RTflow Help

The RTflow Help is organized into the following main parts:

- Definitions. One single table with explanations of the most common terms and concepts used in the RTflow Help and in the software itself.
- Tutorials. Three tutorials that together provide a complete overview of RTflow's functions and uses.
- The Main Window. Detailed reference for all menu commands and for the project tree.
- The Schematic Editor. Detailed reference for the schematic editor.
- Blocks. Detailed reference for all built-in blocks.
- The Simulator. Detailed reference for the simulator.
- The Code Generator. Detailed reference for the code generator.
- Compiler Errors. Detailed reference for all compiler error messages.

Definitions

Term	Description
Active file	The file that is in view in the tab area, if any. A file is made active by clicking its tab.
Block	A building block for schematics, for example an input port, an adder or a user block. When there is no risk for confusion, "block" will also refer to a block instance.
Block instance	An instance of a block in a schematic. For example, there may be several instances of the block Add in a schematic.
Blocks tree	A tree containing all available blocks for use in the schematic editor, available under the **Blocks** tab in the tree view in the left part of the main window.
Causality loop	A set of blocks that are connected in a loop without any delay (Pre) block. Causality loops are illegal in RTflow and will generate a compiler error.
Command	A function in RTflow that can be invoked by the user through a menu, a toolbar button or a keyboard shortcut.
Connection	A connection between port instances in a schematic. A connection specifies that all input and output ports that the connection is attached to have the same value at each cycle.
Constant	An input whose value is a constant value specified at design/compile time for each instance. In a schematic, the user specifies the value in editable fields directly in the block, or in the block instance properties window. For example, the primitive block Gain has a constant specifying the amount of gain.
Cycle	A discrete time step, also called execution cycle or instant. During the execution of a model, all signals are updated exactly once per cycle.
Element	See Schematic element.
Equation	A subnode or the usage of a primitive function within a node. Each block instance corresponds to an equation with the same name.
Execution cycle	See cycle.
Frame	An add-on of graphics to schematics, typically used to enhance the appearance of a printed schematic.
Graph area	The right part of the simulator view, showing graph plots of signals values over time.
Instance	See Block instance.
Joint view	A display mode of the graph area, where several graphs are displayed in one common coordinate system.
Main window	The main application window that opens when RTflow starts, and that contains the menu, the tree view, the tab area and the message view.
Node	The function of a non-primitive block. Each schematic corresponds to a node with the same name.
Primitive	A block whose function is hard-coded in RTflow, as opposed to a standard block or a

block	user block, whose functions are defined in schematics or other source files. The Add block is an example of a primitive block.
Project	A set of source files and settings for simulation, code generation et c for these source files.
Project tree	A visualization of the set of source file, available under the **Project** tab in the tree view in the left part of the main window.
Port	An input or output of a block or a schematic. When there is no risk for confusion, "port" will also refer to port instance.
Port instance	A port on a block instance in a schematic, symbolized by a small line on the edge of the block instance. Port instances are the endpoints of connections.
Sample time	The real time duration of one execution cycle.
Schematic	A sheet containing blocks, connections and other graphical elements that define the behavior of a system or a sub-component of the system.
Schematic element	A block instance or a connection.
Signal	The bearer of a value that changes with time during simulation or execution. This is the same as a *variable* in many other programming languages. Each connection in a schematic corresponds to a signal with the same name.
Signal area	The left part of the simulator view, showing signal names, values and other signal attributes in a table.
Simulator view	The view that opens under the **Simulator** tab when the simulator is started. It consists of the signal area, the graph area and a toolbar.
Source file	A file that is part of the project and that can be compiled. In the current version of RTflow, schematic files are the only source files.
Split view	A display mode of the graph area, where each graph is displayed in an individual row.
Subnode	The instance of a node within another node. Each non-primitive block instance corresponds to a subnode with the same name.
Standard block	A block whose function is defined in a source file that is delivered with RTflow, for example the Integral block.
Symbol	The graphical representation of a block.
Tab area	The main area of the application window where the source file editors and the simulator open.
Tree view	The left part of the main window, showing either the project tree or the blocks tree.
User block	A block whose function is defined in a source file not delivered with RTflow. Typically these blocks have been compiled from source files in the user's project.

Tutorials

The purpose of the tutorials is to present the different aspects of RTflow and guide through the most common functions. After completing the three tutorials, which should take no more than 15-20 minutes each, you should be fully prepared to start making your own models with help from the subsequent reference sections.

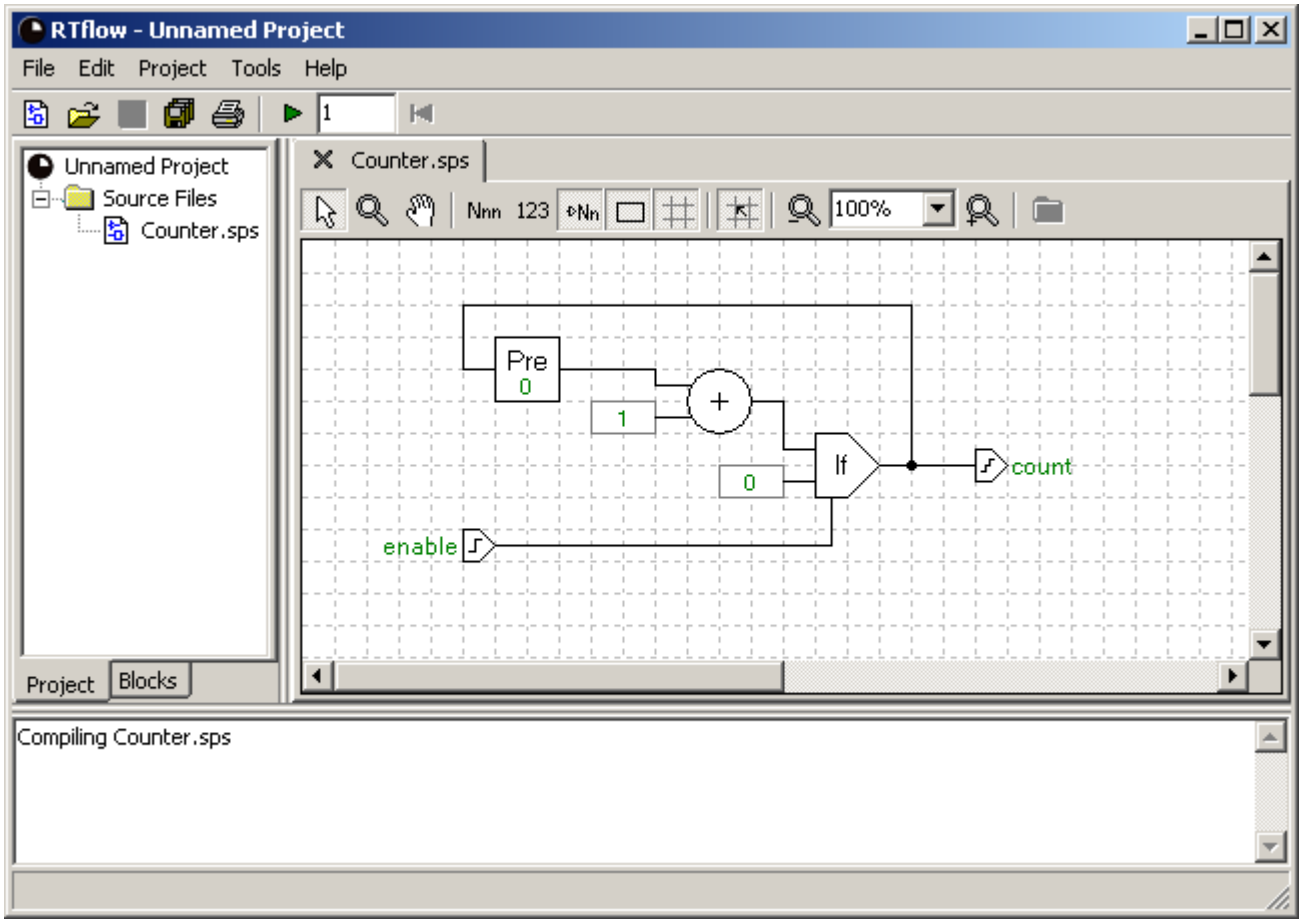
The three tutorials are:

- A simple counter. The simple counter tutorial demonstrates the very basics of the schematic editor, the simulator and the code generator. Here, you build a simple model from scratch, you verify it by running it in the simulator, and finally you generate C++ and VHDL code from it.
- A PID controller. The PID controller tutorial goes more into details on the simulator. Starting with a project containing the controller connected to a model of a simple physical system, you use the simulator to fine-tune the parameters of the controller.
- A Java game. The java game tutorial shows how RTflow can be used for desktop software development. In this example, RTflow is used to develop the physics engine and the artificial intelligence of an enemy player in a simple Java game. You start off from a simple but complete version of the game, and iteratively improve the AI, building and running the game from within RTflow.

Tutorial 1: A Simple Counter

This tutorial presents the basic functions in RTflow by explaining how to design a simple counter, simulate it and finally generate C++ and VHDL code from it. The complete files for the design can be found in the /Examples/Counter folder under the application folder; however, this tutorial will explain how to create the same files so they are not needed for the exercise.

The functionality of the counter is very simple: it has one boolean input signal **enable** and one integer valued output signal **count**. When **enable** is 0, **count** is 0. When **enable** is 1, **count** is the number of cycles that **enable** has been 1. The schematic for the counter, which will be created during this tutorial, is shown below.



The counter illustrates two basic properties of RTflow's execution model:

- *Reactive.* All RTflow models have a set of input signals and a set of output signals, which can be of different data types. The model describes how the values of the output signals are continuously computed from the values of the input signals. There is no specification of where these inputs come from, because it is the surrounding environment's responsibility to feed the system with data. Likewise, there is no specification of what happens with the outputs, because it is also the environment's responsibility to read and apply them.
- *Cyclic.* RTflow models are executed in discrete time steps or *cycles*, where the input values are sampled and the output values are produced exactly once every cycle. It is possible to obtain values from the previous cycle using the block **Pre**.

Mathematically, the counter can be expressed as a function from sequences of boolean numbers to sequences of integers as follows:

$$count[0] = enable[0],$$
$$count[n] = 0, \text{ if } enable[n] = 0 \text{ for } n > 0,$$
$$count[n] = count[n\text{-}1] + 1, \text{ if } enable[n] = 1 \text{ for } n > 0$$

Note: Throughout this tutorial, there will be references to toolbar buttons, such as the **New Schematic** toolbar button. Here, **New Schematic** refers to the tooltip that is shown when the mouse cursor is placed over the button. Notice that there may be two toolbars in view, both the main toolbar right under the menu and a page-specific toolbar under the tabs. The toolbar buttons are summarized in the sections Main Window Toolbar Buttons, Schematic Editor Toolbar Buttons and Simulator Toolbar Buttons.

The Schematic Editor (Tutorial)

This section explains step-by-step how to create a schematic that models the counter.

Create a new project file

It is not strictly necessary to create a project file, but it is recommended to start the project by saving an empty project file in order to settle the project path.

- In the **File** menu, choose **Save Project As...**.
- Choose a location in the file selector, enter a name (e.g. **Counter**; the extension .spp will be added automatically if necessary), and click **OK**.

Create a new schematic

- Create a new schematic by choosing **File:New Schematic**, by clicking the **New Schematic** toolbar button or by pressing Ctrl-N. A new file called Unnamed1.spp is created as can be seen under the **Source Files** folder in the project tree in the left part of the window. Notice that the file does not yet exist on disk. The first time it is saved (**File:Save** or Ctrl-S), RTflow will ask for its actual location and filename.

Add blocks

A schematic consists of blocks and connections. The blocks represent the functions to be computed, and the connections, shown as simple lines between the blocks, represent the signals that flow between these functions. There are also port blocks to represent input and output signals. Blocks are added to the schematic by dragging them from the blocks tree, on the **Blocks** tab in the left part of the window, into the schematic area.

- Click the **Blocks** tab in the lower left part of the window to show the blocks tree.
- Drag the **BoolInput** block from the blocks tree into the schematic area. This block represents a boolean input signal.
- Drag the **IntOutput** block from the blocks tree into the schematic area. This block represents an integer output signal.
- Open the **Logic** folder in the blocks tree by clicking on its + sign.
- Drag the **If** block from the blocks tree into the schematic area. This block represents an "If" function that selects either of two values depending on a third boolean value.

Rename the inputs and outputs

RTflow gives input and output blocks default names like **BoolInput1** and **IntOutput1**. These can be changed by clicking on the name.

- Click on **BoolInput1**, enter **enable** and press Enter.
- Click on **IntOutput1**, enter **count** and press Enter.

In general, text that appears in green color in the schematic can be edited by clicking it. Editing can be aborted by pressing the Esc key. For details, see the Renaming an Element section.

Add connections

Connections are created by clicking once on the small line representing the output port of the source block and then once on the input port of the destination block.

- Move the mouse cursor to the right edge of the **enable** input block. A blue square appears around its port, indicating that clicking will start creating a connection from this port. Click the port.
- Move the mouse cursor to the small vertical line at the bottom of the **If** block. A blue square appears around its port, indicating that clicking will finalize the connection to this port. Click the port. A connection has now been created, indicating that the **enable** input will be used as the selector value of the **If** function.
- Create a connection from the port on the right edge of the **If** block to the port on the left edge of the **count** output in the same way as above. The output block will move up or down vertically to align with the block that it is connected to. (This behavior can be turned off as follows: Right-click the schematic, choose **Properties...**, click the **Automation** tab, uncheck **Align ports vertically with blocks** and click **OK**. However, keeping this function on typically makes it easier to maintain the layout of simpler schematics.)

RTflow does not allow open-ended connections. This means that clicking anywhere else than on a destination port when a connection has been started (as after the first step above) will cancel the connection. It also means that if a block is deleted, all connections to and from the deleted block will also be deleted automatically. For details, see the Adding a Connection section.

Finalize the schematic

Complete the schematic with the blocks and connections as shown in the screenshot above. The box with a single number in it is the **Value** block in the **Arithmetic** folder. The value in it can be edited by clicking it, just as with input and output names. The **Add** block is likewise in the **Arithmetic** folder, and the **Pre** block can be found in the **Dynamic** folder.

- At any time, the schematic can be saved with **File:Save** or Ctrl-S. Name the schematic file **Counter.sps**.
- Blocks can be moved by dragging them, and the inner segments of connections can also be moved by dragging. See the Moving a Block and Moving a Connection sections for details.
- Blocks and connections can be deleted by clicking them once and pressing the Del key.
- Also notice **Undo**, **Redo** and the clipboard functions in the **Edit** menu.

The Simulator (Tutorial)

This section explains how to verify the counter using the built-in simulator.

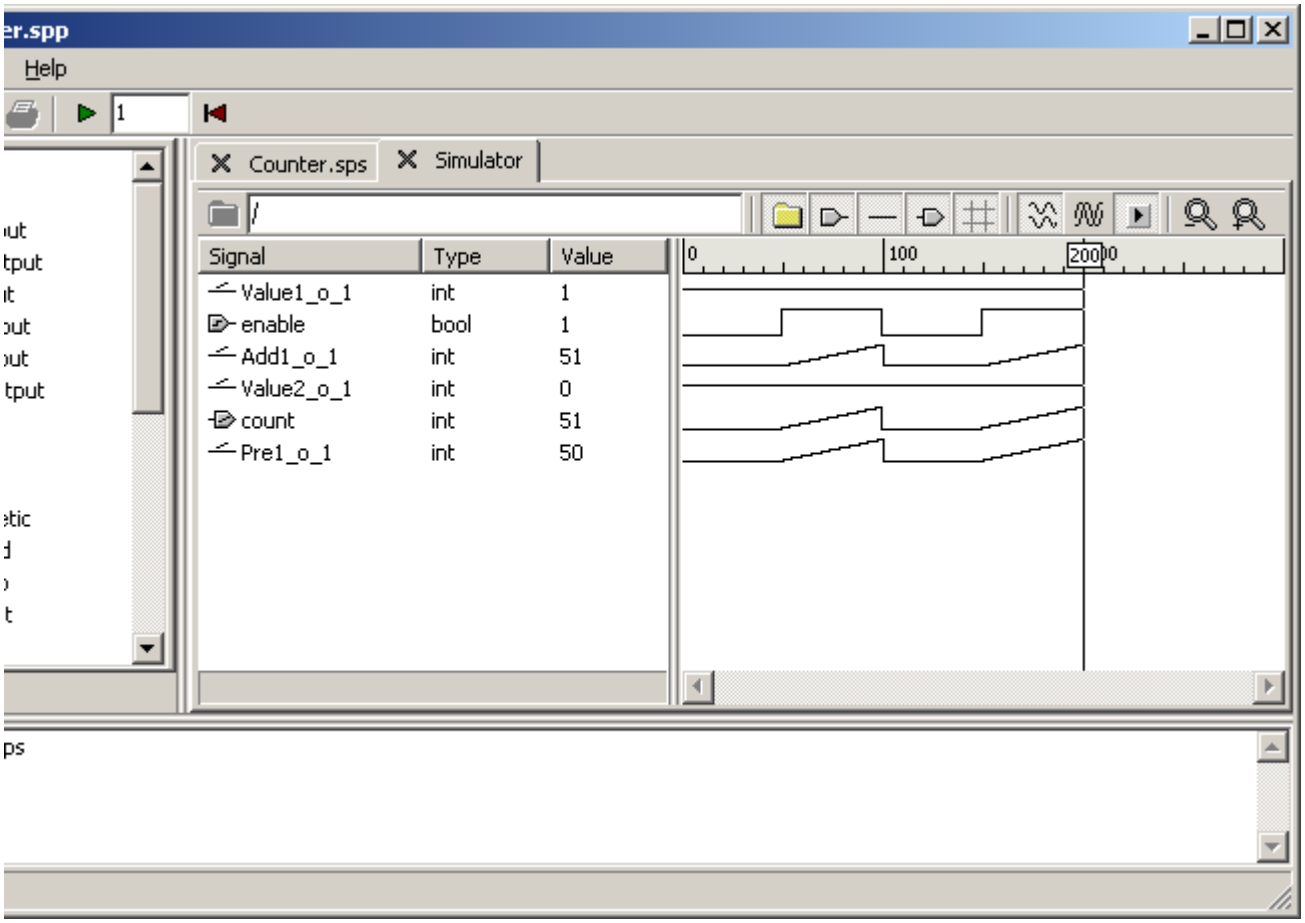
Set up the simulator

Before simulation can start, the schematic needs to be compiled. Moreover, it is necessary to set up a few simulation parameters. In particular, since the project can contain many schematics, it is necessary to specify which schematic to simulate.

- Start simulation by choosing **Project:Simulate**, by clicking the **Simulate** toolbar button or by pressing F9.
- If the schematic has not yet been compiled, it must now be given a name. Enter **Counter** and press Enter.
- The schematic is now automatically compiled. If there are any errors, they will be printed in the message pane in the bottom of the window. Fix the errors and start simulation again. See the Compiler Errors chapter for documentation of error messages.
- When asked to set up simulation, choose **OK**.
- The **Simulation** page of the **Project Settings** dialog appears. In the **Top-Level Node** drop-down list, choose **Counter** as the schematic to simulate. Then click OK. (The **Sample Time** setting is irrelevant for a model that doesn't contain time-dependent blocks like integration, so it can be left at its default setting.)

Simulate with the simulator view

When the simulation has started, the simulator view appears under a new tab in the same area as the schematic editor. The simulator view is divided into two main areas: the *signal area* to the left shows the names and the current values of the signals, and the *graph area* to the right shows the simulated data graphically. As the simulation progresses, the graphs will be plotted towards the right. By default, the graphs are shown individually in rows aligned with their names, but the graphs can also be shown in full size in the graph area by clicking the **Joint View** toolbar button.



- Execute one cycle by clicking the **Simulate** toolbar button or by pressing F9. No values will change because **enable** is 0, but a pixel is added to each graph in the graph area.
- Press and hold F9 to execute more cycles and see the graph being plotted.
- Enter 10 in the simulation time box to the right of the **Simulate** toolbar button and then click the button or press F9 to execute ten cycles at once.

Force a signal

Individual signal values can be forced to any value during the simulation by clicking in the **Value** or the **Force** column. For boolean signals, the value will switch between 0 and 1 when clicked, while for other types, a box appears where a new value can be entered. Forcing is removed by right-clicking the value and choosing **Unforce** in the context menu that appears.

- Click the 0 in the **Value** column for the **enable** signal to force this signal to 1. Notice that **count** also immediately switches to 1, since the value of **count** is dependent on the value of **enable** in the same cycle.
- Click the **Simulate** toolbar button or press F9 a few times to see the value of **count** increase.
- Right-click the 1 and click **Unforce** to remove the forcing.

Move the cursor to examine simulation results

The *cursor* is a vertical line that is always present in the graph area, by default at the right end of the graphs. However, the cursor can be moved by clicking in the graph area or by pressing Ctrl-Left arrow or Ctrl-Right arrow. The values shown in the signal area always reflect the values at the cursor's position.

- Click in the middle of the graph. The cursor will move to this position, and the signal area will show the values at this position.
- Press Ctrl-Left arrow and Ctrl-Right arrow to move the cursor one cycle to the left and to the right, respectively.

View values in the schematic

Simulated signal values can also be viewed in the schematic right by the corresponding connections. As with the signal area, the schematics reflect the values of the cursor, even if the cursor is not explicitly visible.

- Click the Counter.sps tab to show the schematic.
- Click the **Show Values** toolbar button. The current signal values are now shown above the connections.
- Click the **Simulate** toolbar button or press F9 to see the values change.
- Press Ctrl-Left arrow and Ctrl-Right arrow to step backwards and forwards in the simulation result.

Set up a testbench

A common way to perform simulation is to set up a testbench where input data is fed automatically into the model. In RTflow, this can be done by using the model as a block in another schematic and connecting this block to a signal generator.

- Create another schematic.
- Find the **Counter** block in the **User** folder at the bottom of the blocks tree. This block was automatically added when the Counter schematic was compiled in the beginning of this section.
- Drag the **Counter** block into the new schematic.
- Find the block **Clock** under **Standard** and **Generator** in the blocks tree, and drag it into the same schematic. This block produces a boolean signal that switches between 0 and 1 at a given period.
- Connect the **o** port of **Clock1** to the **enable** port of **Counter1**.
- Add a **Value** block with value 100 and connect it to the **t** port of **Clock1**. This sets the period of **Clock1** to 100 cycles.

The above procedure exemplifies how models can be structured into a hierarchy of schematics. A typical model has a top-level schematic containing the model's inputs and outputs and a number of connected user-defined blocks. These blocks have been defined in schematics that may again contain other user-defined blocks. The definition of a block can be opened by double-clicking it, and one can return to the parent schematic by clicking the **Pop Context** toolbar button in the schematic editor.

Simulate with the testbench

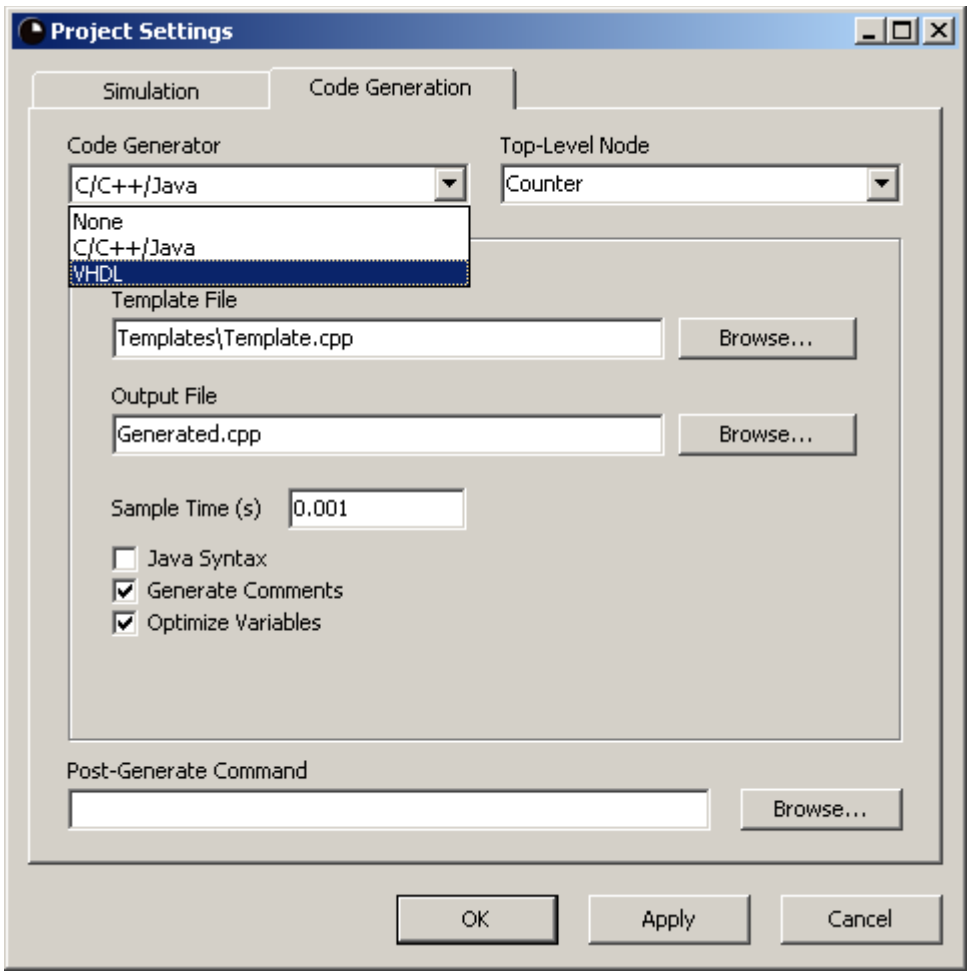
Since the simulator is now set up to simulate the **Counter** schematic, it is necessary to change this setting to simulate the **Testbench** schematic and restart the simulator.

- Compile the new schematic by choosing **Project:Compile** or by pressing F11. Give it the name Testbench.
- Open the project settings dialog by choosing **Project:Settings...**.

- Change the top-level node to **Testbench** and click **OK**.
- Restart the simulator by choosing **Project:Reset Simulator** or by clicking the **Reset Simulator** toolbar button. The simulator view appears, showing the signals of the testbench schematic.
- Double-click the **Counter1** folder to view the signals of the counter model.
- Press and hold F9 to simulate at least 100 cycles to see that **enable** switches automatically every 50th cycle.

The Code Generator (Tutorial)

RTflow can generate C, C++, Java and VHDL code from the model, so that it can be implemented in both software and hardware in a real-world application. First of all, you must supply a code generation template in the target language with special keys (e.g. %OUTPUT_UPDATE_EQUATIONS%) indicating where the generated information should be inserted. The code generator reads the template file and makes a copy of it, but where all keys have been replaced with the corresponding information or code. Default template files for all four target languages are supplied with RTflow and can be found under the Templates folder in the application folder.



Generate C++ code

- First open the file Templates/Template.cpp in any text editor. Notice that it is a valid C++ file with the exception for the code generation keys with their % characters.
- In RTflow, choose **Project:Generate Code** or press F10.
- When asked to set up code generation, choose **OK**.

- The **Code Generation** page of the **Project Settings** dialog appears. In the **Code Generator** drop-down list, choose **C/C++/Java**.
- In the **Top-Level Node** drop-down list, choose **Counter** as the schematic from which code should be generated.
- Click **Browse...** to the right of the **Template File** edit box and choose Templates/Template.cpp in the file selector that appears.
- Select a suitable output folder and enter Generated.cpp in the **Output File** edit box.
- Leave the three remaining checkboxes unchanged and click **OK**.
- Open the generated file Generated.cpp in any text editor and examine the results.

Generate VHDL code

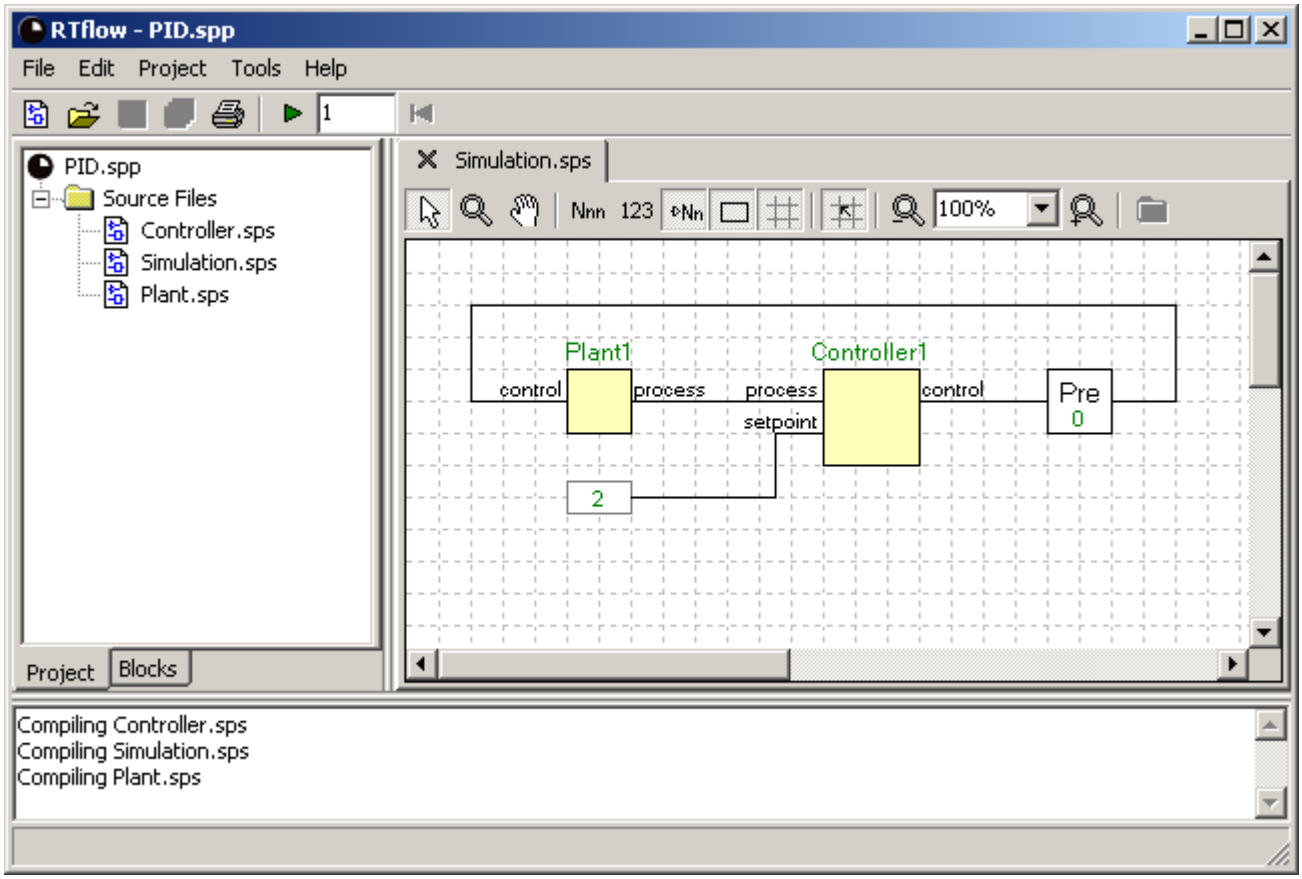
- Now open the file Template/Template.vhd in the text editor. Notice that it is a valid VHDL file with the exception for the code generation keys with their % characters.
- In RTflow, choose **Project:Settings...** to open the Project Settings dialog and click the **Code Generation** tab.
- In the **Code Generator** drop-down list, choose **VHDL**.
- Click **Browse...** to the right of the **Template File** edit box and choose Templates/Template.vhd in the file selector that appears.
- Select a suitable output folder and enter Generated.vhd in the **Output File** edit box.
- Leave the two remaining checkboxes unchanged and click **OK**.
- Choose **Project:Generate Code** or press F10 to generate the code.
- Open the generated file Generated.vhd in any text editor and examine the results.

Tutorial 2: A PID Controller

This tutorial provides a closer look into the simulator by explaining how to use the simulator to tune the parameters of a PID controller. The PID controller and a model of the physical system are already defined in an example project that came with the installation of RTflow, so there is no modeling work involved in this tutorial. Instead, you will iteratively modify the controller parameters and run simulations until a satisfying step response has been achieved. Notice that a basic knowledge of PID controllers is helpful for fully understanding this tutorial.

Open and Examine the Model

- Open the project Examples/PID/PID.spp under the application folder.
- Open the **Simulation.sps** schematic by double-clicking it in the project view in the left part of the window.



This schematic is the top-level schematic for simulation. It shows clearly how the controller (**Controller1**) is connected to a model of the physical system (**Plant1**) in a closed loop. Typically, it is the controller block that is the top-level block for code generation. The physical system model block is only active during simulation, and describes how a real physical system would react to the control value. The physical system model in this example is quite arbitrary (it contains some gain, offset, delay and integration - common elements in a typical system model), and should for the purpose of this tutorial be considered as a black box.

The **Pre** block between the controller and the plant can be seen as a model of the slight electrical delay of the control signal as it flows from the controller to the physical system. More importantly, however, it prevents the closed loop from becoming a *causality loop*, that is, a set of blocks that are connected in a loop with no delay. Causality loops are forbidden in RTflow, because it may be difficult or impossible to compile them into executable code. Since the controller contains data paths without delays, and the physical system is a black box that could potentially also contain data paths without delays, a **Pre** block must be inserted to ensure that there is a delay in the loop.

As can be seen, the setpoint to the controller is 2. Therefore, if the controller is tuned properly, we can expect that the **process** value stabilizes at 2 after some time. The goal of this tutorial is to use the simulator in RTflow to tune the controller further, so that **process** stabilizes more quickly.

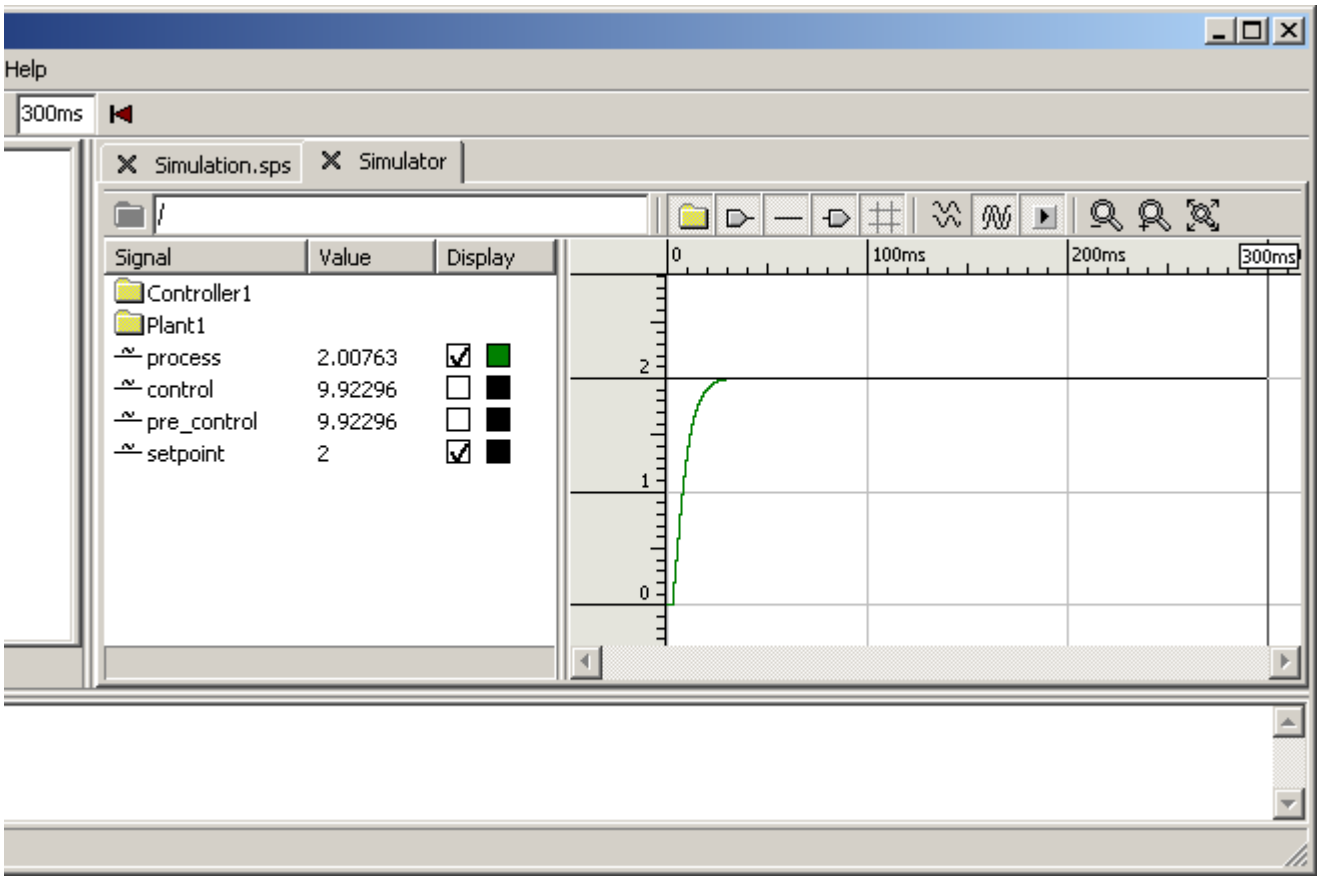
Run a simulation

First we will see how the controller performs with the current parameters.

- Click the **Simulate** toolbar button or press F9 to start the simulator. The simulator is now positioned on the very first cycle, and it can be seen that **process** is at 0 and **setpoint** is at 2. The controller consequently tries to raise the **process** value by issuing a **control** value of 2002.
- In the simulation time box to the right of the **Simulate** toolbar button, enter **300ms** to indicate that we want to run a simulation of 300 milliseconds.

- Click the **Simulate** toolbar button or press F9 to run the simulation for 300 milliseconds. It can be seen that the **process** value has stabilized slightly above 2.

Set Up the Simulator View



The default display settings of the simulator view are not ideal for studying the simulation output. However, there are a number of possibilities for setting up the display.

Switch to joint view

RTflow offers two different views for displaying a number of signals: *Split view*, which shows each graph individually on its own row, and *joint view*, which shows all signals in one common coordinate system. In split view, the graphs will always be vertically scaled automatically to fit into its narrow row. In joint view, the graphs will be automatically scaled so that all graphs fit into the view. However, in joint view, it is possible to zoom vertically.

- Click the **Joint View** button in the simulator's toolbox. The graphs will be plotted in one common area. However, since the **control** value varies between 0 and over 2000, it completely dominates the plot. The **process** value, which we are more interested in, varies between 0 and slightly over 2, so it has been scaled down vertically so that it looks like a horizontal line.

Remove uninteresting graphs

By using the checkboxes by the signals' names in the signal area, it is possible to choose the graphs to be displayed in joint view. A maximum of three graphs can be viewed at the same time. For details, see the The Display Column section. When adding or removing a graph, the graph area will be rescaled so that all of the graphs in the new set fit into the view.

- Remove the graphs of **control** and **pre_control** by unchecking their checkboxes. The graph area will automatically rescale so that the variation of **process** can be seen.
- Add the graph of **setpoint** by checking its checkbox.

Change graph color

Each graph can be given an individual color.

- Click the black square to the right of the checkbox for the **process** signal.
- Select a color in the color picker, for example green.
- Click **OK**.

Expand the graph area

You can freely resize and remove columns in the signal area and resize the graph area to get a better view.

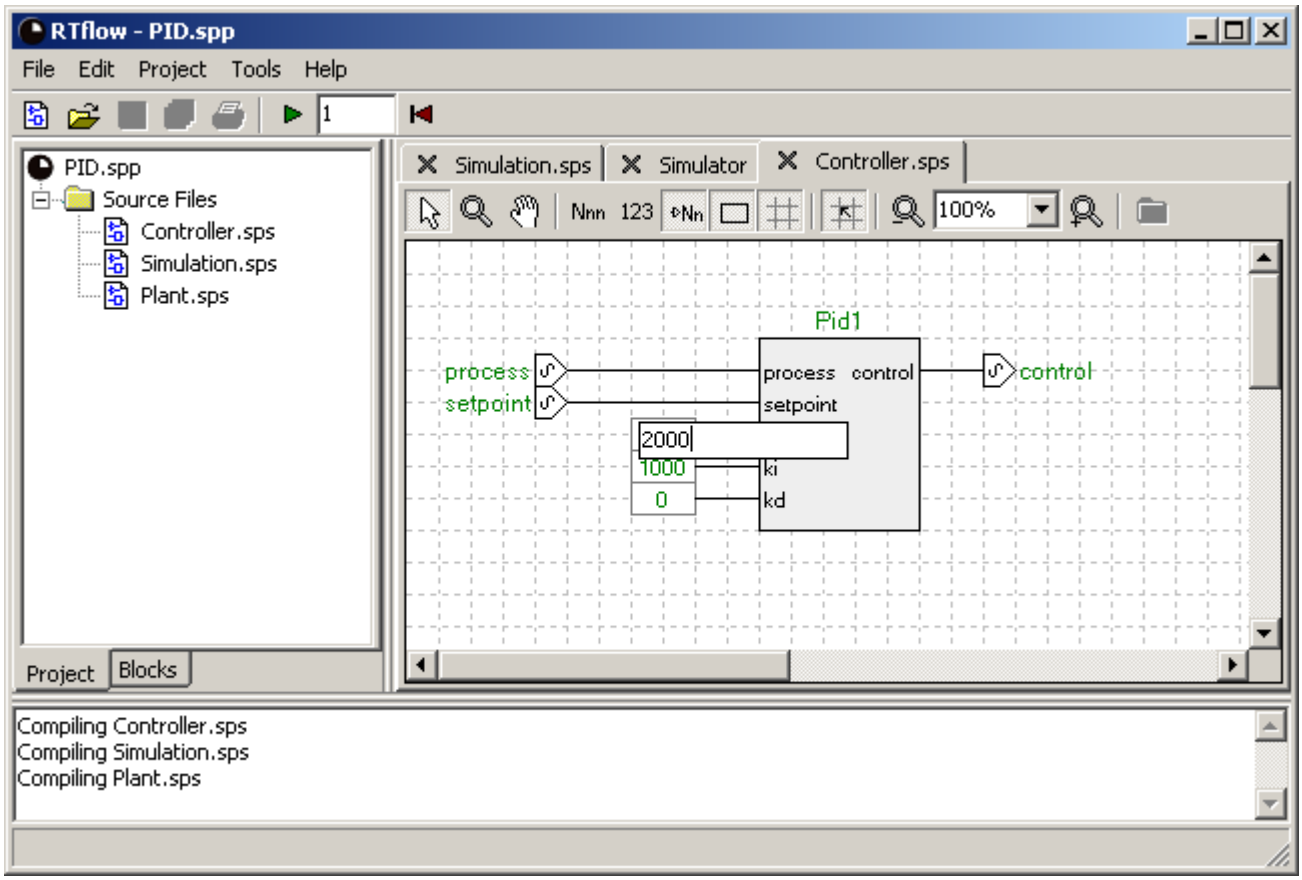
- Remove the Type column by right-clicking on the title bar above the signal area and uncheck Type.
- Remove the Force column in the same manner.
- Reduce the size of the **Signal** column by dragging the separator between the **Signal** column and the **Value** column in the title bar to the left, so that there is just about room to show the signal names.
- Drag the separator between the signal area and the graph area to the left, so that it coincides with the right edge of the Display column.

Tune the Controller

Now that the view is properly set up, we focus our attention on tuning the controller to see if it can perform better. Looking at the results of the simulator, we can see that it takes almost 20 milliseconds to reach a process value over 1.8. This time vill serve as the *step response time*. Moreover, we can see that after 300 milliseconds, the process value is still not exactly 2, but 2.00763, and very slowly decreasing. Our objective by tuning the controller is to reduce the step response time and to ensure that the process value reaches (and stays at) 2 sooner.

- Open the **Controller.sps** schematic by double-clicking it in the project view in the left part of the window.

Modify the PID parameters



The PID parameters k_p, k_i and k_d can be seen as the inputs to the **Pid1** block, and that these values are 1000, 1000 and 0, respectively. To tune the controller, it will suffice to modify these values in this schematic.

- Change the value going into the **kp** input of the **Pid1** block to **2000**.

Rerun the simulation

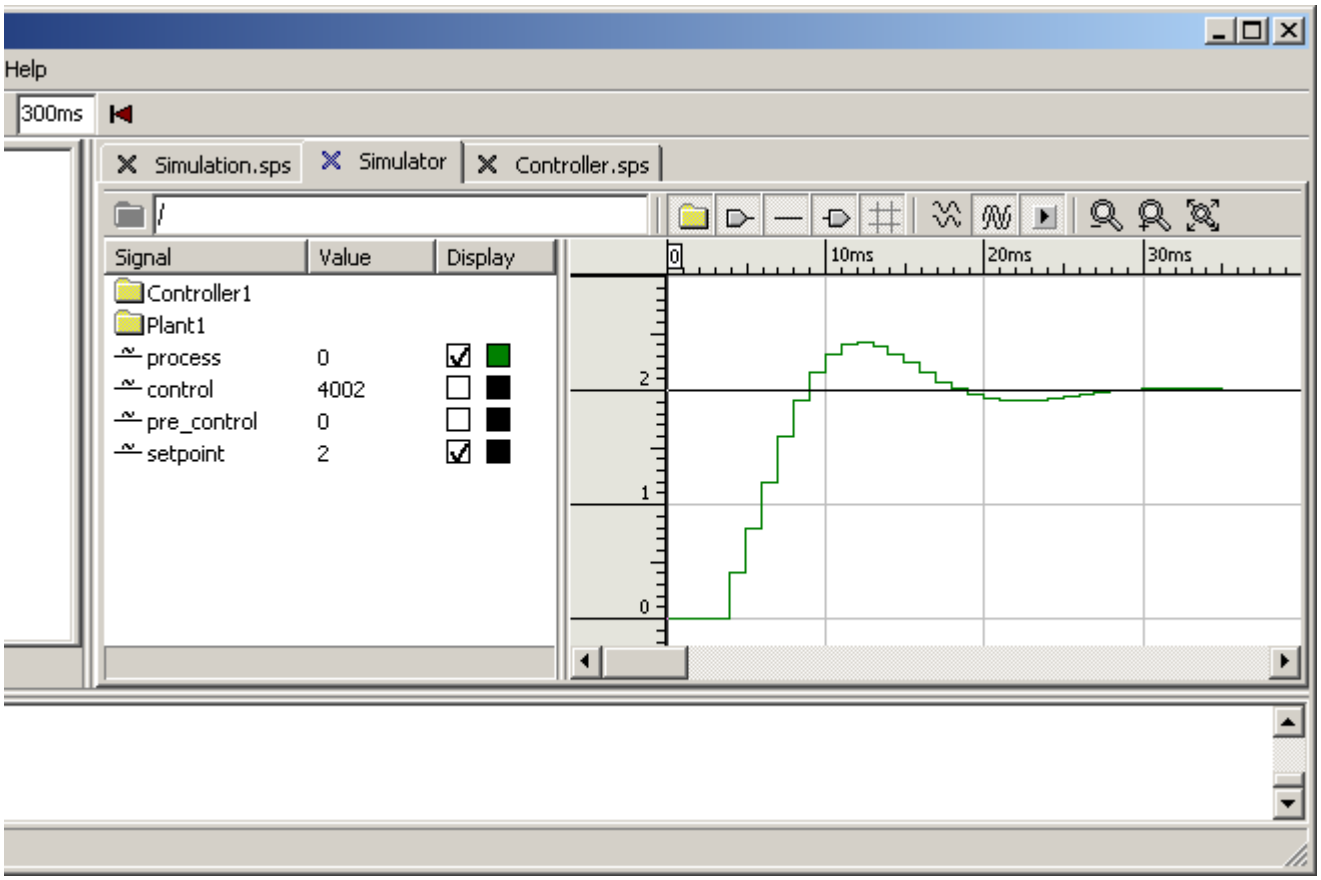
If you change the model while the simulation is still running and then invoke the Simulate/Run command (F9), you will be asked if you want to apply the changes and restart the simulation. If you answer **Yes**, the model will be compiled automatically, and the simulation will restart at time 0. If you answer **No**, the old simulation will continue without your recent changes. It is not possible to continue the old simulation using the new model.

- Click the **Simulate** toolbar button or press F9 to start the simulator.
- When asked whether to apply changes and restart the simulator, choose **Yes**. The simulator will restart and is thereafter again positioned on the first cycle.
- Click the **Simulate** toolbar button or press F9 to run the simulation for 300 milliseconds. It can be seen that the step response time is closer to 10 milliseconds, and that the process value is exactly 2 after 300 milliseconds. However, there is a significant overshoot - the **process** value is up to almost 2.5 before turning down towards 2 again.

Zoom into the graph

There are many possibilities for zooming in and out of the graph, so that you can study parts of it at a higher resolution. The **Zoom In** toolbar button zooms in horizontally around the cursor (the vertical line going through the graph area). The **Zoom Out** toolbar button zooms out horizontally.

By dragging with the mouse inside the graph area, you can create a rectangle into which the graph will zoom when you release the mouse button. Right-clicking in the graph and selecting **Zoom Out Vertically** in the context menu that appears will zoom out so that the complete vertical extension of all graphs is in view. See the Joint View section for details. Finally, clicking the **Restore Zoom** toolbar button zooms out both vertically and horizontally so that one pixel corresponds to one cycle.



- Click and drag in the graph area to create a rectangle around the first 40 milliseconds of the graph. The chosen area will expand to fill the whole graph area.
- Right-click in the graph and select **Zoom Out Vertically** to zoom out vertically.
- Click the **Restore Zoom** toolbar button to zoom out completely.

Fine-tune the controller

Keep fine-tuning the controller until you are satisfied. You should be able to achieve a step response time at about 7 milliseconds, while the overshoot is no more than 10% (that is, 2.2). As a hint, try applying a small amount of derivative gain using the **kd** input of the PID block.

Tutorial 3: A Java Game

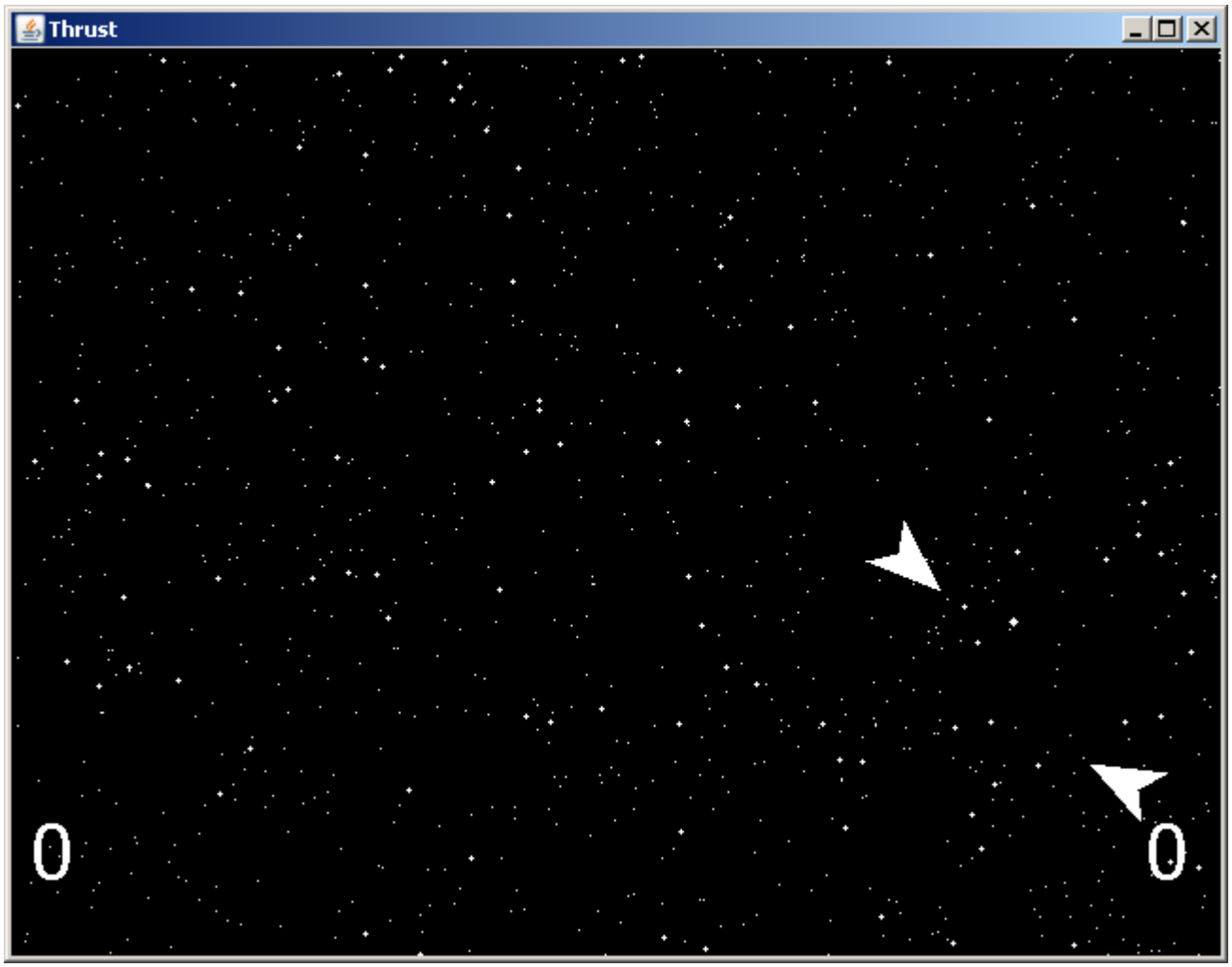
This tutorial shows how RTflow can be used for development of desktop software, in this case a Java game. In this game, you control a spaceship, subject to gravity and physical laws that make it quite challenging to control, and you will try to shoot down an enemy spaceship, controlled by the computer. For the implementation of the game, Java is used for the player interface such as keyboard handling and graphics, while RTflow is used for the physics engine and the artificial intelligence (AI) of the enemy spaceship. The tutorial doesn't present many new features, but it gives you a good and entertaining opportunity to try out some serious development with RTflow.

Set up the Java environment

You don't need a complete Java development environment on your computer to run and develop the game, but you do need the Java development kit (JDK), if you don't already have it. Moreover, it is recommended that you set your **PATH** user variable to point to the JDK bin folder, so that the java compiler can be called without specifying an absolute path.

- If you don't already have it on your computer, download and install JDK from http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- Add the path to the JDK bin folder to your PATH user variable. Typically, the path to the JDK bin folder is something like **C:\Program Files\Java\jdk1.7.0\bin**. In Windows XP, you access the **PATH** user variable through the Control Panel, System, Advanced and Environment Variables. In the dialog that appears, select the **PATH** line under User variables, choose **Edit** and append a semicolon (**;**) and the path to the JDK bin folder. Then click **OK** in each open dialog to close them.
- Check that the environment is properly set up by opening a command prompt and entering **javac**. You should now see instructions for usage of javac.

Open and run the game



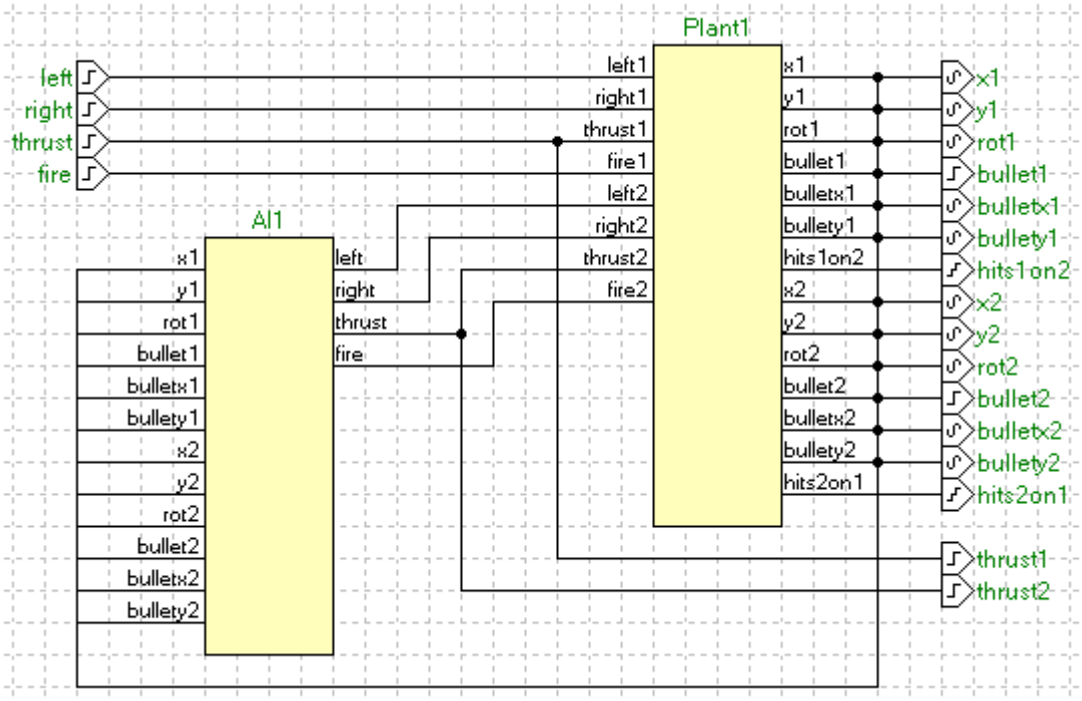
A Java (or any kind of) application can be invoked directly from within RTflow using the *post-generate command* setting in the code generation settings dialog. The post-generate command specifies an executable file, a batch file or a system command to be executed after code

generation. For example, the post-generate command can be a batch file that first compiles the generated Java file and then runs the compiled Java application.

- In RTflow, open the project Examples/Thrust/Thrust.spp under the application folder.
- Choose **Project:Settings...** to open the Project Settings dialog and click the **Code Generation** tab. Notice that the batch file **buildandrun.bat** has been specified under **Post-Generate Command**. This batch file has the following contents:
```
javac -sourcepath src -d classes src\rtflow\*.java src\thrust\*.java
java -classpath classes thrust.Main
```
- Close the dialog without changing anything.
- Choose **Project:Generate Code** or press F10. RTflow will generate Java code from the model, and then the above batch file will be executed. As a result, a window with the running game will appear. Use the left and right cursor keys to turn your ship clockwise and counterclockwise, respectively, and use the down cursor key to thrust. Use the space key to fire. Notice that you can only have one bullet in the air at a time, so you have to wait until it disappears before you can fire again.

Improve the AI

The AI that controls the enemy ship in this example project is very simple, as can bee seen when trying out the game. In this tutorial, your task is to improve and refine the AI to the level that you think is appropriate. To your help, you have the reference sections in the help system.



A good way to get started would be to create a new AI block by copying the inputs and outputs of the existing AI block:

- Open **Main.sps**, the top-level schematic. You can see that the model is divided into two main parts: The physics engine (**Plant1**), which calculates the ships' positions, the score and other outputs given the control actions of the two players, and the AI (**AI1**), which calculates the control actions for ship 2.
- Double-click the **AI1** block to open **AI.sps**.
- Choose **Edit:Select All** or press Ctrl-A to select all elements.
- Choose **Edit:Copy** or press Ctrl-C to copy the elements to the clipboard.
- Choose **File:New Schematic** or press Ctrl-N to create a new schematic.
- Choose **Edit:Paste** or press Ctrl-V to paste the elements into the new schematic.

- Delete all elements except the ports by selecting them and pressing the Del key.
- Choose **Project:Compile** or press F11 to compile the new schematic into a new block. Name the new block appropriately, for example **AI2** or **MyAI**. Don't worry about the compiler errors for now.
- Go back to **Main.sps** by clicking its tab.
- Depending on how you want to test your AI, you now have two options:
    1. If you want to test your AI by playing against it yourself, then you should replace the existing AI with your own. You do this by selecting the **AI1** block, pressing delete, dragging in your own AI block from the blocks tree and recreate the connections.
    2. If you want to test your AI by letting the existing AI play against it, then you should replace the input ports of **Main.sps** with your AI, so that your AI controls ship 1 instead of the player. You do this by selecting the four input ports, pressing delete, dragging in your own AI block from the blocks tree and connect it appropriately. Notice that the AI by convention should consider the inputs with names ending with "1" be the enemy ship information, and inputs with names ending with "2" be its own ship information. Therefore, the "1" outputs of **Plant1** should be connected to the "2" inputs of your AI, and vice versa.

Your own AI block is now ready to be filled out with logic. At any time, you can press F10 to test the game. Notice, however, that the code generator will fail as long as any of the four output ports of your AI schematic is unconnected. At least you will have to connect them to **Value** blocks to be able to start the game.

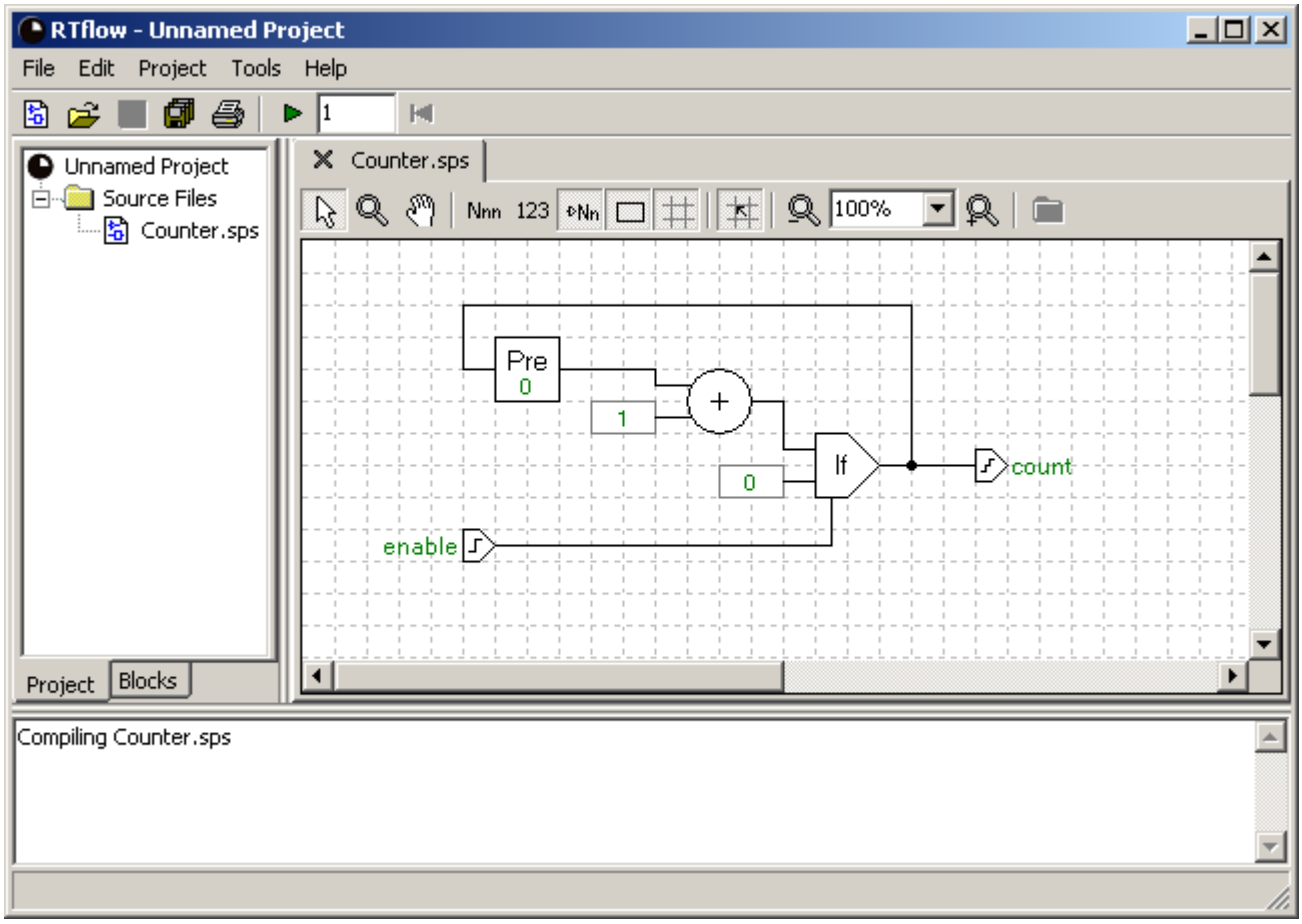
The Main Window

This chapter gives an overview of the main window, followed by a detailed reference of all commands available via the menus, the toolbar buttons, the project tree and the keyboard shortcuts. The sections in this chapter are:

- Main Window - General. Overview of the different elements of the main window.
- The File Menu. Detailed reference of all commands available through the **File** menu of the main window.
- The Edit Menu. Detailed reference of all commands available through the **Edit** menu of the main window.
- The Project Menu. Detailed reference of all commands available through the **Project** menu of the main window.
- The Tools Menu. Detailed reference of all commands available through the **Tools** menu of the main window.
- The Help Menu. Detailed reference of all commands available through the **Help** menu of the main window.
- Main Window Toolbar Buttons. Summarizes the commands available as toolbar buttons in the top of the main window.
- The Project Tree. Describes the structure of the project tree and the commands available in its context menus.
- Main Window Keyboard Shortcuts. Summarizes the keyboard shortcuts of the menu commands.

Main Window - General

The main window consists of five main parts: The menu bar at the top, the toolbar immediately below, the tree view in the left part of the window, the message view in the bottom, and finally the tab area that occupies the main part of the window. Above the message view, and between the tree view and the tab area, there are separators that can be dragged with the mouse to resize or hide the different parts.



The menu bar

The menu bar allows you to invoke commands. These commands are described in detail in the following sections, sorted under what menu they can be found in. Many commands are also available as toolbar buttons and as keyboard shortcuts, and these are summarized in the Main Window Toolbar Buttons and Main Window Keyboard Shortcuts sections.

It should be noticed that some commands are only available in certain states of the application. For example, **Cut** is only available if some kind of editor is open and some elements are selected in this editor.

The toolbar

The commands available as toolbar buttons are listed in the Main Window Toolbar Buttons section.

The tree view

The tree view in the left part of the window can show two different trees depending on what tab has been selected at the bottom of the view. The *project tree* shows the list of source files in the project, and the commands available on the project tree are listed in the The Project Tree section. The *blocks tree* is an integral part of the schematic editor and will be described in the The Blocks Tree section.

The message view

The message view shows textual messages from various functions in RTflow, including the compiler and the simulator. Compiler error messages commonly includes a link to the failing

element, shown in brackets (< and >) and in blue color instead of the ordinary black text. These links can be clicked to open the containing schematic and select the failing element.

The tab area

The *tab area* is the main area of the window where source file editors and the simulator view open. Each editor and the simulator is associated to a tab at the top of the area such as when the tab is clicked, the contents are brought into view. Clicking the cross in a tab closes it. Right-clicking a tab brings up a context menu with the Close and Close All commands. Refer to the help sections for these commands for details.

The File Menu

New Schematic

Creates a new schematic. The new schematic is added to the currently open project, and it is opened for editing under a new tab in the tab area. No file is created on disk; the schematic is only opened in memory, and it obtains a temporary filename until you explicitly save it to disk.

Toolbar button

Keyboard shortcut

Ctrl-N

Availability

Always available.

Open...

Opens an existing file. A file dialog appears, where you select an existing file. If the selected file is a project file (*.spp), then the currently open project and all open tabs will first be closed as with the Close All command, and then the selected project is opened. If the selected file is a schematic file (*.sps), then the schematic will be opened for editing under a new tab in the tab area.

It should be noticed that when opening a schematic, it is not added to the project. As a consequence, if the schematic was not in the project before, then it is not possible to compile the schematic into a block for use in other schematics. To use a block that a schematic file defines, the file must be added to the project using the Add Files... command.

Also notice that when opening a schematic that itself uses blocks that are not in the project, then these block instances and the connections to them will be removed in the schematic editor. When this happens, warning messaged will be written to the message view. Study the warning messages to find out what blocks are missing and add the missing blocks to the project using the Add Files... command. Then close the failing schematic file without saving the changes and open it again.

Toolbar button

Keyboard shortcut

Ctrl-O

Availability

Always available.

Open Recent

Opens a recently used file without going through a file dialog. Instead, the files to choose between appear in a submenu under the **Open Recent** item when it is clicked. See Open... for details on the effect of opening a file.

Availability

Always available.

Save

Saves the active file. The active file is the file currently in view in the tab area. If the file is new and hasn't been saved to disk before, a file dialog appears, where you choose a folder and enter a name for the file. Notice that if the file is part of a project, then it is recommended to save it in the same folder as, or in a subfolder of, the project file. If not, the project file will contain an absolute path, which makes it difficult to move the project to another location or another computer.

Toolbar button

Keyboard shortcut

Ctrl-S

Availability

Available when an unsaved file is in view in the tab area.

Save As...

Saves the active file with a new name. The active file is the file currently in view in the tab area. A file dialog appears, where you choose a folder and enter a name for the file. Notice that if the file is part of a project, then it is recommended to save it in the same folder as, or in a subfolder of, the project file. If not, the project file will contain an absolute path, which makes it difficult to move the project to another location or another computer.

Availability

Available when a schematic file is in view in the tab area.

Save Project As...

Saves the active project with a new name. A file dialog appears, where you choose a folder and enter a name for the project file. It is recommended to save a project file in the same folder as, or in a parent folder of, all the files that it contains. If not, the project file will contain absolute paths, which makes it difficult to move the project to another location or another computer.

Availability

Always available.

Save All

Saves all modified files. Modified files can be the project file or open source files. For all modified files that are new and haven't been saved before, a file dialog appears, where you choose a folder and enter a name for the file.

Toolbar button

Availability

Available if the project or any of the open files have been modified.

Close

Closes the active tab. If the tab is a modified source file, you will be asked whether to save the file to disk before closing. In this case, if the you choose **Cancel**, the file will stay open.

Keyboard shortcut

Ctrl-F4

Availability

Close All

Closes the project and all open tabs. If one or two files are modified, then for each modified file, you will be asked whether to save the file to disk before closing. If three or more files are modified, then the common save dialog appears, where you can choose with a checkbox for each modified file whether to save it or not. The checked files will be saved when you click **OK** in this dialog. If you click **Cancel** in this dialog, the files will stay open.

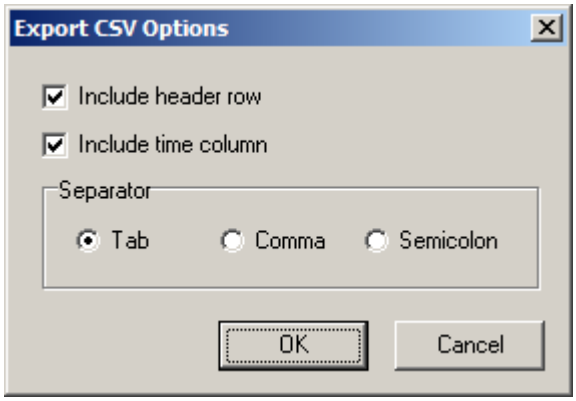
Availability

Always available.

Export CSV...

Exports data produced by the simulator to a CSV file. The file will contain one column for each signal in the signals view and one row for each cycle that has been simulated so far.

First, a file dialog appears, where you choose a folder and enter a name for the file. Thereafter, the **Export CSV Options** dialog appears, where you can choose some options for the format of the file. The file will be produced when you click **OK** in this dialog.



Include header row. When checked, an extra row will be added in the beginning of the file, indicating the names of all signals.

Include time column. When checked, an extra column will be added to the left of the data columns, indicating the cycle number or simulation time in seconds for each row.

Separator. The character to separate the elements with within each row.

Availability

Available when the simulator is active in the tab area.

Print Setup...

Changes the printer settings. A standard print setup dialog appears, where you can set up printer and page layout settings for source file printing.

Availability

Always available.

Print...

Prints the schematic if a schematic file is active in the tab area or the graphs if the simulator is active. A standard print dialog appears, where you to set up print parameters. When you click **OK**, the contents of the active tab is printed.

Toolbar button

Keyboard shortcut

Ctrl-P

Availability

Available when a tab is open in the tab area.

Exit

Quits the application. Before the application quits, all files are closed as with the Close All command. If you choose **Cancel** in any of the save file dialogs that may appear, the exit process aborts and the application will not quit.

Availability

Always available.

The Edit Menu

Undo

Reverts the last editing operation. An *editing operation* is an operation that modifies a source file. Hence, commands like save, compile, and add file can not be undone. Undo never affects the clipboard contents. There is one undo stack for each open source file, so the Undo command will always undo the last action on the currently active file. RTflow supports multi-level undo, which means that a sequence of many editing actions can be undone by using the Undo command repeatedly. The size of the undo stack can be changed with the Options command.

Notice that in the current version of RTflow, undoing a port rename will not properly update other files with references to the port. After undoing a port rename, the schematic should be compiled, and the message view should be inspected for possibly removed connections in other files.

Keyboard shortcut

Ctrl-Z

Availability

Available if the active tab is a source file and this source file has been modified since it was opened. In addition, the editor must be focused; if it is not, click it to focus it.

See also

Redo

Redo

Reverts the last undo. Redo never affects the clipboard contents. Since RTflow supports multi-level undo, one can use the Undo and Redo commands to step backwards and forwards in a sequence of many editing operations.

Keyboard shortcut

Ctrl-Alt-Z

Availability

Available if the active tab is a source file and the Undo command has been used since the last editing operation on the file. In addition, the editor must be focused; if it is not, click it to focus it.

Cut

Cuts the selection and puts it on the clipboard.

Keyboard shortcut

Ctrl-X

Availability

Available if the active tab is a source file and at least one element of the source file is selected. In addition, the editor must be focused; if it is not, click it to focus it.

Copy

Copies the selection and puts it on the clipboard.

Keyboard shortcut

Ctrl-C

Availability

Available if the active tab is a source file and at least one element of the source file is selected. In addition, the editor must be focused; if it is not, click it to focus it.

Paste

Inserts clipboard contents. When pasting schematic elements into a schematic, the elements will by default obtain the same position as the original elements. However, pasted elements will always be shifted downwards and to the right so that they don't cover existing copies of the same element. For example, when copying and pasting into the same schematic, the pasted elements will be positioned below and to the right of the original elements. Subsequent pastes will be inserted further down and to the right.

In order to keep the name of every element unique in a schematic, names of copied elements may be augmented with a number.

Keyboard shortcut

Ctrl-V

Availability

Available if the active tab is a source file and the clipboard contents are compatible with the source file. For schematics, the contents of the clipboard must be schematic elements. In addition, the editor must be focused; if it is not, click it to focus it.

Select All

Selects all elements.

Keyboard shortcut

Ctrl-A

Availability

Available if the active tab is a source file. In addition, the editor must be focused; if it is not, click it to focus it.

Delete

Erases the selection. If a block is deleted in a schematic editor, then all connections to and from the block are also deleted.

Keyboard shortcut

Del

Availability

Available if the active tab is a source file and at least one element of the source file is selected. In addition, the editor must be focused; if it is not, click it to focus it.

Properties...

Changes properties of the selected element. A window appears, where you can edit properties of the selected element. The window is non-modal, which means that you can still interact with the main window while the Properties window is open. If another element is selected while the Properties window is open, the window will change its contents to show the properties for the new element. If no element is selected, the properties for the entire source file will be shown.

The controls in the Properties window are different depending on what type of element is selected. Descriptions of available properties for schematics and schematic elements are given in the Properties section.

Clicking the **OK** button applies the chosen property values to the selected element and closes the dialog. Clicking the **Apply** button applies the chosen property values to the selected element but leaves the dialog open. Finally, clicking the **Cancel** dialog closes the dialog without applying the values.

Keyboard shortcut

Alt-Enter

Availability

Available if the active tab is a source file. In addition, the editor must be focused; if it is not, click it to focus it.

The Project Menu

Add Files...

Adds files to the project. A source file must be added to the project if the block that it defines is to be used in other schematics. When this command is invoked, a file dialog appears, where you select files to be added. When you click **OK** in the file dialog, the selected files are added to the project tree, and they are also compiled, so the blocks that they define appear in the blocks tree.

Keyboard shortcut

Ctrl-+

Availability

Always available.

Remove File...

Removes the active file from the project. Notice that when a file is removed from a project, the block that it defines is also removed. As a consequence, if there are instances of the removed block in another file in the project, then those instances will also be removed, and the project will be invalid. Therefore, it is important to ensure that the block to remove is not used anywhere before proceeding with this command. This can be checked with the Find Usages command.

The file will not be removed from disk, and it will not be closed.

Availability

Available if the active tab is a source file in the project.

Compile

Compiles the active file. The main reason for using the Compile command is to create or update a block in the blocks tree. There is no reason to use the Compile command for simulation or code generation, since all relevant files will be compiled automatically in those functions.

First, if the file is a schematic file that hasn't been compiled before, then you will be asked for a name for the block that will be created. This name must adhere to the following rules:

- It must start with a letter, and the other characters can only be letters, numbers, or the underscore character (_).
- It must not be the name of a block (primitive, standard or user) that already exists in the blocks tree. Names are case-sensitive, but it is recommended to avoid having names that are different only by letter case.

Next, the file is checked for syntactic and semantic errors. If errors are found, they will be output to the message view. Some error messages include a link that can be clicked to have the failing

element selected in the source file. Refer to the Compiler Errors section for information about error messages.

If no errors are found, a block is created and added to the blocks tree under the **User** folder. If the file had already been compiled to a block before, then that block is replaced with the new one.

Finally, all open source files are scanned for instances of the block to ensure that they match the new definition. Hence, if the interface of the block was changed (that is, inputs or outputs were added or removed), then all existing instances of the block in open source files will be updated. In addition, connections to removed inputs or outputs will be removed, and for each removed connection, a message will be printed into the message view.

As an example of the above situation, consider two schematics A and B, where the block defined by B is used in A. In A, all the inputs and outputs are connected. Now, if you remove an input port **i** from the schematic B and then compile B, then the instance of block B in schematic A will be updated accordingly. Hence, the port **i** will be removed from that instance, and so will the connection to it.

Notice that no files will be changed on disk as a result of the Compile command. If connections are removed from a schematic as in the above example, then those changes take place only in memory. Hence, if you close the modified schematic without saving, then the connections will still be present in the file. Similarly, files that are not open will not be modified. However, as soon as a schematic file is opened, its block instances will be updated to match the current blocks.

Keyboard shortcut

F11

Availability

Available if the active tab is a source file in the project.

Simulate

Starts or steps forward the simulator. If the simulation is not already started (that is, the simulator view is not open), then the simulation startup procedure is initiated. The first step in the startup procedure is to check whether the simulation settings have been set. If not, you are prompted to do so, and if you accept, the Project Settings dialog is opened with the Simulation page in front. Refer to the Simulation Settings section for a description of available settings. If you don't accept to set up simulation settings, or if you click **Cancel** in the Project Settings dialog, simulation is aborted.

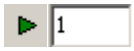
Next, all modified source files are compiled as necessary, and then a thorough semantic analysis is performed on the whole model. If errors are found, they will be output to the message view. Some error messages include a link that can be clicked to have the failing element selected in the source file. Refer to the Compiler Errors section for information about error messages.

If no errors are found, simulation is started at cycle 0 and the simulator view is opened. Refer to the The Simulator chapter for information on how to use the simulator view.

If the simulation was already started when the Simulate command is invoked, it is first checked whether any source file or the project settings have been modified since the simulation was started. If so, and you accept to restart the simulation in the dialog that appears, the simulation is reset as described in the Reset Simulator section. Otherwise, the existing simulation is stepped forward with the amount of time specified in the *simulation time box* in the toolbar. The simulation time box is located to the right of the **Simulate** toolbar button. Simulation time can be specified as number of cycles or, if appended with the time units "s", "ms" or "us", as real time. For example, entering **10** in the simulation time box will make the simulation step forward by 10

cycles each time the Simulate command is invoked, while entering **10 ms** will make it step forward by 10 milliseconds. The real time corresponding to one cycle can be set up in the Simulation Settings.

Toolbar button



Keyboard shortcut

F9

Availability

Available if there is at least one source file in the project.

Reset Simulator

Resets the simulator. If any source file has been changed since the simulation was started, then the whole model is rebuilt as described in the Simulate section. If no errors are found, all simulated data is erased and simulation is restarted at cycle 0.

If there were forced signals in the simulator before the simulator is reset, then you are asked whether these force settings should be kept. If you reply **Yes**, these force settings are immediately applied in the new simulation. If you reply **No**, the force settings are discarded.

Toolbar button

Availability

Available if the simulator is started.

Generate Code

Generates implementation code. The first step in the code generation procedure is to check whether the code generation settings have been set. If not, you are prompted to do so, and if you accept, the Project Settings dialog is opened with the Code Generation page in front. Refer to the Code Generation Settings section for a description of available settings. If you don't accept to set up code generation settings, or if you click cancel in the Project Settings dialog, code generation is aborted.

Next, all modified source files are compiled as necessary, and then a thorough semantic analysis is performed on the whole model. If errors are found, they will be output to the message view. Some error messages include a link that can be clicked to have the failing element selected in the source file. Refer to the Compiler Errors chapter for information about error messages.

If no errors are found, code generation is performed. Refer to the The Code Generator chapter for information on how the code generator works. Finally, if a post-generate command has been specified in the Project Settings dialog, then this command will be executed.

Keyboard shortcut

F10

Availability

Available if there is at least one source file in the project.

Find Usages

Finds usages of the block defined by the currently open source file in other source files. The result is output in the message view. First, the number of usages is provided. Then, links to all usages of the block are listed.

Keyboard shortcut

Ctrl-U

Availability

Available if the active tab is a source file in the project.

Find Unused Blocks

Finds blocks in the project that are not used in any source files. Links to all source files that define unused blocks are listed in the message view. Notice that the top-level schematic is considered unused since it, by definition, is not used in any other source, although it may be used for simulation or code generation.

Availability

Available if there is at least one source file in the project.

Settings...

Changes settings for the project. The Project Settings dialog appears, where you can set up settings related to the project. For descriptions of settings under the **Simulation** tab, refer to the Simulation Settings section. For settings under the **Code Generation** tab, refer to the Code Generation Settings section.

Clicking the **OK** button applies the chosen settings to the project and closes the dialog. Clicking the **Apply** button applies the chosen settings to the project but leaves the dialog open. Finally, clicking the **Cancel** button closes the dialog without applying the settings.

Notice that project settings are saved with the project file. Hence, if you change a project setting, the project file will be considered modified, and you will be prompted to save the project file if trying to close it without having saved it first.

Availability

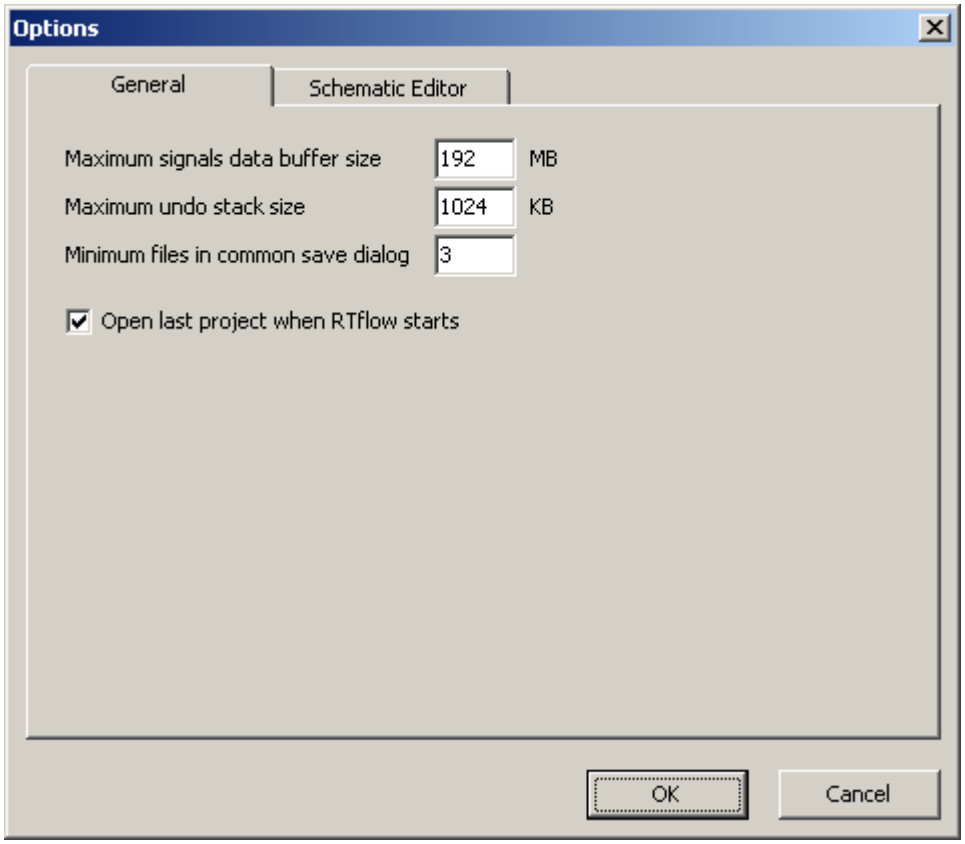
Always available.

The Tools Menu

Options...

Changes user options for the environment. The Options dialog appears, where you can set up options related to the RTflow application.

General page



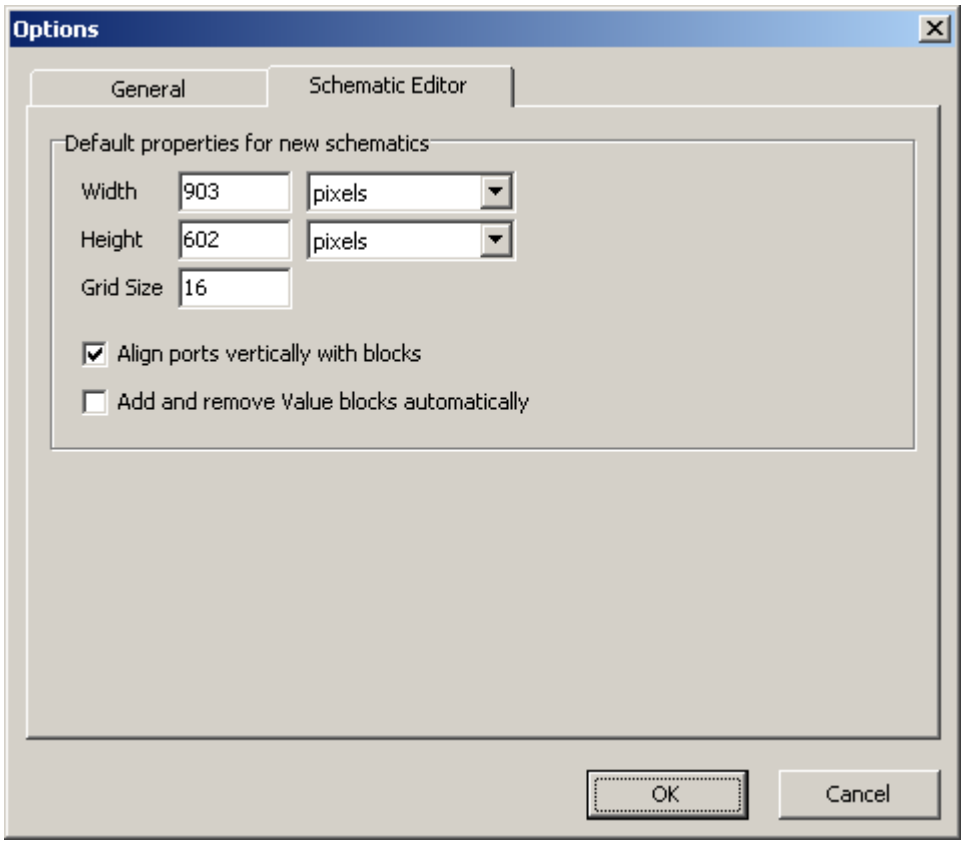
Maximum signals data buffer size. The size in megabytes that the simulator data buffer is allowed to grow to. The simulator will halt when it needs more memory than this to store the simulation result. Increase this value to allow the simulator to run further.

Maximum undo stack size. The size in kilobytes of the editor undo stack. Increase this value to be able to undo more editing opererediting operations.

Minimum files in common save dialog. The minimum number of files that must be modified if the common save dialog (described in the Close All section) is to be used, rather than prompting to save each modified individually, when the project is closed. Increase this value to avoid getting the common save dialog when only a few files have been modified.

Open last project when RTflow starts. When checked, RTflow will open the last opened project on startup.

Schematic Editor page



Default properties for new schematics. These options determine the schematic properties for all new schematics that are created with the New Schematic command. Refer to the Schematic Properties section for information about these properties.

Availability

Always available.

The Help Menu

Help

Opens help.

Keyboard shortcut

F1

Availability

Always available.

About RTflow...

Shows version information.

Availability

Always available.

Main Window Toolbar Buttons

Some commands are available as toolbar buttons in the toolbar immediately under the menu bar. These commands are summarized in the following table.

Toolbar Button	Command	Description
	New Schematic	Creates a new schematic.
	Open...	Opens an existing file.
	Save...	Saves the active file.
	Save All	Saves all modified files.
	Print...	Prints the active file.
	Simulate	Starts or steps forward the simulator.
	Simulation Time Box	The number of cycles, or the time, to simulate.
	Reset Simulator	Resets the simulator.

The Project Tree

The project tree displays the structure and the members of the project. The top-most node represents the project itself, and therefore it carries the name of the project file. The project node is followed by the **Source Files** folder that contains one node for each source file in the project. Double-clicking a source file node opens that file for editing. Moreover, right-clicking a node may bring up a context menu with commands that affect the file associated with the node.

Project node context menu

Right-clicking the project node brings up a context menu with the following commands:

Command	Description
Add Files...	Adds files to the project.
Simulate	Starts or steps forward the simulator.

Generate Code	Generates implementation code.
Settings...	Changes settings for the project.

Source file node context menu

Right-clicking a source file node brings up a context menu with the commands summarized in the following table. Notice that the commands in this menu affect the associated source file, and not the active file, as stated in the detailed reference for the commands. As a consequence, the commands in this menu are always available (that is, never disabled).

Command	Description
Open	Opens the source file.
Remove	Removes the source file from the project.
Compile	Compiles the source file.

Main Window Keyboard Shortcuts

Some commands are available as keyboard shortcuts. These commands are summarized in the following table.

If the schematic editor is open, additional keyboard shortcuts for editing operations are available. These shortcuts are listed in the Schematic Editor Keyboard Shortcuts section.

Shortcut	Command	Description
Ctrl-N	New Schematic	Creates a new schematic.
Ctrl-O	Open...	Opens an existing file.
Ctrl-S	Save	Saves the active file.
Ctrl-F4	Close	Closes the active tab.
Ctrl-P	Print...	Prints the active file.
Ctrl-Z	Undo	Reverts the last editing operation.
Ctrl-Alt-Z	Redo	Reverts the last undo.
Ctrl-X	Cut	Cuts the selection and puts it on the clipboard.
Ctrl-C	Copy	Copies the selection and puts it on the clipboard.
Ctrl-V	Paste	Inserts clipboard contents.
Ctrl-A	Select All	Selects all elements.
Alt-Enter	Properties...	Changes properties of the selected element.
Ctrl-+	Add Files...	Adds files to the project.

F11	Compile	Compiles the active file.
F9	Simulate	Starts or steps forward the simulator.
F10	Generate code	Generates implementation code.
Ctrl-U	Find Usages	Finds usages of the active file.
F1	Help	Opens help.

The Schematic Editor

This chapter provides detailed documentation of schematics and the schematics editor. The sections in this chapter are:

- Schematic Editor - General. Definitions and instructions on how to open the schematic editor.
- Editing Operations. Describes the basic editing operations available by clicking and dragging.
- Schematic Editor Toolbar Buttons. Describes the effect of using the toolbar buttons in the schematic editor.
- Schematic Editor Keyboard Shortcuts. Summarizes the keyboard shortcuts in the schematic editor.
- Properties. Describes the properties of schematics and schematic elements, available for modification in the properties window.
- The Blocks Tree. Describes the structure of the blocks tree and the commands available in its context menu.

Schematic Editor - General

A *schematic* is a sheet containing blocks, connections and other graphical elements that define the behavior of a system model or a sub-component of the system model. In RTflow version 1.1, schematics are the only kind of source files, so the model is always completely defined by a set of schematic files. Schematic files have the extension **.sps**.

Schematics open in a *schematic editor*, which appears under a new tab in the tab area. A schematic file can be created or opened for editing in several ways:

- A new, empty schematic is created by selecting **New Schematic** in the **File** menu, by clicking the **New Schematic** toolbar button or by pressing Ctrl-N.
- A schematic in the project can be opened by double-clicking its name in the project tree, or by right-clicking the name and selecting **Open** in the menu that appears.
- An existing schematic file can be opened by selecting **Open...** in the **File** menu, by clicking the **Open** toolbar button or by pressing Ctrl-O and then selecting it in the file dialog that appears.
- An existing schematic file can also be opened by dragging it from the Windows desktop into the RTflow main window.
- A schematic defining a non-primitive block can be opened by right-clicking the block in the blocks tree and selecting **Edit Source** in the menu that appears.
- A schematic defining a non-primitive block can also be opened by double-clicking an instance of it in another schematic.

Editing Operations

This section describes basic editing operations available by clicking and dragging. These operations are:

- Adding a block
- Adding a connection
- Resizing a block
- Moving a connection
- Renaming an element
- Selecting elements
- Moving a block

- Cutting, copying, pasting and deleting elements

Adding a Block

A block is added to the active schematic by dragging the block from the blocks tree to the schematic with the mouse. The blocks tree is opened by clicking the **Blocks** tab in the tree view. Read more about the blocks tree in the The Blocks Tree section.

A block can also be added by right-clicking the block in the blocks tree and selecting **Add to Schematic** in the context menu that appears, or by selecting the block in the blocks tree and pressing the Enter key.

When the new block instance is added, it will be given an instance name that is unique in the schematic. The instance name will be the block name with a number appended on it. For standard and user blocks, the name is displayed with green text above the block instance, and it can be changed as described in the Renaming an Element section.

In RTflow, it is not possible to drag empty blocks from the blocks tree into the schematic, as is possible in some similar software. Instead, create a new schematic, add ports to that schematic and compile it. Then the new block will be available in the blocks tree.

Adding a Connection

A connection, which is a line connecting two port instances in the schematic, is added by first clicking on the first port instance and then on the second port instance, or by dragging with the mouse from the first to the second port instance.

Since port instances are rather small objects in a schematic at 100% zoom, it may be difficult to find the accurate mouse cursor position where one should click to start the connection. To aid in finding this position, a blue square is shown around the port instance when the mouse cursor is over it, and the name of the port is shown beside it.

After having clicked a port instance, the schematic editor is in *connect mode*, which means that a line is continuously drawn between the port instance and the mouse cursor, and that if you click another port instance, a connection will be created between the two port instances. If you click somewhere else in the schematic when in connect mode, the connection will be cancelled. It is not possible to create open-ended connections in RTflow.

Connections are only allowed between an output port and an input port, and the two ports must have compatible types. If you try to connect two incompatible port instances, for example two input ports, then an error message will appear in red text by the mouse cursor. To resolve type incompatibility problems, it may be necessary to insert a type conversion block - see the documentation for the Bool, Int and Real blocks.

An output port instance can be attached to many connections. If there are two or more connections emerging from the same output port instance, then there will be one or more points, called *connection points*, where the paths of two connections split. These points are marked in the schematic with a black dot. The set of connections emerging from the same port may appear as one single connection net, but it is important to remember that these are actually all individual connections. For example, if one of those connections is selected and deleted, the other connections will not be affected.

An input port instance can be attached to only one connection. If a connection is made to an input port instance that is already attached to a connection, then the previous connection will be deleted.

When the new connection is added, it will be given a connection name that is unique in the schematic. By default, the name is hidden, but it can be made visible by clicking the Show

Connection Names toolbar button or by pressing N. See the Renaming an Element section for details on how to change the connection name and on the rules governing what names are allowed.

Resizing a Block

A block is resized by dragging its bottom-right corner with the mouse. The correct mouse cursor position for resizing is indicated by that the cursor is changed from the ordinary arrow to a small double-ended diagonal arrow.

Some blocks, including most primitive blocks, can not be resized.

Moving a Connection

An interior segment of a connection can be moved by dragging it with the mouse in a direction that is perpendicular to the segment. A *segment* of a connection is a straight part of a connection and can be either horizontal or vertical. An *interior segment* is a segment that is not attached to a block instance. Notice that only interior segments can be moved, since segments that are attached to a block have their positions fixed by the positions of the port instances.

When the mouse cursor is over a segment that can be moved, it will change from the ordinary arrow to a small double-ended arrow that is either horizontal (for vertical segments, indicating it can be moved horizontally) or vertical (for horizontal segments, indicating it can be moved vertically).

It is not possible to add or remove segments to the connection. The schematic editor will always minimize the number of segments of a connection, given the positions of its port instances and the directions in which they emerge from the port instances.

Renaming an Element

There are two different ways to rename an element:

- By clicking the element name, which appears in green text in the schematic, editing the name in the text box that appears and pressing the Enter key.
- By selecting the element, opening the Properties dialog and changing the name there.

By default, connection names are hidden, but they can be made visible by clicking the Show Connection Names toolbar button or by pressing N. In addition, some block instance names are also hidden, but it can be made visible by selecting the element, opening the Properties dialog and changing the **Display** settings.

Schematic element names (both block instance and connection names) must adhere to the following rules:

- It must start with a letter, and the other characters can only be letters, numbers, or the underscore character (_).
- It must not be the name of another element in the schematic, except for in special cases specified below. Names are case-sensitive, but it is recommended to avoid having names that are different only by letter case.

In addition, connection names must adhere to the following rules:

- The names of all connections emerging from the same output port instance must have the same names. In other words, the connections emerging from an output port instance has one name in common.

- The name of a connection that is attached to an input or output port block must have the same name as the port. It is not possible to change the names of such connections directly. Instead, the name of the port block must be changed.

Renaming an input or output port may affect other source files that use the block defined by the edited schematic. In this case, a dialog will appear, listing all affected source files. Clicking OK will perform the name change in the schematic editor and update the references to the port in the listed source files. Open source files will be modified in the editor, while files that are not open will be modified directly on disk. For example, consider two schematics Main.sps and SubFunction.sps, where Main uses SubFunction. If a port name is changed in SubFunction.sps, a dialog will appear, informing that the reference to the port in Main.sps will be modified.

Selecting Elements

Schematic elements can be selected in several ways:

- One single element can be selected by clicking it. Elements that were selected before will be unselected.
- Several elements that are close to each other can be selected by dragging a rectangle with the mouse. All elements that are completely inside the rectangle will be selected, while other elements will be unselected.
- One single element can be selected or unselected without affecting the other elements by clicking it while holding the Ctrl key down.
- All elements in the schematic can be selected by using the Select All command or by pressing Ctrl-A.

Moving a Block

A block can be moved by dragging it with the mouse. Several blocks can be moved by first selecting them and then dragging one of them with the mouse. Connections attached to the moved blocks will also be moved.

If the **Align ports vertically with blocks** property is checked in the Schematic Properties, port blocks can not be moved vertically unless the block it is connected to is also moved.

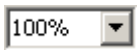
Cutting, Copying, Pasting and Deleting Elements

Cutting, copying, pasting and deleting blocks and connections are performed by first selecting the elements and then invoking the command, either from the **Edit** menu or by using the command's shortcut. For details, refer to the Cut, Copy, Paste and Delete commands.

Schematic Editor Toolbar Buttons

The schematic editor has a toolbar, situated above the schematic and below the tabs, where you can change functionality and display settings of the schematic editor. The settings apply to all schematic editors and are saved with the user options. The toolbar buttons are summarized in the following table, and the commands that they invoke are described in detail in the subsequent subsections.

Toolbar Button	Command	Description
	Select Tool	Enables the select tool.

	Zoom Tool	Enables the zoom tool.
	Scroll Tool	Enabled the scroll tool.
	Show Connection Names	Shows or hides all connection names.
	Show Values	Shows or hides the simulation values of all signals.
	Show Port Names	Shows or hides port names.
	Show Frame	Shows or hides the frame.
	Show Grid	Shows or hides the grid.
	Snap to Grid	Enables or disables snapping of schematic elements to the grid.
	Zoom Out	Decreases the zoom level.
	Zoom Level	Sets the zoom level.
	Zoom In	Increases the zoom level.
	Pop Context	Opens the source file from which this schematic was opened.

Select Tool

Enables the select tool. When the select tool is enabled, which it is by default, you can use the mouse to select, add, move and resize elements. The select tool is one of three available tools, of which exactly one is enabled at a time.

Toolbar button

Keyboard shortcut

S

Availability

Always available.

Zoom Tool

Enables the zoom tool. When the zoom tool is enabled, you can zoom onto a specific point by clicking with the mouse. This is equal to the Zoom In command, but the zoom in point is the point under the cursor, rather than the point in the middle of the view. In the same way, clicking while

holding the Ctrl key down zooms out from the point under the cursor. Moreover, you can drag a rectangle over the schematic to zoom such that the area inside the rectangle will cover the view.

Toolbar button

Keyboard shortcut

Z

Availability

Always available.

Scroll Tool

Enables the scroll tool. When the scroll tool is enabled, you can scroll the schematic by dragging it with the mouse. The scroll tool is one of three available tools, of which exactly one is enabled at a time.

Toolbar button

Keyboard shortcut

R

Availability

Always available.

Show Connection Names

Shows or hides all connection names. When the button is down, connection names are shown above all connections. When the button is up, connection names are not shown.

Toolbar button

Keyboard shortcut

N

Availability

Always available.

Show Values

Shows or hides the simulation values of all signals. When the button is down, signal values calculated by the simulator are shown by their corresponding connections. When the button is up, signal values are not shown.

Notice that the simulator must be started for signal values to be sensible. Moreover, you should ensure that the schematic shows the values of the correct instance. This is accomplished by opening the top-level schematic and then double-clicking block instances until the wanted block instance has been reached. The path to the instance will then be shown in the toolbar, to the right of the **Pop Context** toolbar button, and only then the displayed signal values can be trusted.

Toolbar button

Keyboard shortcut

V

Availability

Always available.

Show Port Names

Shows or hides port names. When the button is down, port names are shown by each port instance in the schematic, but only on block instances that are set to display port names. The setting for whether a block instance displays port names can be found in the Block Instance Properties. When the button is up, port names are not shown.

Toolbar button

Keyboard shortcut

P

Availability

Always available.

Show Frame

Shows or hides the frame. When the button is down, the frame of the schematic is shown. When the button is up, the frame is not shown. See the Schematic Properties button for details on frames.

Toolbar button

Keyboard shortcut

F

Availability

Always available.

Show Grid (Schematic Editor)

Shows or hides the grid. When the button is down, a light gray grid is shown on the schematic sheet, so that schematic elements can be aligned more easily. When the button is up, the grid is not shown.

Toolbar button

Keyboard shortcut

G

Availability

Always available.

Snap to Grid

Enables or disables snapping of schematic elements to the grid. When the button is down, all elements will automatically be snapped to the grid when they are added, moved or resized. Block instances will be moved so that the upper left corner of the block will coincide with a corner of a square in the grid. Connection segments will be moved to the nearest grid line. No elements are moved when this command is invoked; grid snapping only has effect when operating on an element.

Toolbar button

Keyboard shortcut

T

Availability

Always available.

Zoom Out (Schematic Editor)

Decreases the zoom level. Zoom level will be set to the preceding zoom factor in the list of predefined zoom factors: 25%, 50%, 75%, 100%, 125%, 150%, 200%, 300% and 400%. For example, if the current zoom level is 180%, pushing the **Zoom Out** button will set the zoom level to 150%. If possible, the schematic will be scrolled in the view such that the point in the center of the view before zooming out is also in the center of the view afterwards.

Toolbar button

Keyboard shortcut

F2

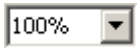
Availability

Available if the current zoom level is greater than 25%.

Zoom Level

Sets the zoom level. A zoom factor can be chosen in the drop-down listbox, or it can be entered by typing in the boxed, followed by an Enter key press. The minimum allowed zoom level is 25% and the maximum allowed zoom level is 400%. If possible, the schematic will be scrolled in the view such that the point in the center of the view before zooming is also in the center of the view afterwards.

Toolbar button



Availability

Always available.

Zoom In (Schematic Editor)

Increases the zoom level. Zoom level will be set to the succeeding zoom factor in the list of predefined zoom factors: 25%, 50%, 75%, 100%, 125%, 150%, 200%, 300% and 400%. For example, if the current zoom level is 180%, pushing the **Zoom In** button will set the zoom level to 200%. The schematic will be scrolled in the view such that the point in the center of the view before zooming is also in the center of the view afterwards.

Toolbar button

Keyboard shortcut

F3

Availability

Available if the current zoom level is less than 400%.

Pop Context (Schematic Editor)

Opens the source file from which this schematic was opened. This command is only available if the currently active schematic was opened by double-clicking a block instance in another schematic. In that case, this command opens that other schematic. In effect, if this schematic has been reached by double-clicking block instances several times to walk down in the model hierarchy, this command is used to walk back, upwards in the model hierarchy.

Toolbar button

Keyboard shortcut

X

Availability

Available if this schematic was opened by double-clicking a block instance in another schematic.

Schematic Editor Keyboard Shortcuts

All toolbar buttons have associated keyboard shortcuts. These shortcuts are summarized in the following table.

Shortcut	Command	Description
S	Select Tool	Enables the select tool.
Z	Zoom Tool	Enables the zoom tool.
R	Scroll Tool	Enabled the scroll tool.
N	Show Connection Names	Shows or hides all connection names.
V	Show Values	Shows or hides the simulation values of all signals.
P	Show Port Names	Shows or hides port names.
F	Show Frame	Shows or hides the frame.
G	Show Grid (Schematic Editor)	Shows or hides the grid.
T	Snap to Grid	Enables or disables snapping of schematic elements to the grid.
F2	Zoom Out	Decreases the zoom level.
F3	Zoom In	Increases the zoom level.

X	Pop Context	Opens the source file from which this schematic was opened.

Properties

Schematics and schematic elements have properties that can be edited in the Properties window. The Properties window opens when the Properties... command is invoked. The contents of the window depends on what element is selected as follows:

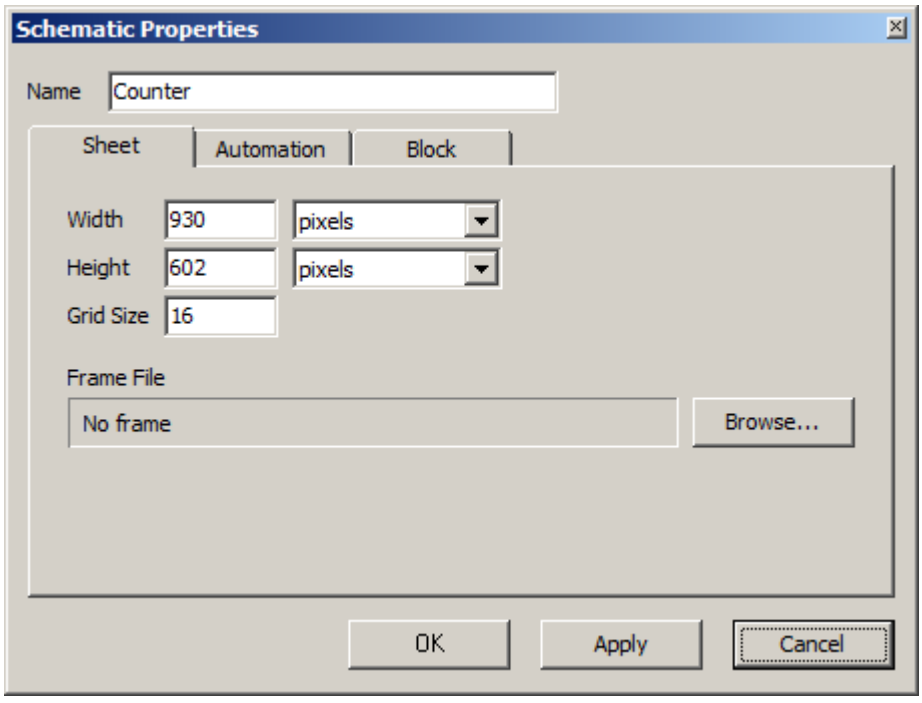
- If no element is selected, then Schematic Properties are shown for the currently active schematic.
- If a block instance that is not a port block is selected, then Block Instance Properties are shown for that block instance.
- If a connection is selected, then Connection Properties are shown for that connection.
- If a port block is selected, then Port Properties are shown for that port block.
- If several elements are selected, then the properties for one of the selected elements will be shown.

In all cases, the name of the schematic or the schematic element is shown in the top of the window. Refer to the Renaming an Element section for details on changing names of schematic elements. Changing the name of the schematic itself will automatically change the name of the block that it defines and may affect other source files that use this block. In this case, a dialog will appear, listing all affected source files. Clicking OK will perform the name change and update the references to the block in the listed source files. Open source files will be modified in the editor, while files that are not open will be modified directly on disk.

Schematic Properties

Schematic properties apply to the currently active schematic, and will be saved in the schematic file. No other schematic will be affected. To set default properties for new schematics, use the Options dialog.

Sheet page



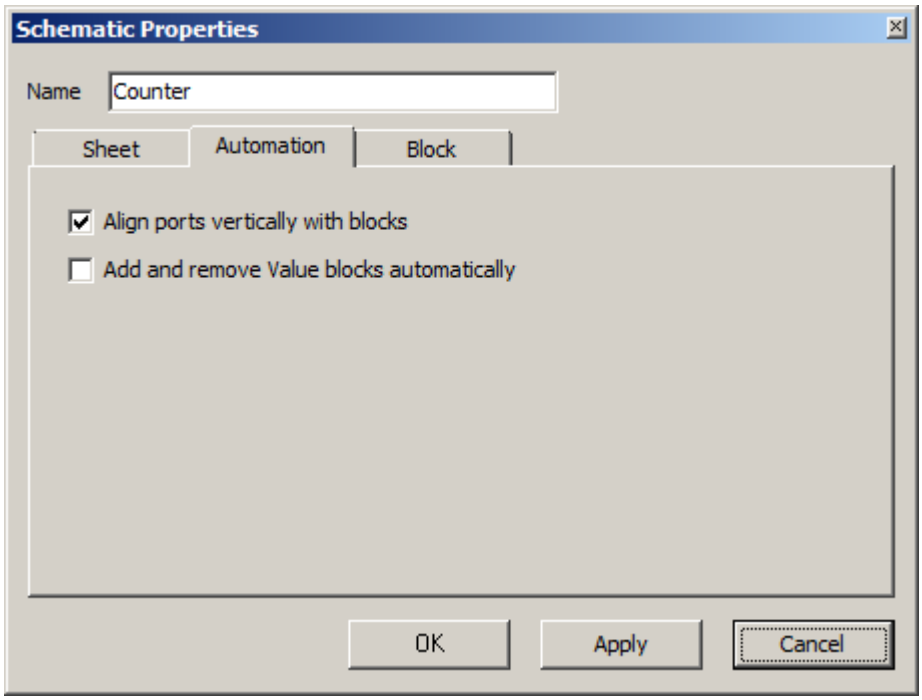
Width. The width of the schematic, given in the unit specified in the drop-down list immediately to the right of the number.

Height. The height of the schematic, given in the unit specified in the drop-down list immediately to the right of the number.

Grid Size. The length in pixels of the sides of the squares that constitute the grid.

Frame File. The full path to the file containing the frame definition, or **No frame** if no frame has been added. A new frame can be added to the schematic by clicking the **Browse...** button and selecting a frame file. A *frame* is an add-on of graphics to schematics, typically used to enhance the appearance of a printed schematic. However, there is no way to create a frame file in the current version of RTflow, so this should be seen as a future feature.

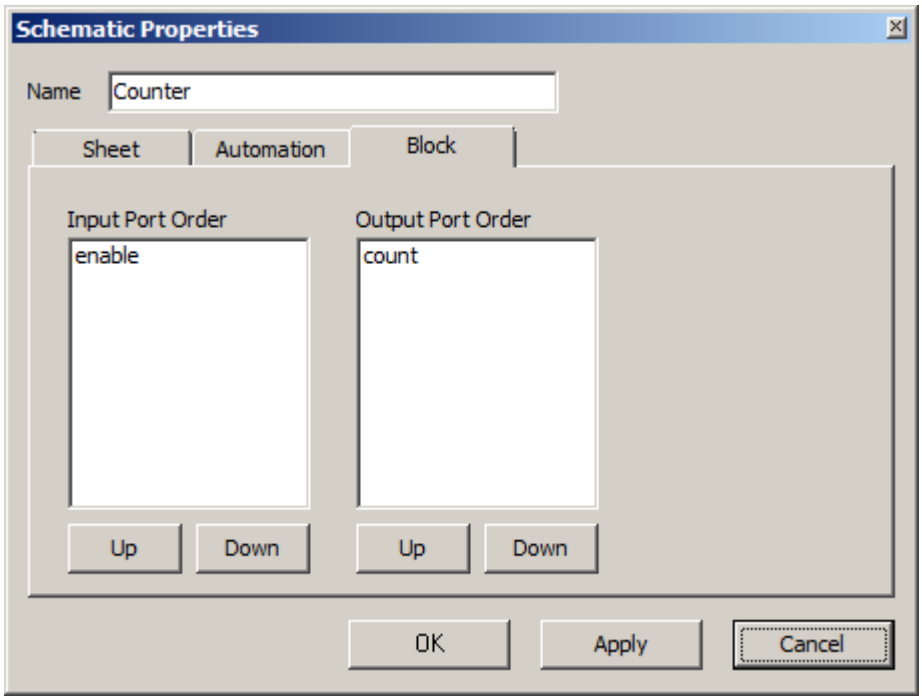
Automation page



Align ports vertically with blocks. When checked, all port blocks in the schematic are vertically aligned with the port instance it is connected to. As a result, it is impossible to move a port block vertically by dragging it with the mouse, unless the block instance it is connected to is also moved.

Add and remove Value blocks automatically. When checked, a Value block is always inserted automatically as soon as there is an unconnected input port instance in the schematic, and the new Value block is connected to that port instance. Moreover, unconnected Value blocks are always deleted. As a result, you can not add or remove Value blocks manually as with other blocks - this happens automatically under the mentioned conditions.

Block page



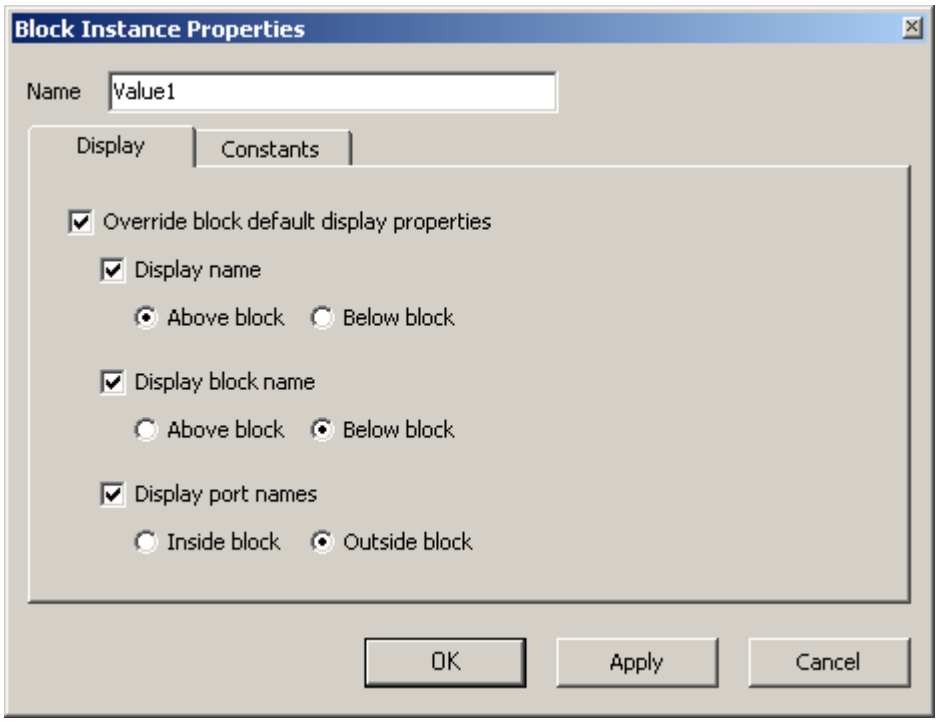
Input port order. The order in which the input ports will appear on the left side of the block that this schematic defines. Select a port in the list and press the **Up** or **Down** buttons below the list to move the port up and down. After changing the order and clicking OK or Apply, the schematic must also be compiled to apply the changes to the block.

Output port order. The order in which the output ports will appear on the right side of the block that this schematic defines. Select a port in the list and press the **Up** or **Down** buttons below the list to move the port up and down. After changing the order and clicking OK or Apply, the schematic must also be compiled to apply the changes to the block.

Block Instance Properties

Block instance properties apply to the selected block instance.

Display page



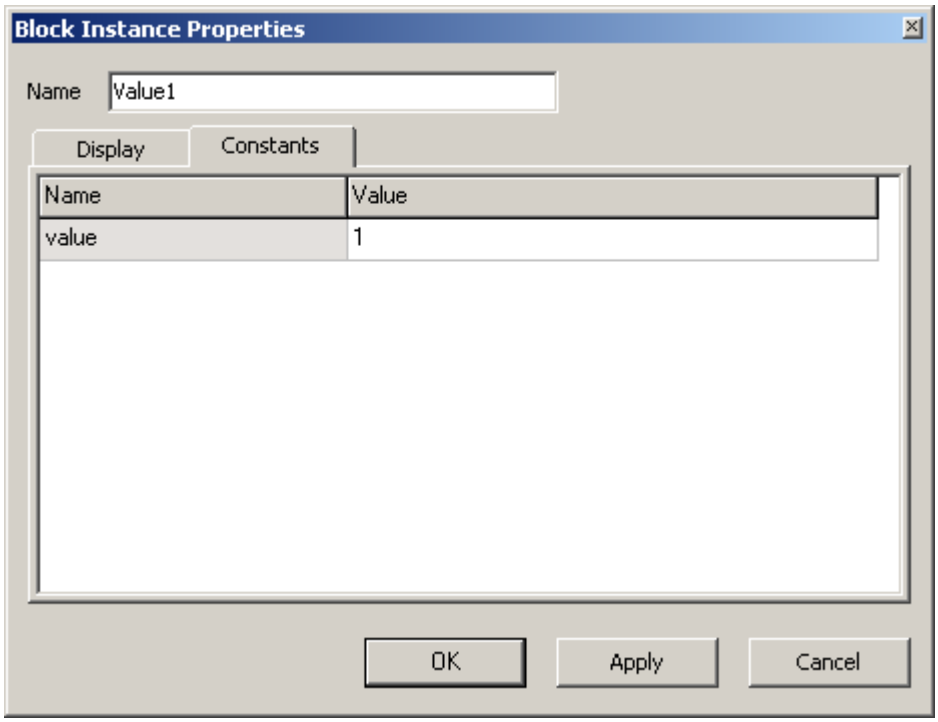
Override block default display properties. When checked, the block instance is displayed using the display properties that are set on this page. When unchecked, the block instance is displayed using the default display properties for the block itself. Default display settings for blocks can not be edited in the current version of RTflow.

Display name. When checked, the instance name is displayed by the block instance. If **Above block** is selected, the name is displayed above the block instance. If **Below block** is selected, the name is displayed below the block instance.

Display block name. When checked, the name of the block is displayed by the block instance. If **Above block** is selected, the name is displayed above the block instance. If **Below block** is selected, the name is displayed below the block instance.

Display port names. When checked, port names are displayed by each port instance of the block instance, provided that the Show Port Names toolbar button is down. If **Inside block** is selected, the names are displayed in the interior of the block symbol. If **Outside block**, the name is displayed outside of the block (immediately above the emerging connection, if any).

Constants page



A *constant* is an input whose value is a constant value specified at design/compile time. For example, the value that you enter in a Value block is a constant for that block. In the current version of RTflow, constants only exist in primitive blocks, and they are all available for editing directly in the schematic by clicking the value inside the symbol. However, to accommodate for future blocks with so many constants that they can't be edited directly in the schematic, constant values for a block instance can also be edited in the **Constants** page.

Connection Properties

There are no properties for connections.

Port Properties

There are no properties for ports.

The Blocks Tree

The *blocks tree* is the container of all available blocks for use in the schematic editor, available under the **Blocks** tab in the tree view in the left part of the main window. The blocks are organized in a tree, rather than in a list, in order to facilitate navigation. On the first level of subdivision, all blocks are categorized into one of four different kinds of blocks, as follows:

- Ports. Input and output port blocks of different types, for example BoolInput and RealOutput.
- Primitive Blocks. Blocks whose function is hard-coded in RTflow, and not defined in schematics or other source files, for example Add and Value.
- Standard Blocks. Blocks whose function is defined in a source file that is delivered with RTflow, for example Integral and Max.
- **User Blocks.** Blocks that have been compiled from source files created by the user.

In the current version of RTflow, there is no way to reorganize the blocks tree. The only modifications that are possible are to add and remove blocks of the **User** folder. User blocks are added by adding a new schematic to the project and compiling it, and a user block is deleted by removing its source file from the project.

Right-clicking a block in the blocks tree brings up a context menu with the following commands:

Command	Description
Add to Schematic	Adds an instance of the block to the currently active schematic.
Edit Source	Opens the source file that defines the block.

Add to Schematic

Adds an instance of the block to the currently active schematic. The block instance will be placed in an unoccupied area of the schematic, if possible. For details on adding blocks to schematics, see the Adding a Block section.

Availability

Available if the currently active tab in the tab area is a schematic.

Edit Source

Opens the source file that defines the block.

Availability

Available if the block is a standard or user block.

Blocks

This chapter provides detailed documentation of all built-in blocks in RTflow. The sections in this chapter are:

- Ports. Documentation of blocks under the **Ports** folder in the blocks tree.
- Primitive Blocks. Documentation of blocks under the **Primitive** folder in the blocks tree.
- Standard Blocks. Documentation of blocks under the **Standard** folder in the blocks tree.

Ports

Name	Symbol	Description
BoolInput		Boolean input port.
BoolOutput		Boolean output port.
IntInput		Integer input port.
IntOutput		Integer output port.
RealInput		Real input port.
RealOutput		Real output port.

BoolInput

Boolean input port. When the schematic is compiled, an input port of type bool will be generated for the block.

Blocks tree path

Ports\BoolInput

Outputs

Name	Type	Description
o	bool	The incoming signal through this port.

BoolOutput

Boolean output port. When the schematic is compiled, an output port of type bool will be generated for the block.

Blocks tree path

Ports\BoolOutput

Inputs

Name	Type	Description
i0	bool	The signal to send out through this port.

IntInput

Integer input port. When the schematic is compiled, an input port of type int will be generated for the block. Integers are 32-bit signed integers in the simulator, while it is dependent on the target language in the code generator.

Blocks tree path

Ports\IntInput

Outputs

Name	Type	Description
o	int	The incoming signal through this port.

IntOutput

Integer output port. When the schematic is compiled, an output port of type int will be generated for the block.

Blocks tree path

Ports\IntOutput

Inputs

Name	Type	Description
i0	int	The signal to send out through this port.

RealInput

Real input port. When the schematic is compiled, an input port of type real will be generated for the block. Reals are 32-bit IEEE floating point values in the simulator, while it is dependent on the target language in the code generator.

Blocks tree path

Ports\RealInput

Outputs

Name	Type	Description
o	real	The incoming signal through this port.

RealOutput

Real output port. When the schematic is compiled, an output port of type real will be generated for the block.

Blocks tree path

Ports\RealOutput

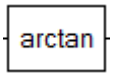
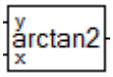
Inputs

Name	Type	Description
i0	real	The signal to send out through this port.

Primitive

Name	Symbol	Description
Bool		Converts a signal of any type to boolean.
Int		Converts a signal of any type to an integer.
Real		Converts a signal of any type to a real.

If		Selects one of two signals depending on a third boolean signal.
Not		Inverts a boolean signal.
And		Computes the logical and of two boolean signals.
Or		Computes the logical or of two boolean signals.
Xor		Computes the logical xor of two boolean signals.
LeftShift		Shifts the bits of an integer to the left.
RightShift		Shifts the bits of an integer to the right.
Add		Adds two numeric signals.
Sub		Subtracts two numeric signals.
Mult		Multiplies two numeric signals.
Div		Divides two numeric signals.
Mod		Computes the modulus of two numeric signals.
Sqrt		Computes the square root of a real-valued signal.
Less		Determines whether one numeric signal is less than another one.
LessEqual		Determines whether one numeric signal is less than or equal to another one.
Equal		Determines whether one signal is equal to another one.

NotEqual		Determines whether one signal is different from another one.
Greater		Determines whether one numeric signal is greater than another one.
GreaterEqual		Determines whether one numeric signal is greater than or equal to another one.
Gain		Multiplies a real-valued signal by a constant factor.
Identity		Copies a signal.
Value		Constant value.
Pre		Delays a signal one cycle.
Dt		The real time of one execution cycle in seconds.
Sin		Computes the sine of a real-valued signal.
Cos		Computes the cosine of a real-valued signal.
Tan		Computes the tangent of a real-valued signal.
Arctan		Computes the arctangent of a real-valued signal.
Arctan2		Computes the angle of a two-dimensional vector.
Ln		Computes the natural logarithm of a real-valued signal.
Exp		Computes e raised to the power of a real-valued signal.
Pow		Computes a real-valued signal raised to the power of another real-valued signal.

Bool

Converts a signal of any type to boolean. If i0 is 0, the resulting boolean signal o will be 0. In all other cases, o will be 1.

Blocks tree path

Primitive\Types\Bool

Inputs

Name	Type	Description
i0	any	The signal to convert.

Outputs

Name	Type	Description
o	bool	The resulting boolean signal.

Int

Converts a signal of any type to an integer. If i0 is boolean, then the resulting signal o will be 0 or 1. If i0 is real, then the fractional part will be truncated. Hence, 2.8 will be converted to 2, while -2.8 will be converted to -2. Integers are 32-bit signed integers in the simulator, while it is dependent on the target language in the code generator.

Blocks tree path

Primitive\Types\Int

Inputs

Name	Type	Description
i0	any	The signal to convert.

Outputs

Name	Type	Description

o	int	The resulting integer signal.

Real

Converts a signal of any type to a real. The resulting signal o will be the closest possible value to i0. Notice that there are high integer values that can not be represented exactly by the binary representation of real. Reals are 32-bit IEEE floating point values in the simulator, while it is dependent on the target language in the code generator.

Blocks tree path

Primitive\Types\Real

Inputs

Name	Type	Description
i0	any	The signal to convert.

Outputs

Name	Type	Description
o	real	The resulting real signal.

If

Selects one of two signals depending on a third boolean signal. If the boolean signal i2 is 0, o will be equal to i0. If i2 is 1, o will be equal to i1.

Blocks tree path

Primitive\Logic\If

Inputs

Name	Type	Description
i0	any	The signal to output if i2 is 0.
i1	any	The signal to output if i2 is 1.
i2	bool	The boolean signal that determine whether to output i0 or i1.

Outputs

Name	Type	Description
o	any	The selected signal.

Not

Inverts a boolean signal.

Blocks tree path

Primitive\Logic\Not

Inputs

Name	Type	Description
i0	bool	The boolean signal to invert.

Outputs

Name	Type	Description
o	bool	The resulting inverted signal.

And

Computes the logical and of two boolean signals. The resulting signal o is 1 if both i0 and i1 are 1, 0 otherwise.

Blocks tree path

Primitive\Logic\And

Inputs

Name	Type	Description
i0	bool	Operand.
i1	bool	Operand.

Outputs

Name	Type	Description
o	bool	Logical and of i0 and i1.

Or

Computes the logical or of two boolean signals. The resulting signal o is 1 if any of i0 or i1 is 1, 0 otherwise.

Blocks tree path

Primitive\Logic\Or

Inputs

Name	Type	Description
i0	bool	Operand.
i1	bool	Operand.

Outputs

Name	Type	Description
o	bool	Logical or of i0 and i1.

Xor

Computes the logical xor of two boolean signals. The resulting signal o is 1 if exactly one of i0 or i1 is 1, 0 otherwise.

Blocks tree path

Primitive\Logic\Xor

Inputs

Name	Type	Description

i0	bool	Operand.
i1	bool	Operand.

Outputs

Name	Type	Description
o	bool	Logical xor of i0 and i1.

LeftShift

Shifts the bits of an integer to the left. Zeroes are shifted in to the right side. This is equal to multiplying by 2 to the power of i1.

Blocks tree path

Primitive\Logic\LeftShift

Inputs

Name	Type	Description
i0	int	The integer signal to be shifted.
i1	int	The number of bit positions to shift.

Outputs

Name	Type	Description
o	int	The resulting shifted signal.

RightShift

Shifts the bits of an integer to the right. Zeroes are shifted in to the right side. This is equal to dividing by 2 to the power of i1.

Blocks tree path

Primitive\Logic\RightShift

Inputs

Name	Type	Description
i0	int	The integer signal to be shifted.
i1	int	The number of bit positions to shift.

Outputs

Name	Type	Description
o	int	The resulting shifted signal.

Add

Adds two numeric signals. The two signals must be of the same type.

Blocks tree path

Primitive\Arithmetic\Add

Inputs

Name	Type	Description
i0	numeric	Operand.
i1	numeric	Operand.

Outputs

Name	Type	Description
o	numeric	The sum of i0 and i1.

Sub

Subtracts two numeric signals. The two signals must be of the same type.

Blocks tree path

Primitive\Arithmetic\Sub

Inputs

Name	Type	Description
i0	numeric	The first term of the subtraction.
i1	numeric	The term to subtract from i0.

Outputs

Name	Type	Description
o	numeric	The difference between i0 and i1.

Mult

Multiplies two numeric signals. The two signals must be of the same type.

Blocks tree path

Primitive\Arithmetic\Mult

Inputs

Name	Type	Description
i0	numeric	Operand.
i1	numeric	Operand.

Outputs

Name	Type	Description
o	numeric	The product of i0 and i1.

Div

Divides two numeric signals. The two signals must be of the same type.

Blocks tree path

Primitive\Arithmetic\Div

Inputs

Name	Type	Description
i0	numeric	The numerator.
i1	numeric	The denominator.

Outputs

Name	Type	Description
o	numeric	The quotient of i0 and i1.

Mod

Computes the modulus of two numeric signals. The two signals must be of the same type. If they are real numbers, o has the same sign as i0, and the absolute value of o is less than the absolute value of i1.

Blocks tree path

Primitive\Arithmetic\Mod

Inputs

Name	Type	Description
i0	numeric	The numerator.
i1	numeric	The denominator.

Outputs

Name	Type	Description
o	numeric	The modulus of i0 and i1.

Sqrt

Computes the square root of a real-valued signal.

Blocks tree path

Primitive\Arithmetic\Sqrt

Inputs

Name	Type	Description
i0	numeric	Operand.

Outputs

Name	Type	Description
o	numeric	The square root of i0.

Less

Determines whether one numeric signal is less than another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is less than i1, 0 otherwise.

Blocks tree path

Primitive\Arithmetic\Less

Inputs

Name	Type	Description
i0	numeric	Left operand.
i1	numeric	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is less than i1, 0 otherwise.

LessEqual

Determines whether one numeric signal is less than or equal to another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is less than or equal to i1, 0 otherwise.

Blocks tree path

Primitive\Arithmetic\LessEqual

Inputs

Name	Type	Description
i0	numeric	Left operand.
i1	numeric	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is less than or equal to i1, 0 otherwise.

Equal

Determines whether one signal is equal to another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is equal to i1, 0 otherwise.

Blocks tree path

Primitive\Arithmetic\Equal

Inputs

Name	Type	Description
i0	any	Left operand.
i1	any	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is equal to i1, 0 otherwise.

NotEqual

Determines whether one signal is different from another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is different from i1, 0 if i0 is equal to i1.

Blocks tree path

Primitive\Arithmetic\NotEqual

Inputs

Name	Type	Description
i0	any	Left operand.
i1	any	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is different from i1, 0 if i0 is equal to i1.

Greater

Determines whether one numeric signal is greater than another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is greater than i1, 0 otherwise.

Blocks tree path

Primitive\Arithmetic\Greater

Inputs

Name	Type	Description

i0	numeric	Left operand.
i1	numeric	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is greater than i1, 0 otherwise.

GreaterEqual

Determines whether one numeric signal is greater than or equal to another one. The two signals must be of the same type. The resulting signal o is 1 if i0 is greater than or equal to i1, 0 otherwise.

Blocks tree path

Primitive\Arithmetic\GreaterEqual

Inputs

Name	Type	Description
i0	numeric	Left operand.
i1	numeric	Right operand.

Outputs

Name	Type	Description
o	bool	1 if i0 is greater than or equal to i1, 0 otherwise.

Gain

Multiplies a real-valued signal by a constant factor.

Blocks tree path

Primitive\Arithmetic\Gain

Inputs

Name	Type	Description
i0	real	The signal to apply gain to.

Constants

Name	Description
gain	The constant value to multiply with i0.

Outputs

Name	Type	Description
o	real	The product of i0 and gain.

Identity

Copies a signal. This block is only useful if the same signal need to appear with two different names in the simulator or in the generated code.

Blocks tree path

Primitive\Arithmetic\Identity

Inputs

Name	Type	Description
i0	any	The signal to copy.

Outputs

Name	Type	Description
o	any	The resulting copy.

Value

Constant value.

Blocks tree path

Primitive\Arithmetic\Value

Constants

Name	Description
value	The value.

Outputs

Name	Type	Description
o	any	The value as a signal.

Pre

Delays a signal one cycle. The output signal o is the value that i0 had in the previous execution cycle. If it's the first execution cycle, the output signal o is the value of the constant initial.

Blocks tree path

Primitive\Dynamic\Pre

Inputs

Name	Type	Description
i0	any	The signal to delay.

Constants

Name	Description
initial	The value that the output signal o shall have in the first execution cycle.

Outputs

Name	Type	Description
o	any	The delayed signal, or initial if it's the first execution cycle.

Dt

The real time of one execution cycle in seconds. This value is set in the Project Settings dialog in RTflow for simulation and code generation, respectively.

Blocks tree path

Primitive\Dynamic\Dt

Outputs

Name	Type	Description
o	real	The real time of one execution cycle in seconds.

Sin

Computes the sine of a real-valued signal.

Blocks tree path

Primitive\Trigonometric\Sin

Inputs

Name	Type	Description
i0	real	Operand in radians.

Outputs

Name	Type	Description
o	real	The sine of i0.

Cos

Computes the cosine of a real-valued signal.

Blocks tree path

Primitive\Trigonometric\Cos

Inputs

Name	Type	Description
i0	real	Operand in radians.

Outputs

Name	Type	Description
o	real	The cosine of i0.

Tan

Computes the tangent of a real-valued signal.

Blocks tree path

Primitive\Trigonometric\Tan

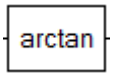
Inputs

Name	Type	Description
i0	real	Operand in radians.

Outputs

Name	Type	Description
o	real	The tangent of i0.

Arctan



Computes the arctangent of a real-valued signal. The resulting signal o will be in the interval [-pi/2,+pi/2].

Blocks tree path

Primitive\Trigonometric\Arctan

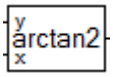
Inputs

Name	Type	Description
i0	real	Operand.

Outputs

Name	Type	Description
o	real	The arctangent of i0 in radians in the interval [-pi/2,+pi/2].

Arctan2



Computes the angle of a two-dimensional vector. i0 is the y-coordinate of the vector and i1 is the x-coordinate of the vector. The resulting signal o will be the angle of the given vector counterclockwise from the x-axis in radians and in the interval [-pi,+pi]. Notice that the length of the vector must be greater than 0, or a math error will occur.

Blocks tree path

Primitive\Trigonometric\Arctan2

Inputs

Name	Type	Description
i0	real	Y-coordinate of the vector to calculate the angle of.
i1	real	X-coordinate of the vector to calculate the angle of.

Outputs

Name	Type	Description
o	real	The angle of the given vector in radians in the interval [-pi,+pi].

Ln

Computes the natural logarithm of a real-valued signal.

Blocks tree path

Primitive\Exponential\Ln

Inputs

Name	Type	Description
i0	real	Operand.

Outputs

Name	Type	Description
o	real	The natural logarithm of i0.

Exp

Computes e raised to the power of a real-valued signal.

Blocks tree path

Primitive\Exponential\Exp

Inputs

Name	Type	Description
i0	real	Operand.

Outputs

Name	Type	Description
o	real	e raised to the power of i0.

Pow

Computes a real-valued signal raised to the power of another real-valued signal. i0 is the base and i1 is the exponent.

Blocks tree path

Primitive\Exponential\Pow

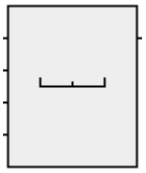
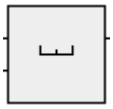
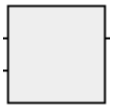
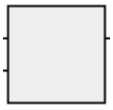
Inputs

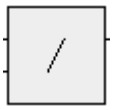
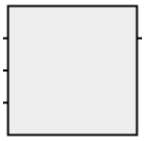
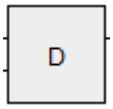
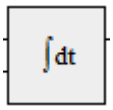
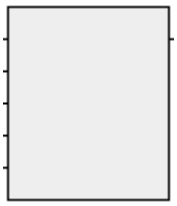
Name	Type	Description
i0	real	The base.
i1	real	The exponent.

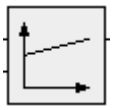
Outputs

Name	Type	Description
o	real	i0 raised to the power of i1.

Standard

Name	Symbol	Description
Abs		Computes the absolute value of a real-valued signal.
Distance		Computes the distance between two points.
Length		Computes the length of a vector.
Max		Selects the greatest value out of two real-valued signals.
Min		Selected the smallest value out of two real-valued signals.
Negation		Negates a real-valued signal.
SafeArctan2		Computes the angle of a two-dimensional vector and avoids math errors.

SafeDiv		Divides two real-valued signals and avoids math errors.
Saturation		Limits a real-valued signal between a lower and an upper bound.
Sqr		Computes the square of a real-valued signal.
Average		Computes the average of a real-valued signal over time.
Count		Counts the number of cycles that a boolean signal has been 1.
D		Holds the value of a real-valued signal.
Derivative		Differentiates a real-valued signal.
Integral		Integrates a real-valued signal.
IntegralWithReset		Integrates a real-valued signal and allows resetting.
Pid		Computes a control value given a setpoint signal and process signal using a PID algorithm.
PositiveEdge		Detects when a boolean signal switches from 0 to 1.
SetReset		Stores which out of two boolean signals last had a positive edge.
T		Computes the elapsed real time.

Clock		Generates a square wave.
Ramp		Generates a signal that increases or decreases at a fixed rate.
Degrees2Radians		Converts an angle in degrees to radians.
NormalizeAngle		Transposes an angle in radians to the interval [-pi,+pi].

Abs

Computes the absolute value of a real-valued signal.

Blocks tree path

Standard\Arithmetic\Abs

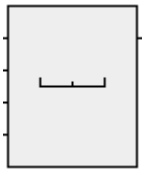
Inputs

Name	Type	Description
i	real	Operand.

Outputs

Name	Type	Description
o	real	The absolute value of i.

Distance



Computes the distance between two points. The points are given as two pairs of real cartesian coordinates (x1,y1) and (x2,y2).

Blocks tree path

Standard\Arithmetic\Distance

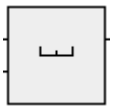
Inputs

Name	Type	Description
x1	real	X coordinate of first point.
y1	real	Y coordinate of first point.
x2	real	X coordinate of second point.
y2	real	Y coordinate of second point.

Outputs

Name	Type	Description
d	real	The distance between (x1,y1) and (x2,y2).

Length



Computes the length of a vector. The vector is given as a pair of real cartesian coordinates (dx,dy).

Blocks tree path

Standard\Arithmetic\Length

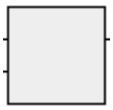
Inputs

Name	Type	Description
dx	real	X component of the vector.
dy	real	Y component of the vector.

Outputs

Name	Type	Description
l	real	The length of the vector (dx,dy).

Max



Selects the greatest value out of two real-valued signals.

Blocks tree path

Standard\Arithmetic\Max

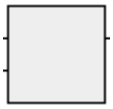
Inputs

Name	Type	Description
i0	real	Operand.
i1	real	Operand.

Outputs

Name	Type	Description
o	real	The greatest value of i0 and i1.

Min



Selected the smallest value out of two real-valued signals.

Blocks tree path

Standard\Arithmetic\Min

Inputs

Name	Type	Description
i0	real	Operand.
i1	real	Operand.

Outputs

Name	Type	Description
o	real	The greatest value of i0 and i1.

Negation

Negates a real-valued signal.

Blocks tree path

Standard\Arithmetic\Negation

Inputs

Name	Type	Description
i	real	The signal to negate.

Outputs

Name	Type	Description
o	real	The negation of i.

SafeArctan2

Computes the angle of a two-dimensional vector and avoids math errors. The vector is given as a pair of real cartesian coordinates (x,y). The resulting signal o will be the angle of the given vector counterclockwise from the x-axis in radians and in the interval [-pi,+pi]. If the length of the vector is 0, the resulting signal o will be 0.

Blocks tree path

Standard\Arithmetic\SafeArctan2

Inputs

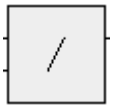
Name	Type	Description

y	real	Y-coordinate of the vector to calculate the angle of.
x	real	X-coordinate of the vector to calculate the angle of.

Outputs

Name	Type	Description
rad	real	The angle of the given vector in radians in the interval [-pi,+pi], or 0 if the length of (x,y) is 0.

SafeDiv



Divides two real-valued signals and avoids math errors. If i1, the denominator, is 0, the resulting signal o will be 0.

Blocks tree path

Standard\Arithmetic\SafeDiv

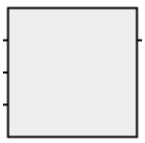
Inputs

Name	Type	Description
i0	real	The numerator.
i1	real	The denominator.

Outputs

Name	Type	Description
o	real	The quotient of i0 and i1, or 0 if i1 is 0.

Saturation



Limits a real-valued signal between a lower and an upper bound. The bounds are given as the real-valued signals min and max, where max must be greater than or equal to min. If value is less than min, the resulting signal limited will be min. If value is greater than max, the resulting signal limited will be max. In all other cases, limited will be value.

Blocks tree path

Standard\Arithmetic\Saturation

Inputs

Name	Type	Description
value	real	The signal to limit.
min	real	The lower bound.
max	real	The upper bound.

Outputs

Name	Type	Description
limited	real	The limited signal.

Sqr

Computes the square of a real-valued signal.

Blocks tree path

Standard\Arithmetic\Sqr

Inputs

Name	Type	Description
i	real	Operand.

Outputs

Name	Type	Description
o	real	The square of i.

Average

Computes the average of a real-valued signal over time. Values are only accumulated when enable is 1. Before any values have been accumulated, the resulting signal average will be 0. Notice that since the counter of accumulated cycles may be stored in a 32-bit signed integer, the resulting signal may be invalid after 2147483648 accumulated cycles.

Blocks tree path

Standard\Dynamic\Average

Inputs

Name	Type	Description
i	real	The value to compute the average of.
enable	bool	1 to accumulate i this cycle, 0 to ignore i this cycle.

Outputs

Name	Type	Description
average	real	The time average of i, or 0 if enable has not yet been 1.

Count

Counts the number of cycles that a boolean signal has been 1.

Blocks tree path

Standard\Dynamic\Count

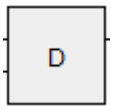
Inputs

Name	Type	Description
enable	bool	1 to increase the counter this cycle, 0 otherwise.

Outputs

Name	Type	Description
count	int	The number of cycles that enable has been 1.

D



Holds the value of a real-valued signal. When the boolean input signal set is 1, input signal value is copied to the resulting signal o. When set is 0, the resulting signal o holds the same value since set was 1. Before any value has been set, the resulting signal o is 0.

Blocks tree path

Standard\Dynamic\D

Inputs

Name	Type	Description
value	real	The value to hold.
set	bool	1 to set o to value, 0 to let o keep the old value.

Outputs

Name	Type	Description
o	real	The value of the signal value when set was last 1, or 0 if set has not yet been 1.

Derivative

Differentiates a real-valued signal. The derivative is computed by taking the difference between i0 at this cycle and the previous cycle and dividing it by the sample time.

Blocks tree path

Standard\Dynamic\Derivative

Inputs

Name	Type	Description
i0	real	The signal to differentiate.

Outputs

Name	Type	Description
o	real	The derivative of i0.

Integral

Integrates a real-valued signal. The integral is computed by multiplying i0 by the sample time and accumulating the result. The integral is 0 in the first execution cycle.

Blocks tree path

Standard\Dynamic\Integral

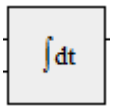
Inputs

Name	Type	Description
i0	real	The signal to integrate.

Outputs

Name	Type	Description
o	real	The integral of i0.

IntegralWithReset



Integrates a real-valued signal and allows resetting. The integral is computed by multiplying i by the sample time and accumulating the result. If rst is 1, the integration is reset to 0. The integral is 0 in the first execution cycle.

Blocks tree path

Standard\Dynamic\IntegralWithReset

Inputs

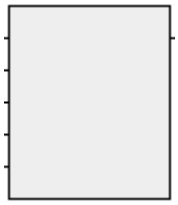
Name	Type	Description

i	real	The signal to integrate.
rst	bool	1 to reset the integration to 0, 0 otherwise.

Outputs

Name	Type	Description
o	real	The integral of i since rst was 1.

Pid



Computes a control value given a setpoint signal and process signal using a PID algorithm. A PID controller is used to produce a control signal that brings a process signal, typically a sensor signal from a physical system, as close as possible to a given setpoint value. The control signal is connected so that it controls the process (the physical system) that produces the process signal. The parameters kp, ki and kd specify the amount of proportional, integral and derivative gain, respectively, in the resulting control signal. To set these parameters correctly, some knowledge of PID controllers is required. Notice that the PID block reacts instantaneously to changes in the process signal, which means that it is necessary to include a delay (a Pre block) in the model of the process to avoid causality loops when simulating.

Blocks tree path

Standard\Dynamic\Pid

Inputs

Name	Type	Description
process	real	The process signal that is to be brought closer to setpoint.
setpoint	real	The value that the process value should be brought closer to.
kp	real	The amount of proportional gain in the PID algorithm.
ki	real	The amount of integral gain in the PID algorithm.
kd	real	The amount of derivative gain in the PID algorithm.

Outputs

Name	Type	Description

control	real	The control signal to control the process.

PositiveEdge

Detects when a boolean signal switches from 0 to 1.

Blocks tree path

Standard\Dynamic\PositiveEdge

Inputs

Name	Type	Description
i	bool	The signal whose positive edges to detect.

Outputs

Name	Type	Description
o	bool	1 if i is 1 in this cycle and was 0 in the previous cycle, 0 otherwise.

SetReset

Stores which out of two boolean signals last had a positive edge. If set was the last signal to have a positive edge, that is switched from 0 to 1, then the resulting signal o is 1. If reset was the last signal to have a positive edge, then the resulting signal o is 0. If both have a positive edge at the same time, reset (0) will have priority. If no signal has yet had a positive edge, the resulting signal is 0.

Blocks tree path

Standard\Dynamic\SetReset

Inputs

Name	Type	Description
set	bool	The boolean signal whose positive edges sets the resulting signal o to 1.
reset	bool	The boolean signal whose positive edges sets the resulting signal o to 0.

Outputs

Name	Type	Description
o	bool	1 if set was the last signal to have a positive edge, 0 otherwise.

T

Computes the elapsed real time. The time is accumulated in a floating-point signal that will eventually grow so big that further accumulation does not modify it. Hence, the computed time will become invalid. If 32-bit floats with 23 mantissa bits are used, then this happens at latest after 2^23 execution cycles.

Blocks tree path

Standard\Dynamic\T

Outputs

Name	Type	Description
t	real	The elapsed real time.

Clock

Generates a square wave. A square wave switches regularly between 0 and 1. The input signal t specifies the period time in execution cycles.

Blocks tree path

Standard\Generators\Clock

Inputs

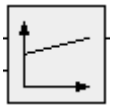
Name	Type	Description
t	int	The period time in execution cycles.

Outputs

Name	Type	Description

o	bool	A square wave with period t.

Ramp



Generates a signal that increases or decreases at a fixed rate. The start value is given by the input signal start, and the rate is given by the input signal step in change per execution cycle. Both parameters must be fed by constants in order to obtain a fixed change rate.

Blocks tree path

Standard\Generators\Ramp

Inputs

Name	Type	Description
start	real	The start value, that is, the value that the output signal will have in the first cycle.
step	real	The change for each execution cycle.

Outputs

Name	Type	Description
value	real	The generated signal.

Degrees2Radians

Converts an angle in degrees to radians.

Blocks tree path

Standard\Trigonometric\Degrees2Radians

Inputs

Name	Type	Description
deg	real	The angle in degrees to convert.

Outputs

Name	Type	Description
rad	real	The converted angle in radians.

NormalizeAngle

Transposes an angle in radians to the interval [-pi,+pi]. The resulting signal o is i + n*2*pi for some integer n such that o is in the interval [-pi,+pi].

Blocks tree path

Standard\Trigonometric\NormalizeAngle

Inputs

Name	Type	Description
i	real	The angle to transpose in radians.

Outputs

Name	Type	Description
o	real	The angle in radians in the interval [-pi,+pi].

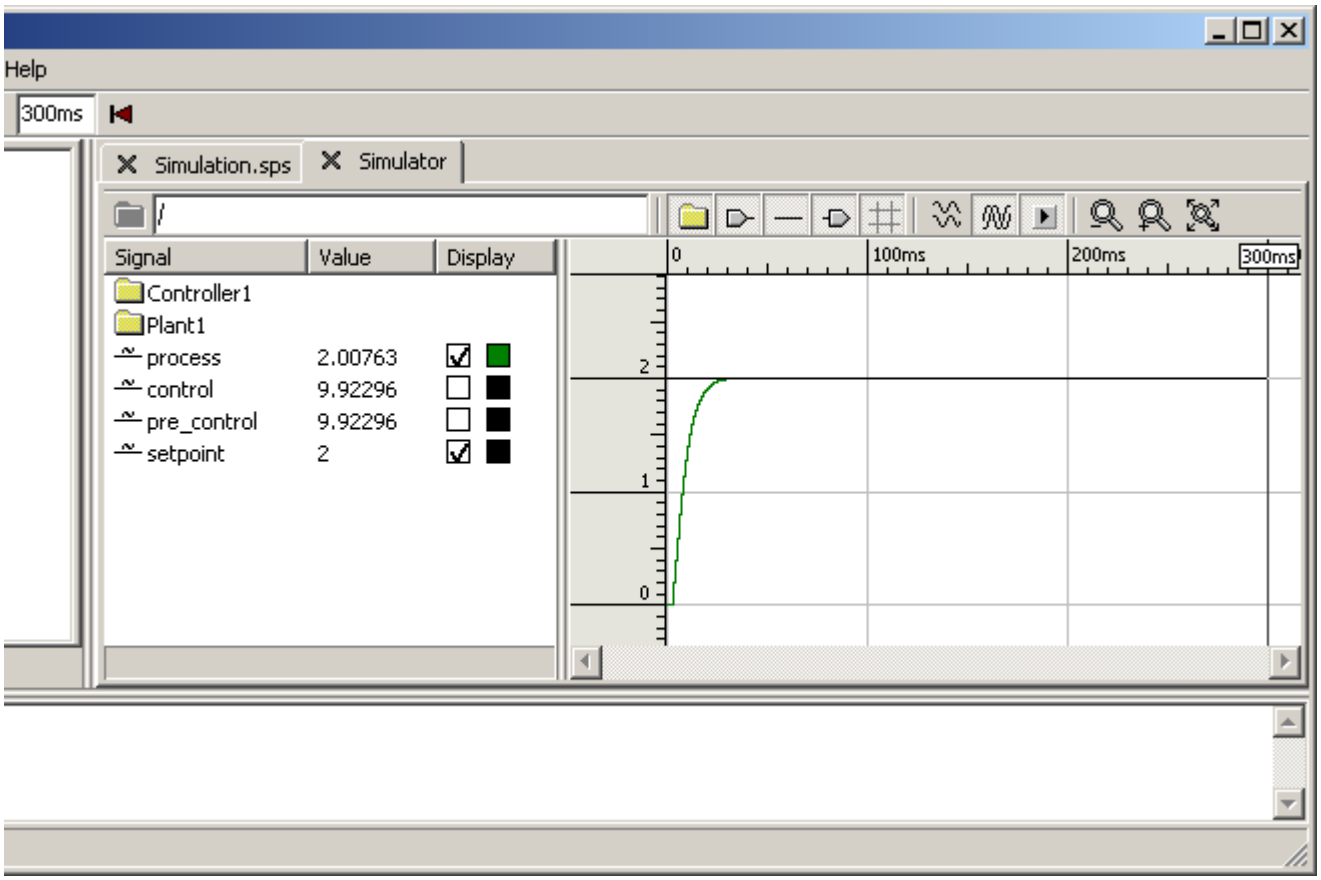
The Simulator

This chapter provides detailed documentation of the simulator and the simulator view. The sections in this chapter are:

- Simulator - General. Provides an overview of the simulator view and introduces some terms used in this chapter.
- Simulation Settings. Describes the settings available under the **Simulation** page in the **Project Settings** dialog.
- Starting and Running the Simulator. Describes how to start, step forward and reset the simulator.
- The Signal Area. Describes the different columns of the signal area and the effect of clicking them.
- The Graph Area. Describes the elements and contents of the graph area and the effect of clicking and dragging in it.
- Simulator Toolbar Buttons. Describes the effect of using the toolbar buttons in the simulator.
- Simulator Keyboard Shortcuts. Describes the effect of key commands in the simulator.

Simulator - General

The *simulator view*, which opens under a special tab in the tab area labeled **Simulator** when simulation is started, consists of three main parts: The signal area in the left half, the graph area in the right half, and the toolbar above the two areas. Between the signal area and the graph area, there is a separator that can be dragged with the mouse to resize or hide the areas.



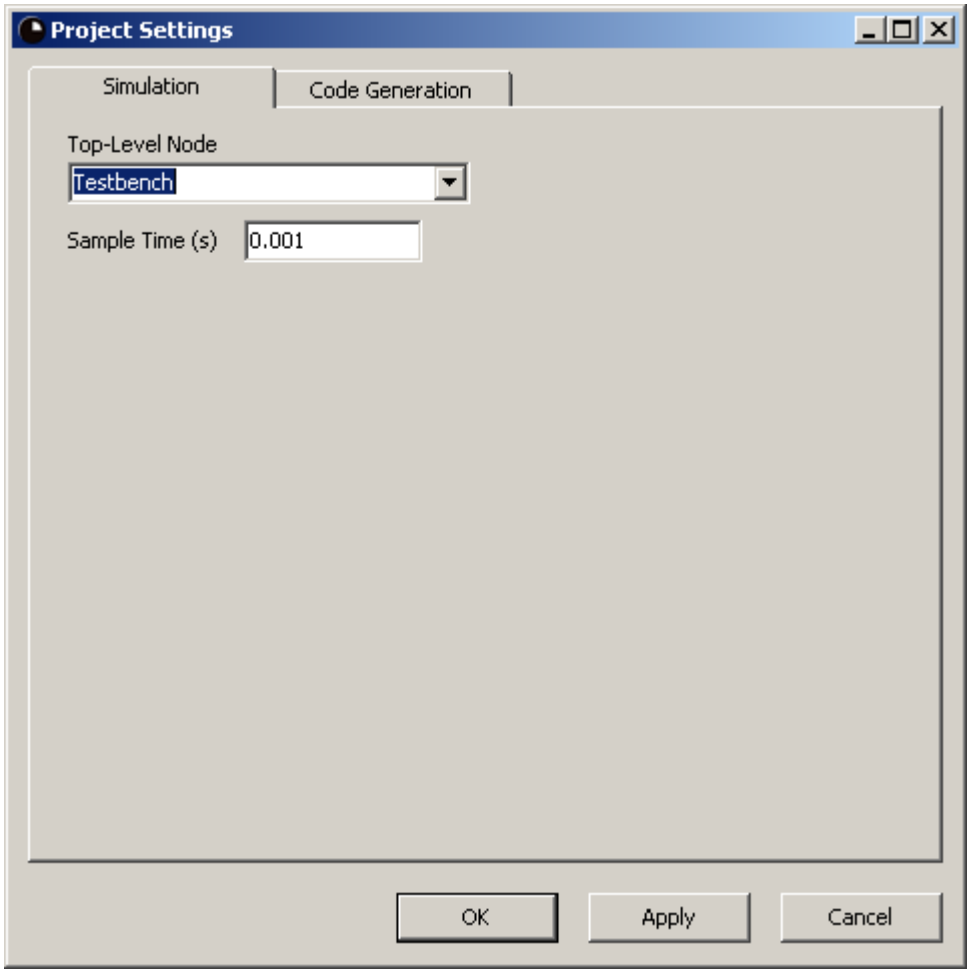
Terminology

In this chapter, a different terminology is used for entities that at first may appear as equivalent. However, while the previous chapter dealt with syntactic elements of the schematic language, this chapter deals with their semantic counterparts - mathematical objects involved in the computation and execution of the model, and this distinction calls for a new set of terms. The following table lists the syntactic elements in the previous chapter and the semantic counterparts used in this chapter:

Syntactic Element	Semantic Counterpart
Schematic	Node
Non-primitive block instance	Subnode
Primitive block instance	Primitive equation
Connection	Signal

Simulation Settings

This section describes the settings available under the **Simulation** page in the **Project Settings** dialog.



Top-Level Node. The node to be simulated. Naturally, all nodes directly or indirectly used by the top-level node will also be simulated.

Sample Time (s). The real time in seconds corresponding to one execution cycle. This setting specifies the output value of the Dt block, which is used in the definitions of, among others, the Derivative and the Integral blocks. Moreover, it is used to translate real time into execution cycles if you enter a real time value in the simulation time box in the toolbar of the main window. The value must be greater than 0.

Starting and Running the Simulator

The simulator is started and stepped forward by invoking the Simulate command. When the simulator is active, it can be reset by invoking the Reset Simulator command.

The simulator is running as long as the simulator view is open. To stop the simulator, simply close the simulator view.

The Signal Area

The signal area shows the state of the simulator as a table. Each row of this table corresponds to one subnode or one signal, and the different columns show different attributes of the signals. When the simulator starts, the signal area shows the subnodes and signals of the top-level node, corresponding to the block instances and connections of the top-level schematic. You can double-click the name of a subnode to enter this node, so that the simulator view shows the subnodes and signals of that node instead. Hence, the signal area works quite similar to the Windows File Explorer, where you double-click a folder to view the contents of it. Therefore, subnodes are symbolized by a folder icon in the signal area.

To optimize the view, you can show or hide individual columns by right-clicking the header row and checking or unchecking the name of the column in the context menu that appears.

The columns of the signal area are:

- Signal. Shows the name of the subnode or the signal. This column can not be hidden.
- Type. Shows the type of the signal.
- Value. Shows the value of the signal.
- Force. Shows whether the signal is forced, and to what value.
- Display. Shows the graph display settings for the signal.

The Signal column

The **Signal** column shows the name of the subnode or the signal. To the left of the name, an icon shows the type and kind of the subnode or signal, as follows:

Icon	Type and kind
	Subnode
	Boolean input signal
	Boolean local signal
	Boolean output signal

	Integer input signal
	Integer local signal
	Integer output signal
	Real input signal
	Real local signal
	Real output signal

Double-clicking a subnode name opens the subnode for viewing in the simulator view.

The Type Column

The **Type** column shows the type of the signal. Clicking in the **Type** column has no effect.

The Value Column

The **Value** column shows the value of the signal at the execution cycle indicated by the cursor. The *cursor* is the vertical line in the graph area. By default, the cursor is positioned at the end of the simulation data, but it can be moved by clicking in the graph area - see the The Graph Area section for details.

Clicking in the **Value** column enables forcing of the signal - see the The Force Column section for details on forcing.

The Force Column

The **Force** column shows whether the signal is currently forced, and to what value. If this column is empty for a signal, then the signal is not being forced, otherwise, it is forced to the shown value.

Forcing means that you override the simulator's computation of a signal and forces it to a specific value. All signals that depend on the forced signal are affected accordingly. It is typically used on input signals in order to see how the model reacts to specific input values, but it is also possible to force local and output signals. Forcing applies immediately to the last cycle of the simulation data and to all cycles being simulated while it is enabled. However, it is not possible to go back and force signals in cycles before the last cycle.

As an example, let's say that you start out by simulating your model for 1000 cycles without any forcing enabled. Hence, the last cycle of the simulation data so far will then be cycle number 1000. Then you force one of the input signals to another value. The value of the input signal itself and all the signals that depend on it will immediately be updated in cycle number 1000, while cycles 0 through 999 will stay unaffected. Finally, you run the simulation for another 1000 cycles, stopping at cycle number 2000. Now all the cycles 1000 through 2000 are affected by the forcing. If you now disable the forcing again, cycle number 2000 will immediately be updated to reflect that no signal is forced, while cycles 1000 through 1999 are still affected by the forcing.

Forcing is enabled by clicking in the **Value** or the **Force** column. Depending on the type of the signal, the following happens:

- If the signal is of type **boolean**, it is forced to the negation of the previous value. In other words, clicking switches the value between 0 and 1.

- Otherwise, a text box appears, where you can enter a force value. The forcing is enabled when you press the Enter key, while pressing the Esc key cancels the forcing.

Forcing is disabled by right-clicking the signal and selecting **Unforce** in the context menu that appears.

The Display Column

The **Display** column shows the graph display settings for the signal. The column contains two boxes; the *visible* box and the *color* box.

The visible box

The visible box is a check box such that when checked, the signal's graph is visible in the graph area, otherwise it is invisible. It has effect only in joint view. Clicking the check box switches between checked (visible) and unchecked (invisible). At most three graphs can be visible at a time. If you check the visible box of a fourth signal, another signal's visible box will be unchecked, and its graph will accordingly disappear.

The color box

The color box shows the color of the graph of the signal. Clicking the color box opens a color picker dialog, where you can choose another color for the graph. Clicking **OK** in the color picker dialog applies the new color to the graph.

The Graph Area

The graph area shows the simulated data as graphs of signal values over time. It can operate in two different modes:

- Split View. Shows each graph in its own row, which is the extension of the signal's row in the signal area.
- Joint View. Shows at most three graphs, all in the same coordinate system.

You can switch between split view and joint view using the Split View and Joint View toolbar buttons.

For both views, the horizontal axis is (simulated) time, and the scale is shown in the top of the view. Depending on whether you have specified simulation time as a number of cycles or as real time in the simulation time box in the toolbar of the main window, the scale shows the time either as the number of cycles elapsed or as the real time elapsed accordingly. In the latter case, a time unit is shown in the scale, or, if more than one minute has elapsed, the time is shown as hours, minutes and seconds, separated by colons.

The *cursor* is a vertical line through the whole graph area. Its exact position is shown in a box in the scale, in the top of the area. By default, the cursor is positioned at the last cycle of the simulated data, but you can move it by clicking in the graph or the scale at the time to which the cursor should be moved. The position of the cursor decides what cycle the values in the **Value** column are shown for.

Split View

In split view, the graph area shows each graph in its own row, which is the extension of the signal's row in the signals view. The vertical scales of the individual graphs adjust automatically, so that the entire graph computed so far fits in the row. Split view is typically used when you want to have an overview of the trends of all signals in view.

If you drag with the mouse horizontally over the graph area, the graphs will zoom in horizontally such that the area covered by the mouse will fill the whole graph area. There is no way to zoom in vertically or increase the height of the rows in split view in the current version of RTflow. To zoom out again, right-click in the graph area and select one of the following:

- **Zoom Out Horizontally**. Zooms out so that the whole graph from the first to the last cycle can be seen.
- **Zoom To 100% Horizontally**. Zooms out so that one pixel corresponds to one cycle. This is equivalent to clicking the Restore Zoom toolbar button in split view.

Joint View

In joint view, the graph area shows all graphs in the same coordinate system. At most three graphs can be displayed, and these are selected by clicking the visible box in the **Display** column of the signal in the signal area. By default, the vertical scale adjusts automatically, so that all of the graphs computed so far fit in the area. Joint view is typically used when you want to study the details of a few specific signals.

If you drag with the mouse over the graph area, the graphs will zoom in such that the area covered by the mouse will fill the whole graph area. As a consequence, the automatic vertical scaling is disabled, and further simulation will not rescale the graphs. To zoom out and/or reenable automatic vertical scaling, right-click the graph area and select one of the following:

- **Zoom Out Completely**. Zooms out so that the whole graph from the first to the last cycle can be seen and reenables automatic vertical scaling.
- **Zoom Out Horizontally**. Zooms out horizontally so that the whole graph from the first to the last cycle can be seen.
- **Zoom To 100% Horizontally**. Zooms out horizontally so that one pixel corresponds to one cycle.
- **Zoom Out Vertically**. Reenables automatic vertical scaling.

Simulator Toolbar Buttons

The simulator view has a toolbar, situated above the signal and graph areas and below the tabs, where you can change functionality and display settings of the simulator view. The toolbar buttons are summarized in the following table, and the commands that they invoke are described in detail in the subsequent subsections.

Toolbar Button	Command	Description
	Pop Context	Shows the signals of the supernode of the current instance.
	Current Instance	Shows the path from the top-level node to the current instance.
	Show Subnodes	Shows or hides rows with subnodes in the signal area.
	Show Inputs	Shows or hides rows with input signals in the signal area.
	Show Locals	Shows or hides rows with local signals in the signal area.
	Show Outputs	Shows or hides rows with output signals in the signal area.

	Show Grid	Shows or hides the grid in the graph area.
	Split View	Switches the graph area to split view.
	Joint View	Switches the graph area to joint view.
	Auto Scroll	Enables or disables automatic horizontal scrolling of the graph area.
	Zoom Out	Decreases the horizontal zoom level of the graph area.
	Zoom In	Increases the horizontal zoom level of the graph area.
	Restore Zoom	Restores vertical and horizontal zoom of the graph area to the default setting.

Pop Context Toolbar Button (Simulator)

Shows the signals of the supernode of the current instance. This command is only available if you have double-clicked a subnode in the signal area, so that the signals of a subnode are currently being shown. In that case, this command returns to the parent node. In effect, if the current instance has been reached by double-clicking subnodes several times to walk down in the model hierarchy, this command is used to walk back, upwards in the model hierarchy.

Toolbar button

Availability

Available when the signal area is showing the signals of another node than the top-level node.

Current Instance

Shows the path from the top-level node to the current instance. The *current instance* is the node instance (the top-level node or any subnode) whose signals are currently being shown in the simulator view. The path is a sequence of subnode names, separated by slashes, such that one can start from the top-level node and double-click the subnodes specified in the sequence in order to reach the current instance. You can also manually enter a path in the field, but if the entered path is invalid, no signals will be shown.

Toolbar button

Availability

Always available.

Show Subnodes Toolbar Button

Shows or hides rows with subnodes in the signal area. When the button is down, subnodes of the current instance are shown in the signal area. When the button is up, the subnodes are hidden.

Toolbar button

Availability

Always available.

Show Inputs Toolbar Button

Shows or hides rows with input signals in the signal area. When the button is down, input signals of the current instance are shown in the signal area. When the button is up, the input signals are hidden.

Toolbar button

Availability

Always available.

Show Locals Toolbar Button

Shows or hides rows with local signals in the signal area. When the button is down, local signals of the current instance are shown in the signal area. When the button is up, the local signals are hidden. *Local signals* correspond to connections that are internal in the schematics, that is, not attached to a port block.

Toolbar button

Availability

Always available.

Show Outputs Toolbar Button

Shows or hides rows with output signals in the signal area. When the button is down, output signals of the current instance are shown in the signal area. When the button is up, the output signals are hidden.

Toolbar button

Availability

Always available.

Show Grid Toolbar Button (Simulator)

Shows or hides the grid in the graph area. When the button is down, a light gray grid is shown in the graph area, indicating major time units by vertical lines and, if the graph area is in joint view, major value units by horizontal lines. When the button is up, the grid is not shown.

Toolbar button

Availability

Always available.

Split View Toolbar Button

Switches the graph area to split view. See the Split View section for a description of the split view.

Toolbar button

Availability

Available when the graph area is in joint view.

Joint View Toolbar Button

Switches the graph area to joint view. See the Joint View section for a description of the joint view.

Toolbar button

Availability

Available when the graph area is in split view.

Auto Scroll Toolbar Button

Enables or disables automatic horizontal scrolling of the graph area. When the button is down, invoking the Simulate command for stepping forward the simulator will automatically cause the graph area to scroll horizontally to the end of the simulated data. When the button is up, the graph area will not scroll automatically.

Toolbar button

Availability

Always available.

Zoom Out Toolbar Button (Simulator)

Decreases the horizontal zoom level of the graph area. The zoom level is decreased by half, such that the same time interval covers half the horizontal extension after zooming out. If possible, the graph area will also scroll such that the cursor stays at the same screen position.

All elements of the graphs are shown, also in zoom levels of less than one pixel per cycle. For instance, consider a graph of a signal that is 0 for all cycles 0 through 1,000,000, except for one single cycle in the middle, where it is 1. This spike will be seen in the graph, regardless of how far out you zoom. However, a consequence of this feature is that drawing can become slow at very low zoom levels.

Toolbar button

Availability

Available when the current zoom level is greater than 1/1,000,000 pixels per cycle.

Zoom In Toolbar Button (Simulator)

Increases the horizontal zoom level of the graph area. The zoom level is doubled, such that the same time interval covers double the horizontal extension after zooming in. If possible, the graph area will also scroll such that the cursor stays at the same screen position.

Toolbar button

Availability

Available when the current zoom level is less than 16 pixels per cycle.

Restore Zoom Toolbar Button

Restores vertical and horizontal zoom of the graph area to the default setting. Horizontal zoom is restored to one pixel per cycle. Moreover, if the graph area is in joint view, automatic vertical scaling is reenabled so that all of the graphs computed so far fit in the area.

Toolbar button

Availability

Always available.

Simulator Keyboard Shortcuts

The following table summarizes the keyboard shortcuts available in the simulator.

Shortcut	Command	Description
Ctrl-Left arrow	Cursor Back	Moves the cursor one cycle back in the simulation data.
Ctrl-Right arrow	Cursor Forward	Moves the cursor one cycle forward in the simulation data.

Cursor Back

Moves the cursor one cycle back in the simulation data. As a consequence, the signal area shows the signal values for the previous cycle. This command can also be used when studying signal values in a schematic editor.

Keyboard shortcut

Ctrl-Left arrow

Availability

Available when the cursor is not at the first cycle.

Cursor Forward

Moves the cursor one cycle forward in the simulation data. As a consequence, the signal area shows the signal values for the next cycle. This command can also be used when studying signal values in a schematic editor.

Keyboard shortcut

Ctrl-Right arrow

Availability

Available when the cursor is not at the last cycle of the simulated data.

The Code Generator

This chapter provides detailed documentation of the code generator. The sections in this chapter are:

- Code Generator - General. Describes the possibilities with the code generator, how to invoke it and how to choose a template file.
- Code Generation Settings. Describes the settings available under the **Code Generation** page in the **Project Settings** dialog.
- Key Reference. Documentation of all code generation keys available for template files.

Code Generator - General

RTflow can generate implementation code from the model, so that it can be implemented in a real-world application. To be more precise, RTflow can generate C, C++ and Java for software implementations and synthesizable VHDL for FPGA/ASIC implementations. Code generation is performed by invoking the Generate Code command. The first time, you will be asked to set up some code generation settings in the project settings dialog. Subsequent invocations require no further user interaction.

Template files

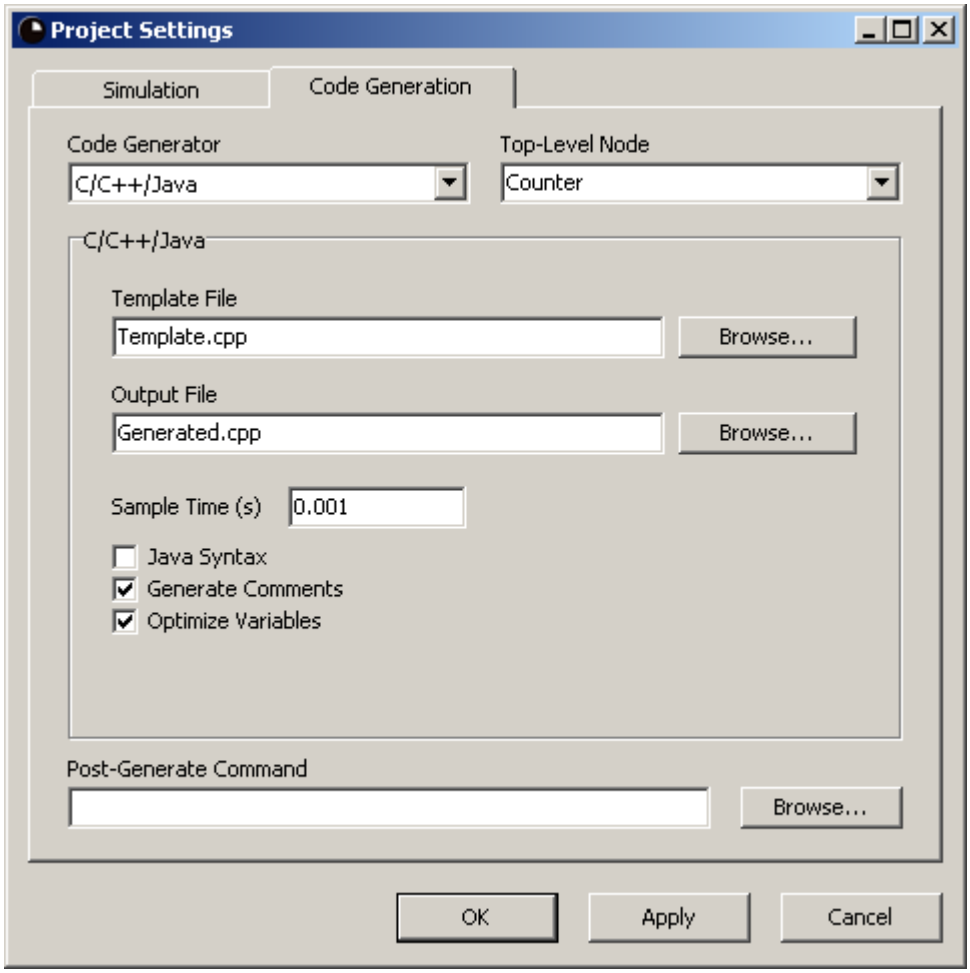
Among a few other settings for code generation, you have to specify a *template file*. The template file defines the interface of the generated file by means of skeleton functions that contain code generation keys. The code generator reads the template file and outputs a copy of it, but with the keys replaced by actual generated code. Most users don't need to write their own template file, but can use the ones supplied with RTflow - see the table below. However, advanced users may prefer to write their own template file. The Key Reference provides the primary documentation for those users.

The template files supplied with RTflow are (paths are relative to the application folder):

Target Language	Path
C	Templates/Template.c
C++	Templates/Template.cpp
Java	Templates/Template.java
VHDL	Templates/Template.vhd

It is recommended to copy the template file to the project folder, to ensure that the project can be moved to other computers.

Code Generation Settings



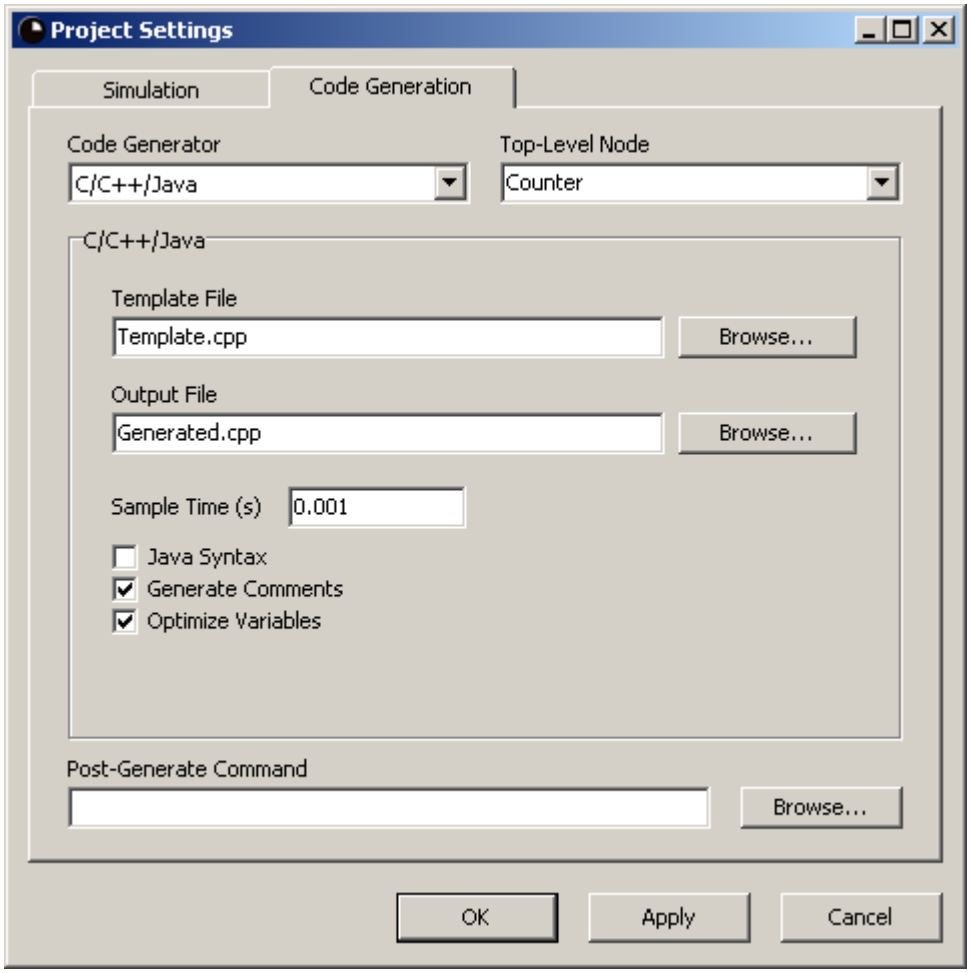
Code Generator. The code generator module to use, and hence, what target language to generate.

Top-Level Node. The node to generate code for. Naturally, code will also be generated for all nodes directly or indirectly used by the top-level node.

Post-Generate Command. A system command, an executable file or a batch file that will be executed after each code generation. In the case of an executable file or a batch file, the path can be both absolute or relative to the project folder, that is, the folder containing the project file. Click the **Browse...** button to the right of the field to select the executable or batch file using a file dialog.

The remaining settings are dependent on what code generator was chosen in the **Code Generator** drop-down box. For **C/C++/Java**, see the C/C++/Java Generator Settings section. For **VHDL**, see the VHDL Generator Settings section.

C/C++/Java Generator Settings



Template File. The template file for code generation. See the Code Generator - General section for an explanation of the template file. The path can be both absolute or relative to the project folder, that is, the folder containing the project file. Click the **Browse...** button to the right of the field to select the template file using a file dialog.

Output File. The location and name of the file to generate. The path can be both absolute or relative to the project folder, that is, the folder containing the project file. Click the **Browse...** button to the right of the field to select the location and name using a file dialog.

Sample Time (s). The real time in seconds corresponding to one execution cycle. This setting specifies what the `%SAMPLE_TIME%` key in the template file will be translated to in the code generation. It also specifies the output value of the Dt block, which is used in the definitions of, among others, the Derivative and the Integral blocks. The value must be greater than 0.

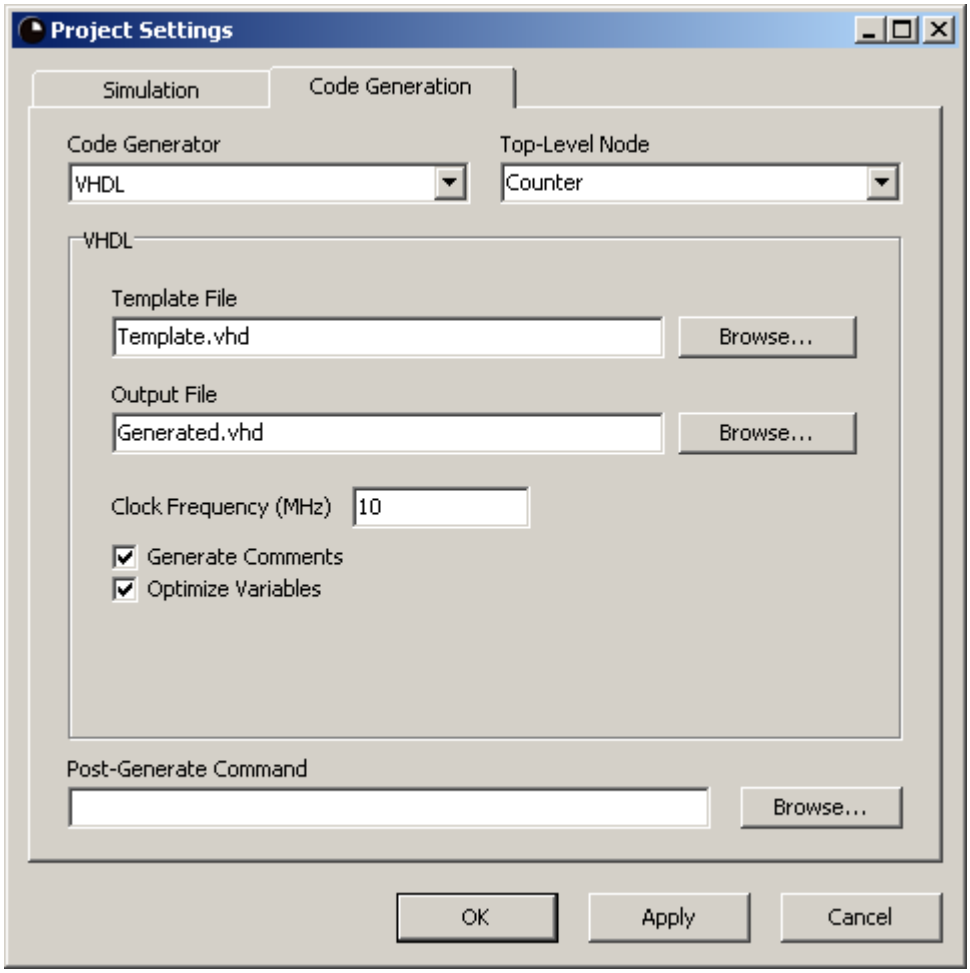
Java Syntax. When checked, Java code will be generated instead of C/C++ code. For example, the call to a sine function will be `Math.sin(x)` instead of just `sin(x)`.

Generate Comments. When checked, each generated assignment will be followed by a comment that indicates what signals are involved in the operation. As an example, `i[1] = i[2] + i[3];` could be followed by `/* sum = pre_count + v1; */`. While this option makes the generated file bigger, it also makes it a little more human-readable.

Optimize Variables. When checked, the generated code will be optimized. As a consequence, some internal signals may be removed. This means that the caller of the generated code can not

assume that all signals in the source files will also exist in the code. However, the input and output signals of the top-level node will always be available.

VHDL Generator Settings



Template File. The template file for code generation. See the Code Generator - General section for an explanation of the template file. The path can be both absolute or relative to the project folder, that is, the folder containing the project file. Click the **Browse...** button to the right of the field to select the template file using a file dialog.

Output File. The location and name of the file to generate. The path can be both absolute or relative to the project folder, that is, the folder containing the project file. Click the **Browse...** button to the right of the field to select the location and name using a file dialog.

Clock Frequency (MHz). The frequency in MHz of the clock that will drive the implementation. This setting specifies what the `%FREQUENCY_MHZ%` key in the template file will be translated to in the code generation. It also determines the output value of the Dt block, which is used in the definitions of, among others, the Derivative and the Integral blocks. The value of **Dt** will be 1 / (clock frequency * 1,000,000). The clock frequency must be greater than 0.

Generate Comments. When checked, each generated assignment will be followed by a comment that indicates what signals are involved in the operation. As an example, `i(1) <= i(2) + i(3);` could be followed by `-- sum <= pre_count + v1;`. While this option makes the generated file bigger, it also makes it a little more human-readable.

Optimize Variables. When checked, the generated code will be optimized. As a consequence, some internal signals may be removed. This means that the caller of the generated code can not assume that all signals in the source files will also exist in the code. However, the input and output signals of the top-level node will always be available.

Key Reference

The following table summarizes the code generation keys that will be recognized in the template file. *type* can be any of BOOL, INT and REAL. Keys are case sensitive and are always in capital letters. Most of the keys can be used with both the C/C++/Java generator and the VHDL, while a few keys are unique for one of the generators. The table specifies with an X for which code generators the key can be used.

Key	Description	C/C++/Java	VHDL
%*type*_VECTOR%	The name of the *type* array.	X	X
%IF_HAS_*type*%	Includes the line if and only if there are *type* signals.	X	X
%*type*_NAMES%	The names of the signals and states in the *type* array.	X	
%*type*_INPUT_NAMES%	The names of the input signals of type *type*.		X
%*type*_OUTPUT_NAMES%	The names of the output signals of type *type*.		X
%NUMBER_*type*%	The total number of *type* signals.	X	X
%NUMBER_*type*_INPUT%	The number of *type* input signals.	X	X
%NUMBER_*type*_OUTPUT%	The number of *type* output signals.	X	X
%NUMBER_*type*_LOCAL%	The number of *type* local signals.	X	X
%NUMBER_*type*_STATE%	The number of *type* state variables.	X	X
%OFFSET_*type*_INPUT%	The index of the first input signal in the *type* array.	X	X
%OFFSET_*type*_OUTPUT%	The index of the first output signal in the *type* array.	X	X
%OFFSET_*type*_LOCAL%	The index of the first local signal in the *type* array.	X	X
%OFFSET_*type*_STATE%	The index of the first state variable in the *type* array.	X	X
%INITIAL_EQUATIONS%	Code that resets all state variables to the initial state.	X	X
%OUTPUT_UPDATE_EQUATIONS%	Code that computes all local and output signals from the current inputs and states.	X	X
%STATE_UPDATE_EQUATIONS%	Code that transitions the state variables	X	X

	to the next cycle.		
%SAMPLE_TIME%	The real time in seconds corresponding to one execution cycle.	X	
%FREQUENCY_MHZ%	The frequency in MHz of the clock that will drive the implementation.		X
%INPUT_MAP%	Code that copies input signals from the entity's interface to the arrays.		X
%OUTPUT_MAP%	Code that copies output signals from the arrays to the entity's interface.		X

%*type*_VECTOR%

The name of the *type* array. *type* can be any of BOOL, INT and REAL, such that the actual keys are %BOOL_VECTOR%, %INT_VECTOR% and %REAL_VECTOR%.

Rather than declaring one single variable (in VHDL: signal) for each signal of the RTflow model, the code generator expects the existence of arrays (in VHDL: vectors), in which all signals reside. Hence, the generated code is merely a sequence of computations on elements of these arrays. There must be one array for each of the basic types boolean, integer and real, and it is the template author's responsibility to declare these appropriately. To do this, the array names that are used in the generated code must be known, and these are given by the %*type*_VECTOR% key.

Generators

C/C++/Java, VHDL

Example (Java)

```
%IF_HAS_REAL%   public float[] %REAL_VECTOR% = new float[%NUMBER_REAL%];
```

Example (VHDL)

```
%IF_HAS_BOOL%   signal %BOOL_VECTOR% : std_logic_vector(%NUMBER_BOOL%-1
downto 0);
```

%IF_HAS_*type*%

Includes the line if and only if there are *type* signals. *type* can be any of BOOL, INT and REAL.

If this key is found in a line of the template file during code generation, then that whole line will only be processed if the number of variables of type *type* is greater than 0. This is useful for avoiding declarations of arrays with zero elements, which is forbidden in some target languages.

Generators

C/C++/Java, VHDL

Example (Java)

```
%IF_HAS_REAL%   public float[] %REAL_VECTOR% = new float[%NUMBER_REAL%];
```

%*type*_NAMES%

The names of the signals and states in the *type* array. type can be any of BOOL, INT and REAL.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are signals of type *type*, and in each copy, the %*type*_NAMES% key is replaced by the name of the signal residing in the corresponding cell of the array. As a result, this key generates the complete list of the names of all signals in the *type* array. This information is used to associate an array index with an actual signal in the RTflow model.

The names of signals in the top-level node are provided without modifications. However, for signals in subnodes, the names are prepended with the instance path to the subnode containing the signal. Hence, if the subnode A is a subnode of the top-level node, and subnode A has a signal i, then that signal will be given the name "A/i" in the generated name list.

Generators

C/C++/Java

Example (Java)

```
public String[] boolNames = new String[]
{
   "%BOOL_NAMES%",
   ""
};
```

%*type*_INPUT_NAMES%

The names of the input signals of type *type*. *type* can be any of BOOL, INT and REAL.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are input signals of type *type*, and in each copy, the %*type*_INPUT_NAMES% key is replaced by the name of an input signal.

An *input* signal in the code generation context is an input signal of the top-level node. Inputs of subnodes count as local signals.

Generators

VHDL

Example (VHDL)

```
entity RTflowCode is
   port(
      %BOOL_INPUT_NAMES% : in std_logic;
      %INT_INPUT_NAMES% : in integer;
      %REAL_INPUT_NAMES% : in float32;
      %BOOL_OUTPUT_NAMES% : out std_logic;
      %INT_OUTPUT_NAMES% : out integer;
      %REAL_OUTPUT_NAMES% : out float32;
      clk : in std_logic
   );
end RTflowCode;
```

%*type*_OUTPUT_NAMES%

The names of the output signals of type *type*. *type* can be any of BOOL, INT and REAL.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are output signals of type *type*, and in each copy, the %*type*_OUTPUT_NAMES% key is replaced by the name of an output signal.

An *output* signal in the code generation context is an output signal of the top-level node. Outputs of subnodes count as local signals.

Generators

VHDL

Example

See %*type*_INPUT_NAMES%.

%NUMBER_*type*%

The total number of *type* signals. *type* can be any of BOOL, INT and REAL.

This key provides the size of the array for each basic type. See %*type*_VECTOR% for details about the arrays in code generation.

Generators

C/C++/Java, VHDL

Example (Java)

```
%IF_HAS_REAL%   public float[] %REAL_VECTOR% = new float[%NUMBER_REAL%];
```

Example (VHDL)

```
%IF_HAS_BOOL%   signal %BOOL_VECTOR% : std_logic_vector(%NUMBER_BOOL%-1
downto 0);
```

%NUMBER_*type*_INPUT%

The number of *type* input signals. *type* can be any of BOOL, INT and REAL.

An *input* signal in the code generation context is an input signal of the top-level node. Inputs of subnodes count as local signals.

Generators

C/C++/Java, VHDL

%NUMBER_*type*_OUTPUT%

The number of *type* output signals. *type* can be any of BOOL, INT and REAL.

An *output* signal in the code generation context is an output signal of the top-level node. Outputs of subnodes count as local signals.

Generators

C/C++/Java, VHDL

%NUMBER_*type*_LOCAL%

The number of *type* local signals. *type* can be any of BOOL, INT and REAL.

An *local* signal in the code generation context is any signal that is not an input or an output of the top-level node. Hence, all signals of the subnodes count as local signals in this context.

Generators

C/C++/Java, VHDL

%NUMBER_*type*_STATE%

The number of *type* state variables. *type* can be any of BOOL, INT and REAL.

A *state variable* is the holder of a value from the previous cycle, corresponding to a Pre block. State variables don't exist explicitly in the abstract RTflow model, but are necessary in the generated implementation code. See %STATE_UPDATE_EQUATIONS% for details about how state variables are used.

Generators

C/C++/Java, VHDL

%OFFSET_*type*_INPUT%

The index of the first input signal in the *type* array. *type* can be any of BOOL, INT and REAL.

Each of the arrays (one for each of the basic types boolean, integer and real) is partitioned into input signals, output signals, local signals and state variables. This key provides the index of the first input signal in the array. Hence, the index of the *n*:th boolean input signal in the boolean array is %OFFSET_BOOLEAN_INPUT% + *n*.

An *input* signal in the code generation context is an input signal of the top-level node. Inputs of subnodes count as local signals.

Generators

C/C++/Java, VHDL

%OFFSET_*type*_OUTPUT%

The index of the first output signal in the *type* array. *type* can be any of BOOL, INT and REAL.

Each of the arrays (one for each of the basic types boolean, integer and real) is partitioned into input signals, output signals, local signals and state variables. This key provides the index of the first output signal in the array. Hence, the index of the *n*:th boolean output signal in the boolean array is %OFFSET_BOOLEAN_OUTPUT% + *n*.

An *output* signal in the code generation context is an output signal of the top-level node. Outputs of subnodes count as local signals.

Generators

C/C++/Java, VHDL

%OFFSET_*type*_LOCAL%

The index of the first local signal in the *type* array. *type* can be any of BOOL, INT and REAL.

Each of the arrays (one for each of the basic types boolean, integer and real) is partitioned into input signals, output signals, local signals and state variables. This key provides the index of the first local signal in the array. Hence, the index of the *n*:th boolean local signal in the boolean array is %OFFSET_BOOLEAN_LOCAL% + *n*.

An *local* signal in the code generation context is any signal that is not an input or an output of the top-level node. Hence, all signals of the subnodes count as local signals in this context.

Generators

C/C++/Java, VHDL

%OFFSET_*type*_STATE%

The index of the first state variable in the *type* array. *type* can be any of BOOL, INT and REAL.

Each of the arrays (one for each of the basic types boolean, integer and real) is partitioned into input signals, output signals, local signals and state variables. This key provides the index of the first state variable in the array. Hence, the index of the *n*:th boolean state variable in the boolean array is %OFFSET_BOOLEAN_LOCAL% + *n*.

A *state variable* is the holder of a value from the previous cycle, corresponding to a Pre block. State variables don't exist explicitly in the abstract RTflow model, but are necessary in the generated implementation code. See %STATE_UPDATE_EQUATIONS% for details about how state variables are used.

Generators

C/C++/Java, VHDL

%INITIAL_EQUATIONS%

Code that resets all state variables to the initial state.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are state variables of all types, and in each copy, the %INITIAL_EQUATIONS% key is replaced by an assignment statement of a state variable to its initial value. As a result, this key generates the code that should be executed once in the initialization of the process. None of the input, output or local signals are assigned or referred in this code.

Generators

C/C++/Java, VHDL

Example (C)

```
void initialize()
{
   %INITIAL_EQUATIONS%
}
```

%OUTPUT_UPDATE_EQUATIONS%

Code that computes all local and output signals from the current inputs and states.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are output and local signals of all types, and in each copy, the %OUTPUT_UPDATE_EQUATIONS% key is replaced by a statement where a local or output signal is computed and assigned. As a result, this key generates the code that, together with %STATE_UPDATE_EQUATIONS%, should be executed for each cycle.

Generators

C/C++/Java, VHDL

Example (C)

```
void run()
{
   %OUTPUT_UPDATE_EQUATIONS%
   %STATE_UPDATE_EQUATIONS%
}
```

Example (VHDL)

```
process (clk)
begin
   if reset = '1' then
      %INITIAL_EQUATIONS%
   elsif clk'event and clk = '1' then
      %STATE_UPDATE_EQUATIONS%
   endif;
end process;
%OUTPUT_UPDATE_EQUATIONS%
```

%STATE_UPDATE_EQUATIONS%

Code that transitions the state variables to the next cycle.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are state variables of all types, and in each copy, the %STATE_UPDATE_EQUATIONS% key is replaced by an assignment statement of a state variable to a value computed by the last %OUTPUT_UPDATE_EQUATIONS% execution. In schematic terms, this corresponds to that the values of signals connected to the input of Pre blocks are transferred into the memory registers (state variable) that the Pre blocks constitute. As a result, this key generates the code that, together with %OUTPUT_UPDATE_EQUATIONS%, should be executed for each cycle.

Generators

C/C++/Java, VHDL

Example

See %OUTPUT_UPDATE_EQUATIONS%.

%SAMPLE_TIME%

The real time in seconds corresponding to one execution cycle.

Generators

C/C++/Java

Example (Java)

```
public float getSampleTime()
{
   return %SAMPLE_TIME%;
}
```

%FREQUENCY_MHZ%

The frequency in MHz of the clock that will drive the implementation.

Generators

VHDL

%INPUT_MAP%

Code that copies input signals from the entity's interface to the arrays.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are input signals of all types, and in each copy, the %INPUT_MAP% key is replaced by an assignment statement that copies a VHDL entity input signal to an input signal in the array. The entity input signal has the same name as the corresponding input of the top-level node. Hence, the entity should be declared as in the example for %*type*_INPUT_NAMES%.

Generators

VHDL

Example (VHDL)

```
process (clk)
begin
   if clk'event and clk = '1' then
      %INPUT_MAP%
   end if;
end process;
```

%OUTPUT_MAP%

Code that copies output signals from the arrays to the entity's interface.

If this key is found in a line of the template file during code generation, then that whole line will be copied the same number of times as there are output signals of all types, and in each copy, the %OUTPUT_MAP% key is replaced by an assignment statement that copies an output signal in the array to a VHDL entity output signal. The entity output signal has the same name as the corresponding output of the top-level node. Hence, the entity should be declared as in the example for *%type* INPUT_NAMES%.

Generators

VHDL

Compiler Errors

The following table summarizes error messages that may be output to the message view as a result of a Compile, Simulate or Generate Code command. Detailed descriptions of the cause of these errors and suggestions for how to resolve them are given in the subsequent subsections.

Notice. The compiler may output error messages that are not listed here. There may be two different reasons for this:

- A project or a schematic file has been modified manually.
- There is an internal error in the compiler. In this case, it should be reported in the RTflow forums.

Number	Message
2001	Causality loop involving the following signals: *signals*
2002	Output *port* has no source in node *node*.
2006	Recursion in node *node*.
2007	Equation *equation* not resolved in node *node*.
3004	Unbound input *port* in equation *equation* of node *node*.
3017	Type mismatch for signal *signal* (*type1* / *type2*) in node *node*.
4001/4101	Could not open template file *filename*.
4002/4102	Could not open *filename* for writing.
4004/4104	Error writing *filename*.
4010/4110	Unknown key *key*.

2001: Causality loop involving the following signals: *signals*

Cause

There is a causality loop in the model, which means that a set of blocks that are connected in a loop without any delay (Pre) block. Causality loops are forbidden in RTflow, because it may be difficult or impossible to compile them into executable code. The sequence of signals that constitute the loop are given in the error message.

Resolution

Insert a Pre block anywhere in the loop.

2002: Output *port* has no source in node *node*.

Cause

An output port block is not connected to any source, which would leave the output value undefined.

Resolution

Connect the block to any output port instance, or create a new Value block instance in the schematic and connect it to the output port block.

2006: Recursion in node *node*.

Cause

A schematic contains a block instance of the block that it defines, possibly through other blocks, which causes an infinite recursion. For example, if schematic A contains an instance of block B, and schematic B contains and instance of block A, then the schematics A and B constitute a recursion.

Resolution

Remove the block instance involved in the recursion from any of the involved schematics.

2007: Equation *equation* not resolved in node *node*.

Cause

A schematic file contains an instance of a block that doesn't exist. This can happen if a schematic file has been removed from the project without properly updating other schematics that use the removed block.

Resolution

Either add the missing schematic file to the project, or open the failing file, in which case the failing block instances will be removed automatically.

3004: Unbound input *port* in equation *equation* of node *node*.

Cause

An input port instance is not connected to any source, which would leave the input value undefined.

Resolution

Connect the input port instance to any output port instance, or create a new Value block instance in the schematic and connect it to the input port instance.

3017: Type mismatch for signal *signal* (*type1* / *type2*) in node *node*.

Cause

A block has been connected to signals of different types, violating a requirement that all connected signals must have the same types. For example, if an Add block is connected to two integer input port blocks and to a real output port block, then a type mismatch occurs, since all ports of the Add block must be connected to signals of the same type.

Resolution

Insert a type conversion block (Bool, Int or Real) in the failing connection.

4001/4101: Could not open template file *filename*.

Cause

The template file given in the code generation settings could not be opened, probably because it doesn't exist.

Resolution

In the Code Generation Settings, click **Browse...** by **Template File** and select an existing template file.

4002/4102: Could not open *filename* for writing.

Cause

The output file given in the code generation settings could not be opened for writing, probably due to one of the following:

1. The folder doesn't exist.
2. The file already exists and is write-protected.
3. The file already exists and is open in another application.

Resolution

In the Code Generation Settings, click **Browse...** by **Output File** and select an existing folder and a new file name.

4004/4104: Error writing *filename*.

Cause

A disk write error occurred while generating code, possibly because the disk is full or corrupt.

Resolution

Solve the disk problem, or open the Code Generation Settings dialog, click **Browse...** by **Output File** and select another disk.

4010/4110: Unknown key *key*.

Cause

The template file contains a key, that is, a % followed by a number of capital letters and underscores followed by a %, that is not known by the code generator. Possibly the key was misspelled, or it is only available for another code generator.

Resolution

Open the template file and remove or correct the key.