Release Summary

v0.11

```
Import["https://qtechtheory.org/questlink.m"];
CreateDownloadedQuESTEnv[];
```

This *major* release significantly extends QuESTlink's analytic processing of symbolic circuits.
It introduces new symbols and functions:

- Matr
- GetCircuitInverse
- SimplifyCircuit
- GetKnownCircuit
- CalcCircuitMatrix
- GetCircuitGeneralised
- GetCircuitSuperoperator

in addition to some other changes.

New features

Matr

The new circuit symbol **Matr** works just like **U** except it does not enforce unitarity. This is convenient to effect general non-unitary operations, or operators which are only approximtely unitary due to numerical imprecision

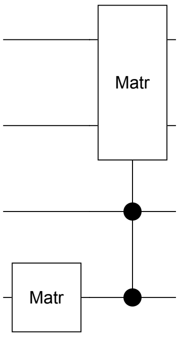
```
? Matr
```

Symbol

Matr[matrix] is an arbitrary operator with any number of target qubits,
specified as a completely general (even non–unitary) square complex matrix.

$$\texttt{circ = Circuit}\left[\mathrm{Matr}_0\left[\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}\right] C_{0,1}\left[\mathrm{Matr}_{2,3}\left[\begin{pmatrix}.1 & 0 & 0 & e^{.2 i}\\ 0 & i & 0 & 0\\ -2 & 0 & 0 & 0\\ 0 & 0 & \pi & 1\end{pmatrix}\right]\right]\right];$$

```
DrawCircuit[circ]
```




```
CalcCircuitMatrix[circ][[;; 8, ;; 8]] // MatrixForm
```

$$\begin{pmatrix} 1.+0.i & 2.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i \\ 3.+0.i & 4.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i \\ 0.+0.i & 0.+0.i & 1.+0.i & 2.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i \\ 0.+0.i & 0.+0.i & 0.3+0.i & 0.4+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i \\ 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 1.+0.i & 2.+0.i & 0.+0.i & 0.+0.i \\ 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 3.+0.i & 4.+0.i & 0.+0.i & 0.+0.i \\ 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 1.+0.i & 2.+0.i \\ 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+0.i & 0.+3.i & 0.+4.i \end{pmatrix}$$

```
InitPlusState @ CreateQureg[4];
ApplyCircuit[%, circ];
GetQuregMatrix[%%] // MatrixForm
```

$$\begin{pmatrix} 0.75+0.i \\ 1.75+0.i \\ 0.75+0.i \\ 1.89012+0.347671i \\ 0.75+0.i \\ 1.75+0.i \\ 0.75+0.i \\ 0.+1.75i \\ 0.75+0.i \\ 1.75+0.i \\ 0.75+0.i \\ -3.5+0.i \\ 0.75+0.i \\ 1.75+0.i \\ 0.75+0.i \\ 7.24779+0.i \end{pmatrix}$$

GetCircuitInverse

GetCircuitInverse[circ] returns a symbolic circuit description of the *inverse* of the input circuit.

```
? GetCircuitInverse
```

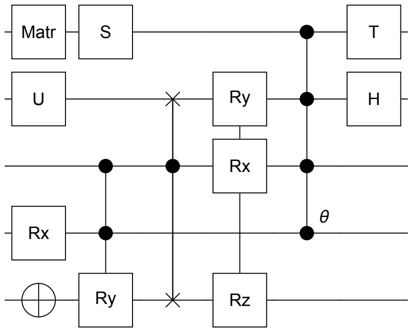
Symbol

GetCircuitInverse[circuit] returns a circuit prescribing the inverse unitary operation of the given circuit.

For instance, this non-trivial input circuit...

$$\texttt{circ = Circuit}\Big[X_0\,Rx_1[a]\,C_{1,2}[Ry_0[b]]\,U_3\Big[\begin{pmatrix}a & b\\ c & d\end{pmatrix}\Big]\,Matr_4\Big[\begin{pmatrix}a & b\\ c & d\end{pmatrix}\Big]\,G[e]\,C_2\big[SWAP_{0,3}\big]\,R[f, X_2\,Y_3\,Z_0]\,S_4\,C_4\big[Ph_{3,2,1}[g]\big]\,T_4\,H_3\Big];$$

```
DrawCircuit[circ]
```

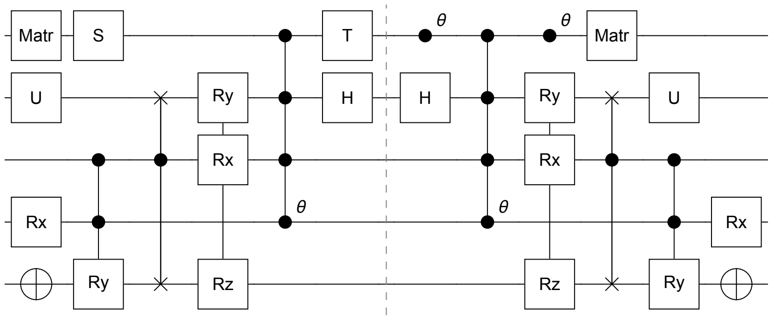


is inverted to

```
inv = GetCircuitInverse[circ]
```

$$\Big\{H_3,\ Ph_4\Big[-\frac{\pi}{4}\Big],\ C_4\big[Ph_{3,2,1}[-g]\big],\ Ph_4\Big[-\frac{\pi}{2}\Big],\ R[-f,\ X_2\,Y_3\,Z_0],\ C_2\big[SWAP_{0,3}\big],\ G[-e],\ Matr_4\Big[\Big\{\Big\{\frac{d}{-b\,c+a\,d},\ -\frac{b}{-b\,c+a\,d}\Big\},\ \Big\{-\frac{c}{-b\,c+a\,d},\ \frac{a}{-b\,c+a\,d}\Big\}\Big\}\Big],\ U_3[\{\{\mathrm{Conjugate}[a],\ \mathrm{Conjugate}[c]\},\ \{\mathrm{Conjugate}[b],\ \mathrm{Conjugate}[d]\}\}],\ C_{1,2}[Ry_0[-b]],\ Rx_1[-a],\ X_0\Big\}$$

```
DrawCircuit[{circ, inv}]
```



Beware that not every gate has an inverse.

```
GetCircuitInverse @ Circuit[M₀]
```

GetCircuitInverse : Could not determine the inverse of gate M_0.

```
$Failed
```

SimplifyCircuit

SimplifyCircuit[circ] performs basic but comprehensive simplification of the circuit, useful as a pre-step before advanced topological or approximate simplification. **SimplifyCircuit** will...

- remove adjacent idempotent operations
- sort gate qubit indices, even if this requires adjusting the gate arguments
- combine arguments of adjacent parameterised gates
- multiply matrices of adjacent unitaries
- merge global phases
- merge adjacent Pauli operators, and with Pauli rotations
- remove zero-parameter and identity gates
- mod arguments of rotation gates to within their periods
- simplify single-target Pauli gadgets to rotations
- replace special-param rotation gates with global phases

```
? SimplifyCircuit
```

Symbol

SimplifyCircuit[circuit] returns an equivalent but simplified circuit.

```
SimplifyCircuit @ Circuit[ Ph0,1[x] C1[Ph0[y]] C0[S1] C1[T0] ]
```

$\left\{\mathrm{Ph}_{0,1}\left[\frac{3\pi}{4}+x+y\right]\right\}$

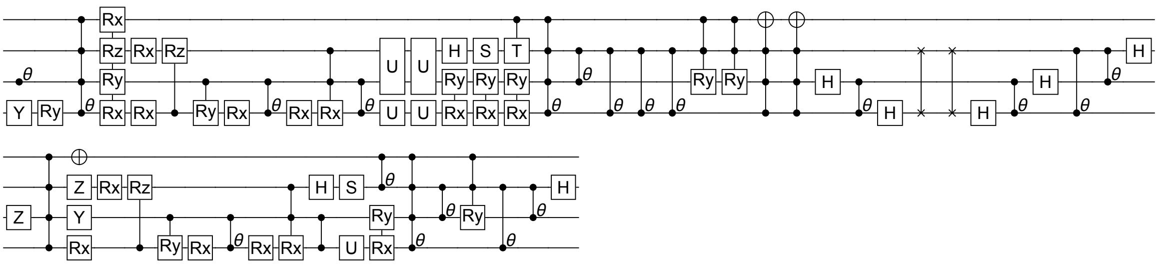
```
SimplifyCircuit @ Circuit[
    hi0 SWAP1,2 H2 H0 X1 Y2 Z3 X1 Y2 H2 Z3 C1[X0] C2[Y3] C2[Y3] C1[X0] H0 SWAP2,1]
```

$\{\mathrm{hi}_0\}$

Expect the number and nature of these simplifications to grow and improve as QuESTlink matures

$$
\begin{aligned}
&\texttt{circ = Circuit}\Big[Y_0\, Ry_0[\pi]\; Ph_1[\pi]\; Ph_{0,1,2,3}[\pi]\; R[\pi,\, X_0\, Y_1\, Z_2\, X_3] \times R[eh,\; X_0]\; Rx_2[-\pi] \\
&\quad C_0[Rz_2[\pi]]\; C_1[R[\phi,\, Y_0]]\; Rx_0[a]\; Ph_{0,1}[13]\; Rx_0[11\,\pi]\; G[x]\; C_{2,1}[Rx_0[11\,\pi]]\; Ph_{1,0}[-\pi] \\
&\quad U_{2,1}\left[\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}\right]\; U_{2,1}\left[\texttt{Inverse@}\begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{pmatrix}\right]\; U_0[\{\{a,\, b\},\, \{c,\, d\}\}]\; G[z]\; U_0\left[\begin{pmatrix} e & f \\ g & h \end{pmatrix}\right] \\
&\quad H_2\, R[3,\, X_0\, Y_1] \times R[200,\, X_0\, Y_1] \times R[-5,\, X_0\, Y_1]\;\; S_2\, C_3[T_2]\; C_{3,2}\big[Ph_{1,0}[x]\big]\; Ph_{2,1}\left[\frac{\pi}{2}\right] \\
&\quad Ph_{2,0}\left[\frac{\pi}{4}\right]\; Ph_{0,2}\left[\frac{\pi}{4}\right]\; C_2\left[Ph_0\left[-\frac{\pi}{2}\right]\right]\; C_{3,2,2}[Ry_1[e]]\; C_{\{3,2\}}[Ry_1[e]]\; C_{0,2,1}[X_3]\; C_{0,1,2}[X_3] \\
&\quad H_1\; Ph_{1,0}\left[\frac{\pi}{2}\right]\; H_0\; SWAP_{0,2}\; SWAP_{0,2}\; H_0\; G[eh]\; Ph_{1,0}\left[-\frac{\pi}{2}\right]\; H_1\; Ph_{2,0}\left[-\frac{\pi}{4}\right]\; Ph_{2,1}\left[-\frac{\pi}{2}\right]\; H_2\Big];
\end{aligned}
$$

```
DrawCircuit[circ, ImageSize → 600]
DrawCircuit[SimplifyCircuit@circ, ImageSize → 300]
```



GetKnownCircuit

GetKnownCircuit[] can dynamically generate canonical quantum circuits.

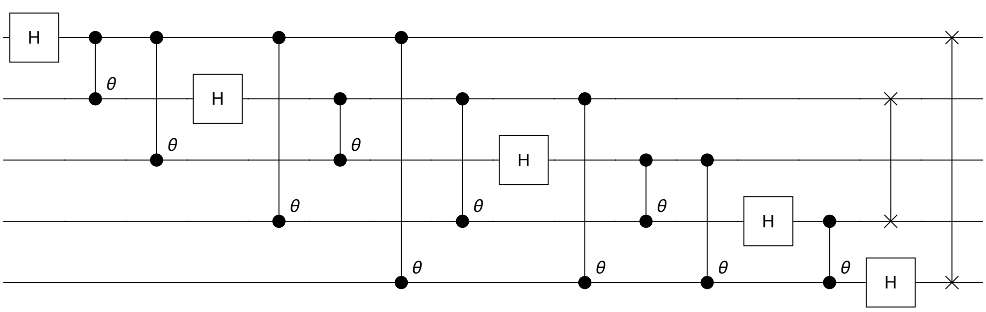
```
? GetKnownCircuit
```

Symbol

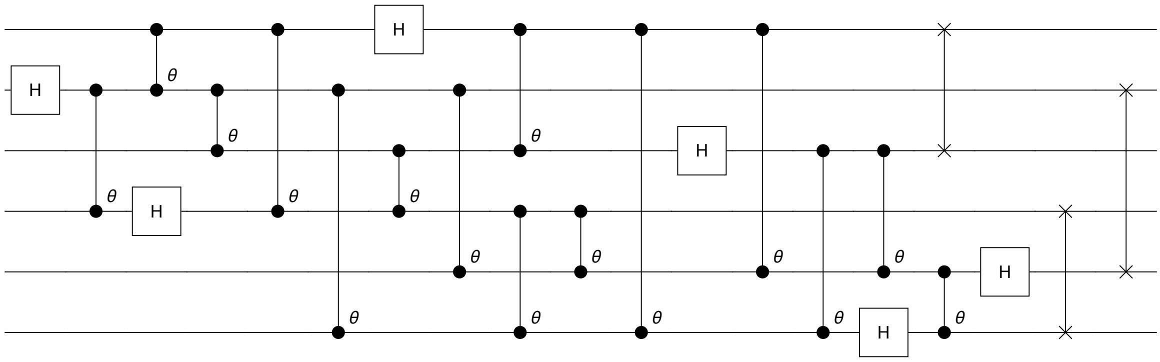
GetKnownCircuit["QFT", qubits]

GetKnownCircuit["Trotter", hamil, order, reps, time]

```
DrawCircuit @ GetKnownCircuit["QFT", 5]
```

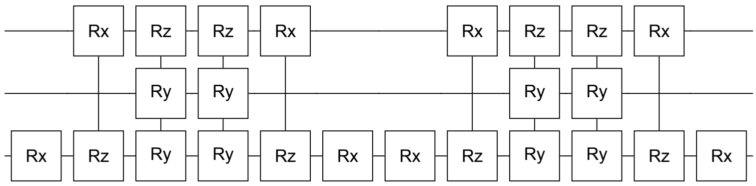


```
DrawCircuit @ GetKnownCircuit["QFT", {1, 0, 3, 5, 2, 4}]
```




```
GetKnownCircuit["Trotter", a X0 + b Y0 Y1 Z2 + c Z0 X2, 2, 2, t]
DrawCircuit[%]
```

$$\left\{R\left[\frac{a\,t}{4}, X_0\right], R\left[\frac{c\,t}{4}, X_2 Z_0\right], R\left[\frac{b\,t}{4}, Y_0 Y_1 Z_2\right], R\left[\frac{b\,t}{4}, Y_0 Y_1 Z_2\right], R\left[\frac{c\,t}{4}, X_2 Z_0\right], R\left[\frac{a\,t}{4}, X_0\right],\right.$$
$$\left. R\left[\frac{a\,t}{4}, X_0\right], R\left[\frac{c\,t}{4}, X_2 Z_0\right], R\left[\frac{b\,t}{4}, Y_0 Y_1 Z_2\right], R\left[\frac{b\,t}{4}, Y_0 Y_1 Z_2\right], R\left[\frac{c\,t}{4}, X_2 Z_0\right], R\left[\frac{a\,t}{4}, X_0\right]\right\}$$



We expect this family of circuits to quickly grow and include canonical variational circuits.

CalcCircuitMatrix

CalcCircuitMatrix[] can now analytically evaluate *channels*!

```
? CalcCircuitMatrix
```

Symbol

CalcCircuitMatrix[circuit] returns an analytic matrix for the given unitary circuit, which may contain symbolic parameters. The number of qubits is inferred from the circuit indices (0 to maximum specified).

CalcCircuitMatrix[circuit] returns an analytic superoperator for the given non-unitary circuit, expressed as a matrix upon twice as many qubits. The result can be multiplied upon a column-flattened density matrix.

CalcCircuitMatrix[circuit, numQubits] forces the number of present qubits.

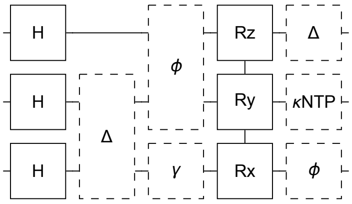
CalcCircuitMatrix accepts optional argument AsSuperoperator->True to obtain a superoperator from a unitary circuit.

```
CalcCircuitMatrix @ Circuit[ Deph_0[λ] ]
```

$$\{\{\sqrt{1-\lambda}\ \text{Conjugate}[\sqrt{1-\lambda}] + \sqrt{\lambda}\ \text{Conjugate}[\sqrt{\lambda}], 0, 0, 0\},$$
$$\{0, \sqrt{1-\lambda}\ \text{Conjugate}[\sqrt{1-\lambda}] - \sqrt{\lambda}\ \text{Conjugate}[\sqrt{\lambda}], 0, 0\},$$
$$\{0, 0, \sqrt{1-\lambda}\ \text{Conjugate}[\sqrt{1-\lambda}] - \sqrt{\lambda}\ \text{Conjugate}[\sqrt{\lambda}], 0\},$$
$$\{0, 0, 0, \sqrt{1-\lambda}\ \text{Conjugate}[\sqrt{1-\lambda}] + \sqrt{\lambda}\ \text{Conjugate}[\sqrt{\lambda}]\}\}$$

$$\text{circ} = \text{Circuit}\Big[H_0\ H_1\ H_2\ \text{Depol}_{0,1}[a]\ \text{Deph}_{1,2}[b]\ \text{Damp}_0[c]\ R[2, X_0\ Y_1\ Z_2]\ \text{KrausNonTP}_1\Big[\Big\{\begin{pmatrix} a & b \\ c & d \end{pmatrix}, \begin{pmatrix} b & c \\ c & a \end{pmatrix}\Big\}\Big]\ \text{Deph}_0[a]\ \text{Depol}_2[b]\Big];$$

```
DrawCircuit[circ]
```




```
matr = CalcCircuitMatrix[circ /. {a → .1, b → .2, c → .3, d → .4}];
N @ matr[[1, 1]]
```

0.0218241 + 0. i

The result is a **superoperator matrix** which can be multiplied upon a **column-flattened density matrix**. The latter is obtained by **Flatten @ Transpose @** matrix

```
ρ = InitPlusState @ CreateDensityQureg[3];
ρv = Flatten @ Transpose @GetQuregMatrix[ρ];
σv = matr . ρv // Chop
```

{0.0149415, 0.0225279, 0.0243866, 0.016442, 0, 0, 0, 0, 0.0225279, 0.0759462,
0.0426938, 0.065502, 0, 0, 0, 0, 0.0243866, 0.0426938, 0.0477619, 0.0422399,
0, 0, 0, 0, 0.016442, 0.065502, 0.0422399, 0.102294, 0, 0, 0, 0, 0, 0, 0, 0,
0.00229868, 0.00346584, 0.00375179, 0.00252954, 0, 0, 0, 0, 0.00346584,
0.011684, 0.00656828, 0.0100772, 0, 0, 0, 0, 0.00375179, 0.00656828, 0.00734799,
0.00649845, 0, 0, 0, 0, 0.00252954, 0.0100772, 0.00649845, 0.0157375}

It is trivial to reformat this back to a matrix for comparison to QuESTlink’s numerical methods.

```
ApplyCircuit[ρ, circ /. {a → .1, b → .2, c → .3, d → .4}];
GetQuregMatrix[ρ] - Transpose @ ArrayReshape[σv, {2^3, 2^3}] // Chop
```

{{0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}, {0, 0, 0, 0, 0, 0, 0, 0}}

GetCircuitGeneralised

GetCircuitGeneralised[circ] produces an equivalent circuit composed only of general operators. This is likely only useful as a subroutine in user-implemented recompilation schemes.

```
? GetCircuitGeneralised
```

Symbol

GetCircuitGeneralised[circuit] returns an equivalent circuit composed only of general unitaries (and Matr operators) and Kraus operators of analytic matrices.

```
GetCircuitGeneralised @ Circuit[X0 Y1 C0[Z1]]
```

$\{U_0[\{\{0, 1\}, \{1, 0\}\}], U_1[\{\{0, -i\}, \{i, 0\}\}], U_{1,0}[\{\{1, 0, 0, 0\}, \{0, 1, 0, 0\}, \{0, 0, 1, 0\}, \{0, 0, 0, -1\}\}]\}$

```
GetCircuitGeneralised @ Circuit[Depol0[a]]
```

$$\left\{\text{Kraus}_0\left[\left\{\left\{\left\{\sqrt{1-a}, 0\right\}, \left\{0, \sqrt{1-a}\right\}\right\}, \left\{\left\{0, \frac{\sqrt{a}}{\sqrt{3}}\right\}, \left\{\frac{\sqrt{a}}{\sqrt{3}}, 0\right\}\right\}, \left\{\left\{0, -\frac{i\sqrt{a}}{\sqrt{3}}\right\}, \left\{\frac{i\sqrt{a}}{\sqrt{3}}, 0\right\}\right\}, \left\{\left\{\frac{\sqrt{a}}{\sqrt{3}}, 0\right\}, \left\{0, -\frac{\sqrt{a}}{\sqrt{3}}\right\}\right\}\right\}\right]\right\}$$

GetCircuitSuperoperator

GetCircuitSuperoperator[circ] produces a Choi--Jamiolkowski superoperator of the input circuit. This will again be most useful for user subroutines.

```
? GetCircuitSuperoperator
```

Symbol

GetCircuitSuperoperator[circuit] returns the corresponding superoperator circuit upon doubly-many qubits as per the Choi–Jamiolkowski isomorphism. Decoherence channels become Matr[] superoperators.

GetCircuitSuperoperator[circuit, numQubits] forces the circuit to be assumed size numQubits, so that the output superoperator circuit is of size 2*numQubits.

```
GetCircuitSuperoperator @ Circuit[X0]
```

$\{X_0, X_1\}$

```
GetCircuitSuperoperator @ Circuit[Rx1[a]]
```

$\{Rx_1[a], Rx_3[-a]\}$

```
GetCircuitSuperoperator @ Circuit[Depol₁[a]]
```

$$\left\{\text{Matr}_{1,3}\left[\left\{\left\{\sqrt{1-a}\ \text{Conjugate}\left[\sqrt{1-a}\right]+\frac{1}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right],0,0,\frac{2}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right]\right\},\right.\right.\right.$$
$$\left\{0,\sqrt{1-a}\ \text{Conjugate}\left[\sqrt{1-a}\right]-\frac{1}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right],0,0\right\},$$
$$\left\{0,0,\sqrt{1-a}\ \text{Conjugate}\left[\sqrt{1-a}\right]-\frac{1}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right],0\right\},$$
$$\left.\left.\left.\left\{\frac{2}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right],0,0,\sqrt{1-a}\ \text{Conjugate}\left[\sqrt{1-a}\right]+\frac{1}{3}\sqrt{a}\ \text{Conjugate}\left[\sqrt{a}\right]\right\}\right\}\right]\right\}$$

Changes

Gate symbols are now protected

You'll never accidentally override them again!

```
? C
```

Symbol

C is a declaration of control qubits (subscript),
which can wrap other gates to conditionally/controlled apply them.

```
C = 2;
```

Set: Symbol C is Protected.

CalcCircuitMatrix will report unrecognised gates

```
CalcCircuitMatrix @ Circuit[X₀ Y₁ Shplee₂]
```

CalcCircuitMatrix: Circuit contained an unrecognised or unsupported gate: Shplee_2

```
$Failed
```

Pauli Hamiltonians will ignore zero scalars

Previously, functions like **ApplyPauliSum**, **CalcExpecPauliSum** and **CalcPauliSumMatrix** would report a "stand-alone scalar" error when they contained a numerical zero, **0.`** . Now, this innocuous term often resulting from **SimplifyPaulis** will be ignored

```
h = X₀ Y₁ Z₂ + 0.;
{ψ, ϕ} = InitPlusState /@ CreateQuregs[3, 2];
CalcExpecPauliSum[ψ, h, ϕ]
```

```
0.
```