QuICScript

Release 2.0.0

pQCee.com

Dec 20, 2024

TABLE OF CONTENTS

CHAPTER

ONE

INTRODUCTION

The QuIC in QuICScript stands for Quantum Interpreted Circuits. QuICScript is a software-based quantum emulator, which can support quantum computations of up to 20-qubits and scale to gate depths beyond 100,000 gates. Quantum programs written in QuICScript can be executed in real-time via a QuICScript Engine.

This is the user manual for QuICScript.

You can get to QuICScript here or visit pQCee for more information.

CHAPTER

TWO

GETTING STARTED WITH QUICSCRIPT (QUANTUM-IN-A-BROWSER)

This section covers all you need to know for QuICScript before you start using QuICScript.

2.1 Before you start

QuICScript runs in a browser. To access QuICScript, you simply need a browser and a Wifi connection. There is no pre-installation required and circuit execution results are immediate, no compilation is required… it's that simple!

If you are new to quantum, take our Introduction to Quantum Gates and QuICScript course. *Note: you can also access this course via the quantumnft homepage.*

If you are a physicist and are already familiar with quantum gates, then skip to the Using QuICScript section of this manual to get a quick understanding of the syntax and the gates supported. Or you can refer to our QuICScript cheat sheet at QuICScript Cheat Sheet developed internally by Senior Cryptographic Engineer, Nicholas Ho.

2.2 Getting to QuICScript

When you are ready to start building and testing your quantum circuits, you can access QuICScript here.

CHAPTER

THREE

USING QUICSCRIPT

3.1 Overview

This section is designed to get you up and running with QuICScript quickly. It covers the following sections:

- **Building Circuits**: the quantum gates supported in QuICScript.
- **Circuit Syntax**: how to create a quantum circuit in QuICScript.
- **Results Display**: Outputs are shown in QuICScript engine section the Results section and can be formatted in decimal or binary.
- **Commands Guide**: Detailed explanations of fundamental, universal, controlled gates, and measurement operations.
- **API Reference**: Detailed function documentation to interact programmatically with QuICScript.

3.2 QuICScript cheat sheet

For a summary of the QuICScript commands for the current version of QuICScript, read the QuICScript Cheat Sheet.

3.3 Building a circuit in QuICScript

When building your circuit in QuICScript, supported gates include: `X`, `Y`, `Z`, `H`, `CN`, `P`, `T`, `I`, `m`, `J`, and `U`.

3.3.1 Circuit syntax

The language used to write the quantum program is "QuIC" (Quantum Interpreted Circuits) where the BNF notation of QuIC quantum program is as follows:

```
::= | ::= | ::= H | C | N | X | Y | Z | P | p | T | t | I | m ::= , | .
```

The use of QuICScript allows for efficient description and execution of the quantum program which is important since we want to run the quantum program interactively.

The gate operations supported by QuIC are as follows:

- Hadamard (H)
- Control-Not (CN)
- Pauli-X (X)
- Pauli-Y (Y)
- Pauli-Z (Z)

- pi/4 and pi/8 Phase-shift (P, T)
- Conjugate pi/4 and pi/8 Phase-shift (p, t)
- Identity (I)
- measurement (m)

Note: gates are separated by a comma delimiter and a period signifies the end of a circuit.

3.4 Results

The results of the circuit is displayed on the **Results** section on the right hand side of the page.

Note that the results are displayed in decimal. For displaying of results in binary, simply add a `:` at the end of your circuit.

Example result for `HX:`

```
01,50.00%;
11,50.00%;
```

Note

The percentage value represents the probability of the state. $|01\rangle$ has a 50% probability, and $|11\rangle$ has a 50% probability.

3.5 QuICScript Commands

The QuICScript Engine supports the execution of many discrete quantum gates. The commands are listed in the left column, and the associated behaviour is listed in the right column.

Types of Gates:

- *Fundamental Gates*
- *Universal Gates*
- *Controlled Gates*
- *Measurements*
- *Circuit Modifiers*

3.5.1 Fundamental Gates

QuICScript Command	Description
`I`	Identity Gate
`X`	Pauli-X Gate
`Y`	Pauli-Y Gate
`Z`	Pauli-Z Gate / Phase shift by $\varphi = \pi$
`H`	Hadamard Gate
`P`	Phase (S, P) Gate / Phase shift by $\varphi = \frac{\pi}{2}$
`p`	Inverse Phase (S, P) Gate
`T`	Phase (T) Gate / Phase shift by $\varphi = \frac{\pi}{4}$
`t`	Inverse Phase (T) Gate
`s`	Swap Gate

3.5.2 Universal Gates

QuICScript Command	Description
`U`	U Gate. Enables arbitrary rotation of a single qubit. Parameters are the three Euler rotation angles, θ, φ and λ.

The matrix for the U gate is defined as:

$$U(\theta, \varphi, \lambda) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\varphi}\sin(\frac{\theta}{2}) & e^{i(\varphi+\lambda)}\cos(\frac{\theta}{2}) \end{pmatrix}$$

If you wish to implement the rotation operators about the x-axis, y-axis, they can be defined via the U gate as follows:

$$R_x(\theta) = U(\theta, -\frac{\pi}{2}, \frac{\pi}{2})$$

$$R_y(\theta) = U(\theta, 0, 0)$$

To implement the phase shift via the U gate, it is defined as:

$$P(\varphi) = U(0, \varphi, 0)$$

3.5.3 Controlled Gates

QuICScript Command	Description
`CN`	Controlled Not Gate (CNOT)
`CY`	Controlled Y Gate
`CZ`	Controlled Z Gate (CZ)
`CP`	Controlled Phase (S, P) Gate
`CT`	Controlled Phase (T) Gate
`Cs`	Fredkin Gate (CSWAP)
`CU`	Controlled U Gate
`CCN`	Toffoli Gate (CCNOT)
`C...CN`	n-bit Toffoli Gate

Similar to the **Toffoli gate**, QuICScript also supports controlled-controlled-gate syntax for the above listed controlled gates. E.g., `CCY`, `CYC`, `ZCC`, etc. Note that QuICScript Engine supports any permutation of the QuICScript commands describing the controlled gates.

Similar to the n-bit **Toffoli gate**, QuICScript also supports n-bit controlled-gate syntax for the above listed controlled gates. E.g., `CCCY`, `CYCC`, `ZCCCC`, etc.

3.5.4 Measurements

QuICScript Command	Description
`m`	Measurement operation that resolves at random, per execution, according to the quantum state probabilities to either $\|0\rangle$ or $\|1\rangle$

3.5.5 Circuit Modifiers

QuICScript Command	Description
`,`	One QuICScript command is assigned to each qubit in a top-to-bottom vertical sequence. The comma is a delimiter to indicate that all available qubits have been each assigned a command; the next set of command-to-qubit assignments is expected after the comma.
`d`	Delete a qubit from the output state (system wavefunction). The probabilities of the remaining qubits will be normalized in the QuICScript Engine output.

3.6 API Reference

void **QuICScript_clearbuf**()

Clears any internal buffers used by the QuICScript engine. This function should be called to reset the state between different QuICScript circuit executions.

char ***QuICScript_setState**(int numQubits, unsigned long Value, double Count)

Sets the initial quantum state for a quantum circuit.

Parameters

- **numQubits** (`int`) – The number of qubits used in the quantum circuit.
- **Value** (`unsigned long`) – The integer value used to set the initial quantum state.
- **Count** (`double`) – The probability count used to set the initial quantum state.

Returns

A string (outString1) that represents the initial quantum state.

char ***QuICScript_begin**(int numQubits)

Initializes the QuICScript engine for a quantum circuit with a specific number of qubits.

Parameters

- **numQubits** (`int`) – The number of qubits used in the quantum circuit.

Returns

integer value that represents the success of the initialization.

Example Usage:

```
// Initializes a circuit for the first time
var inited = 0;
if(inited == 0) {
    Module._QuICScript_begin(numQubits);
    inited = numQubits;
    message = "State is reset, working on " + numQubits + " Qubits\n";
}
```

char ***QuICScript_cont**(int numQubits, char *myQuICScript, double X1_real, double X1_imag, double Y1_real, double Y1_imag, double X2_real, double X2_imag, double Y2_real, double Y2_imag)

Parameters

- **numQubits** (int) – The number of qubits used in the quantum circuit.
- **myQuICScript** (char*) – The string that represents the quantum circuit in QuICScript.
- **X1_real** (double) – The real part of the first X gate parameter.
- **X1_imag** (double) – The imaginary part of the first X gate parameter.
- **Y1_real** (double) – The real part of the first Y gate parameter.
- **Y1_imag** (double) – The imaginary part of the first Y gate parameter.
- **X2_real** (double) – The real part of the second X gate parameter.
- **X2_imag** (double) – The imaginary part of the second X gate parameter.
- **Y2_real** (double) – The real part of the second Y gate parameter.
- **Y2_imag** (double) – The imaginary part of the second Y gate parameter.

Returns

A string (outString1) that represents the output of the quantum circuit.

Example Usage:

```
// Executes the Bell state circuit
resultstate = Module.ccall("QuICScript_cont", "string", ["number", "string", "number
↪", "number", "number", "number", "number", "number", "number", "number"], [2, "HI,
↪CN", 2,0,0,0,0,0,1,0]);
```

void **QuICScript_end**()

Ends the QuICScript engine for a quantum circuit.

Example Usage:

```
// Resets the state
Module._QuICScript_end();
inited = 0;
message = "State is cleared.";
document.getElementById('quicDisplay').innerHTML = "<textarea readonly >" + message␣
↪+ "</textarea>";
```

char ***QuICScript_Qibo**(int numQubits, char *myQuICScript, double theta, double phi, double lamda)

Translates the QuICScript string into Qibo code.

Parameters

- **numQubits** (int) – The number of qubits used in the quantum circuit.

- **myQuICScript** (char*) – A string that represents the QuICScript, detailing the quantum circuit's operations.
- **theta** (double) – The theta parameter, typically used in rotational quantum gates.
- **phi** (double) – The phi parameter, typically used in rotational quantum gates.
- **lamda** (double) – The lambda parameter, typically used in rotational quantum gates.

Returns

Qibo code that represents the quantum circuit.

Example Usage:

```
// Generates Qibo code for the Bell State circuit
resultstate = Module.ccall("QuICScript_Qibo", "string", ["number", "string", "number
↪", "number", "number"], [2, "HI,CN", 0,0,0]);
```

CHAPTER

FOUR

USING CIRCUIT BUILDER

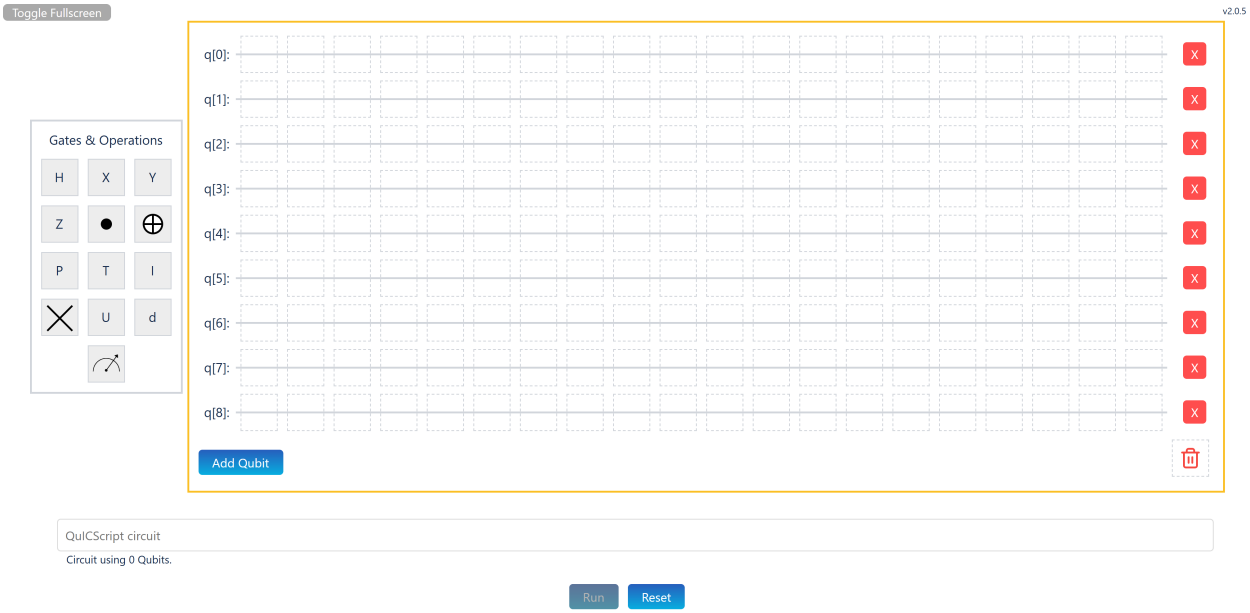


Fig. 1: QuICScript Circuit Builder (Default Configration)

4.1 Folder structure

The QuICScript Circuit Builder is built using HTML, Javascript, and CSS.

The core files are as follows:

```
├──index.html - Modify this file to change the default configuration
└──assets/
   ├──index.js
   ├──index.css
   ├──QuICScript.js
   └──QuICScript.wasm
```

1. `index.html` - Web server to point to `index.html`. For customization, see following sections for more information.
2. `index.js` - Logic for Circuit Builder
3. `index.css` - Styling for Circuit Builder

4. `QuICScript.js` - QuICScript Engine WASM library
5. `QuICScript.wasm` - QuICScript Engine WebAssembly file

Warning

Do not modify the `js` and `css` files directly as it may break the application. They are bundled assets through the build process.

4.2 Circuit Builder Configuration

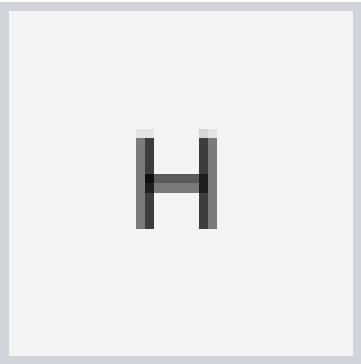
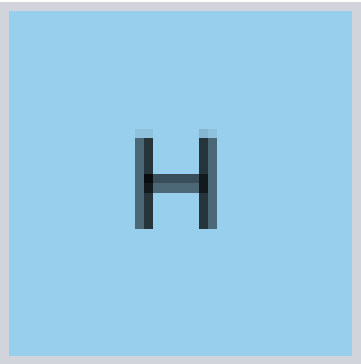
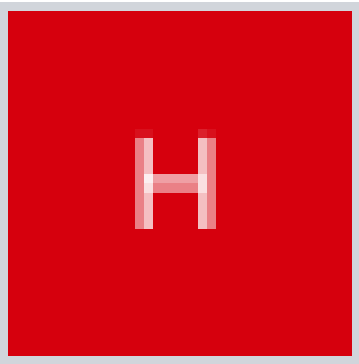
QuICScript Circuit Builder comes with fully customizable features suited for your needs. From changing styling to setting default configurations, the builder is designed to be flexible enabling personalized experience.

1. *Theme configuration*
    - *Modify Gate Style*
    - *Modify Gate Palette*
2. *Feature configuration*
    1. *URL Params*
    2. *defaultConfig*

4.2.1 Theme configuration

Theme configuration allows you to change the look and feel of the Circuit Builder.

Modify Gate Style

Update in `index.html` under `<style>` tag.

```
<style>
   div.gate {
      /* Change Gate background color here: Default (#f3f4f6) */
      background-color: #98CFEC;
   }
</style>
```

Note

Any CSS styling works, including removing border `border: none;`

Modify Gate Palette

Update in index.html under `<script>` tag.

```
<script>
   /** Update Gates present in the palette: Default [
↪"H", "X", "Y", "Z", "C", "N", "P", "T", "I", "s", "U", "d", "m"]  */
   /** Do check the documentation for allowed gates */
   const␣
↪gates = ["H", "X", "Y", "Z", "C", "P", "T", "I", "s", "U", "m"];
</script>
```

Note

Check out *Using QuICScript* for available gates.

Fig. 2: Gate Palette

4.2.2 Feature configuration

The Circuit Builder comes with a set of features that can be customized to suit your needs. Feature configuration enable you to set default parameters for the builder.

Check *Parameters* for feature list.

There are 2 methods to customize the circuit builder.

1. *URL Params*
2. *defaultConfig*

Note

The circuit builder piorities `URL Params` over `defaultConfig`, if `URL Params` is set, `defaultConfig` will be disregarded.

URL Params

URL Params follows Query String.

Using URL Params allows user to insert default parameters into the system for configurations.

Usage

Single Parameter `<url>?<param>=<data>`

Multiple Parameter (Separated with &) `<url>?<param>=<data>&<param>=<data>`

Warning

Make sure <param> follow exactly to Available Params. Any invalid param/data will not reflect in the builder.

Parameters

- `quicscript` - QuICScript String
- `qubits` - Number of qubits
- `columns` - Number of columns
- `qibo` - Enable/Disable of Qibo Feature
- `type` - Type of output
- `input` - Enable/Disable of Input Feature

Table 1: Parameters

Query string parameter	Type	Description	Constraint	Default	Example
`quicscript`	string	QuICScript String (qubits are calculated)	$0 <$ qubits ≤ 20	—	Link
`qubits`	integer	Number of qubits	$0 <$ qubits ≤ 20	`9`	Link
`columns`	integer	Gate depth	—	`20`	Link
`qibo`	boolean	Toggle Qibo Feature	`true` | `false`	`false`	Link
`type`	string	Type of output	`decimal` | `binary` | `invbinary`	`binary`	Link
`input`	boolean	Enable QuICScript input	`true` | `false`	`true`	Link

Table 2: Examples use cases

quicscript	qubits	columns	Remarks	Example
not-in-use	not-in-use	not-in-use	Default configuration	Link
use	not-in-use	not-in-use	Set Bell State	Link
use	use	use	3-qubit QFT circuit with 6 columns	Link
not-in-use	use	use	Empty 5-qubit circuit with gate depth of 10	Link

Note

In this example we are using `https://pqcee.github.io/quicscript-dev-react/` as the base URL, do replace it with where you are hosting the website application

defaultConfig

Similar to *URL Params*, `defaultConfig` has all the parameters as configurable as `URL Params`. `defaultConfig` enables config to be set in the `index.html` for standardized usage when served.

Confguration is done via editing `index.html`, uncommenting the comments to set default config:

```
/* Allow config of the QuICScript Builder
*  Remove the comment to enable the config
*/
const defaultConfig = {
   // defaultQubits: 9, // Default: null
   // defaultColumns: 20, // Default: null
   // defaultQuic: "HI,CN", // Default: null
   // qibo: true, // Default: false
   // defaultResultDelimiter: "binary", // Default: binary (decimal, invbinary, binary)
   // displayInput: false, // Default: true
};
```

4.3 Circuit Builder Setup (iFrame)

Follow these steps:

1. Basic Embedding
   - To embed the link `https://pqcee.github.io/quicscript-dev-react/` in an HTML file using an iframe:

```
<iframe src="https://pqcee.github.io/quicscript-dev-react/?qibo=true&input=false"
        title="iFrame example"
        width="50%"
        height="600"
        allow="fullscreen">
</iframe>
```

> **Note**
>
> This code embeds the page, allows full-screen, enables qibo, disables input field, and adjusts the width and height.*

2. Adjusting `<iframe ...>` Parameters:
   - Width: The width of the frame in CSS pixels. Default is 300.
   - Height: The height of the frame in CSS pixels. Default is 150.
3. Allow fullscreen:
   - To enable fullscreen mode for the iframe, add the `allow="fullscreen"` attribute.
4. Optional Parameters:
   - See *Feature configuration* for more information on the URL parameters you can modify to create custom configurations.

4.4 Preloading Circuits

The Circuit Builder comes with a `callback_render` fuction which enables developers to interact with the visualizer via `index.html`.

This can be triggered after `ReactOnLoad`. Uncommenting the following code in `index.html` will set the circuit to render `H,CN` onload.

```
ReactOnLoad = () => {
   // Example Preload Circuit with H,CN
   // Uncomment the line below to preload a circuit
   // callback_render("H,CN");
};
```

INDEX

Q