qvtrace for SIMULINK

User Manual
Version 0.11.7

Table of Contents

QRA critical thinking

1. Overview

Thank you for choosing QVtrace to help you build complex systems with confidence. QVtrace is a powerful analysis tool that enables systems engineers to easily and rigorously probe Simulink models to verify the behaviour of their complex designs.

The following guide provides an overview of the installation and use of QVtrace, including supported mathematical expressions, input formats, general workflow, and the QCT querying language syntax.

The scope of analysis capabilities and component support in QVtrace is growing rapidly. We work closely with our customers and partners to ensure the tool grows and meets their specific needs.

The current coverage in QVtrace is as follows:

Linear and non-linear components and query conditions (constraints) including: ‣ Trigonometric ‣ Exponential ‣ Logarithmic ‣ Matrix arithmetic	Standard data types: ‣ Booleans ‣ Integers ‣ Reals ‣ Arrays ‣ Buses	Input formats ‣ Simulink models in MDL and SLX file formats ‣ Matlab .mat files (for parameter values)

This guide will walk you through the necessary steps to install and begin working with QVtrace for Simulink. If you have questions regarding any topic relating to QVtrace after reading through this guide, please contact us at support@qracorp.com or visit our Help Center at support.qracorp.com.

2. Installing and Starting QVtrace

There are two main components to QVtrace for Simulink, the QVtrace server that houses the analysis engine, and the QVtrace web interface.

2.1. System Requirements for the QVtrace server

- OS: QVtrace will run on any OS that supports the Docker Container Platform.
- HD: The server size will vary depending on the number of users. For a single user the server will require approximately 1.5GB of space, additional user support will increase by a factor of ~0.7.
- RAM: Minimum 8GB, 16GB or higher suggested.
- Docker: Latest version of Docker CE (see below for a link to obtain Docker).
- Web browser: Latest version of Google Chrome (recommended)

2.2. Installing the QVtrace server

1. With your license confirmation you will receive a link to download the QVtrace TAR file. Download and place the TAR file on the computer and folder where the QVtrace server will be located. If the QVtrace server is not connected to the internet, download this file elsewhere and manually transfer it to the server computer.

2. The QVtrace analysis server requires the Docker container platform. If not already installed in your system, you can download Docker for your specific environment from: https://store.docker.com/search?offering=community&type=edition

3. Double-click on the downloaded Docker file and follow the installation instructions.

2.3. Starting the QVtrace server

1. Launch Docker (the Docker icon should display "Docker is running" when clicked on).

2. Open a command-line terminal and move into the folder where the QVtrace TAR file is located.

3. Load and run QVtrace type (Press 'Enter' after each line):

```
docker load --input QVtrace-#.#.#.docker.tar
docker run --name=qvtrace --detach --publish=2999:2999 instance
```

(The # in the load command above should match that of the actual .tar file)

4. To completely shut down the server and remove any previous instances of QVtrace on Docker type (Press 'Enter' after each line):

```
docker rmi -f instance
docker rm -f qvtrace
```

2.4. Accessing QVtrace:

Once the QVtrace server is running, QVtrace will be accessed through a web browser with the address: http://localhost:2999

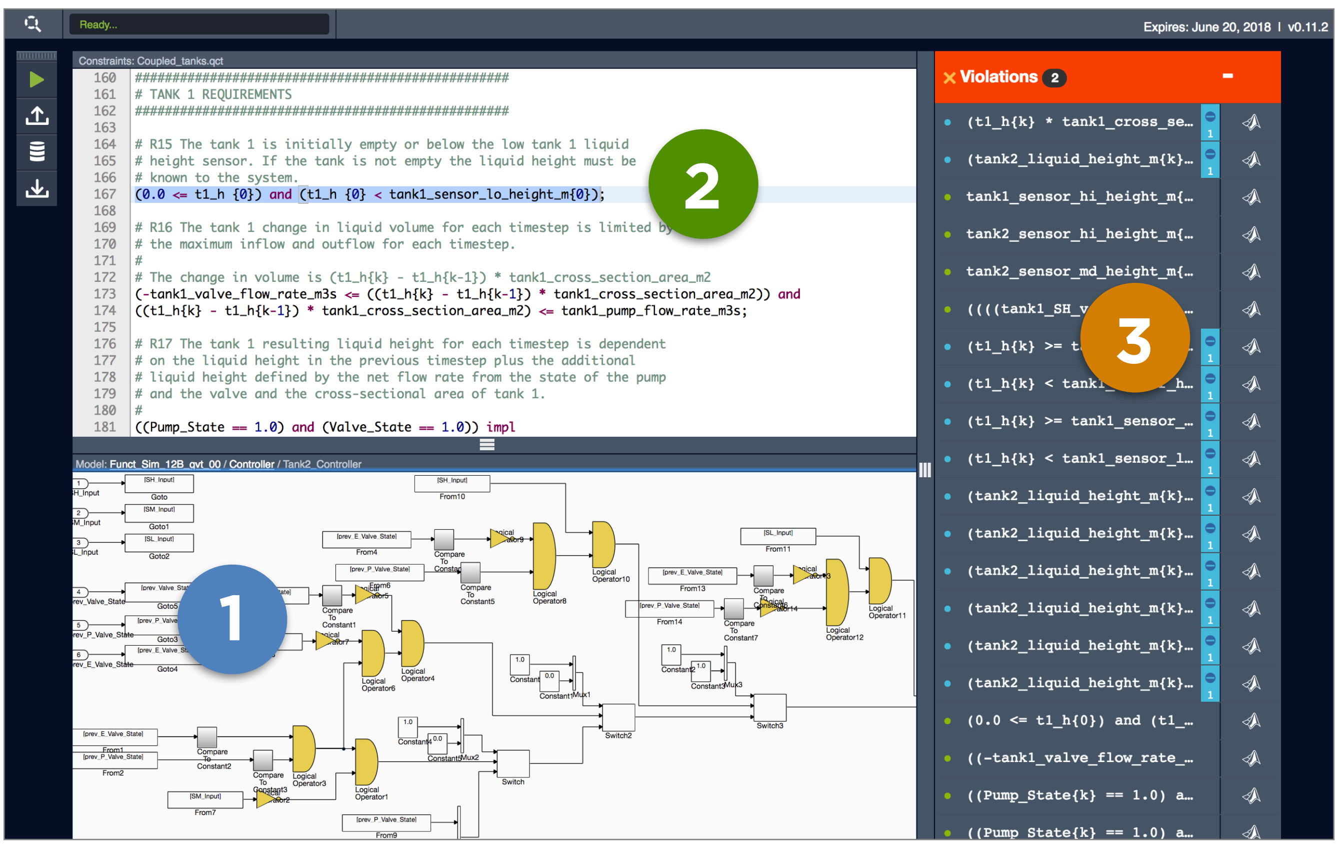
If accessing the QVtrace server on a networked computer then use the address: `http://[server_name]:2999.`

QVtrace has been fully tested to be accessed with the Google Chrome web browser. Although other browsers may render QVtrace appropriately, these have not been fully tested and their performance is not well known. We recommend you use the Google Chrome browser for QVtrace.

3. Using QVtrace

3.1. Understanding the QVtrace user interface

QVtrace has been designed to optimize the workflow for model-based design analysis. The interface has three main sections as shown in the image below and described in detail on the next page.



3.2. The Model Navigation Window

You can visualize and navigate through your Simulink design directly inside QVtrace. The block position and labelling are made to be as close as possible to their Simulink representation, making it easy to navigate through the different subsystems. Similar to navigation within Simulink, double-clicking on any subsystem will display the contents of that subsystem. To return to higher-levels in the design, you click on the desired level in the breadcrumb at the top left corner of the model navigation window.

3.3. The Constraints Window

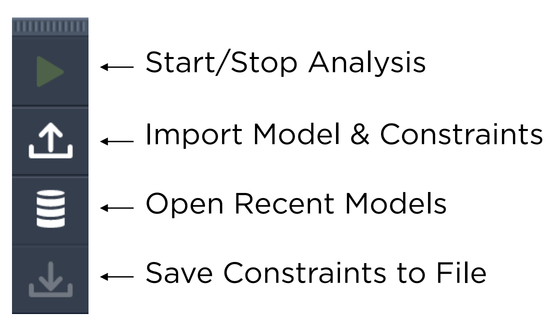
The constraints window is where you will enter the queries used to analyze the design. These can be direct translations of the requirements specifications for the design into the QCT language, as well as sanity checks for bounds on any variable. You can learn more about the QCT language in section.

3.4. The Information Window

The information window provides information actionable information on the importing of models, constraints, and analysis. The window is divided into three tabs:

- The Results tab: Provides analysis results information. You can learn more about this tab in the analysis results section.
- The Problems tab: Provides information on any importing errors or warnings such as unsupported components or unspecified data types, as well as any issues with the constraints written into the constraints window such as improperly stated mathematical expressions.
- The Console tab: Provides information on analysis progress and total analysis time, as well as high-level information on analysis results.

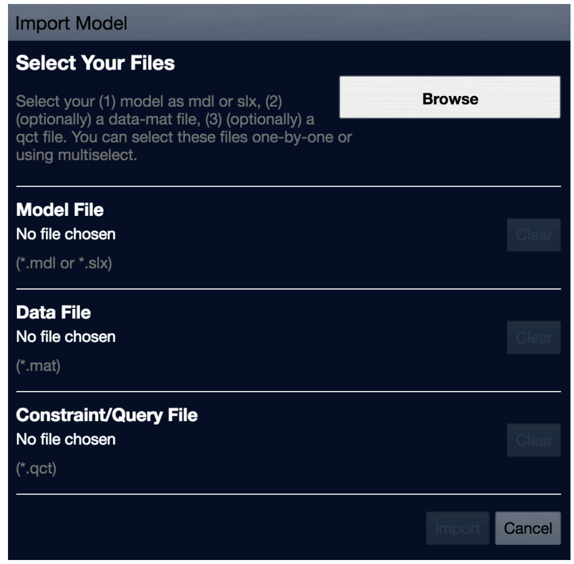
Finally, a floating toolbar gives easy access to the main actions in QVtrace. The image to the right shows the toolbar and the function of the different buttons.



3.5.General workflow in QVtrace

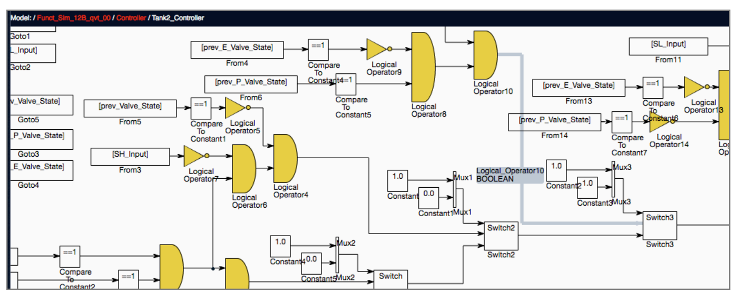
3.5.1.Importing your Simulink Models

QVtrace accepts direct import of Simulink .MDL and .SLX model files. If the model requires additional parameter data, this can imported as a .MAT file simultaneously with the model file..



Note that QVtrace's support is continuously growing and some components may not be currently supported. In situations where a model is imported and contains unsupported components, the Problems Tab will turn yellow and give a description of the unsupported components. In these situations, we recommend contacting us at support@qracorp.com and sending us the details shown in the Problems Tab to help add these components to the import support queue for development.

Once the model has been imported you can explore it by double clicking each subsystem to go into it and using the breadcrumb path at the top left corner of the model navigation window to move to higher levels. Hovering over any wire displays the corresponding parameter name and data type, and right-clicking on any component gives access to its properties for review.



3.5.2.Entering and importing design Constraints

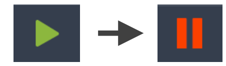
Constraints are conditions you expect the design to meet. These can be requirements specification (translated into formal language), or specific queries to check behaviour such as output bounds or logic/temporal conditions.

When writing a constraint on the fly, you can click on any specific input, output, or wire on the model to place the corresponding parameter where the cursor is on the constraints window. This makes it easier to build queries and conditions. Constraints for a specific model can also be saved and loaded in a text file with the .QCT extension. You can learn more about the constraint language in the QCT Query Language section in this guide.

3.5.3. Analyzing the model

Once the model and constraints have been entered into QVtrace, you are ready to do analysis. QVtrace will rigorously verify if the model and the stated constraints are consistent, and show if and when these are not.

To Start/Stop the analysis press on the Play button on the QVtrace toolbar.



Analysis in QVtrace can be approached in two ways:

a) By formally translating sets of requirements specifications and verifying the model meets these, or
b) As an interactive querying process where the domain expert iteratively queries the model for expected behaviour as the system components are modelled.

Analysis will always be done on all constraints present in the Constraints Window and can be run from any subsystem in the model. It is important to note that the analysis will always check the entire model against all constraints present, and not just the subsystem being shown in the Design Navigation Window.

When running analysis, the constraints will first be verified to ensure these are consistent with the QCT language syntax (see Section 5 for a guide to the QCT language syntax). For example writing “param_1 == 5” where param_1 is a boolean variable will return an error message stating that the constraint is inappropriately written, and no analysis will be run on the model.

4. Interpreting QVtrace Analysis Results

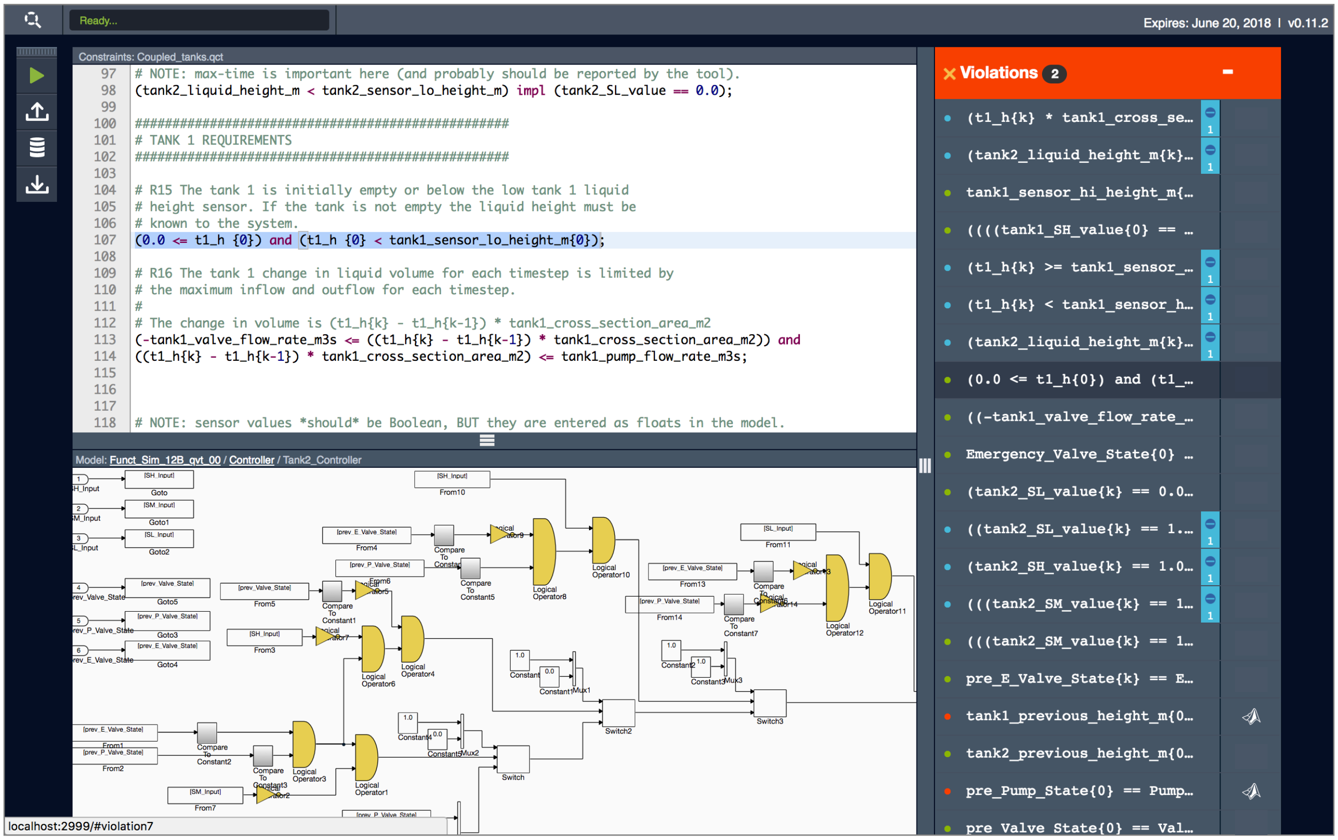
4.1. Possible analysis results

No violations exist: This implies that the model is consistent with the stated constraints for all possible input values, and at all times. As shown in the left image, the Results tab will turn green when no violations exist.

No violations exist up to a maximum time (k_{max}): This implies that the model has been proven to be consistent with the constraints within the implicit temporal logic of the system. However, there is no guarantee that at some greater time step a violation may occur. In these cases, the results tab turns blue, and absolute time references may be required to assess the validity of results over larger timeframes. This is accomplished by including an explicit time reference {t} to the parameters present in the constraints.

Violations found: This states that inconsistencies between the constraints and the model have been found (the term violation arises because QVtrace works by initially assuming the model and the constraints are consistent with each other, and then proving this is the case or finding that this consistency has been violated). The Results tab turns red in these cases.

When violations have been found, the Results tab will be expandable to show which constraints are violated by the design.



As seen in the example image above, two of the constraints have been violated by the design and this number of violations is shown on the red tab. Pressing the '+' sign on the right-hand side of the tab expands it to show the violated constraints. Each of the violated constraints can, in turn, be expanded in the same manner to show the evaluated time steps as well as the variable values at each time step[1]. Each time step can also be expanded to show the variable values used to demonstrate the violation.

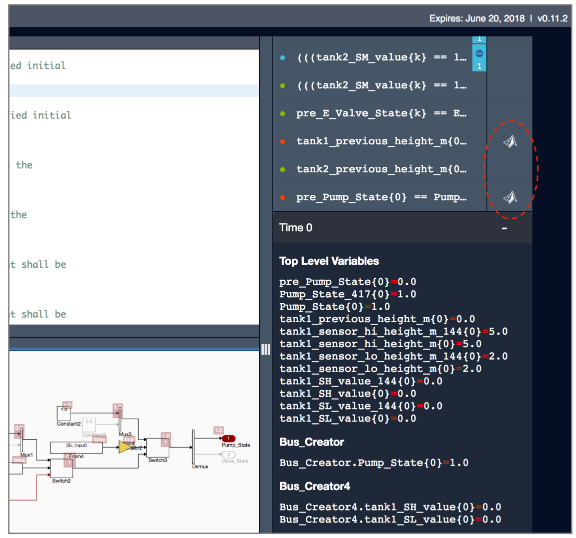
Vertical tabs on the left-hand side of each expanded time step in the results tab show either top-level variables (highest level design inputs and outputs) or all of the variables including all inputs and outputs for each component in each subsystem.

To search for specific variables in the violated constraints, click on the magnifying glass icon on the right side of the red tab. This helps focus on a single variable's values for all evaluated time steps.

[1] Note that the number of time steps evaluated depends on the implicit temporal logic of the design. The more complex the time dependence in the model (due to delays, integrators, etc...) the more time steps need to be evaluated to complete an entire period of this temporal logic of the design.

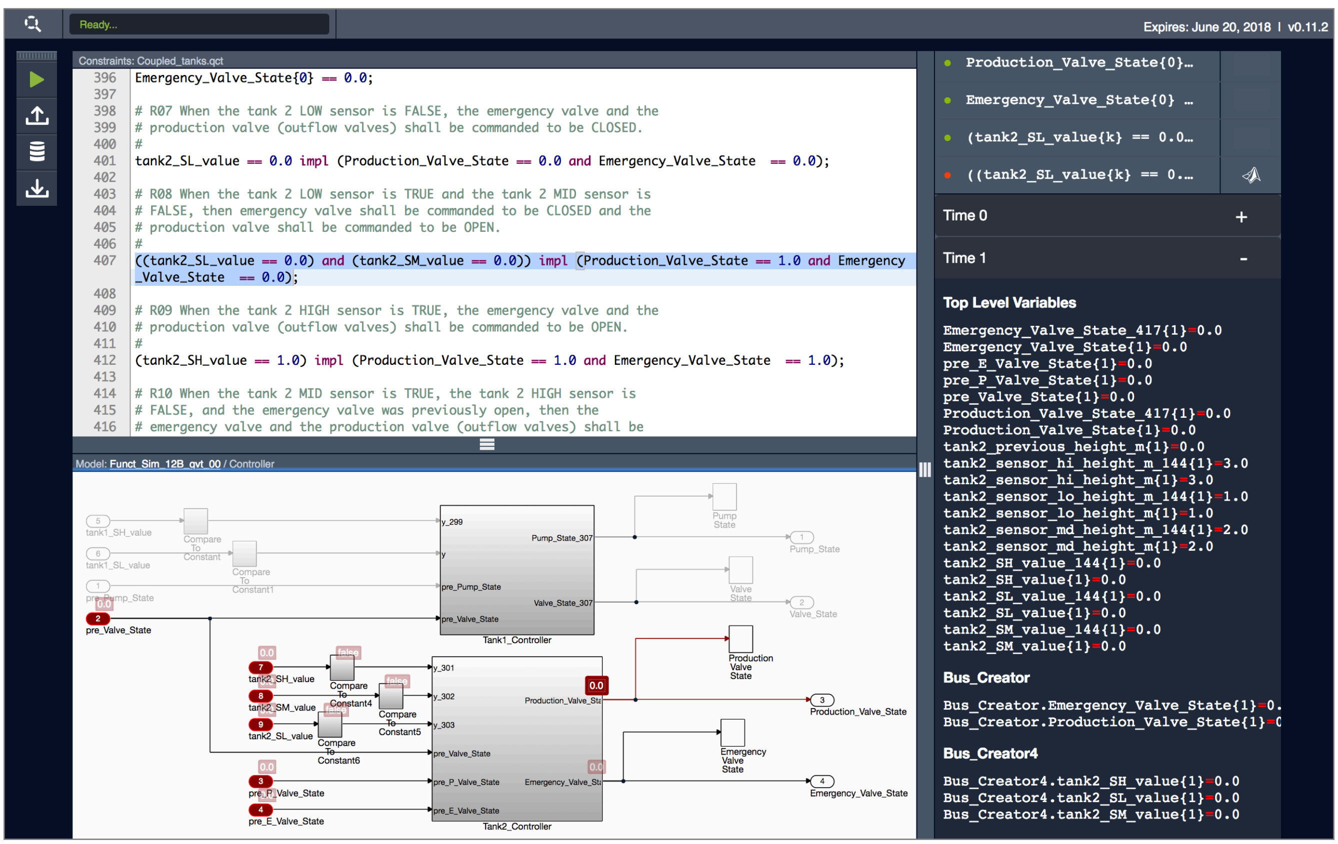
4.2.Counterexamples

The variable values given under a violation in the results tab provide a counterexample that surfaces the inconsistency (violation) between the constraint and design. These values can be exported into a Simulink-friendly format by pressing on the icon beside each violated constraint in the results tab, as circled in the image on the right.



4.3.Violation tracing on the model

When a violation has been found, clicking on the constraint in the results tab will also show a trace on the model highlighting the propagation of the violation throughout all subsystems.



In the image above, the inputs and subsystems that do not contribute to the violation of the constraint are greyed out. This helps the systems engineer more rapidly hone-in on the root cause of error by following the trace in the culprit subsystems.

5. QCT Query Language

The QCT language is a logical predicate language used in QVtrace to state queries and constraints on the behaviour of the Simulink model being analyzed. During analysis, QVtrace evaluates the model against these user-defined constraints to determine if the model and constraints are consistent with each other. The analysis results show the user when there are no violations (i.e. the model meets all stated constraints), or when and where violations are present (i.e. the model and constraints are inconsistent).

In cases where violations are present, QVtrace generates a counterexample with specific values that surface the violation. This values can be exported in a Simulink-readable format to surface the error within the simulation environment.

5.1. Querying a Design

Once a Simulink model has been imported into QVtrace, constraints specific to the model variables can be manually typed in the query window, or they can be imported from a previously created constraints file with the '.qct' extension.

Constraints are expressed using standard mathematical operators and functions on model variables, defined variables, and constants. Variables can be scalars, vectors, or matrices with integer, real, or boolean values.

Each constraint must represent a condition that can be evaluated as either true or false.

Example	
Property	#Tank 1 shall not overflow from its maximum volume of 7.0m^3.
Constraint	tank1_height * tank1_cross_section_area <= 7.0;

Constraints must each be terminated with a semicolon and can be multi-line.

Example	
Constraint 1	(Input1 >= 0.0) and (Input2 == false) impl (Output1 == Input1);
Constraint 2	not (Input2 or Input 3) impl (Output1 == -Input1*2.5);

5.2.Constants

Explicit integer and real numbers can be used as constants within a constraint.

Example
To state that 'output0' should be larger than 5, write 'output0 > 5'

Boolean constants ('true' and 'false') are supported in the QCT language. However, there is no difference between simply stating a boolean expression and stating the boolean expression is true.

Example
'boolVariable1' is equivalent to 'boolVariable1 == true'
'not boolVariable2' is equivalent to 'boolVariable1 == false'

5.3.Variables

Constraints reference variables present in the model by explicitly calling the variable names. The general format for referencing variables is shown in the example below:

Example	
Single variable	subsystemName.variableName
Matrix variable	subsystemName.matrixVariableName(row,column)

where 'variableName' in the single variable above references a variable within a system or subsystem, and 'subsystemName' is optional for referencing variables inside subsystems.

Example
To reference the 85th iteration of the first row and second column element of the 'standby' matrix variable of the 'manager' subsystem, one would write: manager.standby(1,2){85}

Note that clicking on any variable (or connecting wire) within a system or subsystem of the model in QVtrace immediately writes this variable name to the cursor position on the constraints window. This helps rapidly build constraints and query the model.

5.4.New Variables

New variables can be defined and assigned within QCT to help in building constraints. These variables can be scalars, vectors, and matrices with boolean, integer, or real values.

New Variable Type	**Example**
New integer variable 'a1'.	a1 = 2;
New real variable 'a2`.	a2 = 2.5;
New boolean vector variable 'V'	V = [true; false; true; true];
New matrix variable `A` of integers	A = [1, 2, 3; 4, 5, 6; 7, 8, 9];

New variables can also be defined as expressions including model variables.

Example
a = [3,4] + sub1.In3; #where `sub1.In3` is a scalar or a vector variable of length 2 present in the model.

New variables can also be defined as compound boolean expressions.

Example

preconditions = (In1 >= 0.5 and In2 <= 100 and In3 == false);
preconditions and (In1 <= 1) impl Out1 == true;
preconditions and (valueY > 3.5 or valueX == 2) impl Out2 >= 0.0;

New variables allow to easily build more elaborate constraints.

Example

assumption1 = C == 0.1;
assumption2 = reset;
assumption3 = xin == 1.0;
assumption4 = (TL >= BL) and (BL <= yout) and (yout <= TL);
assumptions = and(assumption1, assumption2, assumption3 or assumption4);
all_k(assumptions) impl (fabs(yout{100} - 10.0) <= 0.1);

NOTE: QCT requires that all assignments of new variables precede any constraints. I.e. All assignments must be above all constraints in the QCT window

5.5.Comments

Adding comments to constraints is optional but considered good practice. Comments can add helpful context and may directly reference the original property or requirement being checked. QCT interprets any line beginning with '//' or '#' as a comment.

Example

```
1  #When the sensor measures 0.5 volts or less, the pump shall be on and valve shall be closed.
2  (sensor_value <= 0.5) impl (pump_state  == true and valve_state == false);
```

5.6.Matrices

It is important to mention the way QCT handles some matrix operations.

Currently, for two matrices 'A' and 'B', the expressions 'A == B' and 'A != B' return boolean matrices (following the Matlab approach), in the same element-wise way as the equality function 'eq(A,B)' and distinctness function 'ne(A,B)'.

Note also that you can also write expressions with both vectors and matrices.

Example
For 'V1' and 'V2' vectors, and 'M' a matrix, the following expression can be evaluated V1 * M == V2;

5.7.Time Dependence

The value of a node at a particular time-step can be referenced absolutely, relatively, or for all time-steps. The following table details how variableName can be referenced at different time-steps in QCT constraints:

Reference	Relationship	Description	Example
variableName or variableName{k}	Relative	variableName at the current time-step. Equivalent to variableName{k}.	Stating that Input0 should be greater than 10 at all time-steps can be written ‘Input0 > 10’ or ‘Input0{k} > 10’
variableName{k-1}	Relative	variableName at the previous time-step.*	Stating that Out2 should change each time-step could be stated as: 'Out2{k} != Out2{k-1}'
variableName{k-n}	Relative	variableName at n time-steps prior to the current time-step.*	Stating that Out2 should be the same every 10 time-steps can be stated as: 'Out2{k} != Out2{k-10}’
variableName{0}	Absolute	variableName at the initial condition or 0th time-step.	Input5{0} == 0.0;
variableName{n}	Absolute	variableName at the nth time-step, where n is an integer.	Input0 at time-step 3 would be referenced using 'Input0{3}'
all_k(variableName)	Universal	Evaluates a condition of variableName to satisfied for all time-steps	If In1 is false at time-step 0, then Out1 is always true would be expressed as: '(In1{0} == false) impl all_k(Out1 == true)'
set k_max	Absolute	Sets the maximum number of time-steps to evaluate.	To evaluate the first 12 time-steps of the set of constraints write: ‘set k_max = 12’

*Note that QVtrace uses the convention that relative time-steps are made to time-steps prior to the current time-step, ’k’. Referencing time-steps ahead of the current time is not supported.

Example
The condition that a variable remain unchanged from one step to the next should be stated as ‘variable{k}==variable{k-1};’, whereas ‘variable{k+1}==variable{k};’ is not acceptable.

Time-steps ‘k’ can also be directly referenced within expressions

Example	
Requirement	#The output shall not go over 10 volts after the first 5 time-steps
Constraint	all_k(k>= 5 imp output(k)<=10)

5.8.Glossary of operators and functions

The following table summarize the operators and functions currently supported by QVtrace. Scalar operations and functions are all supported, and many behave in similar manner to corresponding ones in matlab. However, some functions and operations are not currently supported for matrices or vectors. Such functionality is continually growing, and QVtrace will report back to the user when an operation or function is not supported for a given input type.

Operators

Operators	**Description**	**Equivalent function**
+	Addition	plus(X,Y)
&	And	and(X,Y)
&&	And	
and	And	
!=	Distinctness	ne(X,Y)
/	Matrix or scalar division	mrdivide(X,Y)
==	Equality	eq(X,Y)
xor	Exclusive or	xor(X,Y)
>	Greater than	gt(X,Y)
>=	Greater than or equal to	ge(X,Y)
iff	If and only if (biconditional)	iff(X,Y)
impl	Implication (conditional)	impl(X,Y)
|	Inclusive or	or(X,Y)
||	Inclusive or	
or	Inclusive or	
<	Less than	lt(X,Y)
<=	Less than or equal to	le(X,Y)
*	Matrix or scalar multiplication	mtimes(X,Y)
nand	Negative And	nand(X,Y)
nor	Negative or	nor(X,Y)
~	Not	not(X)
./	Piecewise division	rdivide(X,Y)
.*	Piecewise multiplication	times(X,Y)
.^	Piecewise power	power(X,Y)
^	Power	mpower(X,n)
-	Subtraction	minus(X,Y)
X′	transpose of X	transpose(X)

Note: Boolean operators are strictly logical and not bitwise. When comparing matrices, the results are boolean (e.g. A < B if “<” element-wise).

Functions

Functions	Description
abs	Absolute value. Syntax: **abs(X)**
acos	Inverse cosine. Syntax: **acos(X)**
all	For all. Specific to elements of boolean matrices. Syntax: **all(A)**, where A is a boolean matrix. That is, 'all(A)' is true if all elements of matrix 'A' are true.
all_k	For all time-steps. Syntax: **all_k(condition)**, where 'condition' must be en expression that evaluates to true or false for all time-steps k.
and	Logical conjunction. Syntax: **and(X, B, ...)**, where the inputs 'X', 'B' are boolean variables or expressions that evaluate to a boolean (e.g. X = realVariable < 10).
any	There exists. Specific to elements of boolean matrices. Syntax: **any(A)**, where A is a boolean matrix. That is, 'any(A)' is true if there exists an element of matrix `A` that is true.
asin	Inverse sine. Syntax: **asin(X)**
atan	Inverse tanget. Syntax: **atan(X)**
atan2	Four-quadrant inverse tangent Syntax: **atan2(Y,X)**
cos	Cosine. Syntax: **cos(X)**
cosh	Hyperbolic cosine. Syntax: **cosh(X)**
dot	Dot product. Syntax: **dot(A,B)**
eq	Equality function (X = Y). Syntax: **eq(X, Y, ...)**
exp	Natural exponential. Syntax: **exp(X)**
exp10	Exponential base 10. Syntax: **exp10(X)**
ge	Greater than or equal function (X >= Y). Syntax: **ge (X, Y)**
gt	Greater than function (X > Y). Syntax: **gt(X, Y)**
horzcat	Concatenate matrices horizontally. Syntax: **horzcat(X1,...,Xn) == [X1,X2,...,Xn]** where all Xi have the same number or rows.
hypot	Hypotenuse (sqrt(x^2+y^2)). Syntax: **hypot(X,Y)**
iff	if and only if. Syntax: **iff(A, B)**, where the inputs 'A' and 'B' are boolean variables or expressions that evaluate to a boolean (e.g. B = realVariable < 10).
impl	Imply. Syntax: **iff(A, B)**, where the inputs 'A' and 'B' are boolean variables or expressions that evaluate to a boolean (e.g. B = realVariable < 10).
inv	Matrix inv. Syntax: **inv(X)**
ite	Conditional expression 'if-then-else'. Syntax: **ite(b, X, Y)**. For example, y == ite(x<0, -x, x) is equivalent to y == abs(x)
le	Less than or equal function (X <= Y). Syntax: **lt(X, Y)**
log	Natural logarithm. Syntax: **log(X)**
log10	Logarithm base 10. Syntax: **log10(X)**
log2	Logarithm base 2. Syntax: **log2(X)**
logical	Casting function to boolean type. Syntax: **logical(realVariable)**
lt	Less than function (X < Y). Syntax: **lt(X, Y)**
max	Largest element of a vector or matrix. Syntax: **max(A)**, where 'A' is a vector or matrix variable. Should always be used to state a constraint (e.g. max(A) < 10).
min	Smallest element of a vector or matrix. Syntax: **min(A)**, where 'A' is a vector or matrix variable. Should always be used to state a constraint (e.g. max(A) >= 0).
minus	Element-wise subtraction. Syntax: **minus(A, B)**, where 'A' can be a scalar, vector, or matrix variable and 'B' must be the same type as 'A' or scalar.
mpower	Currently equivalent to 'power'.
mrdivide	Division of scalars and matrices. Syntax: **mrdivide(A, B)**, where 'A' and 'B' are scalar or matrix variables.

Functions	Description
mtimes	Multiplication of matrices and scalars. Syntax: **mtimes(A, B)**, where ‘A’ and ‘B’ are matrix or scalar variables.
nand	Negative AND function. Returns FALSE only when both arguments are TRUE. Syntax: **nand(X, Y)**, where ‘X’ and ‘Y’ are boolean expressions.
ne	Not equal function (X != Y). Syntax: **eq(X, Y, ...)**
nor	Negative OR function. Returns TRUE only when both inputs are FALSE. Syntax: **nor(X, Y)**, where ‘X’ and ‘Y’ are boolean expressions.
not	Logical negation. Reverses the logical value of the boolean argument. Syntax: **not(X)**, where ‘X’ is a boolean variable.
nrt	Two argument (Nth) root. Syntax: **nrt(X, n)** would be the nth root of X, where ‘X’ must be a scalar and ’n’ an integer.
or	Logical disjunction. Syntax: **or(X, B, ...)**, where the inputs ‘X’, ‘B’ are boolean variables or expressions that evaluate to a boolean (e.g. X = realVariable < 10).
plus	Matrix and scalar addition. Syntax: **plus(A, B)**, where ‘A’ and ‘B’ are scalar or matrix variables.
power	Raised to the power. Syntax: **power(a, b)**, where ‘a’ and ‘b’ are scalar variables.
prod	Product of array elements. Syntax: **prod(A)**
rdivide	Division of scalars and matrices. Syntax: **mrdivide(A, B)**, where ‘A’ and ‘B’ are scalar or matrix variables.
real	Casting function to real type. Syntax: **real(booleanVariable)**
sign	Element-wise signum function. Syntax **sign(A)**, where ‘A’ is a scalar or matrix variable (e.g. sign([3 -2; 0 17]) == [1 -1; 0 1] verifies as true).
sin	Sine. Syntax: **sin(X)**
sinh	Hyperbolic sine. Syntax: **sinh(X)**
sqrt	Square root. Syntax: **sqrt(X)**
sum	Sum of array elements. Syntax: **sum(A)**
tan	Tangent. Syntax: **tan(X)**
tanh	Hyperbolic tangent. Syntax: **tanh(X)**
times	Element-wise multiplication. Syntax: **times(A,B)**, where ‘A’ and ‘B’ are scalar or matrix variables.
transpose	Matrix transpose. Syntax: **transpose(X)**
uminus	Unary minus. Syntax: **uminus(X)**, where ‘X’ is a vector or matrix variable.
vertcat	Concatenate matrices vertically. Syntax: **vertcat(X1,...,Xn) == [X1;X2;...;Xn]** where all Xi have the same number of columns.
xor	Exclusive OR function. Syntax: **xor(X,Y)**, where ‘X’ and ‘Y’ must be boolean expressions.
isequals	Returns TRUE if A and B are the same size and equal element-wise; False otherwise. **Syntax: isequal(A,B)**
size	Returns the dimension of an array or matrix in a 1x2 vector. Syntax: **size(A)**
det	Returns the determinant of square matrix A. Syntax: **det(A)**

5.9.Glossary of QCT editor commands

Line Operations		
Action	**Windows/Linux**	**Mac**
Remove line	Ctrl-D	Command-D
Copy lines down	Alt-Shift-Down	Command-Option-Down
Copy lines up	Alt-Shift-Up	Command-Option-Up
Move lines down	Alt-Down	Option-Down
Move lines up	Alt-Up	Option-Up
Remove to line end	Alt-Delete	Ctrl-K
Remove to linestart	Alt-Backspace	Command-Backspace
Remove word left	Ctrl-Backspace	Option-Backspace, Ctrl-Option-Backspace
Remove word right	Ctrl-Delete	Option-Delete
Selection		
Action	**Windows/Linux**	**Mac**
Select all	Ctrl-A	Command-A
Select left	Shift-Left	Shift-Left
Select right	Shift-Right	Shift-Right
Select word left	Ctrl-Shift-Left	Option-Shift-Left
Select word right	Ctrl-Shift-Right	Option-Shift-Right
Select line start	Shift-Home	Shift-Home
Select line end	Shift-End	Shift-End
Select to line end	Alt-Shift-Right	Command-Shift-Right
Select to line start	Alt-Shift-Left	Command-Shift-Left
Select up	Shift-Up	Shift-Up
Select down	Shift-Down	Shift-Down
Select page up	Shift-PageUp	Shift-PageUp
Select page down	Shift-PageDown	Shift-PageDown
Select to start	Ctrl-Shift-Home	Command-Shift-Up
Select to end	Ctrl-Shift-End	Command-Shift-Down
Duplicate selection	Ctrl-Shift-D	Command-Shift-D
Select to matching bracket	Ctrl-Shift-P	---
Go to		
Action	**Windows/Linux**	**Mac**
Go to left	Left	Left, Ctrl-B
Go to right	Right	Right, Ctrl-F
Go to word left	Ctrl-Left	Option-Left
Go to word right	Ctrl-Right	Option-Right
Go line up	Up	Up, Ctrl-P
Go line down	Down	Down, Ctrl-N

Go to line start	Alt-Left, Home	Command-Left, Home, Ctrl-A
Go to line end	Alt-Right, End	Command-Right, End, Ctrl-E
Go to page up	PageUp	Option-PageUp
Go to page down	PageDown	Option-PageDown, Ctrl-V
Go to start	Ctrl-Home	Command-Home, Command-Up
Go to end	Ctrl-End	Command-End, Command-Down
Go to line	Ctrl-L	Command-L
Scroll line down	Ctrl-Down	Command-Down
Scroll line up	Ctrl-Up	---
Go to matching bracket	Ctrl-P	---
Other		
Action	**Windows/Linux**	**Mac**
Indent	Tab	Tab
Outdent	Shift-Tab	Shift-Tab
Undo	Ctrl-Z	Command-Z
Redo	Ctrl-Shift-Z, Ctrl-Y	Command-Shift-Z, Command-Y
Toggle comment	Ctrl-/	Command-/
Transpose letters	Ctrl-T	Ctrl-T
Enter full screen	Ctrl-Enter	Command-Enter
Change to upper case	Ctrl-U	Ctrl-U
Overwrite	Insert	Insert
Macros replay	Ctrl-Shift-E	Command-Shift-E
Macros recording	Ctrl-Alt-E	---
Delete	Delete	---

Simulink Block Support

1-D Lookup Table	Degrees to Radians	Interval Test	Rounding Function
2-D Lookup Table	Delay	Interval Test Dynamic	S-R Flip-Flop
3x3 Cross Product	Demux	Invert 3x3 Matrix	Saturation
Abs	Detect Change	J-K Flip-Flop	Saturation Dynamic
ActionPort	Detect Decrease	Log	Scope
Add	Detect Fall Negative	Logical Operator	Selector
Adjoint of 3x3 Matrix	Detect Fall Nonpositive	Manual Switch	Sign
Algebraic Constraint	Detect Increase	Math Function	Signal Conversion
Assertion	Detect Rise Nonnegative	Math Reciprocal	Signal Specification
Assignment	Detect Rise Positive	Matrix Concatenate	Signed Sqrt
atan	Determinant of 3x3 Matrix	Max	Sin
Atan2	Difference	Memory	Sincos
Backlash	Digital Clock	Merge	Sine Wave
Band-Limited White Noise	Direct Lookup Table (n-D)	Min	Sine Wave Function
Bias	Discrete Derivative	MinMax	Slider Gain
Bus Assignment	Discrete Filter	Model Info	Spherical to Cartesian
Bus Creator	Discrete Pulse Generator	Multiport Switch	Sqrt
Bus Selector	Discrete Transfer Fcn	Mux	Squeeze
Bus to Vector	Discrete Zero-Pole	n-D Lookup Table	Step
Cartesian to Polar	Discrete-Time Integrator	Out1	SubSystem
Cartesian to Spherical	Display	Permute Dimensions	Subtract
Ceil	Divide	Polar to Cartesian	Sum
Celsius to Fahrenheit	DocBlock	Polynomial	Sum of Elements
Chirp Signal	Dot Product	Prelookup	Switch
Clock	Enable	Product	Switch Case
Combinatorial Logic	Environment Controller	Product of Elements	Terminator
Compare To Constant	Exp	Quantizer	To File
Compare To Zero	Fahrenheit to Celsius	Quaternion Conjugate	To Workspace
Constant	Fcn	Quaternion Division	Transfer Fcn First Order
Cos	First-Order Hold	Quaternion Inverse	Transfer Fcn Lead or Lag
Coulomb & Viscous Friction	Fix	Quaternion Modulus	Transfer Fcn Real Zero

Simulink Block Support

Counter Free-Running	Floating Scope	Quaternion Multiplication	Transpose
Counter Limited	Floor	Quaternion Norm	Trigger
Create 3x3 Matrix	From	Quaternion Normalize	Trigonometric Function
	From Workspace	Quaternion Rotation	Unary Minus
D Latch	Gain	Radians to Degrees	Unit Delay
Data Type Conversion	Goto	Ramp	Unit Delay External IC
Data Type Conversion Inherited	Ground	Random Number	Vector Concatenate
Data Type Duplicate	Hit Crossing	Rate Limiter	
Data Type Propagation	IC	Rate Limiter Dynamic	Weighted Sample Time
Data Type Scaling Strip	If	Reciprocal	Weighted Sample Time Math
Dead Zone	In1	Reciprocal Sqrt	Wrap To Zero
Dead Zone Dynamic	Increment Real World	Relational Operator	XY Graph
Decrement Real World	Increment Stored Integer	Relay	Zero-Order Hold
Decrement Stored Integer	Index Vector	Repeating Sequence Stair	
Decrement Time To Zero	Inport Shadow	Reshape	
Decrement To Zero	Interpolation Using Prelookup	Round	

support.qracorp.com