qSim How-To User Guide – v1.1

©G. Casonato – Mar 2023

Table of Contents

1 Introduction

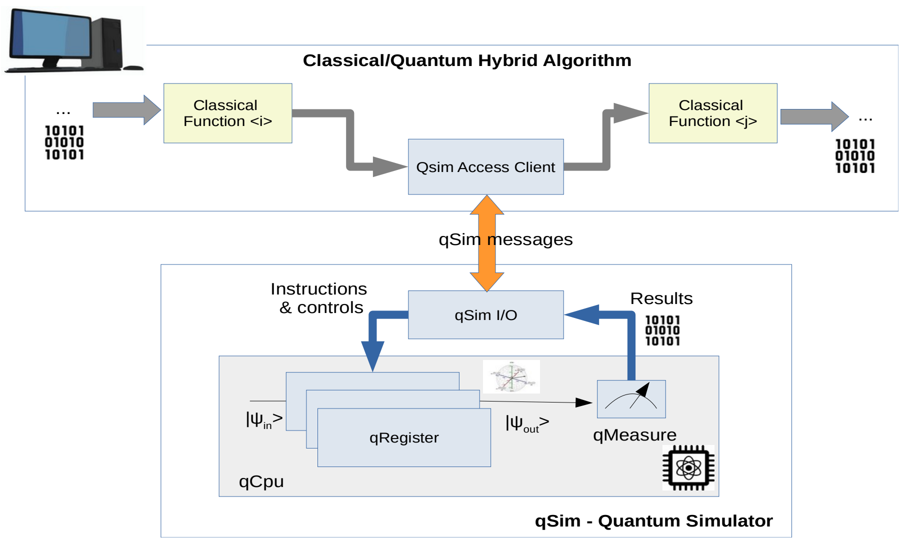
1.1 Purpose and Scope

The purpose of this note is to describe how to use the Quantum Simulator (qSim) from an external client for running QC algorithms. It focuses on the qSim external interfaces and the relevant data units transferred back and forth.

The scope is limited to external interfaces only and qSim internal details are not described here.

1.2 qSim Overview

The Quantum Simulator (qSim) is simulator able to provide an high performance universal QC machine, running on CPU or GPU based hardware.



1.2.1 Main functionalities

The functionalities provided by qSim are:

- qubit registers (quregs) handling
- universal 1-qubit and 2-qubits quantum gate set (e.g. Pauli, Hadamard, identity, phase shift, CX, CY, CZ, etc.)
- high level blocks for common n-qubits transformations (e.g. swap and controlled swap) and QML extensions (feature maps and QVC networks)
- quregs measurement

In addition, some diagnostic & test helper functionalities are also supported:

- arbitrary qureg state setup
- qureg state access (max 10 qubits qureg)

1.2.2 Constraints and Limitations

The qSim limitations are the following:

- max qubit size for transformations: 30 qubits
- max qubit size for arbitrary state setup or peek: 10 qubits

1.2.3 Dependencies

The qSim requires CUDA libraries installed in the target platform. The following version has been used for the simulator development ant tests.

CUDA Toolkit	Device CUDA Driver	Device CUDA Capability
9.2	11.4	6.1

1.3 qSim Interfaces

The qSim supports the following client access interfaces:

- TCP/IP Socket, for direct access to qSim

Their description is provided in section 2.3 .

2 Use & Access How-To

2.1 Build

The qSim executable can be built in two ways:

1. from Eclipse importing the repository
2. manually, using the Makefile provided in the build_make folder

To use "make" or "make gpu" for GPU target building, or "make cpu" for CPU one.

In both cases the CUDA development toolkit for the target hardware has to be available and configured in the include and library build variables using the actual paths and CUDA capability, namely:

- NVCC := nvcc
- CUDA_ARCH := -gencode arch=compute_61,code=sm_61
- CUDA_INC :=/usr/local/cuda-9.2/include
- CUDA_LIB := /usr/local/cuda-9.2/lib64

2.2 Launch

The qSim executable can be launched with optional command line arguments which allows specific functionalities setup, when needed. More specifically:

- silent mode (default)

  ```
  ./qSim_gpu
  ```

- diagnostic mode

  ```
  ./qSim_gpu -verbose
  ```

- TCP/IP port configuration

  ```
  ./qSim_gpu -port=27050
  ```

2.3 Interfaces

qSim provides the following interfaces to external clients for accessing and use the QC functionalities provided:

- TCP/IP socket based access

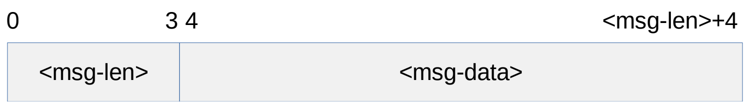
For best performance it is recommended to set the client TCP/IP socket protocol parameter NODELAY.

2.3.1 TCP/IP Socket Interface

The default connection parameters for this interface are specified in the following table.

Parameter	Value
IP address	127.0.0.1 (localhost)
Port	27020 (default)
Connection initiator	<client>
Stream type	ASCII

The messages are in ASCII format and their syntax is the following:



The actual content of the message-data part depends on the specific message, as indicated in the next sections. The general format is ASCII, and using "|" as field separators, namely

<encoded_instruction> = <counter>"|"<id>"|"<params>

with ":" as param tag-value pair separator, i.e.

<params> = <tag_1>=<value_1>:<tag_2>=<value_2>: ...<tag_n>=<value_n>:

2.3.2 General Conventions

The data types supported in qSim messages are the following:

- integer values: encoded as string value
- double values: encoded in decimal format with decimal digits as per required precision
- index range values: encoded as string value pair in parenthesis “(<index$_1$>, <index$_2$>)”
- complex state values: encoded in decimal pair format for real and imaginary parts, with decimal digits as per required precision “(<real_val>, <imag_val>)”
- complex state value array: encoded as comma separated sequence of complex state values “(<real_val$_1$>, <imag_val$_1$>), …, (<real_val$_n$>, <imag_val$_n$>)”

The convention for qubit ordering in a qureg is similar to the one used for bits into bytes, namely

- least significant qubit (LSQ): the leftmost qubit in the qureg tensor product
- most significant qubit (MSQ): the rightmost qubit in the qureg tensor product

2.4 Datastream Details

2.4.1 TCP/IP Socket Datastreams

This case allows a client to exchange socket messages with qSim, through the following datastreams:

- client => qSim
  - qSim-connection
  - qSim-qureg-control
  - qSim-qreg-transformation
- qSim => client
  - qSim-response

The client <==> qSim handshake is pretty simple and it foresee a synchronous exchange of messages from client to qSim and back.

2.4.1.1 qSim-connection Datastream

This datastream allows a client to establish and manage the connection with qSim. The messages part of the datastream are:

- Client Register
- Client Unregister

Message details are in 2.4.2 .

2.4.1.2 qSim-qureg-control Datastream

This datastream allows a client to setup and manage qureg within qSim. The messages part of the datastream are:

- Qureg Allocate
- Qureg Release
- Qureg State Reset
- Qureg State Set (including arbitrary state preparation - diagnostic)
- Qureg State Measurement
- Qureg State Expectation (diagnostic)
- Qureg State Peek (diagnostic)

Message details are in 2.4.2 .

2.4.1.3 qSim-qureg-transformation Datastream

This datastream allows a client to perform qureg transformations with qSim. The messages part of the datastream are:

- Qureg State Transform

Message details are in 2.4.2 .

2.4.1.4 qSim-response Datastream

This datastream allows a client to receive responses for all the messages sent to qSim. The messages part of the datastream are:

- Message Response

Message details are in 2.4.2 .

2.4.2 TCP/IP Socket Messages and Parameters

The description of the qSim messages is provided in this section. Each sub-para includes the client => qSim message and the relevant response.

2.4.2.1 Client Register

This message allow a client to register to qSim for being allowed to operate on quregs. It receives a token to be provided in all qreg related messages. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	1

Parameter	Tag	Type	Value
Client ID	id	string	<client mnemonic id>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok
Client token	token	string	<client registration token>

2.4.2.2 Client Unregister

This message allow a client to deregister to qSim and to stop the connection in a clean way. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	2

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok

2.4.2.3 Qureg Allocate

This message allow a client to allocate a qureg in qSim, and to receive an handler to be used to reference the qureg for further operations. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	10

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg size	qr_n	int	<qureg size in qubits>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok
Qureg handler	qr_h	int	<qureg handler>

2.4.2.4 Qureg Release

This message allow a client to deallocate a qureg in qSim, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	11

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok

2.4.2.5 Qureg State Reset

This message allow a client to reset the state of a qureg in qSim, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	12

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value

Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok

2.4.2.6 Qureg State Set

This message allow a client to set an excited state or an arbitrary state (optional – diagnostic case) in a qureg in qSim, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	13

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>
Qureg state values	qr_stVal	Complex array	<qureg state values> - optional for arbitrary state setup only

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok

2.4.2.7 Qureg State Transformation

This message allow a client to apply a transformation function to a qureg states in qSim, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	14

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>

Function type	f_type	int	<function type> from following list *1-qubit* • Q1_I = 0 • Q1_H = 1 • Q1_X = 2 • Q1_Y = 3 • Q1_Z = 4 • Q1_SX = 5 • Q1_PS = 6 • Q1_T = 7 • Q1_S = 8 • Q1_Rx = 9 • Q1_Ry = 10 • Q1_Rz = 11 *2 qubits* • Q2_CU = 12 • Q2_CX = 13 • Q2_CY = 14 • Q2_CZ = 15 *n qubits* • Qn_MCSLRU = 16 *blocks* • Q1_SWAP = 100 • Qn_SWAP = 101 • Q1_CSWAP = 102 • Qn_CSWAP = 103 *QML blocks* • Qn_FMAP = 200 • Qn-QNET = 201
Function size	f_size	int	<function size in states> n.a. for QML blocks
Function repetitions	f_rep	int	<function repetitions in the qureg> n.a. for QML blocks
Function LSQ	f_lqs	int	<function LSQ index in the qureg> n.a. for QML blocks
Function control range	f_cRange	Int (start, stop) interval	<function control start-stop indexes> only for n-qubits case n.a. for QML blocks
Function target range	f_tRange	Int (start, stop) interval	<function target start-stop indexes> only for n-qubit case n.a. for QML blocks
Controlled U-function type	f_uType	int	<controlled U-function type>, only for n-qubit case n.a. for QML blocks
Function arguments	f_args	Double list	<function or controlled U-function

			arguments> *1-qubit function* • arg #0: phase value (PS, Rx, Ry, Rz cases) *2-qubit controlled U-function* • arg #0: controlled U-function control range • arg #1: controlled U-function target range • arg #2: phase value (PS, Rx, Ry, Rz cases) n.a. for QML blocks

QML block repetitions	fqml_rep	int	<feature map or QVC network layout repetitions>
QML layout entanglement type	fqml_entang_type	int	<feature map or QVC network layout entanglement type> from following list • linear = 0 • circular = 1
QML block subtype	fqml_subtype	Int	<feature map or QVC network subtype> from following list *Feature Map* • Pauli Z= 0 • Pauli ZZ = 1 *QVC network* • Real Amplitudes = 0
QML feature map vector	fqml_fmap_fvec	Double list	<feature map classical values>

qSim => Client response

Message ID	20

Parameter	**Tag**	**Type**	**Value**
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok

2.4.2.8 Qureg State Peek

This message allow a client to peek the state values of a qureg in qSim, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	15

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok
Qureg state values	qr_stVal	Complex array	<qureg state values>

Note: the usage of this message is constrained by the size of the qureg to inspect, with a limit of 10 qubits size applied.

2.4.2.9 Qureg State Measurement

This message allow a client to measure fully or partially the state a qureg in qSim, based on given handler, and collapsing the state due to the measure. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	16

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>
Measure qubit start index	qr_mQidx	int	<(sub)qureg start index> (-1 for whole qureg)
Measure qubit length	qr_mQlen	int	<(sub)qureg length in qubits>
Measure random flag	qr_mRand	int	<random vs. expectation measure flag >
Measure state collapse flag	qr_mStColl	int	<real-state collapse vs. simulated measure flag>

qSim => Client response

Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok
Measure state index	m_stIdx	int	<measure state index>
Measure state probability	m_stPr	double	<measure state probability value>
Measure state collapse indexes	qr_mStColl	Int list	<list of collapsed qureg state indexes>

2.4.2.10 Qureg State Expectation

This message allow a client to calculate the expectation value for all qureg states or a single state, and for a whole qureg or a sub-qureg, based on given handler. The relevant arguments and parameters are the following.

Client => qSim message

Message ID	17

Parameter	Tag	Type	Value
Client token	token	string	<client registration token>
Qureg handler	qr_h	int	<qureg handler>
Expectation state index	qr_exStIdx	int	<qureg state index> (-1 for all states)
Measure qubit start index	qr_exQidx	nt	<(sub)qureg start index> (-1 for whole qureg)
Measure qubit length	qr_exQlen	int	<(sub)qureg length in qubits>
Measure random flag	qr_exObsOp	int	<observation operator type> from following list • Computational-Basis = 0 • Pauli-Z-Basis = 1

qSim => Client response

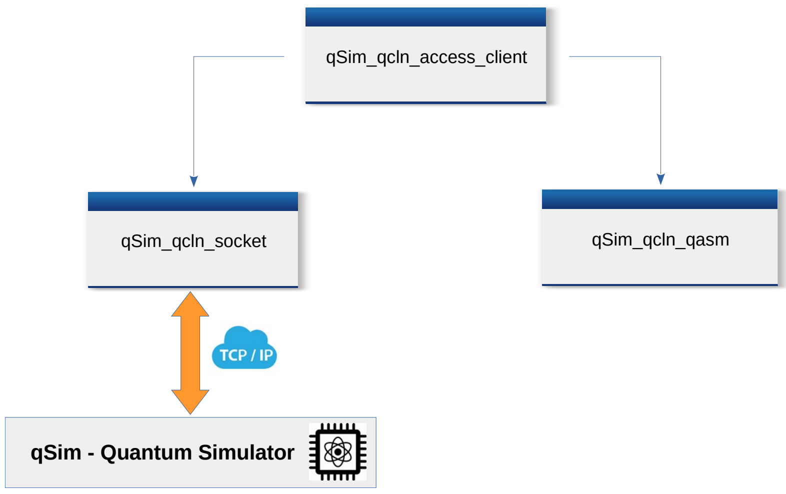
Message ID	20

Parameter	Tag	Type	Value
Result	result	string	<message exec result (Ok/Not-Ok)>
Error	error	string	<error details> only for result Not-Ok
Measure state index	m_exStVal	double	<state expectation value>

2.5 Python Access Client

2.5.1 Access Classes

The qSim package provides also an access client Python module, to make easier to connect and use qSim from Python applications.



The classes included are:

- qSim_qcln_access_client: entry point class for Python applications, providing a class able to connect to the qSim at given address and port, and to perform all supported access requests as described in section 2 .
- qSim_qcln_qasm: helper class performing encoding/decoding of TCP/IP raw messages from/to high level parameters.
- qSim_qcln_socket: helper class wrapping the TCP/IP connection handling with qSim server.

The module is written in Python 3 and it doesn't use any third party library (built-in socket package only).

2.5.2 Example

Here below some basic examples for using the qClient access module with qSim, simply by importing the entry point class, namely

```
# import client access module
import qSim_qcln_access_client as qacc
```

Connection test

```
# instantiate class and connect
qcln = qacc.qSim_qcln_access_client(verbose=True)
qcln.connect()
print('=> isConnected:', qcln.isConnected())
print()

...

# disconnect
qcln.disconnect()
print()
```

Qureg handling test

```
# allocate a test qureg and peek states
qr_h = qcln.qreg_allocate(3)
qr_st = qcln.qreg_state_getValues(qr_h)
print('=> qr-states:', qr_st)
print()

...

# release qureg
qcln.qreg_release(qr_h)
print('=> qr-release')
print()
```

Qureg transformation test

```
# apply a transformation
f_type = qasm.QASM_F_TYPE_H
f_size = 2
f_rep = 1
f_lsq = 1
res = qcln.qreg_state_transform(qr_h, f_type, f_size, f_rep, f_lsq)
print('=> qr-transformation res:', res)
print()
```