQSOAPySDK -U
```


© QuantumPath® 2021 | All rights reserved

Config file

In order to speed up the use of the SDK, and make use of all its functions, it is possible to create an optional configuration file. This file must be created in the home path of the computer, that is: C:\Users\user. And its name should be `.qpath`

The file has the following structure:

```
[pro-credentials]
username = username
password = password

[active-environment]
('default-environments', 'pro')

[custom-environments]
custom1 = ('url', True)
custom2 = ('url', False)
```

It contains three types of sections:

- **[pro-credentials]**
  Section that must always contain the username and password as keys, with their respective values. If new environments are added, new sections of this type could be also added, following the following nomenclature: [environment_name-credentials].

- **[active-environment]**
  It contains a Tuple of two Strings, where the first element is the environment type chosen for the use of qsoa (default-environment or custom-environment), and the second is its name.
  By default, qsoa has the pro environment.

- **[custom-environment]**
  Key-value list, where the key is the environment name, and a tuple of a string as URL and a Boolean to validate SSL certificate as value. There is no limit of number to add.

© QuantumPath® 2021 | All rights reserved

QPath® SDK quickstart

In the following points, the functions offered by the SDK will be presented in-depth. Also, when using Strings as inputs, if they contain sensitive characters such as quotes or slashes, three single quotes can be used to mark the beginning and end of the String. For example: `'''string/with"sensitive"{characters}'''`

Import SDK

Once the SDK is installed, it must be imported using the following command:

```
from QuantumPathQSOAPySDK import QSOAPlatform
```

Create new qSOA workspace

It can be done through different ways.

Create a new qSOA workspace to work with, without login in by using the following command:

```
qsoa = QSOAPlatform()
```

Create a new qSOA workspace to work with, login in manually by using the following command:

```
username = 'username'
password = 'password' # password encrypted in SHA-256

qsoa = QSOAPlatform(username, password)
```

Create a new qSOA workspace to work with, login in using .qpath config file by using the following command:

```
qsoa = QSOAPlatform(configFile=True)
```


© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.
- **ConfigFileError:** Raised when some error occurs during reading the configuration file.

Further information can be found in the Custom Exceptions summary section.

Get environments

Show QuantumPath® available environments.

It is done through the `getEnvironments` method, which returns a Dictionary showing the available environments. As can be seen in the following example:

```
environments = qsoa.getEnvironments()

print(environments)
```

```
{
   "default-environments":{
      "pro":"https://qsoa.quantumpath.app:8443/api/"
   },
   "custom-environments":{
      "custom":("url/",True)
   }
}
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.getEnvironments()`

Show QuantumPath® available environments.

Prerequisites	None
Inputs	None
Output	Dictionary

Get active environment

Show active QuantumPath® environment.

It is done through the `getActiveEnvironment` method, which returns a Tuple showing if it is a default or custom environment, and its name as result. Being "pro" environment as default. As can be seen in the following example:

```
activeEnvironment = qsoa.getActiveEnvironment()

print(activeEnvironment)
```

```
('default-environments', 'pro')
```

Summarizing the function as follows:

`qsoa.getActiveEnvironment()`

Show active QuantumPath® environment.

Prerequisites	None
Inputs	None
Output	Tuple

© QuantumPath® 2021 | All rights reserved

Set active environment

Set active QuantumPath® environment.

It is done through the `setActiveEnvironment` method, supplying the environment name as argument. It returns a Tuple showing the active environment's name. As can be seen in the following example:

```
setActiveEnvironment = qsoa.setActiveEnvironment('pro')

print(setActiveEnvironment)
```

```
('default-environments', 'pro')
```

In the case of having a custom environment, it can be added and selected as active by entering its name and url and indicating to check its SSL certificate. This will only be active at runtime. Once this is finished to use it again it must create it again. For its persistence, the config file must be used.

Summarizing the function as follows:

`qsoa.setActiveEnvironment(environmentName, qSOATargetURL, validateCert)`

Set active QuantumPath® environment.

Prerequisites	Existing QuantumPath® environment
Inputs	`environmentName` → String `qSOATargetURL` → String (Optional) `validateCert` → Boolean (Optional)
Output	Tuple

© QuantumPath® 2021 | All rights reserved

Check security service status

Test to validate if the security service is enabled.

It is done through the `echoping` method, which returns a Boolean variable showing the status of the service as a result. As can be seen in the following example:

```
ping = qsoa.echoping()

print(ping)
```

```
True
```

Summarizing the function as follows:

`qsoa.echoping()`

Test to validate if the security service is enabled.

Prerequisites	None
Inputs	None
Output	Boolean

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Basic user Authentication

Performs the user authentication process.

It is done through the `authenticateBasic` method, supplying the username and password as arguments. Returns Boolean variable showing the status of the authentication as a result. As can be seen in the following example:

```
username = 'username'
password = 'password'

authenticated = qsoa.authenticateBasic(username, password)

print(authenticated)
```

```
True
```

To avoid entering the user's credentials as plain text, it is possible to authenticate using the `.qpath` configuration file.

Once the configuration file has been created, the authentication can be done using the `authenticateBasic` function without arguments. As can be seen in the following example:

```
authenticated = qsoa.authenticateBasic()

print(authenticated)
```

```
True
```


© QuantumPath® 2021 | All rights reserved

Summarizing the functions as follows:

`qsoa.authenticateBasic(username, password)`

Performs the user authentication process.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String
Output	Boolean

`qsoa.authenticateBasic()`

Performs the user authentication process.

Prerequisites	User created in QPath® `.qpath` created
Inputs	None
Output	Boolean

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.
- **ConfigFileError:** Raised when some error occurs during reading the configuration file.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

User Authentication with Base64 password

Performs the user authentication process. With Base64 password.

It is done through the `authenticate` method, supplying the username and password as arguments (password enconded in Base64). Returns Boolean variable showing the status of the authentication as a result.

To encode password in Base64, it can be used the `encodePassword` method:

`qsoa.encodePassword(password)`

Encode password in Base64.

Prerequisites	None
Inputs	`password` → String
Output	String

It can be seen in the following example:

```
username = 'username'
password = qsoa.encodePassword('password')

authenticated = qsoa.authenticate(username, password)

print(authenticated)
```

```
True
```

To avoid entering the user's credentials as plain text, it is possible to authenticate using the `.qpath` configuration file.

© QuantumPath® 2021 | All rights reserved

Once the configuration file has been created, the authentication can be done using the `authenticate` function without arguments. As can be seen in the following example:

```
authenticated = qsoa.authenticate()

print(authenticated)
```

```
True
```

Summarizing the functions as follows:

`qsoa.authenticate(username, password)`

Performs the user authentication process. With Base64 password.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String (Base64)
Output	Boolean

`qsoa.authenticate()`

Performs the user authentication process. With Base64 password.

Prerequisites	User created in QPath® `.qpath` created
Inputs	None
Output	Boolean

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.
- **ConfigFileError:** Raised when some error occurs during reading the configuration file.
- **Base64Error:** Raised when some error occurs during Base64 encoding.

Further information can be found in the Custom Exceptions summary section.

User Authentication with SHA-256 password

Performs the user authentication process. With SHA-256 password.

Recommended authentication way. It is done through the `authenticateEx` method, supplying the username and password as arguments (password encrypted in SHA-256). Returns Boolean variable showing the status of the authentication as a result.

To encrypt password in SHA-256, it can be used the `encryptPassword` method:

`qsoa.encryptPassword(password)`

Encrypt password in SHA-256.

Prerequisites	None
Inputs	`password` → String
Output	String

© QuantumPath® 2021 | All rights reserved

It can be seen in the following example:

```
username = 'username'
password = qsoa.encryptPassword('password')

authenticated = qsoa.authenticateEx(username, password)

print(authenticated)
```

```
True
```

To avoid entering the user's credentials as plain text, it is possible to authenticate using the `.qpath` configuration file.

Once the configuration file has been created, the authentication can be done using the `authenticateEx` function without arguments. As can be seen in the following example:

```
authenticated = qsoa.authenticate()

print(authenticated)
```

```
True
```

Summarizing the functions as follows:

`qsoa.authenticateEx(username, password)`

Performs the user authentication process. With SHA-256 password.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String (SHA-256)
Output	Boolean

© QuantumPath® 2021 | All rights reserved

`qsoa.authenticateEx()`

Performs the user authentication process. With SHA-256 password.

Prerequisites	User created in QPath® `.qpath` created
Inputs	None
Output	Boolean

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.
- **ConfigFileError:** Raised when some error occurs during reading the configuration file.

Further information can be found in the Custom Exceptions summary section.

Login session

Check if user session is active.

It is done through the `echostatus` method, which returns a Boolean as a result showing the if user session is active as a result. As can be seen in the following example:

```
status = qsoa.echostatus()

print(status)
```

```
True
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.echostatus()`

Check if user session is active.

Prerequisites	None
Inputs	None
Output	Boolean

Possible custom exceptions to handle:

- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Login status

Check user login status.

It is done through the `echouser` method, which returns a String as a result showing the user's username and their authentication status as a result. As can be seen in the following example:

```
login = qsoa.echouser()

print(login)
```

```
IPrincipal-user: username - IsAuthenticated: True
```


© QuantumPath® 2021 | All rights reserved

Summarizing the functions as follows:

`qsoa.echouser()`

Check user login status.

Prerequisites	User already authenticated
Inputs	None
Output	String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Version of ConnectionPoint service

Check the ConnectionPoint service version.

It is done through the `getVersion` method, which returns a String variable showing the current version of the service as a result. As can be seen in the following example:

```
version = qsoa.getVersion()

print('Version:', version)
```

```
QuantumPath qSOA(tm) services -Development Context-
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.getVersion()`

Check the ConnectionPoint service version.

Prerequisites	User already authenticated
Inputs	None
Output	String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

QuantumPath® account licence

Returns QuantumPath account licence.

It is done through the `getLicenceInfo` method, which returns a Dictionary variable showing the account licence as a result. As can be seen in the following example:

```
licence = qsoa.getLicenceInfo()

print('Licence:', licence)
```


© QuantumPath® 2021 | All rights reserved

```
Licence: {
   "Name":"FULL_ACCESS",
   "Solutions":0,
   "Circuits":0,
   "Flows":0,
   "OnlySimulators":false,
   "UserManagement":true,
   "MaximumUsers":0,
   "qSOAEnabled":true,
   "RenovationScheme":0,
   "DirectCodeEnabled":true,
   "DirectCodeCircuits":0,
   "qSOAPublishFlows":0,
   "BackupJobs":0,
   "qAPPS":["*"]
}
```

Summarizing the function as follows:

`qsoa.getLicenceInfo()`

Returns QuantumPath account licence.

Prerequisites	User already authenticated
Inputs	None
Output	Dictionary

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Solution

Quantum Solution List

Show the list of solutions available to the user along with their IDs.

It is done through the `getQuantumSolutionList` method, which returns a Dictionary with the ID as the key, and the name of the solution as the value as a result. As can be seen in the following example:

```
solutionList = qsoa.getQuantumSolutionList()

print('Solution List:', solutionList)
```

```
Solution List: {
   "2":"BASICS",
   "19":"ANNEALING"
}
```

Summarizing the function as follows:

`qsoa.getQuantumSolutionList()`

Show the list of solutions available to the user along with their IDs.

Prerequisites	User already authenticated
Inputs	None
Output	Dictionary

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Quantum Solutions

Get the solutions available from the user as an object.

It is done through the `getQuantumSolutions` method, which returns a list of `SolutionItem` objects as a result. As can be seen in the following example:

```
solutions = qsoa.getQuantumSolutions()

print('Solutions:', solutions)
```

```
Solutions: [<QuantumPathQSOAPySDK.objects.SolutionItem.SolutionItem
object at 0x0000028F95864DC0>,
<QuantumPathQSOAPySDK.objects.SolutionItem.SolutionItem object at
0x0000028FA7425810>]
```

The object that the function returns is of type `SolutionItem`, which contains the following methods:

`SolutionItem`

Data	Type	Method
Solution ID	Integer	`getId()`
Solution Name	String	`getName()`
Quantum Type	String	`getQuantumType()`

Therefore, the data can be visualized as follows:

```
for solution in solutions:
    print(solution.getId(), solution.getName())
```

```
2 BASICS
19 ANNEALING
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.getQuantumSolutions()`

Get the solutions available from the user as an object.

Prerequisites	User already authenticated
Inputs	None
Output	List of `SolutionItem` objects

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Quantum Solution Name

Get the name of a solution.

It is done through the `getQuantumSolutionName` method, supplying the ID of the solution whose name you want to know as argument. Returns a String with the name of the solution as a result. As can be seen in the following example:

```
idSolution = 2

solutionName = qsoa.getQuantumSolutionName(idSolution)

print('Solution Name:', solutionName)
```

```
Solution Name: BASICS
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

```
qsoa.getQuantumSolutionName(idSolution)
```

Get the name of a solution.

Prerequisites	User already authenticated
Inputs	`idSolution` → Integer
Output	String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Device

Quantum Device List

Show the list of devices available in a solution along with their IDs.

It is done through the `getQuantumDeviceList` method, supplying the ID of the solution from which you want to extract its associated devices as argument. Returns a Dictionary with the ID as the key and the device name as the value as a result. As can be seen in the following example:

```
idSolution = 2

deviceList = qsoa.getQuantumDeviceList(idSolution)

print('Device List:', deviceList)
```

```
Device List: {
   "14":"AMAZON BRAKET 25qbits Local Simulator",
   "2":"QISKIT Local Simulator"
}
```

Summarizing the function as follows:

`qsoa.getQuantumDeviceList(idSolution)`

Show the list of devices available in a solution along with their IDs.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	Dictionary

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Quantum Devices

Get the available devices in a solution as an object.

It is done through the `getQuantumDevices` method, supplying the ID of the solution from which you want to extract its associated devices as argument. Returns a list of `DeviceItem` objects as a result. As can be seen in the following example:

```
idSolution = 2

devices = qsoa.getQuantumDevices(idSolution)

print('Devices:', devices)
```

```
Devices: [<QuantumPathQSOAPySDK.objects.DeviceItem.DeviceItem object
at 0x00000276C9503EB0>,
<QuantumPathQSOAPySDK.objects.DeviceItem.DeviceItem object at
0x00000276C6E0E080>]
```

The object that the function returns is of type `DeviceItem`, which contains the following methods:

`DeviceItem`

Data	Type	Method
Device Short Name	String	`getDeviceShortName()`
Device Provider	String	`getDeviceProvider()`

© QuantumPath® 2021 | All rights reserved

Quantum Machine Type	String	`getQuantumMachineType()`
Is Local Simulator	Boolean	`getIsLocalSimulator()`
Vendor ID	Integer	`getIdVendor()`
Vendor Name	String	`getVendorName()`
Description	String	`getDescription()`
Device Name	String	`getDeviceName()`

Therefore, the data can be visualized as follows:

```
for device in devices:
    print(device.getIdVendor(), device.getDeviceName())
```

```
14 AWS_LocalSimulator_Gates
15 arn:aws:braket:::device/quantum-simulator/amazon/sv1
10 ibmq_santiago
2 qasm_simulator
4 ibmq_jakarta
6 ibmq_qasm_simulator
1 MS_QDK_SIM
8 QX single-node simulator
```

Summarizing the function as follows:

`qsoa.getQuantumDevices(idSolution)`

Get the available devices in a solution as an object.

Prerequisites	User already authenticated Solution created

© QuantumPath® 2021 | All rights reserved

Inputs	`idSolution` → Integer
Output	List of `DeviceItem` objects

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Quantum Device Name

Get the name of a device.

It is done through the `getQuantumDeviceName` method, supplying the ID of the solution whose name you want to know and the name of the solution to which it belongs as arguments. Returns as a String with the name of the device as a result. As can be seen in the following example:

```
idSolution = 2
idDevice = 2

deviceName = qsoa.getQuantumDeviceName(idSolution, idDevice)

print('Device Name:', deviceName)
```

```
Device Name: QISKIT Local Simulator
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.getQuantumDeviceName(idSolution, idDevice)`

Get the name of a device.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `idDevice` → Integer
Output	String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Flow

Quantum Flow List

Show the list of flows available in a solution along with their IDs.

It is done through the `getQuantumFlowList` method, supplying the ID of the solution from which you want to extract its associated flows as argument. It returns a Dictionary with the ID as the key and the name of the flow as the value as a result. As can be seen in the following example:

```
idSolution = 2

flowList = qsoa.getQuantumFlowList(idSolution)

print('Flow List:', flowList)
```

```
Flow List: {
   "1":"ENTANGLEFlow",
   "5":"RANDOMFlow"
}
```

Summarizing the function as follows:

`qsoa.getQuantumFlowList(idSolution)`

Show the list of flows available in a solution along with their IDs.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	Dictionary

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Quantum Flows

Get the flows available in a solution as an object.

It is done through the `getQuantumFlows` method, supplying the ID of the solution from which you want to extract its associated flows as argument. Returns a list of `FlowItem` objects as a result. As can be seen in the following example:

```
idSolution = 2

flows = qsoa.getQuantumFlows(idSolution)

print('Flows:', flows)
```

```
Flows: [<QuantumPathQSOAPySDK.objects.FlowItem.FlowItem object at
0x0000019088685900>, < QuantumPathQSOAPySDK.objects.FlowItem.FlowItem
object at 0x000002B03C863F70>]
```

The object that the function returns is of type `FlowItem`, which contains the following methods:

`FlowItem`

Data	Type	Method
Flow ID	Integer	`getId()`
Flow Name	String	`getName()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
for flow in flows:
    print(flow.getId(), flow.getName())
```

```
1 ENTANGLEFlow
5 RANDOMFlow
```

Summarizing the function as follows:

`qsoa.getQuantumFlows(idSolution)`

Get the flows available in a solution as an object.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	List of `FlowItem` objects

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Quantum Flow Name

Get the name of a flow.

It is done through the `getQuantumFlowName` method, supplying the ID of the flow whose name you want to know and the name of the solution to which it belongs as argument. Returns as a String with the name of the flow as a result. As can be seen in the following example:

```
idSolution = 2
idFlow = 5

flowName = qsoa.getQuantumFlowName(idSolution, idFlow)

print('Flow Name:', flowName)
```

```
Flow Name: RANDOMFlow
```

Summarizing the function as follows:

`qsoa.getQuantumFlowName(idSolution, idFlow)`

Get the name of a flow.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `idFlow` → Integer
Output	String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Execution

Run Quantum Application

Run a created quantum solution.

It is done through the `runQuantumApplication` method, supplying a name that you want to give to this execution, and the IDs of the solution, flow, and device as arguments. Returns an `Application` object. As can be seen in the following example:

```
applicationName = 'Project_Name'
idSolution = 2
idFlow = 5
idDevice = 2

application = qsoa.runQuantumApplication(applicationName, idSolution,
idFlow, idDevice)
```

The object that the function returns is of type `Application`, which contains the following methods:

`Application`

Data	Type	Method
Application Name	String	`getApplicationName()`
Solution ID	Integer	`getIdSolution()`
Flow ID	Integer	`getIdFlow()`
Device ID	Integer	`getIdDevice()`
Execution Token	String	`getExecutionToken()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Application Name:', application.getApplicationName())
print('ID Solution:', application.getIdSolution())
print('ID Flow:', application.getIdFlow())
print('ID Device:', application.getIdDevice())
print('Execution Token:', application.getExecutionToken())
```

```
Application Name: Project_Name
ID Solution: 2
ID Flow: 5
ID Device: 2
Execution Token: fee61be0-9935-4cf6-a47a-3d7eb33c78d1
```

Summarizing the function as follows:

`qsoa.runQuantumApplication(applicationName, idSolution, idFlow, idDevice)`

Run a created quantum solution.

Prerequisites	User already authenticated Solution created
Inputs	`applicationName` → String `idSolution` → Integer `idFlow` → Integer `idDevice` → Integer
Output	`Application` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Quantum Execution Response

Get the response of a quantum solution execution.

It is done through the `getQuantumExecutionResponse` method, supplying an `Application` object whose status you want to know as argument. Returns an `Execution` object as a result. As can be seen in the following example:

```
execution = qsoa.getQuantumExecutionResponse(application)
```

Another way to use the function if you don't have an `Application` object is to manually enter the execution token, and the ID of the solution and flow as arguments. In this way, the results of previous executions can be seen.

```
executionToken = '98cd85b0-53e0-4a36-93a5-e4ae85fa22e5'
idSolution = 2
idFlow = 5

execution = qsoa.getQuantumExecutionResponse(executionToken,
idSolution, idFlow)
```

The return object is type `Execution`, which contains the following methods:

Execution

Data	Type	Method
Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Solution Name	String	`getSolutionName()`
Flow Name	String	`getFlowName()`
Device Name	String	`getDeviceName()`

© QuantumPath® 2021 | All rights reserved

Histogram	Dictionary	`getHistogram()`
Duration	Integer	`getDuration()`

Therefore, the data can be visualized as follows:

```
print('Exit Code:', execution.getExitCode())
print('Exit Message:', execution.getExitMessage())
print('Solution Name:', execution.getSolutionName())
print('Flow Name:', execution.getFlowName())
print('Device Name:', execution.getDeviceName())
print('Histogram:', execution.getHistogram())
print('Duration:', execution.getDuration())
```

```
Exit Code: OK
Exit Message: None
Solution Name: BASICS
Flow Name: RANDOMFlow
Device Name: MS_QDK_SIM
Histogram: {
   "BASICS_2_RandomCircuit_1_0":{
      "110":120,
      "001":133,
      "100":127,
      "111":143,
      "011":143,
      "010":118,
      "000":103,
      "101":113
   }
}
Duration: 135312618
```

Summarizing the function as follows:

`qsoa.getQuantumExecutionResponse(application)`

Get the response of a quantum solution execution.

Prerequisites	User already authenticated

© QuantumPath® 2021 | All rights reserved

	Solution run `Application` object generated
Inputs	`application` → `Application` object
Output	`Execution` Object

`qsoa.getQuantumExecutionResponse(executionToken, idSolution, idFlow)`

Get the response of a quantum solution execution.

Prerequisites	User already authenticated Solution run
Inputs	`executionToken` → String `idSolution` → Integer `idFlow` → Integer
Output	`Execution` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Run Quantum Application Synchronous

Run a created quantum solution synchronous.

It is done through the `runQuantumApplicationSync` method, supplying a name that you want to give to this execution, and the IDs of the solution, flow, and device as arguments. It combines the `runQuantumApplication` and `getQuantumExecutionResponse` methods. Returns an `Application` object. As can be seen in the following example:

```
applicationName = 'Project_Name'
idSolution = 2
idFlow = 5
idDevice = 2

application = qsoa.runQuantumApplicationSync(applicationName,
idSolution, idFlow, idDevice)
```

The object that the function returns is of type `Application`, which contains the following methods:

Application

Data	Type	Method
Application Name	String	`getApplicationName()`
Solution ID	Integer	`getIdSolution()`
Flow ID	Integer	`getIdFlow()`
Device ID	Integer	`getIdDevice()`
Execution Token	String	`getExecutionToken()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Application Name:', application.getApplicationName())
print('ID Solution:', application.getIdSolution())
print('ID Flow:', application.getIdFlow())
print('ID Device:', application.getIdDevice())
print('Execution Token:', application.getExecutionToken())
```

```
Application Name: Project_Name
ID Solution: 2
ID Flow: 5
ID Device: 2
Execution Token: fee61be0-9935-4cf6-a47a-3d7eb33c78d1
```

Summarizing the function as follows:

`qsoa.runQuantumApplicationSync(applicationName, idSolution, idFlow, idDevice)`

Run a created quantum solution synchronous.

Prerequisites	User already authenticated Solution created
Inputs	`applicationName` → String `idSolution` → Integer `idFlow` → Integer `idDevice` → Integer
Output	`Application` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Execution Response Representation

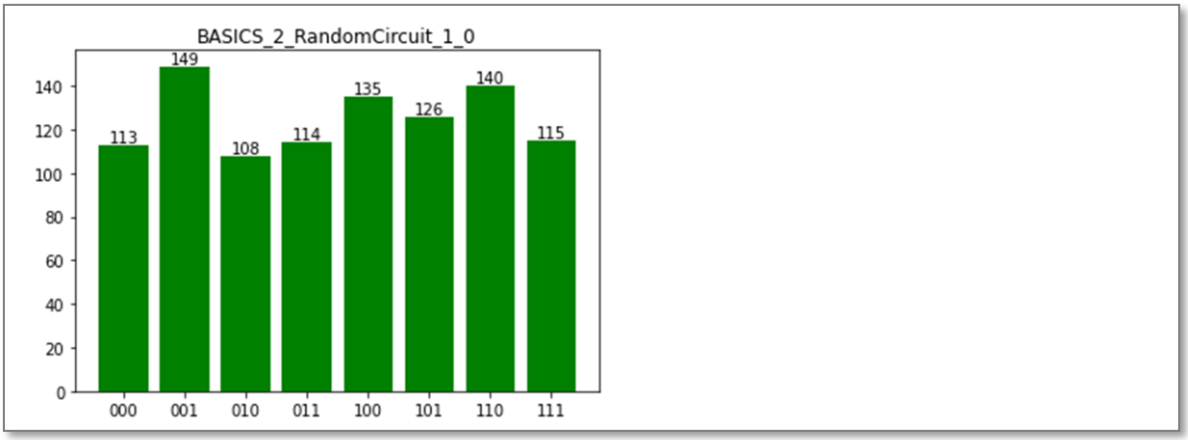
Results visual representation.

It is done through the `representResults` method, supplying an `Execution` to represent it result, and an optional argument `resultIndex` specificizing the index of the result, if there have multiple ones. Returns a png image or string with a table as a result. As can be seen in the following example:

```
representation = qsoa.representResults(execution)
```

Depending on whether it is a quantum gate or annealing circuit, the result will be a graph or a table respectively. In this way, the results would be displayed like this:

```
qsoa.representResults(execution)
```




```
representation = qsoa.representResults(execution)

print(representation)
```


© QuantumPath® 2021 | All rights reserved

```
ANNV2_45_BOXES_TEST_1
+--------------------+---------------------+---------------+------------------+
| number_of_samples | number_of_variables | sample_energy | sample_occurence |
+--------------------+---------------------+---------------+------------------+
|         32         |          5          |      34.0     |         1        |
+--------------------+---------------------+---------------+------------------+
+----------+-------+
|   Name   | Value |
+----------+-------+
| Boxes[1] |   1   |
| Boxes[2] |   1   |
| Boxes[3] |   1   |
| Boxes[4] |   1   |
| Boxes[5] |   0   |
+----------+-------+
```

Summarizing the function as follows:

`qsoa.representResults(execution, resultIndex)`

Results visual representation.

Prerequisites	User already authenticated Execution completed
Inputs	`execution` → `Execution` object `resultIndex` → Integer (Optional)
Output	png image or a table String

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.
- **ExecutionObjectError:** Raised when some error occurs reading an Execution object.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Asset

Asset Catalog

Get asset information from a solution.

It is done through the `getAssetCatalog` method, supplying as argument the ID of the solution from which you want to extract its associated asset information, type of the asset, being CIRCUIT or FLOW, and the level of the language, including VL (Visual Language) or IL (Intermediate Language). Returns a list of `Asset` objects as a result. As can be seen in the following example:

```
idSolution = 2
assetType = 'CIRCUIT'
assetLevel = 'VL'

assetCatalog = qsoa.getAssetCatalog(idSolution, assetType, assetLevel)

print('Assets:', assetCatalog)
```

```
Assets: [<QuantumPathQSOAPySDK.objects.Asset.Asset object at
0x00000117996FEAA0>, <QuantumPathQSOAPySDK.objects.Asset.Asset object
at 0x00000280D267F100>]
```

The object that the function returns is of type `Asset`, which contains the following methods:

Asset

Data	Type	Method
Asset ID	Integer	`getId()`
Asset Name	String	`getName()`
Asset Namespace	String	`getNamespace()`

© QuantumPath® 2021 | All rights reserved

Asset Description	String	`getDescription()`
Asset Body	String	`getBody()`
Asset Type	String	`getType()`
Asset Level	String	`getLevel()`
Asset Last Update	String	`getLastUpdate()`

Therefore, the data can be visualized as follows:

```
for asset in assetCatalog:
    print(asset.getId(), asset.getName())
```

```
20121 ExampleCircuit
20123 ExampleCircuitGates2
```

Summarizing the function as follows:

`qsoa.getAssetCatalog(idSolution, assetType, assetLevel)`

Get asset information from a solution.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetType` → String ('CIRCUIT' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	List of `Asset` objects

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Asset Information

Get specific asset information.

It is done through the `getAsset` method, supplying as argument the ID of the asset from which you want to extract its information, type of the asset, being CIRCUIT or FLOW, and the level of the language, including VL (Visual Language) or IL (Intermediate Language). Returns an `Asset` object as a result. As can be seen in the following example:

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'VL'

asset = qsoa.getAsset(idAsset, assetType, assetLevel)
```

The object that the function returns is of type `Asset`, which contains the following methods:

Asset

Data	Type	Method
Asset ID	Integer	`getId()`
Asset Name	String	`getName()`
Asset Namespace	String	`getNamespace()`

© QuantumPath® 2021 | All rights reserved

Asset Description	String	`getDescription()`
Asset Body	String	`getBody()`
Asset Type	String	`getType()`
Asset Level	String	`getLevel()`
Asset Last Update	String	`getLastUpdate()`

Therefore, the data can be visualized as follows:

```
print('Asset ID:', asset.getId())
print('Asset Name:', asset.getName())
print('Asset Namespace:', asset.getNamespace())
print('Asset Description:', asset.getDescription())
print('Asset Body:', asset.getBody())
print('Asset Type:', asset.getType())
print('Asset Level:', asset.getLevel())
print('Asset Last Update:', asset.getLastUpdate())
```

```
Asset ID: 20121
Asset Name: ExampleCircuit
Asset Namespace: ExampleSolution.ExampleCircuitGates
Asset Description: Example Circuit Description
Asset Body: circuit={"cols":[["H"],["CTRL","X"],["Measure"]]}
Asset Type: GATES
Asset Level: VL
Asset Last Update: 2022-03-31T06:47:06.62
```

Summarizing the function as follows:

`qsoa.getAsset(idAsset, assetType, assetLevel)`

Get specific asset information.

Prerequisites	User already authenticated Asset created

© QuantumPath® 2021 | All rights reserved

Inputs	`idAsset` → Integer `assetType` → String ('CIRCUIT' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`Asset` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Create Asset

Create Asset.

It is done through the `createAsset` method, supplying as argument the ID of the solution to add in, asset name, namespace, description, circuit body, body type being GATES, ANNEAL or FLOW, and the level of the language, including VL (Visual Language) or IL (Intermediate Language).

The circuit body variable admits several inputs:

- In the case of a gates circuit:
    - If the visual language is selected, it can be entered as a String, or a `CircuitGates` object. As shown in the following examples respectively:

© QuantumPath® 2021 | All rights reserved

```
idSolution = 2
assetName = 'ExampleCircuit'
assetNamespace = 'ExampleSolution.Example'
assetDescription = 'Example'
assetBody = 'circuit={"cols":[["H"],["CTRL","X"],["Measure"]]}'
assetType = 'GATES'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

```
circuitGates = qsoa.CircuitGates()
circuitGates.h(0)
circuitGates.cx(0, 1)
circuitGates.measure(0)

idSolution = 2
assetName = 'ExampleCircuit'
assetNamespace = 'ExampleSolution.Example'
assetDescription = 'Example'
assetBody = circuitGates
assetType = 'GATES'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

- If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

```
idSolution = 2
assetName = 'ExampleCircuit'
assetNamespace = 'ExampleSolution.Example'
assetDescription = 'Example'
assetBody = 'H(0,"");CNOT(0,1,"");M(0,"");'
assetType = 'GATES'
assetLevel = 'IL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```


© QuantumPath® 2021 | All rights reserved

- In the case of an annealing circuit:
    - If the visual language is selected, it can be entered as a String, or a `CircuitAnnealing` object. As shown in the following examples respectively:

```
idSolution = 45
assetName = 'ExampleCircuit'
assetNamespace = 'Example'
assetDescription = 'Example'
assetBody = '''{"Parameters":[],"AuxData":[{"uiID":"1dd94b8f-193b-
4db0-94b1-
b3722539c733","Name":"Prices","Value":"[4,1,2,3,5]","_isInvalid":false
},{"uiID":"96cbcec1-ec26-4200-86f5-
bca553855115","Name":"Weights","Value":"[1,2,3,4,4]","_isInvalid":fals
e}],"Classes":[{"Properties":[],"uiID":"cfc3899a-01b0-4060-8c8b-
90a9a861d4fc","Name":"Boxes","NumberOfVars":"5","Description":"","_isI
nvalid":false}],"Variables":[{"Classes":["cfc3899a-01b0-
e4492a88a7b4","Type":"SQUARED","Coefficient":"","Offset":"","From":"",
"To":"","Iterator":"","Term1":{"Indexes":[],"uiID":"c0a5f1c6-cfde-
403c-8cc7-
6a8ea256a6aa","VariableID":""},"Term2":{"Indexes":[],"uiID":"7c9664cc-
8c7f-46fc-9d61-
77bfb37e5ad7","VariableID":""},"Childs":[{"uiID":"5820904a-26cb-491c-
af2f-
15e58fe01417","Type":"SUMMATORY","Coefficient":"","Offset":"","From":"
i","To":"Boxes","Iterator":"i","Term1":{"Indexes":[],"uiID":"ab5303d6-
7e80-4876-ae51-7a7565a3f464","VariableID":""}'''
assetType = 'ANNEAL'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```


© QuantumPath® 2021 | All rights reserved

```
circuit = qsoa.CircuitAnnealing()
circuit.addAuxData([
    circuit.AuxData('Prices', [4,1,2,3,5]),
    circuit.AuxData('Weights', [1,2,3,4,4])
])
boxesClass = circuit.addClass(
    circuit.Class('Boxes', 5)
)
boxesVariable = circuit.addVariable(
    circuit.Variable('Boxes', boxesClass)
)
circuit.addRule([
    circuit.Rule('Rule1', '1/5', 'Maximize the package
price').addExpression(
        circuit.SummationExp(1, 'Boxes',
circuit.LinearExp((boxesVariable, 'i'), '-Prices[i]'))
    ),
    circuit.Rule('Rule2', 1, 'Total weight must be
6kg').addExpression(
        circuit.SquaredExp([
            circuit.SummationExp(1, 'Boxes', [
                circuit.LinearExp((boxesVariable, 'i'), 'Weights[i]')
            ]),
            circuit.OffsetExp(-6)
        ])
    )
])

idSolution = 45
assetName = 'ExampleCircuit'
assetNamespace = 'Example'
assetDescription = 'Example'
assetBody = circuit
assetType = 'ANNEAL'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

- If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

© QuantumPath® 2021 | All rights reserved

```
idSolution = 45
assetName = 'ExampleCircuit'
assetNamespace = 'Example'
assetDescription = 'Example'
assetBody = '''
AUXDATA(Prices|"[4,1,2,3,5]");
AUXDATA(Weights|"[1,2,3,4,4]");
CLASS(Boxes|5|"");
VARIABLE(Boxes|{Boxes}|"");
RULE(Rule1|"Maximize the package price"|"1/5"|
    {
        SUMMATORY(from 1 to Boxes iterate i|
            {
                LINEAR(Boxes[i]| "-Prices[i]")
            }
        )
    }
);
RULE(Rule2|"Total weight must be 6kg"|"1"|
    {
        SQUARED(
            {
                SUMMATORY(from i to Boxes iterate i|
                    {
                        LINEAR(Boxes[i]| "Weights[i]")
                    }
                )
                ,OFFSET("6")
            }
        )
    }
);
'''
assetType = 'ANNEAL'
assetLevel = 'IL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```


© QuantumPath® 2021 | All rights reserved

- In the case of a flow:
    - If the visual language is selected, it can be entered as a String, or a `CircuitFlow` object. As shown in the following examples respectively:

```
idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody = '''{ "class": "go.GraphLinksModel",
  "nodeDataArray": [
{"category":"Start", "text":"Start", "key":-1, "loc":"-294 -309"},
{"category":"Circuit", "text":" Entanglement", "key":-3, "loc":""},
{"category":"Repeat", "text":"1000", "key":-5, "loc":"-155. -70"},
{"category":"End", "text":"End", "key":-6, "loc":"-199. 119"},
{"category":"Init", "text":"0", "key":-2, "loc":"-183. -284"}
 ],
  "linkDataArray": [
{"from":-5, "to":-6, "visible":true, "points":[-155,98], text":"end"},
{"from":-1, "to":-2, "points":[-269,-309,-221,-284,-211,-284]},
{"from":-2, "to":-3, "points":[-183,-264,-183,-179,-227]},
{"from":-3, "to":-5, "points":[-179,-186,-144,-155,-102]} ]}'''
assetType = 'FLOW'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

```
flow = qsoa.CircuitFlow()
startNode = flow.startNode()
initNode = flow.initNode(0)
circuitNode = flow.circuitNode('circuitName')
repeatNode = flow.repeatNode(1000)
endNode = flow.endNode()
flow.linkNodes(startNode, initNode)
flow.linkNodes(initNode, circuitNode)
flow.linkNodes(circuitNode, repeatNode)
flow.linkNodes(repeatNode, endNode)

idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody = flow
assetType = 'FLOW'
assetLevel = 'VL'

assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```


© QuantumPath® 2021 | All rights reserved

- If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

```
idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody =
'ABSTRACT(START,);REPEAT(0|1000,CIRCUIT(Basics.Entanglement));ABSTRACT
(END,);'
assetType = 'FLOW'
assetLevel = 'IL'
assetManagementData = qsoa.createAsset(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

The object that the function returns is of type `AssetManagementData`, which contains the following methods:

`AssetManagementData`

Data	Type	Method
Asset Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Lifecycle Token:', assetManagementData.getLifecycleToken())
print('Solution ID:', assetManagementData.getIdSolution())
print('Asset ID:', assetManagementData.getIdAsset())
print('Asset Name:', assetManagementData.getAssetName())
print('Asset Namespace:', assetManagementData.getAssetNamespace())
print('Asset Type:', assetManagementData.getAssetType())
print('Asset Level:', assetManagementData.getAssetLevel())
print('Asset Compiled Status:',
assetManagementData.getAssetCompiledStatus())
print('Asset Transpiled Status:',
assetManagementData.getAssetTranspiledStatus())
```

```
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: False
Asset Transpiled Status: False
```

Summarizing the function as follows:

`qsoa.createAsset(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Create Asset.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String `assetBody` → String / `CircuitGates` object / `CircuitFlow` object

© QuantumPath® 2021 | All rights reserved

	`assetType` → String ('GATES', 'ANNEAL' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`AssetManagementData` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Create Asset Flow

Create Flow Asset shorter method.

It is done through the `createAssetFlow` method, supplying as argument the ID of the solution to add in, asset name, namespace, description, circuit body, level of the language, including VL (Visual Language) or IL (Intermediate Language), and if it wants to be published.

The circuit body variable admits several inputs:

- If the visual language is selected, it can be entered as a String, or a `CircuitFlow` object. As shown in the following examples respectively:

© QuantumPath® 2021 | All rights reserved

```
idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody = '''{ "class": "go.GraphLinksModel",
  "nodeDataArray": [
{"category":"Start", "text":"Start", "key":-1, "loc":"-294 -309"},
{"category":"Circuit", "text":" Entanglement", "key":-3, "loc":""},
{"category":"Repeat", "text":"1000", "key":-5, "loc":"-155. -70"},
{"category":"End", "text":"End", "key":-6, "loc":"-199. 119"},
{"category":"Init", "text":"0", "key":-2, "loc":"-183. -284"}
 ],
  "linkDataArray": [
{"from":-5, "to":-6, "visible":true, "points":[-155,98], text":"end"},
{"from":-1, "to":-2, "points":[-269,-309,-221,-284,-211,-284]},
{"from":-2, "to":-3, "points":[-183,-264,-183,-179,-227]},
{"from":-3, "to":-5, "points":[-179,-186,-144,-155,-102]} ]}'''
assetLevel = 'VL'
publish = True

assetManagementData = qsoa.createAssetFlow(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetLevel, publish)
```

```
flow = qsoa.CircuitFlow()
startNode = flow.startNode()
initNode = flow.initNode(0)
circuitNode = flow.circuitNode('circuitName')
repeatNode = flow.repeatNode(1000)
endNode = flow.endNode()
flow.linkNodes(startNode, initNode)
flow.linkNodes(initNode, circuitNode)
flow.linkNodes(circuitNode, repeatNode)
flow.linkNodes(repeatNode, endNode)

idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody = flow
assetLevel = 'VL'
publish = True

assetManagementData = qsoa.createAssetFlow(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetLevel, publish)
```

- If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

© QuantumPath® 2021 | All rights reserved

```
idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody =
'ABSTRACT(START,);REPEAT(0|1000,CIRCUIT(Basics.Entanglement));ABSTRACT
(END,);'
assetLevel = 'IL'
publish = True
assetManagementData = qsoa.createAssetFlow(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetLevel, publish)
```

The object that the function returns is of type `AssetManagementData`, which contains the following methods:

`AssetManagementData`

Data	Type	Method
Asset Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Lifecycle Token:', assetManagementData.getLifecycleToken())
print('Solution ID:', assetManagementData.getIdSolution())
print('Asset ID:', assetManagementData.getIdAsset())
print('Asset Name:', assetManagementData.getAssetName())
print('Asset Namespace:', assetManagementData.getAssetNamespace())
print('Asset Type:', assetManagementData.getAssetType())
print('Asset Level:', assetManagementData.getAssetLevel())
print('Asset Compiled Status:',
assetManagementData.getAssetCompiledStatus())
print('Asset Transpiled Status:',
assetManagementData.getAssetTranspiledStatus())
```

```
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: False
Asset Transpiled Status: False
```

Summarizing the function as follows:

`qsoa.createAssetFlow(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetLevel, publish)`

Create Flow Asset shorter method.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String

© QuantumPath® 2021 | All rights reserved

	`assetBody` → String / `CircuitFlow` object `assetLevel` → String ('VL' or 'IL') `publish` → Boolean
Output	`AssetManagementData` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Publish Flow

Change publish flow status.

It is done through the `publishFlow` method, supplying as input the flow identificator and a Boolean with the publish status. Returns a Boolean indicating the new publish status. As can be seen in the following example:

```
idAsset = 609
publish = True

publishUpdated = qsoa.publishFlow(idAsset, publish)
print('Published:', publishUpdated)
```

```
Published: True
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`qsoa.publishFlow(idAsset, publish)`

Change publish flow status.

Prerequisites	User already authenticated Access permission to the flow
Inputs	`idAsset` → Integer `publish` → Boolean
Output	Boolean

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Update Asset

Update Asset values.

It is done through the `updateAsset` method, supplying as argument the Asset object to change their information, and as optional arguments the data to change: asset name, namespace, description, circuit body, body type being GATES, ANNEAL or FLOW, and the level of the language, including VL (Visual Language) or IL (Intermediate Language).

© QuantumPath® 2021 | All rights reserved

In the case that the field to be updated is the circuit body, it admits several inputs:

- In the case of a gates circuit:
    - If the visual language is selected, it can be entered as a String, or a `CircuitGates` object. As shown in the following examples respectively:

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = 'circuit={"cols":[["H"],["CTRL","X"],["Measure"]]}'

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

```
circuitGates = qsoa.CircuitGates()
circuitGates.h(0)
circuitGates.cx(0, 1)
circuitGates.measure(0)

idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = circuitGates

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

    - If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'IL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = 'H(0,"");CNOT(0,1,"");M(0,"");'

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```


© QuantumPath® 2021 | All rights reserved

- In the case of an annealing circuit:
    - If visual or intermediate language is selected, it must be entered as a String. As shown in the following examples:

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = '''{"Parameters":[],"AuxData":[{"uiID":"1dd94b8f-193b-
4db0-94b1-
b3722539c733","Name":"Prices","Value":"[4,1,2,3,5]","_isInvalid":false
},{"uiID":"96cbcec1-ec26-4200-86f5-
bca553855115","Name":"Weights","Value":"[1,2,3,4,4]","_isInvalid":fals
e}],"Classes":[{"Properties":[],"uiID":"cfc3899a-01b0-4060-8c8b-
90a9a861d4fc","Name":"Boxes","NumberOfVars":"5","Description":"","_isI
nvalid":false}],"Variables":[{"Classes":["cfc3899a-01b0-
e4492a88a7b4","Type":"SQUARED","Coefficient":"","Offset":"","From":"",
"To":"","Iterator":"","Term1":{"Indexes":[],"uiID":"c0a5f1c6-cfde-
403c-8cc7-
6a8ea256a6aa","VariableID":""},"Term2":{"Indexes":[],"uiID":"7c9664cc-
8c7f-46fc-9d61-
77bfb37e5ad7","VariableID":""},"Childs":[{"uiID":"5820904a-26cb-491c-
af2f-
15e58fe01417","Type":"SUMMATORY","Coefficient":"","Offset":"","From":"
i","To":"Boxes","Iterator":"i","Term1":{"Indexes":[],"uiID":"ab5303d6-
7e80-4876-ae51-7a7565a3f464","VariableID":""}'''

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```


© QuantumPath® 2021 | All rights reserved

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'IL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = '''
AUXDATA(Prices|"[4,1,2,3,5]");
AUXDATA(Weights|"[1,2,3,4,4]");
CLASS(Boxes|5|"");
VARIABLE(Boxes|{Boxes}|"");
RULE(Rule1|"Maximize the package price"|"1/5"|
    {
        SUMMATORY(from 1 to Boxes iterate i|
            {
                LINEAR(Boxes[i]| "-Prices[i]")
            }
        )
    }
);
RULE(Rule2|"Total weight must be 6kg"|"1"|
    {
        SQUARED(
            {
                SUMMATORY(from i to Boxes iterate i|
                    {
                        LINEAR(Boxes[i]| "Weights[i]")
                    }
                )
                ,OFFSET("6")
            }
        )
    }
);
'''

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

- In the case of a flow:
    - If the visual language is selected, it can be entered as a String, or a `CircuitFlow` object. As shown in the following examples respectively:

© QuantumPath® 2021 | All rights reserved

```
idAsset = 20121
assetType = 'FLOW'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = '''
{ "class": "go.GraphLinksModel",
  "nodeDataArray": [
{"category":"Start", "text":"Start", "key":-1, "loc":"-294 -309"},
{"category":"Circuit", "text":" Entanglement", "key":-3, "loc":""},
{"category":"Repeat", "text":"1000", "key":-5, "loc":"-155. -70"},
{"category":"End", "text":"End", "key":-6, "loc":"-199. 119"},
{"category":"Init", "text":"0", "key":-2, "loc":"-183. -284"}
 ],
  "linkDataArray": [
{"from":-5, "to":-6, "visible":true, "points":[-155,98], text":"end"},
{"from":-1, "to":-2, "points":[-269,-309,-221,-284,-211,-284]},
{"from":-2, "to":-3, "points":[-183,-264,-183,-179,-227]},
{"from":-3, "to":-5, "points":[-179,-186,-144,-155,-102]} ]}
'''

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

```
flow = qsoa.CircuitFlow()
startNode = flow.startNode()
initNode = flow.initNode(0)
circuitNode = flow.circuitNode('circuitName')
repeatNode = flow.repeatNode(1000)
endNode = flow.endNode()

flow.linkNodes(startNode, initNode)
flow.linkNodes(initNode, circuitNode)
flow.linkNodes(circuitNode, repeatNode)
flow.linkNodes(repeatNode, endNode)

idAsset = 20121
assetType = 'FLOW'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = flow

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

- If the intermediate language is selected, it can only be entered as a String. As shown in the following example:

© QuantumPath® 2021 | All rights reserved

```
idAsset = 20121
assetType = 'FLOW'
assetLevel = 'IL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody =
'ABSTRACT(START,);REPEAT(0|1000,CIRCUIT(Basics.Entanglement));ABSTRACT
(END,);'

assetManagementData = qsoa.updateAsset(asset, assetBody=newAssetBody)
```

The object that the function returns is of type `AssetManagementData`, which contains the following methods:

`AssetManagementData`

Data	Type	Method
Asset Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Lifecycle Token:', assetManagementData.getLifecycleToken())
print('Solution ID:', assetManagementData.getIdSolution())
print('Asset ID:', assetManagementData.getIdAsset())
print('Asset Name:', assetManagementData.getAssetName())
print('Asset Namespace', assetManagementData.getAssetNamespace())
print('Asset Type:', assetManagementData.getAssetType())
print('Asset Level:', assetManagementData.getAssetLevel())
print('Asset Compiled Status:',
assetManagementData.getAssetCompiledStatus())
print('Asset Transpiled Status:',
assetManagementData.getAssetTranspiledStatus())
```

```
Lifecycle Token: c7e8d47c-7abc-46ab-abe9-c1568aa9920e
Solution ID: 10095
Asset ID: 20121
Asset Name: ExampleCircuit
Asset Namespace ExampleSolution.ExampleCircuitGates
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: False
Asset Transpiled Status: False
```

Summarizing the function as follows:

`qsoa.updateAsset(asset, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Update Asset values.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object `assetName` → String (Optional) `assetNamespace` → String (Optional) `assetDescription` → String (Optional)

© QuantumPath® 2021 | All rights reserved

	`assetBody` → String (Optional) / `CircuitGates` object (Optional) / `CircuitFlow` object (Optional) `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') (Optional) `assetLevel` → String ('VL' or 'IL') (Optional)
Output	`AssetManagementData` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Asset Management Result

Get Asset Management Result from a lifecycle token.

It is done through the `getAssetManagementResult` method, supplying as argument the asset lifecycle token. Returns an `AssetManagementResult` object as output. As can be seen in the following example:

```
lifecycleToken = 'a22511b3-3a3a-4e7c-b281-b62802533a79'

assetManagementResult = qsoa.getAssetManagementResult(lifecycleToken)
```

The object that the function returns is of type `AssetManagementResult`, which contains the following methods:

`AssetManagementResult`

Data	Type	Method

© QuantumPath® 2021 | All rights reserved

Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

Therefore, the data can be visualized as follows:

```
print('Exit Code:', assetManagementResult.getExitCode())
print('Exit Message:', assetManagementResult.getExitMessage())
print('Lifecycle Token:', assetManagementResult.getLifecycleToken())
print('Solution ID:', assetManagementResult.getIdSolution())
print('Asset ID:', assetManagementResult.getIdAsset())
print('Asset Name:', assetManagementResult.getAssetName())
print('Asset Namespace:', assetManagementResult.getAssetNamespace())
print('Asset Type:', assetManagementResult.getAssetType())
print('Asset Level:', assetManagementResult.getAssetLevel())
print('Asset Compiled Status:',
assetManagementResult.getAssetCompiledStatus())
print('Asset Transpiled Status',
assetManagementResult.getAssetTranspiledStatus())
```


© QuantumPath® 2021 | All rights reserved

```
Exit Code: OK
Exit Message:
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: True
Asset Transpiled Status False
```

Summarizing the function as follows:

`qsoa.getAssetManagementResult(lifecycleToken)`

Get Asset Management Result from a lifecycle token.

Prerequisites	Existing asset lifecycle token
Inputs	`lifecycleToken` → String
Output	`AssetManagementResult` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Create Asset and Get Result

Create Asset and get result.

It is done through the `createAssetSync` method, supplying as argument the ID of the solution to add in, asset name, namespace, description, circuit body, body type being GATES, ANNEAL or FLOW, and the level of the language, including VL (Visual Language) or

© QuantumPath® 2021 | All rights reserved

IL (Intermediate Language). It combines the `createAsset` and `getAssetManagementResult` methods. Returns an `AssetManagementResult` object. As can be seen in the following example:

```
idSolution = 2
assetName = 'ExampleCircuit'
assetNamespace = 'ExampleSolution.Example'
assetDescription = 'Example'
assetBody = 'circuit={"cols":[["H"],["CTRL","X"],["Measure"]]}'
assetType = 'GATES'
assetLevel = 'VL'

assetManagementResult = qsoa.createAssetSync(idSolution, assetName,
assetNamespace, assetDescription, assetBody, assetType, assetLevel)
```

The object that the function returns is of type `AssetManagementResult`, which contains the following methods:

`AssetManagementResult`

Data	Type	Method
Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`

© QuantumPath® 2021 | All rights reserved

Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

Therefore, the data can be visualized as follows:

```
print('Exit Code:', assetManagementResult.getExitCode())
print('Exit Message:', assetManagementResult.getExitMessage())
print('Lifecycle Token:', assetManagementResult.getLifecycleToken())
print('Solution ID:', assetManagementResult.getIdSolution())
print('Asset ID:', assetManagementResult.getIdAsset())
print('Asset Name:', assetManagementResult.getAssetName())
print('Asset Namespace:', assetManagementResult.getAssetNamespace())
print('Asset Type:', assetManagementResult.getAssetType())
print('Asset Level:', assetManagementResult.getAssetLevel())
print('Asset Compiled Status:',
assetManagementResult.getAssetCompiledStatus())
print('Asset Transpiled Status',
assetManagementResult.getAssetTranspiledStatus())
```

```
Exit Code: OK
Exit Message:
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: True
Asset Transpiled Status False
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

```
qsoa.createAssetSync(idSolution, assetName, assetNamespace,
    assetDescription, assetBody, assetType, assetLevel)
```

Create Asset and get result.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String `assetBody` → String / `CircuitGates` object / `CircuitFlow` object `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`AssetManagementResult` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Create Asset Flow and Get Result

Create asset flow and get result.

It is done through the `createAssetFlowSync` method, supplying as argument the ID of the solution to add in, asset name, namespace, description, circuit body, level of the language, including VL (Visual Language) or IL (Intermediate Language), and if it wants to be published. It combines the `createAssetFlow` and `getAssetManagementResult` methods. Returns an `AssetManagementResult` object. As can be seen in the following example:

```
idSolution = 2
assetName = 'ExampleFlow'
assetNamespace = 'ExampleSolution.ExampleFlow'
assetDescription = 'Example'
assetBody =
'ABSTRACT(START,);REPEAT(0|1000,CIRCUIT(Basics.Entanglement));ABSTRACT
(END,);'
assetLevel = 'IL'
publish = True

assetManagementResult = qsoa.createAssetFlowSync(idSolution,
assetName, assetNamespace, assetDescription, assetBody, assetLevel,
publish)
```

The object that the function returns is of type `AssetManagementResult`, which contains the following methods:

`AssetManagementResult`

Data	Type	Method
Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Lifecycle Token	String	`getLifecycleToken()`

© QuantumPath® 2021 | All rights reserved

Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

Therefore, the data can be visualized as follows:

```
print('Exit Code:', assetManagementResult.getExitCode())
print('Exit Message:', assetManagementResult.getExitMessage())
print('Lifecycle Token:', assetManagementResult.getLifecycleToken())
print('Solution ID:', assetManagementResult.getIdSolution())
print('Asset ID:', assetManagementResult.getIdAsset())
print('Asset Name:', assetManagementResult.getAssetName())
print('Asset Namespace:', assetManagementResult.getAssetNamespace())
print('Asset Type:', assetManagementResult.getAssetType())
print('Asset Level:', assetManagementResult.getAssetLevel())
print('Asset Compiled Status:',
assetManagementResult.getAssetCompiledStatus())
print('Asset Transpiled Status',
assetManagementResult.getAssetTranspiledStatus())
```


© QuantumPath® 2021 | All rights reserved

```
Exit Code: OK
Exit Message:
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: True
Asset Transpiled Status False
```

Summarizing the function as follows:

`qsoa.createAssetFlowSync(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetLevel, publish)`

Create asset flow and get result.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String `assetBody` → String / `CircuitFlow` object `assetLevel` → String ('VL' or 'IL') `publish` → Boolean
Output	`AssetManagementResult` object

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Update Asset and Get Result

Update Asset values and get result.

It is done through the `updateAssetSync` method, supplying as argument the Asset object to change their information, and as optional arguments the data to change: asset name, namespace, description, circuit body, body type being GATES, ANNEAL or FLOW, and the level of the language, including VL (Visual Language) or IL (Intermediate Language). It combines the `updateAssetSync` and `getAssetManagementResult` methods. Returns an `AssetManagementResult` object. As can be seen in the following example:

```
idAsset = 20121
assetType = 'CIRCUIT'
assetLevel = 'VL'
asset = qsoa.getAsset(idAsset, assetType, assetLevel)
newAssetBody = 'circuit={"cols":[["H"],["CTRL","X"],["Measure"]]}'

assetManagementResult = qsoa.updateAssetSync(asset,
assetBody=newAssetBody)
```


© QuantumPath® 2021 | All rights reserved

The object that the function returns is of type `AssetManagementData`, which contains the following methods:

`AssetManagementData`

Data	Type	Method
Asset Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Exit Code:', assetManagementResult.getExitCode())
print('Exit Message:', assetManagementResult.getExitMessage())
print('Lifecycle Token:', assetManagementResult.getLifecycleToken())
print('Solution ID:', assetManagementResult.getIdSolution())
print('Asset ID:', assetManagementResult.getIdAsset())
print('Asset Name:', assetManagementResult.getAssetName())
print('Asset Namespace:', assetManagementResult.getAssetNamespace())
print('Asset Type:', assetManagementResult.getAssetType())
print('Asset Level:', assetManagementResult.getAssetLevel())
print('Asset Compiled Status:',
assetManagementResult.getAssetCompiledStatus())
print('Asset Transpiled Status',
assetManagementResult.getAssetTranspiledStatus())
```

```
Exit Code: OK
Exit Message:
Lifecycle Token: a22511b3-3a3a-4e7c-b281-b62802533a79
Solution ID: 10095
Asset ID: 20124
Asset Name: Example
Asset Namespace: ExampleSolution.Example
Asset Type: GATES
Asset Level: VL
Asset Compiled Status: True
Asset Transpiled Status False
```

Summarizing the function as follows:

`qsoa.updateAssetSync(asset, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Update Asset values and get result.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object `assetName` → String (Optional)

© QuantumPath® 2021 | All rights reserved

	`assetNamespace` → String (Optional) `assetDescription` → String (Optional) `assetBody` → String (Optional) / `CircuitGates` object (Optional) / `CircuitFlow` object (Optional) `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') (Optional) `assetLevel` → String ('VL' or 'IL') (Optional)
Output	`AssetManagementResult` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Delete Asset

Delete asset.

It is done through the `deleteAsset` method, supplying an `Asset` object to delete as argument. Returns a Boolean variable as a result. As can be seen in the following example:

```
assetDeleted = qsoa.deleteAsset(asset)

print(assetDeleted)
```

```
True
```

Another way to use the function if you don't have an `Asset` object is to manually enter the ID of the asset, and the type being CIRCUIT or FLOW. In this way, the results can be seen.

© QuantumPath® 2021 | All rights reserved

```
idAsset = 20121
assetType = 'CIRCUIT'

assetDeleted = qsoa.deleteAsset(idAsset, assetType)

print(assetDeleted)
```

```
True
```

Summarizing the function as follows:

`qsoa.deleteAsset(asset)`

Delete asset.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object
Output	Boolean

`qsoa.deleteAsset(idAsset, assetType)`

Delete asset.

Prerequisites	User already authenticated Asset created
Inputs	`idAsset` → Integer `assetType` → String ('CIRCUIT' or 'FLOW')
Output	Boolean

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Execution Historic

Get Quantum Execution Historic

Get a list of quantum execution history entries.

It is done through the `getQuantumExecutionHistoric` method, supplying as optional arguments different filters for the search. There are solution ID, flow ID, device ID, date from to start the search (yyyy-mm-ddThh:mm:ss), Boolean if was executed in simulator, number of top results (10 as default), and a Boolean indicating if were good results with True, or an error result with False. Returns a list of `QuantumExecutionHistoryEntry` objects as a result. As can be seen in the following example:

```
idSolution = 10391 # in a specific solution
idFlow = None
idDevice = None
dateFrom = None
isSimulator = True # only run in simulators
top = 2 # last two results

quantumExecutionHistoryEntryList =
qsoa.getQuantumExecutionHistoric(idSolution, idFlow, idDevice,
dateFrom, isSimulator, top)

print('History Entry List: ', quantumExecutionHistoryEntryList)
```

```
History Entry List:
[<QuantumPathQSOAPySDK.objects.QuantumExecutionHistoryEntry.QuantumExe
cutionHistoryEntry object at 0x000001D48B18F400>,
<QuantumPathQSOAPySDK.objects.QuantumExecutionHistoryEntry.QuantumExec
utionHistoryEntry object at 0x000001D49A624B80>]
```


© QuantumPath® 2021 | All rights reserved

The object that the function returns is of type `QuantumExecutionHistoryEntry`, which contains the following methods:

`QuantumExecutionHistoryEntry`

Data	Type	Method
Result ID	Integer	`getIdResult()`
Solution ID	Integer	`getIdSolution()`
Solution Name	String	`getSolutionName()`
Flow ID	Integer	`getIdFlow()`
Flow Name	String	`getFlowName()`
Device ID	Integer	`getIdDevice()`
Device Name	String	`getDeviceName()`
Device Short Name	String	`getDeviceShortName()`
Device Vendor	String	`getDeviceVendor()`
Is Local Simulator	Boolean	`getIsLocalSimulator()`
Device Type Name	String	`getDeviceTypeName()`
Result Histogram	String	`getResultHistogram()`
Execution Date	String	`getExecutionDate()`
Duration in minutes	Float	`getDurationMinutes()`

© QuantumPath® 2021 | All rights reserved

Result Type	String	`getResultType()`
Result Description	String	`getResultDescription()`

Therefore, the data can be visualized as follows:

```
for quantumExecutionHistoryEntry in quantumExecutionHistoryEntryList:
    print(quantumExecutionHistoryEntry.getIdResult(),
quantumExecutionHistoryEntry.getFlowName())
```

```
22276 Random_Number_Flow
22261 Random_Number_Flow
```

Summarizing the function as follows:

`qsoa.getQuantumExecutionHistoric(idSolution, idFlow, idDevice, dateFrom, isSimulator, top, resultType)`

Get a list of quantum execution history entries.

Prerequisites	User already authenticated
Inputs	`idSolution` → Integer (Optional) `idFlow` → Integer (Optional) `idDevice` → Integer (Optional) `dateFrom` → String (yyyy-mm-ddThh:mm:ss) (Optional) `isSimulator` → Boolean (Optional) `top` → Integer (Optional) `resultType` → Boolean (Optional)
Output	List of `QuantumExecutionHistoryEntry` objects

© QuantumPath® 2021 | All rights reserved

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

Get Quantum Execution Historic Result

Get a quantum execution history entry.

It is done through the `getQuantumExecutionHistoricResult` method, supplying as argument a result ID. Returns a `QuantumExecutionHistoryEntry` object as a result. As can be seen in the following example:

```
idResult = 22276

quantumExecutionHistoryEntry =
qsoa.getQuantumExecutionHistoricResult(idResult)

print('Quantum Execution History Entry: ',
quantumExecutionHistoryEntry)
```

```
Quantum Execution History Entry:
<QuantumPathQSOAPySDK.objects.QuantumExecutionHistoryEntry.QuantumExec
utionHistoryEntry object at 0x0000020C67EA57B0>
```

The object that the function returns is of type `QuantumExecutionHistoryEntry`, which contains the following methods:

`QuantumExecutionHistoryEntry`

Data	Type	Method
Result ID	Integer	`getIdResult()`

© QuantumPath® 2021 | All rights reserved

Solution ID	Integer	`getIdSolution()`
Solution Name	String	`getSolutionName()`
Flow ID	Integer	`getIdFlow()`
Flow Name	String	`getFlowName()`
Device ID	Integer	`getIdDevice()`
Device Name	String	`getDeviceName()`
Device Short Name	String	`getDeviceShortName()`
Device Vendor	String	`getDeviceVendor()`
Is Local Simulator	Boolean	`getIsLocalSimulator()`
Device Type Name	String	`getDeviceTypeName()`
Result Histogram	String	`getResultHistogram()`
Execution Date	String	`getExecutionDate()`
Duration in minutes	Float	`getDurationMinutes()`
Result Type	String	`getResultType()`
Result Description	String	`getResultDescription()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print(quantumExecutionHistoryEntry.getIdResult(),
quantumExecutionHistoryEntry.getFlowName())
```

```
22276 Random_Number_Flow
```

Summarizing the function as follows:

`qsoa.getQuantumExecutionHistoricResult(idResult)`

Get a quantum execution history entry.

Prerequisites	User already authenticated
Inputs	`idResult` → Integer
Output	`QuantumExecutionHistoryEntry` object

Possible custom exceptions to handle:

- **AuthenticationError:** Raised when user is not authenticated.
- **APIConnectionError:** Raised when some error occurs during API connection.

Further information can be found in the Custom Exceptions summary section.

© QuantumPath® 2021 | All rights reserved

Circuit Gates Creation

Available gates

Basic gates
- h → Hadamard
- x → Pauli X
- y → Pauli Y
- z → Pauli Z
- swap → Swap

Basic gates (Control)
- ch → Control Hadamard
- cx → Control Not
- ccx → Toffoli

Prepared gates
- s → S gate
- i_s → Adjoint square root Z gate
- sx → Square root of X gate
- i_sx → Adjoint square root X gate
- sy → Square root of Y gate
- i_sy → Adjoint square root Y gate
- t → T gate
- i_t → Adjoint four root Z gate
- tx → Four root of X gate
- i_tx → Adjoint four root X gate
- ty → Four root of Y gate
- i_ty → Adjoint four root Y gate

Rotation Gates
- p → Phase
- rx → Rotation X
- ry → Rotation Y
- rz → Rotation Z

Utilities
- measure → Measure
- control → Control
- barrier → Barrier
- mcg → Multi Control Gate

Create Circuit Gates

To create a gate circuit, the `CircuitGates` object must be instantiated, generating an empty circuit and being able to add gates through its methods. As shown in the following example:

```
circuit = qsoa.CircuitGates()
```

Object `CircuitGates` contains the following methods:

`CircuitGates`

Function	Method
Get Circuit Body	`getCircuitBody()`
Get Number of Qubits	`getNumberOfQubits()`
Get Default Qubit State	`getDefaultQubitState()`

© QuantumPath® 2021 | All rights reserved

Get Qubit States	`getQubitStates()`
Set Default Qubit State	`setDefaultQubitState()`
Initialize Qubit States	`initializeQubitStates()`
Add Hadamard gate	`h(position, add)`
Add Pauli X gate	`x(position, add)`
Add Swap gates	`swap(position1, position2, add)`
Add Pauli Y gate	`y(position, add)`
Add Pauli Z gate	`z(position, add)`
Add Control Hadamard gate	`ch(position1, position2, add)`
Add Control X gate	`cx(position1, position2, add)`
Add Toffoli gate	`ccx(position1, position2, position3, add)`
Add Square root of Z, S gate	`s(position, add)`
Add Adjoint square root Z gate	`i_s(position, add)`
Add Square root of X gate	`sx(position, add)`
Add Adjoint square root X gate	`i_sx(position, add)`
Add Square root of Y gate	`sy(position, add)`
Add Adjoint square root Y gate	`i_sy(position, add)`

© QuantumPath® 2021 | All rights reserved

Add Fourth root of Z, T gate	`t(position, add)`
Add Adjoint four root Z gate	`i_t(position, add)`
Add Four root of X gate	`tx(position, add)`
Add Adjoint four root X gate	`i_tx(position, add)`
Add Four root of Y gate	`ty(position, add)`
Add Adjoint four root Y gate	`i_ty(position, add)`
Add Phase gate	`p(position, argument, add)`
Add Rotation X gate	`rx(position, argument, add)`
Add Rotation Y gate	`ry(position, argument, add)`
Add Rotation Z gate	`rz(position, argument, add)`
Add Measure	`measure(position)`
Add Control to gate	`control(position, circuit)`
Add Created Gate	`addCreatedGate(gate)`
Add Multi Control Gate	`mcg(position, circuit)`
Add Barrier	`barrier(position)`

© QuantumPath® 2021 | All rights reserved

Get Circuit Body

```
getCircuitBody()
```

Example:

```
circuit = qsoa.CircuitGates()
circuit.h(0)
circuit.x(1)

circuitBody = circuit.getCircuitBody()

print(circuitBody)
```

```
[['H', 'X']]
```

Get Number of Qubits

```
getNumberOfQubits()
```

Example:

```
circuit = qsoa.CircuitGates()

circuit.h(0)
circuit.x(1)

numberOfQubits = circuit.getNumberOfQubits()

print(numberOfQubits)
```

```
2
```


© QuantumPath® 2021 | All rights reserved

Get Default Qubit State

```
getDefaultQubitState()
```

Example:

```
circuit = qsoa.CircuitGates() # create circuit

circuit.h(0)
circuit.x(1)

defaultQubitState = circuit.getDefaultQubitState()

print(defaultQubitState)
```

```
0
```

Get Qubit States

```
getQubitStates()
```

Example:

```
circuit = qsoa.CircuitGates() # create circuit

circuit.h(0)
circuit.x(1)

qubitStates = circuit.getQubitStates()

print(qubitStates)
```

```
['0', '0']
```


© QuantumPath® 2021 | All rights reserved

Set Default Qubit States

```
setDefaultQubitState(qubitState)
```

- qubitState → Set default qubit state. It can be 0, 1, +, -, i or -i.

Example:

```
circuit = qsoa.CircuitGates() # create circuit

circuit.setDefaultQubitState('1')
circuit.h(0)
circuit.x(1)

qubitStates = circuit.getQubitStates()

print(qubitStates)
```

```
['1', '1']
```


© QuantumPath® 2021 | All rights reserved

Initialize Qubit States

```
initializeQubitStates(qubitStates)
```

- qubitStates → List of strings setting up the qubit states. It must be equal than number of qubits. There can be 0, 1, +, -, i or -i.

Example:

```
circuit = qsoa.CircuitGates() # create circuit
circuit.h(0)
circuit.x(1)

circuit.initializeQubitStates(['1', '+'])

qubitStates = circuit.getQubitStates()

print(qubitStates)
```

```
['1', '+']
```

Flexible ways to create circuits

For creating circuits there are two different ways to add gates. As shown later in the description of each one, they have an argument called "add". This changes the behavior of the gate. By default, it is True, which means that once the function is used, the gate is added to the circuit. However, if it is explicitly set to False, the function will return the body of the gate itself, which can be added later with the `addCreatedGate` function.

This type of behavior saves many lines of code if you want to use the gates shown in this guide, while it increases the versatility of circuit construction if you want to create new gates and quickly add them to the circuit.

Below is shown an entanglement circuit first declaring the gates and adding them manually with `addCreatedGate` method.

© QuantumPath® 2021 | All rights reserved

```
circuit = qsoa.CircuitGates()

h_gate = circuit.h(0, False)
cx_gate = circuit.cx(0, 1, False)

circuit.addCreatedGate(h_gate)
circuit.addCreatedGate(cx_gate)

circuit.measure([0, 1])
```

On the other hand, if the use of the “add” argument is omitted (True by default), the gate is added directly to the circuit. Saving lines of code and increasing clarity.

```
circuit = qsoa.CircuitGates()

circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0, 1])
```

Another use of the gates “add” argument is to create new gates using the control method. For example, it can be used the cnot gate in two different ways: the first using the gate created specifically for it, as can be seen in the examples above, or creating it from scratch by adding a control to a not gate and adding lately it to the circuit.

```
circuit = qsoa.CircuitGates()

cx_gate = circuit.control(0, circuit.x(1, False))

circuit.addCreatedGate(cx_gate)
```


© QuantumPath® 2021 | All rights reserved

Basic gates

Hadamard gate

```
h(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.h(0)

print(circuit.getCircuitBody())
```

```
[['H']]
```

Pauli X gate

```
x(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.x(0)

print(circuit.getCircuitBody())
```

```
[['X']]
```

Pauli Y gate

```
y(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.y(0)

print(circuit.getCircuitBody())
```

```
[['Y']]
```


© QuantumPath® 2021 | All rights reserved

Pauli Z gate

```
z(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.z(0)

print(circuit.getCircuitBody())
```

```
[['Z']]
```

Swap gate

```
swap(position1, position2, add)
```

- position1 → Mandatory argument. First qubit position to add the swap.
- position2 → Mandatory argument. Second qubit position to add the swap.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.swap(0, 2)

print(circuit.getCircuitBody())
```

```
[['Swap', 1, 'Swap']]
```

Basic gates (Control)

Control Hadamard gate

```
ch(position1, position2, add)
```

- position1 → Mandatory argument. First qubit position to add the gate.
- position2 → Mandatory argument. Second qubit position to add the gate.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.ch(0, 1)

print(circuit.getCircuitBody())
```

```
[['CTRL', 'H']]
```


© QuantumPath® 2021 | All rights reserved

Control X gate

```
cx(position1, position2, add)
```

- position1 → Mandatory argument. First qubit position to add the gate.
- position2 → Mandatory argument. Second qubit position to add the gate.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.cx(0, 1)

print(circuit.getCircuitBody())
```

```
[['CTRL', 'X']]
```

Toffoli gate

```
ccx(position1, position2, position3, add)
```

- position1 → Mandatory argument. First qubit position to add the gate.
- position2 → Mandatory argument. Second qubit position to add the gate.
- position3 → Mandatory argument. Third qubit position to add the gate.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.ccx(0, 1, 2)

print(circuit.getCircuitBody())
```

```
[['CTRL', 'CTRL', 'X']]
```

Prepared gates

Square root of Z, S gate

```
s(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.s(0)

print(circuit.getCircuitBody())
```

```
[['S']]
```


© QuantumPath® 2021 | All rights reserved

Adjoint square root Z gate

```
i_s(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_s(0)

print(circuit.getCircuitBody())
```

```
[['I_S']]
```

Square root of X gate

```
sx(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.sx(0)

print(circuit.getCircuitBody())
```

```
[['SX']]
```

Adjoint square root X gate

```
i_sx(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_sx(0)

print(circuit.getCircuitBody())
```

```
[['I_SX']]
```


© QuantumPath® 2021 | All rights reserved

Square root of Y gate

```
sy(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.sy(0)

print(circuit.getCircuitBody())
```

```
[['SY']]
```

Adjoint square root Y gate

```
i_sy(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_sy(0)

print(circuit.getCircuitBody())
```

```
[['I SY']]
```

Four root of Z, T gate

```
t(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.t(0)

print(circuit.getCircuitBody())
```

```
[['T']]
```


© QuantumPath® 2021 | All rights reserved

Adjoint four root Z gate

```
i_t(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_t(0)

print(circuit.getCircuitBody())
```

```
[['I_T']]
```

Four root of X gate

```
tx(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.tx(0)

print(circuit.getCircuitBody())
```

```
[['TX']]
```

Adjoint four root X gate

```
i_tx(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_tx(0)

print(circuit.getCircuitBody())
```

```
[['I_TX']]
```


© QuantumPath® 2021 | All rights reserved

Four root of Y gate

```
ty(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.ty(0)

print(circuit.getCircuitBody())
```

```
[['TY']]
```

Adjoint four root Y gate

```
i_ty(position, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.i_ty(0)

print(circuit.getCircuitBody())
```

```
[['I TY']]
```

Rotation Gates

Phase gate

```
p(position, argument, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- argument → Optional argument. Gate angle value. In the case that it is not indicated, it will be pi by default.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.p(0)

print(circuit.getCircuitBody())
```

```
[[{'id': 'P', 'arg': 'pi'}]]
```


© QuantumPath® 2021 | All rights reserved

Rotation X gate

```
rx(position, argument, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- argument → Optional argument. Gate angle value. In the case that it is not indicated, it will be pi by default.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.rx(0)

print(circuit.getCircuitBody())
```

```
[[{'id': 'RX', 'arg': 'pi'}]]
```

Rotation Y gate

```
ry(position, argument, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- argument → Optional argument. Gate angle value. In the case that it is not indicated, it will be pi by default.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.ry(0)

print(circuit.getCircuitBody())
```

```
[[{'id': 'RY', 'arg': 'pi'}]]
```

Rotation Z gate

```
rz(position, argument, add)
```

- position → Optional argument. Qubit position to add the gate. If no position are indicated, gate will be added in all qubits. Argument can also be a list of positions.
- argument → Optional argument. Gate angle value. In the case that it is not indicated, it will be pi by default.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

Example:

```
circuit = qsoa.CircuitGates()

circuit.rz(0)

print(circuit.getCircuitBody())
```

```
[[{'id': 'RZ', 'arg': 'pi'}]]
```


© QuantumPath® 2021 | All rights reserved

Utilities

Measure

```
measure(position)
```

- position → Optional argument. Qubit position to add the measurement. In the case that the position is not indicated, the measurement will be added in all qubits. It can also be a list of positions.

Example:

```
circuit = qsoa.CircuitGates()

circuit.measure(0)

print(circuit.getCircuitBody())
```

```
[['Measure']]
```

Control

```
control(position, circuit)
```

It creates a new gate but doesn't add it to the existing circuit. Use `addCreatedGate` function to add the created gate.

- position → Mandatory argument. Qubit position to add the control.
- circuit → Gate or set of elements to add a control.

For example, it can be used the cnot gate in two different ways: the first using the gate created specifically for it, shown in the Basic gates (Control) section, or creating it from scratch by adding a control to a not gate and adding lately it to the circuit.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

cx_gate = circuit.control(0, circuit.x(1, False))

circuit.addCreatedGate(cx_gate)
```

Add Created Gate

```
addCreatedGate(gate)
```

- gate → Created gate to add to the circuit.

Example:

```
circuit.addCreatedGate(cccx)

print(circuit.getCircuitBody())
```

```
[['CTRL', 'CTRL', 'CTRL', 'X']]
```

Multi Controlled Gate

```
mcg(position, circuit, add)
```

- position → Qubit position or list of positions to add the control.
- circuit → Gate or set of elements to add a control.
- add → Optional argument. True by default. Indicates whether the gate should be added to the circuit or not. In the case of wanting to add it, it is not necessary to introduce that argument. If you want to create a new gate, you must enter False.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.mcg([0, 1], circuit.x(2, False))

print(circuit.getCircuitBody())
```

```
[['CTRL', 'CTRL', 'X']]
```

Barrier

```
barrier(position)
```

- position → Optional argument. Qubit position to add the barrier. In the case that the position is not indicated, the barrier will be added in all qubits. It can also be a list of positions.

Example:

```
circuit = qsoa.CircuitGates()

circuit.barrier(0)

print(circuit.getCircuitBody())
```

```
[['SPACER']]
```

Begin repeat

```
beginRepeat(position, repetitions)
```

- position → Qubit position to add the begin repetition. It can also be a list of positions.
- repetition → Number of repetitions.

© QuantumPath® 2021 | All rights reserved

Example:

```
circuit = qsoa.CircuitGates()

circuit.beginRepeat(0, 2)

print(circuit.getCircuitBody())
```

```
[[{'id': 'BEGIN_R', 'arg': '2'}]]
```

End repeat

```
endRepeat(position)
```

- position → Qubit position to add the end repetition. It can also be a list of positions.

Example:

```
circuit = qsoa.CircuitGates()

circuit.endRepeat(0)

print(circuit.getCircuitBody())
```

```
[['END_R']]
```


© QuantumPath® 2021 | All rights reserved

Annealing Circuit Creation

Create Annealing Circuit

To create an annealing circuit, the `CircuitAnnealing` object must be instantiated, generating an empty circuit and being able to add elements through its methods. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

print(circuit.getCircuitBody())
```

```
{'Parameters': [], 'AuxData': [], 'Classes': [], 'Variables': [],
'Rules': []}
```

Object `CircuitAnnealing` contains the following methods:

`CircuitAnnealing`

Function	Method
Get Circuit Body	`getCircuitBody()`
Add Parameter	`addParameter(parameter)`
Add Auxiliary Data	`addAuxData(auxData)`
Add Class	`addClass(cls)`
Add Variable	`addVariable(variable)`
Add Rule	`addRule(rule)`

© QuantumPath® 2021 | All rights reserved

Parameter

Create Circuit Parameter

To create a circuit parameter, the `Parameter` object must be instantiated. Supplying as arguments the parameter name and value. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

parameter = circuit.Parameter('Parameter', 3)
```

Object `Parameter` contains the following methods:

`Parameter`

Function	Method
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Value	`getValue()`

Therefore, the data can be visualized as follows:

```
print('UI ID:', parameter.getUiID())
print('Name:', parameter.getName())
print('Value:', parameter.getValue())
```

```
UI ID: 3debdf9c-9587-406e-9711-4f2a91a1643a
Name: Parameter
Value: 3
```


© QuantumPath® 2021 | All rights reserved

Add Parameter

Add Circuit Parameter.

It is done through the `addParameter` method, supplying as argument `Parameter` object or list of `Parameter` objects to add to the circuit. As can be seen in the following example:

```
circuit.addParameter(parameter)
```

Summarizing the function as follows:

`circuit.addParameter(parameter)`

Add Circuit Parameter.

Prerequisites	Created circuit
Inputs	`parameter` → `Parameter` object / `Parameter` object list
Output	`Parameter` object / `Parameter` object list

Auxiliary Data

Create Circuit Auxiliary Data

To create a circuit auxiliary data, the `AuxData` object must be instantiated. Supplying as arguments the auxiliary data name and value. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

auxData = circuit.AuxData('AuxData', [1, 2])
```


© QuantumPath® 2021 | All rights reserved

Object `AuxData` contains the following methods:

`AuxData`

Function	Method
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Value	`getValue()`

Therefore, the data can be visualized as follows:

```
print('UI ID:', auxData.getUiID())
print('Name:', auxData.getName())
print('Value:', auxData.getValue())
```

```
UI ID: f33c974a-bb8b-4eef-9883-5f2a287fdc26
Name: AuxData
Value: [1, 2]
```

Add Auxiliary Data

Add Circuit Auxiliary Data.

It is done through the `addAuxData` method, supplying as argument `AuxData` object or list of `AuxData` objects to add to the circuit. As can be seen in the following example:

```
circuit.addAuxData(auxData)
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`circuit.addAuxData(auxData)`

Add Circuit Auxiliary Data.

Prerequisites	Created circuit
Inputs	`auxData` → `AuxData` object / `AuxData` object list
Output	`AuxData` object / `AuxData` object list

Class

Create Circuit Class

To create a circuit class, the `Class` object must be instantiated. Supplying as arguments the class name, number of variables as int or `Parameter` object, and an optional description. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

cls = circuit.Class('Class', 2, 'Description')
```

Object `Class` contains the following methods:

`Class`

Function	Method
Add Property	`addProperty()`
Get Properties	`getProperties()`

© QuantumPath® 2021 | All rights reserved

Get UI ID	`getUiID()`
Get Name	`getName()`
Get Number of Variables	`getNumberOfVars()`
Get Description	`getDescription()`

Therefore, the data can be visualized as follows:

```
print('Properties:', cls.getProperties())
print('UI ID:', cls.getUiID())
print('Name:', cls.getName())
print('Number of variables:', cls.getNumberOfVars())
print('Name:', cls.getName())
```

```
Properties: []
UI ID: d3173f36-70ba-4ca6-a2ad-9b308a203edd
Name: Class
Number of variables: 2
Description: Description
```

To add a property to a created class, `addProperty` method should be used. Supplying as arguments the property name, and value (it may be the same that Class number of variables). As shown in the following example:

```
cls.addProperty('Property', [1, 1])
```


© QuantumPath® 2021 | All rights reserved

Summarizing the function as follows:

`cls.addProperty(name, value)`

Add Class Property.

Prerequisites	Created class
Inputs	`name` → String `value` → Integer / Integer list
Output	`AuxData` object / `AuxData` object list

Add Class

Add Circuit Class.

It is done through the `addClass` method, supplying as argument `Class` object or list of `Class` objects to add to the circuit. As can be seen in the following example:

```
circuit.addClass(cls)
```

Summarizing the function as follows:

`circuit.addClass(cls)`

Add Circuit Class.

Prerequisites	Created circuit
Inputs	`cls` → `Class` object / `Class` object list
Output	`Class` object / `Class` object list

© QuantumPath® 2021 | All rights reserved

Variable

Create Circuit Variable

To create a circuit class, the `Variable` object must be instantiated. Supplying as arguments the variable name, a `Class` object or list of `Class` objects, and an optional description. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

variable = circuit.Variable('Variable', cls, 'Description')
```

Object `Variable` contains the following methods:

`Variable`

Function	Method
Get Classes	`getClasses()`
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Description	`getDescription()`

Therefore, the data can be visualized as follows:

```
print('Classes:', variable.getClasses())
print('UI ID:', variable.getUiID())
print('Name:', variable.getName())
print('Description:', variable.getDescription())
```


© QuantumPath® 2021 | All rights reserved

```
Classes: ['a3ededa5-b992-4383-857e-99d7483528f3']
UI ID: 0f6cc936-f766-4654-af93-7d0b420b5924
Name: Variable
Description: Description
```

Add Variable

Add Circuit Variable.

It is done through the `addVariable` method, supplying as argument `Variable` object or list of `Variable` objects to add to the circuit. As can be seen in the following example:

```
circuit.addVariable(variable)
```

Summarizing the function as follows:

`circuit.addVariable(variable)`

Add Circuit Variable.

Prerequisites	Created circuit
Inputs	`variable` → `Variable` object / `Variable` object list
Output	`Variable` object / `Variable` object list

Rule

Create Rule Expression

There are 5 expression types. To create a rule expression one of the following objects should be instantiated: `OffsetExp`, `LinearExp`, `QuadraticExp`, `SquaredExp` or `SummationExp`.

© QuantumPath® 2021 | All rights reserved

To create an offset expression, the `OffsetExp` object must be instantiated. Supplying as argument the offset value. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

offsetExp = circuit.OffsetExp(1)
```

Object `OffsetExp` contains the following methods:

`OffsetExp`

Function	Method
Get Expression Type	`getExpType()`
Get Offset	`getOffset()`

Therefore, the data can be visualized as follows:

```
print('Expression type:', offsetExp.getExpType())
print('Offset:', offsetExp.getOffset())
```

```
Expression type: OFFSET
Offset: 1
```

To create a linear expression, the `LinearExp` object must be instantiated. Supplying as arguments a tuple with a `Variable` object an its value or list of values (one per class in variable), and an optional coefficient. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

linearExp = circuit.LinearExp((variable, 1), 2)
```


© QuantumPath® 2021 | All rights reserved

Object `LinearExp` contains the following methods:

`LinearExp`

Function	Method
Get Expression Type	`getExpType()`
Get Term 1	`getTerm1()`
Get Coefficient	`getCoefficient()`

Therefore, the data can be visualized as follows:

```
print('Expression type:', linearExp.getExpType())
print('Term 1:', linearExp.getTerm1())
print('Coefficient:', linearExp.getCoefficient())
```

```
Expression type: LINEAR
Term 1: {'Indexes': [{'uiID': '29b7ca4b-8e16-4386-af12-9ff40ce2b8b1',
'Value': '1', 'ClassID': '01129c25-89fa-4fb5-b50d-1b41777ada64'}],
'uiID': '505a790d-e6a1-437d-ac61-8bff3d462288', 'VariableID':
'7e5327ae-46db-449f-91db-8c8742213ecf'}
Coefficient: 2
```

To create a quadratic expression, the `QuadraticExp` object must be instantiated. Supplying as arguments a tuple with a `Variable` object an its value or list of values (one per class in variable), a second tuple with the same logic, and an optional coefficient. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

quadraticExp = circuit.QuadraticExp((variable1, 1), (variable2, 2))
```


© QuantumPath® 2021 | All rights reserved

Object `QuadraticExp` contains the following methods:

`QuadraticExp`

Function	Method
Get Expression Type	`getExpType()`
Get Term 1	`getTerm1()`
Get Term 2	`getTerm2()`
Get Coefficient	`getCoefficient()`

Therefore, the data can be visualized as follows:

```
print('Expression type:', quadraticExp.getExpType())
print('Term 1:', quadraticExp.getTerm1())
print('Term 2:', quadraticExp.getTerm2())
print('Coefficient:', quadraticExp.getCoefficient())
```

```
Expression type: QUADRATIC
Term 1: {'Indexes': [{'uiID': '8c162bac-5fc2-408b-8b36-630491417ea8',
'Value': '1', 'ClassID': '117ce3a7-5ded-40f5-9191-f142ea91b6c6'}],
'uiID': '91e9135e-9082-44fd-93db-1b9d7df8b1b8', 'VariableID':
'6dea0604-9c92-4b90-aba6-7dd274f28920'}
Term 2: {'Indexes': [{'uiID': '45f91771-b380-45e7-aa9a-1b0b986fafbe',
'Value': '2', 'ClassID': '117ce3a7-5ded-40f5-9191-f142ea91b6c6'}],
'uiID': 'a01f43c5-2104-424c-88a3-fcc3fcb6eaf2', 'VariableID':
'66397c93-2c35-4c35-a468-f40c7d76b07c'}
Coefficient: 3
```

To create a squared expression, the `SquaredExp` object must be instantiated. Supplying as argument a `SummationExp`, `LinearExp`, `OffsetExp` object or a list of those objects. As shown in the following example:

© QuantumPath® 2021 | All rights reserved

```
circuit = qsoa.CircuitAnnealing()

squaredExp = circuit.SquaredExp(offsetExp)
```

Object `SquaredExp` contains the following methods:

SquaredExp

Function	Method
Get Expression Type	`getExpType()`
Get Childs	`getChilds()`

Therefore, the data can be visualized as follows:

```
print('Expression type:', squaredExp.getExpType())
print('Childs:', squaredExp.getChilds())
```

```
Expression type: SQUARED
Childs: [{'uiID': '31ab2196-53cf-4a81-85d5-d824d0625f45', 'Type':
'OFFSET', 'Coefficient': '', 'Offset': '1', 'From': '', 'To': '',
'Iterator': '', 'Term1': {}, 'Term2': {}, 'Childs': []}]
```

To create a summation expression, the `SummationExp` object must be instantiated. Supplying as arguments the initial value to iterate and the last, the iterator symbol, "i" as default, and the expression to iterate. It could be a `SummationExp`, `SquaredExp`, `LinearExp`, `QuadraticExp`, `OffsetExp` object or a list of those objects. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

summationExp = circuit.SummationExp(1, 5, circuit.OffsetExp('i'))
```


© QuantumPath® 2021 | All rights reserved

Object `SummationExp` contains the following methods:

`SummationExp`

Function	Method
Get Expression Type	`getExpType()`
Get From Value	`getFrom()`
Get To Value	`getTo()`
Get Iterator	`getIterator()`
Get Childs	`getChilds()`

Therefore, the data can be visualized as follows:

```
print('Expression type:', sumatoryExp.getExpType())
print('From Value:', sumatoryExp.getFrom())
print('To Value:', sumatoryExp.getTo())
print('Iterator:', sumatoryExp.getIterator())
print('Childs:', squaredExp.getChilds())
```

```
Expression type: SUMMATORY
From Value: 1
To Value: 5
Iterator: i
Childs: [{'uiID': 'e76087f0-04bc-40e5-8926-ac47aa4d2633', 'Type':
'OFFSET', 'Coefficient': '', 'Offset': '1', 'From': '', 'To': '',
'Iterator': '', 'Term1': {}, 'Term2': {}, 'Childs': []}]
```


© QuantumPath® 2021 | All rights reserved

Create Rule

To create a circuit rule, the `Rule` object must be instantiated. Supplying as arguments the rule name, lambda value, an optional description and, optionally, if it is disabled. As shown in the following example:

```
circuit = qsoa.CircuitAnnealing()

rule = circuit.Rule('Rule', 1, 'Description')
```

Object `Rule` contains the following methods:

`Rule`

Function	Method
Add Expression	`addExpression()`
Get Expressions	`getExpressions()`
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Lambda	`getLambda()`
Get Description	`getDescription()`
Get Disabled	`getDisabled()`

© QuantumPath® 2021 | All rights reserved

Therefore, the data can be visualized as follows:

```
print('Expressions:', rule.getExpressions())
print('UI ID:', rule.getUiID())
print('Name:', rule.getName())
print('Lambda:', rule.getLambda())
print('Description:', rule.getDescription())
print('Disabled:', rule.getDisabled())
```

```
Expressions: []
UI ID: 1a6d4f92-c3e0-4d05-8572-df78a8698f74
Name: Rule
Lambda: 1
Description: Description
Disabled: False
```

To add an expression to a created rule, `addExpression` method should be used. Supplying as argument a single object or a list of the following objects: `SummationExp`, `SquaredExp`, `LinearExp`, `QuadraticExp`, or `OffsetExp`. As shown in the following example:

```
rule.addExpression([offsetExp, linearExp])
```

Summarizing the function as follows:

`rule.addExpression(expression)`

Add Rule Expression.

Prerequisites	Created rule
Inputs	`expression` → String
Output	`SummationExp` object / `SquaredExp` object / `LinearExp` object / `QuadraticExp` object / `OffsetExp` object / list

© QuantumPath® 2021 | All rights reserved

Add Rule

Add Circuit Rule.

It is done through the `addRule` method, supplying as argument `Rule` object or list of `Rule` objects to add to the circuit. As can be seen in the following example:

```
circuit.addRule(rule)
```

Summarizing the function as follows:

`circuit.addRule(rule)`

Add Circuit Rule.

Prerequisites	Created circuit
Inputs	`rule` → `Rule` object / `Rule` object list
Output	`Rule` object / `Rule` object list

© QuantumPath® 2021 | All rights reserved

Circuit Flow Creation

Create Circuit Flow

To create a flow, the `CircuitFlow` object must be instantiated, generating an empty flow and being able to add elements through its methods. As shown in the following example:

```
flow = qsoa.CircuitFlow()

print(flow.getFlowBody())
```

```
{'class': 'go.GraphLinksModel', 'nodeDataArray': [], 'linkDataArray': []}
```

Object `CircuitFlow` contains the following methods:

`CircuitFlow`

Function	Method
Get Flow Body	`getFlowBody()`
Add Start node	`startNode()`
Add Init node	`initNode(startValue)`
Add Circuit node	`circuitNode(circuitName)`
Add Repeat node	`repeatNode(numReps)`
Add End node	`endNode()`
Add Comment node	`commentNode(comment)`

© QuantumPath® 2021 | All rights reserved

Add Link nodes	`linkNodes(fromNode, toNode)`

Start Node

```
startNode()
```

Example:

```
flow = qsoa.CircuitFlow()

flow.startNode(0)

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'Start', 'text': '0', 'key': -1, 'loc': ''}]
```

Init Node

```
initNode(startValue)
```

- startValue → Initial value for the flow iterations.

Example:

```
flow = qsoa.CircuitFlow()

flow.initNode(0)

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'Init', 'text': '0', 'key': -2, 'loc': ''}]
```


© QuantumPath® 2021 | All rights reserved

Circuit Node

```
circuitNode(circuitName)
```

- circuitName → Circuit name to introduce in the flow.

Example:

```
flow = qsoa.CircuitFlow()

flow.circuitNode('circuitName')

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'Circuit', 'text': 'circuitName', 'key': -3, 'loc': ''}]
```

Repeat Node

```
repeatNode(numReps)
```

- numReps → Number of circuit repetitions.

Example:

```
flow = qsoa.CircuitFlow()

flow.repeatNode(1000)

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'Repeat', 'text': '1000', 'key': -4, 'loc': ''}]
```


© QuantumPath® 2021 | All rights reserved

End Node

```
endNode()
```

Example:

```
flow = qsoa.CircuitFlow()

flow.endNode()

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'End', 'text': 'End', 'key': -5, 'loc': ''}]
```

Comment Node

```
commentNode(comment)
```

- comment → Comment.

Example:

```
flow = qsoa.CircuitFlow()

flow.commentNode('Comment')

print(flow.getFlowBody()['nodeDataArray'])
```

```
[{'category': 'End', 'text': 'End', 'key': -5, 'loc': ''}]
```


© QuantumPath® 2021 | All rights reserved

Link Nodes

```
linkNodes(fromNode, toNode)
```

- fromNode → Origin node to link.
- toNode → Destiny node to link.

Example:

```
flow = qsoa.CircuitFlow()
startNode = flow.startNode()
initNode = flow.initNode(0)

linkNodes = flow.linkNodes(startNode, initNode)

print(flow.getFlowBody()['linkDataArray'])
```

```
[{'from': -1, 'to': -2, 'points': []}]
```


© QuantumPath® 2021 | All rights reserved

Summary: Object types

SolutionItem

Data	Type	Method
Solution ID	Integer	`getId()`
Solution Name	String	`getName()`
Quantum Type	String	`getQuantumType()`

DeviceItem

Data	Type	Method
Device Short Name	String	`getDeviceShortName()`
Device Provider	String	`getDeviceProvider()`
Quantum Machine Type	String	`getQuantumMachineType()`
Is Local Simulator	Boolean	`getIsLocalSimulator()`
Vendor ID	Integer	`getIdVendor()`
Vendor Name	String	`getVendorName()`
Description	String	`getDescription()`
Device Name	String	`getDeviceName()`

© QuantumPath® 2021 | All rights reserved

FlowItem

Data	Type	Method
Flow ID	Integer	`getId()`
Flow Name	String	`getName()`

Application

Data	Type	Method
Application Name	String	`getApplicationName()`
Solution ID	Integer	`getIdSolution()`
Flow ID	Integer	`getIdFlow()`
Device ID	Integer	`getIdDevice()`
Execution Token	String	`getExecutionToken()`

Execution

Data	Type	Method
Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Solution Name	String	`getSolutionName()`

© QuantumPath® 2021 | All rights reserved

Flow Name	String	`getFlowName()`
Device Name	String	`getDeviceName()`
Histogram	Dictionary	`getHistogram()`
Duration	Integer	`getDuration()`

`Asset`

Data	Type	Method
Asset ID	Integer	`getId()`
Asset Name	String	`getName()`
Asset Namespace	String	`getNamespace()`
Asset Description	String	`getDescription()`
Asset Body	String	`getBody()`
Asset Type	String	`getType()`
Asset Level	String	`getLevel()`
Asset Last Update	String	`getLastUpdate()`

© QuantumPath® 2021 | All rights reserved

AssetManagementData

Data	Type	Method
Asset Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`
Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

AssetManagementResult

Data	Type	Method
Exit Code	String	`getExitCode()`
Exit Message	String	`getExitMessage()`
Lifecycle Token	String	`getLifecycleToken()`
Solution ID	Integer	`getIdSolution()`

© QuantumPath® 2021 | All rights reserved

Asset ID	Integer	`getIdAsset()`
Asset Name	String	`getAssetName()`
Asset Namespace	String	`getAssetNamespace()`
Asset Type	String	`getAssetType()`
Asset Level	String	`getAssetLevel()`
Asset Compiled Status	Boolean	`getAssetCompiledStatus()`
Asset Transpiled Status	Boolean	`getAssetTranspiledStatus()`

`QuantumExecutionHistoryEntry`

Data	Type	Method
Result ID	Integer	`getIdResult()`
Solution ID	Integer	`getIdSolution()`
Solution Name	String	`getSolutionName()`
Flow ID	Integer	`getIdFlow()`
Flow Name	String	`getFlowName()`
Device ID	Integer	`getIdDevice()`
Device Name	String	`getDeviceName()`
Device Short Name	String	`getDeviceShortName()`

© QuantumPath® 2021 | All rights reserved

Device Vendor	String	`getDeviceVendor()`
Is Local Simulator	Boolean	`getIsLocalSimulator()`
Device Type Name	String	`getDeviceTypeName()`
Result Histogram	String	`getResultHistogram()`
Execution Date	String	`getExecutionDate()`
Duration in minutes	Float	`getDurationMinutes()`
Result Type	String	`getResultType()`
Result Description	String	`getResultDescription()`

`CircuitGates`

Function	Method
Get Circuit Body	`getCircuitBody()`
Get Number of Qubits	`getNumberOfQubits()`
Get Default Qubit State	`getDefaultQubitState()`
Get Qubit States	`getQubitStates()`
Set Default Qubit States	`getQubitStates()`
Initialize Qubit States	`initializeQubitStates()`
Add Hadamard gate	`h(position, add)`

© QuantumPath® 2021 | All rights reserved

Add Pauli X gate	`x(position, add)`
Add Swap gates	`swap(position1, position2, add)`
Add Pauli Y gate	`y(position, add)`
Add Pauli Z gate	`z(position, add)`
Add Control Hadamard gate	`ch(position1, position2, add)`
Add Control X gate	`cx(position1, position2, add)`
Add Toffoli gate	`ccx(position1, position2, position3, add)`
Add Square root of Z, S gate	`s(position, add)`
Add Adjoint square root Z gate	`i_s(position, add)`
Add Square root of X gate	`sx(position, add)`
Add Adjoint square root X gate	`i_sx(position, add)`
Add Square root of Y gate	`sy(position, add)`
Add Adjoint square root Y gate	`i_sy(position, add)`
Add Fourth root of Z, T gate	`t(position, add)`
Add Adjoint four root Z gate	`i_t(position, add)`
Add Four root of X gate	`tx(position, add)`
Add Adjoint four root X gate	`i_tx(position, add)`

© QuantumPath® 2021 | All rights reserved

Add Four root of Y gate	`ty(position, add)`
Add Adjoint four root Y gate	`i_ty(position, add)`
Add Phase gate	`p(position, argument, add)`
Add Rotation X gate	`rx(position, argument, add)`
Add Rotation Y gate	`ry(position, argument, add)`
Add Rotation Z gate	`rz(position, argument, add)`
Add Measure	`measure(position)`
Add Control to gate	`control(position, circuit)`
Add Created Gate	`addCreatedGate(gate)`
Add Multi Control Gate	`mcg(position, circuit)`
Add Barrier	`barrier(position)`

`CircuitAnnealing`

Function	Method
Get Circuit Body	`getCircuitBody()`
Add Parameter	`addParameter(parameter)`
Add Auxiliary Data	`addAuxData(auxData)`
Add Class	`addClass(cls)`

© QuantumPath® 2021 | All rights reserved

Add Variable	`addVariable(variable)`
Add Rule	`addRule(rule)`

`Parameter`

Function	Method
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Value	`getValue()`

`AuxData`

Function	Method
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Value	`getValue()`

`Class`

Function	Method
Add Property	`addProperty()`
Get Properties	`getProperties()`

© QuantumPath® 2021 | All rights reserved

Get UI ID	`getUiID()`
Get Name	`getName()`
Get Number of Variables	`getNumberOfVars()`
Get Description	`getDescription()`

`Variable`

Function	Method
Get Classes	`getClasses()`
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Description	`getDescription()`

`OffsetExp`

Function	Method
Get Expression Type	`getExpType()`
Get Offset	`getOffset()`

`LinearExp`

Function	Method

© QuantumPath® 2021 | All rights reserved

Get Expression Type	`getExpType()`
Get Term 1	`getTerm1()`
Get Coefficient	`getCoefficient()`

`QuadraticExp`

Function	Method
Get Expression Type	`getExpType()`
Get Term 1	`getTerm1()`
Get Term 2	`getTerm2()`
Get Coefficient	`getCoefficient()`

`SquaredExp`

Function	Method
Get Expression Type	`getExpType()`
Get Childs	`getChilds()`

`SummationExp`

Function	Method
Get Expression Type	`getExpType()`

© QuantumPath® 2021 | All rights reserved

Get From Value	`getFrom()`
Get To Value	`getTo()`
Get Iterator	`getIterator()`
Get Childs	`getChilds()`

`Rule`

Function	Method
Add Expression	`addExpression()`
Get Expressions	`getExpressions()`
Get UI ID	`getUiID()`
Get Name	`getName()`
Get Lambda	`getLambda()`
Get Description	`getDescription()`
Get Disabled	`getDisabled()`

© QuantumPath® 2021 | All rights reserved

CircuitFlow

Function	Method
Get Flow Body	`getFlowBody()`
Add Start node	`startNode()`
Add Init node	`initNode(startValue)`
Add Circuit node	`circuitNode(circuitName)`
Add Shots node	`shotsNode(numShots)`
Add End node	`endNode()`
Add Comment node	`commentNode(comment)`
Add Link nodes	`linkNodes(fromNode, toNode)`

© QuantumPath® 2021 | All rights reserved

Summary: Methods

`qsoa.getEnvironments()`

Show QuantumPath® available environments.

Prerequisites	None
Inputs	None
Output	Dictionary

`qsoa.getActiveEnvironment()`

Show active QuantumPath® environment.

Prerequisites	None
Inputs	None
Output	Tuple

`qsoa.setActiveEnvironment(environmentName, qSOATargetURL, validateCert)`

Set active QuantumPath® environment.

Prerequisites	Existing QuantumPath® environment
Inputs	`environmentName` → String `qSOATargetURL` → String (Optional) `validateCert` → Boolean (Optional)

© QuantumPath® 2021 | All rights reserved

Output	Tuple

`qsoa.echoping()`

Test to validate if the security service is enabled.

Prerequisites	None
Inputs	None
Output	Boolean

`qsoa.encodePassword(password)`

Encode password in Base64.

Prerequisites	None
Inputs	`password` → String
Output	String

`qsoa.encryptPassword(password)`

Encrypt password in SHA-256.

Prerequisites	None
Inputs	`password` → String
Output	String

© QuantumPath® 2021 | All rights reserved

`qsoa.authenticateBasic(username, password)`

Performs the user authentication process.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String
Output	Boolean

`qsoa.authenticateBasic()`

Performs the user authentication process.

Prerequisites	User created in QPath® `.qpath` created
Inputs	None
Output	Boolean

`qsoa.authenticate(username, password)`

Performs the user authentication process. With Base64 password.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String (Base64)

© QuantumPath® 2021 | All rights reserved

Output	Boolean

`qsoa.authenticate()`

Performs the user authentication process. With Base64 password.

Prerequisites	User created in QPath® `.qpath` created
Inputs	None
Output	Boolean

`qsoa.authenticateEx(username, password)`

Performs the user authentication process. With SHA-256 password.

Prerequisites	User created in QPath®
Inputs	`username` → String `password` → String (SHA-256)
Output	Boolean

`qsoa.authenticateEx()`

Performs the user authentication process. With SHA-256 password.

Prerequisites	User created in QPath® `.qpath` created

© QuantumPath® 2021 | All rights reserved

Inputs	None
Output	Boolean

`qsoa.echostatus()`

Check if user session is active.

Prerequisites	None
Inputs	None
Output	Boolean

`qsoa.echouser()`

Check user login status.

Prerequisites	User already authenticated
Inputs	None
Output	String

`qsoa.getVersion()`

Check the ConnectionPoint service version.

Prerequisites	User already authenticated
Inputs	None

© QuantumPath® 2021 | All rights reserved

Output	String

`qsoa.getLicenceInfo()`

Returns QuantumPath account licence.

Prerequisites	User already authenticated
Inputs	None
Output	Dictionary

`qsoa.getQuantumSolutionList()`

Show the list of solutions available to the user along with their IDs.

Prerequisites	User already authenticated Solution created
Inputs	None
Output	Dictionary

`qsoa.getQuantumSolutions()`

Get the solutions available from the user as an object.

Prerequisites	User already authenticated
Inputs	None

© QuantumPath® 2021 | All rights reserved

Output	List of `SolutionItem` objects

`qsoa.getQuantumSolutionName(idSolution)`

Get the name of a solution.

Prerequisites	User already authenticated
Inputs	`idSolution` → Integer
Output	String

`qsoa.getQuantumDeviceList(idSolution)`

Show the list of devices available in a solution along with their IDs.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	Dictionary

`qsoa.getQuantumDevices(idSolution)`

Get the available devices in a solution as an object.

Prerequisites	User already authenticated Solution created

© QuantumPath® 2021 | All rights reserved

Inputs	`idSolution` → Integer
Output	List of `DeviceItem` objects

`qsoa.getQuantumDeviceName(idSolution, idDevice)`

Get the name of a device.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `idDevice` → Integer
Output	String

`qsoa.getQuantumFlowList(idSolution)`

Show the list of flows available in a solution along with their IDs.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	Dictionary

© QuantumPath® 2021 | All rights reserved

`qsoa.getQuantumFlows(idSolution)`

Get the flows available in a solution as an object.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer
Output	List of `FlowItem` objects

`qsoa.getQuantumFlowName(idSolution, idFlow)`

Get the name of a flow.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `idFlow` → Integer
Output	String

`qsoa.runQuantumApplication(applicationName, idSolution, idFlow, idDevice)`

Run a created quantum solution.

Prerequisites	User already authenticated Solution created
Inputs	`applicationName` → String

© QuantumPath® 2021 | All rights reserved

	`idSolution` → Integer `idFlow` → Integer `idDevice` → Integer
Output	`Application` object

`qsoa.getQuantumExecutionResponse(application)`

Get the response of a quantum solution execution.

Prerequisites	User already authenticated Solution run `Application` object generated
Inputs	`application` → `Application` object
Output	`Execution` Object

`qsoa.getQuantumExecutionResponse(executionToken, idSolution, idFlow)`

Get the response of a quantum solution execution.

Prerequisites	User already authenticated Solution run
Inputs	`executionToken` → String `idSolution` → Integer `idFlow` → Integer

© QuantumPath® 2021 | All rights reserved

Output	`Execution` object

`qsoa.runQuantumApplicationSync(applicationName, idSolution, idFlow, idDevice)`

Run a created quantum solution synchronous.

Prerequisites	User already authenticated Solution created
Inputs	`applicationName` → String `idSolution` → Integer `idFlow` → Integer `idDevice` → Integer
Output	`Application` object

`qsoa.representResults(execution, resultIndex)`

Results visual representation.

Prerequisites	User already authenticated Execution completed
Inputs	`execution` → `Execution` object `resultIndex` → Integer (Optional)
Output	png image or a table String

© QuantumPath® 2021 | All rights reserved

`qsoa.getAssetCatalog(idSolution, assetType, assetLevel)`

Get asset information from a solution.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetType` → String ('CIRCUIT' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	List of `Asset` objects

`qsoa.getAsset(idAsset, assetType, assetLevel)`

Get specific asset information.

Prerequisites	User already authenticated Asset created
Inputs	`idAsset` → Integer `assetType` → String ('CIRCUIT' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`Asset` object

© QuantumPath® 2021 | All rights reserved

qsoa.createAsset(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)

Create Asset.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String `assetBody` → String / `CircuitGates` object / `CircuitFlow` object `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`AssetManagementData` object

qsoa.createAssetFlow(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetLevel, publish)

Create Flow Asset shorter method.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String

© QuantumPath® 2021 | All rights reserved

	`assetDescription` → String `assetBody` → String / `CircuitFlow` object `assetLevel` → String ('VL' or 'IL') `publish` → Boolean
Output	`AssetManagementData` object

`qsoa.publishFlow(idAsset, publish)`

Change publish flow status.

Prerequisites	User already authenticated Access permission to the flow
Inputs	`idAsset` → Integer `publish` → Boolean
Output	Boolean

`qsoa.updateAsset(asset, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Update Asset values.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object `assetName` → String (Optional) `assetNamespace` → String (Optional)

© QuantumPath® 2021 | All rights reserved

	`assetDescription` → String (Optional) `assetBody` → String (Optional) / `CircuitGates` object (Optional) / `CircuitFlow` object (Optional) `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') (Optional) `assetLevel` → String ('VL' or 'IL') (Optional)
Output	`AssetManagementData` object

`qsoa.createAssetSync(idSolution, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Create Asset and get result.

Prerequisites	User already authenticated Solution created
Inputs	`idSolution` → Integer `assetName` → String `assetNamespace` → String `assetDescription` → String `assetBody` → String / `CircuitGates` object / `CircuitFlow` object `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') `assetLevel` → String ('VL' or 'IL')
Output	`AssetManagementResult` object

© QuantumPath® 2021 | All rights reserved

`qsoa.updateAssetSync(asset, assetName, assetNamespace, assetDescription, assetBody, assetType, assetLevel)`

Update Asset values and get result.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object `assetName` → String (Optional) `assetNamespace` → String (Optional) `assetDescription` → String (Optional) `assetBody` → String (Optional) / `CircuitGates` object (Optional) / `CircuitFlow` object (Optional) `assetType` → String ('GATES', 'ANNEAL' or 'FLOW') (Optional) `assetLevel` → String ('VL' or 'IL') (Optional)
Output	`AssetManagementResult` object

`qsoa.deleteAsset(asset)`

Delete asset.

Prerequisites	User already authenticated Asset created
Inputs	`asset` → `Asset` object
Output	Boolean

© QuantumPath® 2021 | All rights reserved

`qsoa.deleteAsset(idAsset, assetType)`

Delete asset.

Prerequisites	User already authenticated Asset created
Inputs	`idAsset` → Integer `assetType` → String ('CIRCUIT' or 'FLOW')
Output	Boolean

`qsoa.getAssetManagementResult(lifecycleToken)`

Get Asset Management Result from a lifecycle token.

Prerequisites	Existing asset lifecycle token
Inputs	`lifecycleToken` → String
Output	`AssetManagementResult` object

`qsoa.getQuantumExecutionHistoric(idSolution, idFlow, idDevice, dateFrom, isSimulator, top, resultType)`

Get a list of quantum execution history entries.

Prerequisites	User already authenticated
Inputs	`idSolution` → Integer (Optional) `idFlow` → Integer (Optional) `idDevice` → Integer (Optional)

© QuantumPath® 2021 | All rights reserved

	`dateFrom` → String (yyyy-mm-ddThh:mm:ss) (Optional) `isSimulator` → Boolean (Optional) `top` → Integer (Optional) `resultType` → Boolean (Optional)
Output	List of `QuantumExecutionHistoryEntry` objects

`qsoa.getQuantumExecutionHistoricResult(idResult)`

Get a quantum execution history entry.

Prerequisites	User already authenticated
Inputs	`idResult` → Integer
Output	`QuantumExecutionHistoryEntry` object

`circuit.addParameter(parameter)`

Add Circuit Parameter.

Prerequisites	Created circuit
Inputs	`parameter` → `Parameter` object / `Parameter` object list
Output	`Parameter` object / `Parameter` object list

© QuantumPath® 2021 | All rights reserved

`circuit.addAuxData(auxData)`

Add Circuit Auxiliary Data.

Prerequisites	Created circuit
Inputs	`auxData` → `AuxData` object / `AuxData` object list
Output	`AuxData` object / `AuxData` object list

`cls.addProperty(name, value)`

Add Class Property.

Prerequisites	Created class
Inputs	`name` → String `value` → Integer / Integer list
Output	`AuxData` object / `AuxData` object list

`circuit.addClass(cls)`

Add Circuit Class.

Prerequisites	Created circuit
Inputs	`cls` → `Class` object / `Class` object list
Output	`Class` object / `Class` object list

© QuantumPath® 2021 | All rights reserved

`circuit.addVariable(variable)`

Add Circuit Variable.

Prerequisites	Created circuit
Inputs	`variable` → `Variable` object / `Variable` object list
Output	`Variable` object / `Variable` object list

`rule.addExpression(expression)`

Add Rule Expression.

Prerequisites	Created rule
Inputs	`expression` → String
Output	`SummationExp` object / `SquaredExp` object / `LinearExp` object / `QuadraticExp` object / `OffsetExp` object / list

`circuit.addRule(rule)`

Add Circuit Rule.

Prerequisites	Created circuit
Inputs	`rule` → `Rule` object / `Rule` object list

© QuantumPath® 2021 | All rights reserved

Output	`Rule` object / `Rule` object list

© QuantumPath® 2021 | All rights reserved

Summary: Custom Exceptions

APIConnectionError

Description:

Raised when some error occurs during API connection.

Reason:

Response content describe the connection error code of the http request, reason, and message body from the server.

Solution:

Check the error message to further information about how to solve the specific API connection problem.

AuthenticationError

Description:

Raised when user is not authenticated.

Reason:

Session could be end up by inactivity, or not valid credentials.

Solution:

Authenticate again.

Base64Error

Description:

Raised when some error occurs during Base64 encoding.

Reason:

Some authentication methods precises to introduce by user their password encoded in Base64. If it does not follow that format, an exception will be raised.

© QuantumPath® 2021 | All rights reserved

Solution:

Use a correct Base64 encoding for password.

ConfigFileError

Description:

Raised when some error occurs during reading the configuration file.

Reason:

Mostly produced by a bad configuration file format in the username/password sections.

Solution:

Check the username and password format in the configuration file. Also ensure the section names follow the proper way.

ExecutionObjectError

Description:

Raised when some error occurs reading an Execution object.

Reason:

Produced while trying to read an inexistent property of an Execution object.

Solution:

Read the Execution object properties and make sure that the status code is “OK”.

© QuantumPath® 2021 | All rights reserved