

Qfsm User Manual

Stefan Duffner

Camille Decock

Qfsm User Manual

Stefan Duffner
Camille Decock

Version 0.54

Copyright © 2000-2015 Stefan Duffner, Rainer Strobel

Table of Contents

1. Introduction	1
1.1. What is Qfsm?	1
1.2. Copyright and license information	1
1.3. Installing the Qfsm software	2
1.3.1. Binary installation	2
1.3.2. Installing from source code	2
2. Using Qfsm	4
2.1. Using the main menu	4
2.1.1. File	4
2.1.2. Edit	4
2.1.3. View	5
2.1.4. Machine	5
2.1.5. State	6
2.1.6. Transition	6
2.2. Creating and modifying a Finite State Machine	6
2.3. Using the Working Area	8
2.3.1. Using the Select mode	9
2.3.2. Using the Pan mode	9
2.3.3. Using the Zoom mode	9
2.3.4. Using the Add State mode	10
2.3.5. Using the Add Transition mode	10
2.3.6. Using the Simulate mode	10
2.4. Adding and modifying states	10
2.5. Adding and modifying transitions	12
2.6. Using input ASCII conditions	13
2.6.1. Single character	14
2.6.2. Multiple characters	14
2.6.3. Escape sequences	14
2.6.4. Ranges	14
2.6.5. Mixed formats	14
2.7. Checking the integrity of a FSM	15
2.8. Simulating a FSM	15
2.9. Exporting	17
2.9.1. State diagrams	17
2.9.2. Hardware description languages	17
2.9.3. State Tables	18
2.9.4. Code generation languages	18
2.9.5. Hardware test code	20
2.9.6. I/O description	21
2.9.7. vvvv Automata code	21
2.9.8. State Chart XML	22
2.10. Importing	22
2.11. Setting the options	23
2.11.1. Changing language	23
2.11.2. Changing the display	23
2.11.3. Printing	24
3. Contact	25
Index	26

List of Tables

2.1. Attributes of a Finite State Machine	7
2.2. Attributes of a state	11
2.3. Attributes of a transition	12
2.4. Recognized escape sequences	14

Chapter 1. Introduction

1.1. What is Qfsm?

Qfsm is a graphical editor for finite state machines written in C++ using Qt the graphical Toolkit from Trolltech [<http://www.trolltech.com>].

Finite state machines are models to describe complex objects or systems in terms of the states they may be in. In practice they can be used to create regular expressions, scanners or other program code as well as for integrated circuit design.

Current features of Qfsm are:

- Drawing, editing and printing of states diagrams
- Binary, ASCII and "free text" condition codes
- Integrity check
- Interactive simulation
- Diagram export (EPS, SVG and PNG format)
- AHDL/VHDL/Verilog HDL/KISS/vvvv Automata code export
- State table export in Latex, HTML and plain text format
- Ragel file export (used for C/C++, Java or Ruby code generation, for example for string parsers)
- State Chart XML (SCXML) export
- State Machine Compiler (SMC) export

1.2. Copyright and license information

Copyright (C) 2000-2015 Stefan Duffner, Rainer Strobel

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License [LICENSE] as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

1.3. Installing the Qfsm software

1.3.1. Binary installation

1.3.1.1. Windows

There exists a (graphical) Windows installer that can be downloaded from here [<http://www.sourceforge.net/projects/qfsm>]. The installation should be straightforward and doesn't require any other software to be installed.

1.3.1.2. Linux

An RPM package for openSUSE is provided here [<http://www.sourceforge.net/projects/qfsm>]. RPM packages for other Linux distributions might be provided on other websites.

1.3.1.3. Mac OS X

Binary packages for MacOS X may be provided by third-party websites.

1.3.2. Installing from source code

Qfsm has to be compiled using the CMake buildsystem. Among the advantages of CMake are that it provides excellent cross-platform support, that it can create project files for various IDEs such as KDevelop, XCode and MS Visual Studio and that it builds out-of-source, keeping the source tree clean of temporary files.

1.3.2.1. Requirements

- CMake [<http://www.cmake.org>], version 2.6 or higher
- Qt SDK [<http://qt.nokia.com>], version 4.8
- Graphviz [<http://graphviz.org>], version 2.38 or higher

See the documentation of these tools/libraries for an explanation how to install them.

1.3.2.2. Supported Platforms

In principal, Qfsm compiles and runs on any platform supported by the Qt library version 4.8 (or higher), i.e. Linux/Unix (e.g. AIX, FreeBSD, HP-UX, IRIX, Solaris), Windows (98, NT 4.0, ME, 2000, XP and Vista) and Mac OS X (version 10.3.9 and higher).

1.3.2.3. Building under GNU Linux/Unix

1. Install the above-mentioned required software. If you install them via a package manager make sure you install the development packages (...-dev).
2. Unpack the source code into the current directory
`tar xvfz qfsm-0.54.0-Source.tar.gz --directory .`
and change to that directory.
3. Before invoking CMake you have to make sure that the program qmake is in your global PATH environment variable. The location of qmake depends on your operating system or Linux distribution. On

openSUSE for example it is under: `/usr/lib/qt4/bin/` (or simply under `/usr/bin` on more recent distributions). Thus, if you are using bash, you would have to type:
export PATH=/usr/lib/qt4/bin:\$PATH

4. Type:
cmake .

5. Then:
make

6. Finally, install Qfsm (as root):
make install

This will install the executable "qfsm" to `/usr/bin/` and the documentation to `/usr/share/doc/qfsm/`.

If CMake fails to find any of the dependencies but you know you have the development headers/libraries installed, add `-DLIBRARY_SEARCH_DIRS=<path/to/lib>` and `-DINCLUDE_SEARCH_DIRS=<path/to/include>` to the cmake command.

If you want to create a project file for your IDE (e.g. KDevelop), add `-GKDevelop3` to the cmake command above. This will create a `.kdevelop` file in the build directory which can be opened by KDevelop.

1.3.2.4. Building under Windows

In order to build Qfsm under Windows you have to execute the program CMakeSetup, specify input (source) and output directory and click on "Configure". You may choose between different development environments (e.g. Visual Studio) and CMake will create the respective project files. You can then compile Qfsm as usual with your preferred build tool. For more details, refer to the CMake documentation: <http://www.cmake.org/HTML/Documentation.html>.

Chapter 2. Using Qfsm

2.1. Using the main menu

This section briefly explains the functions available through the main menu.

2.1.1. File

To create a new file:	select “New”. For more details, see Section 2.2, “Creating and modifying a Finite State Machine”
To open an existing Qfsm file:	select “Open”.
To open the most recently opened Qfsm files:	select “Open Recent”.
To save the current FSM to a Qfsm file:	select “Save”.
To save the current FSM under a different name:	select “Save As”.
To export the current FSM to a foreign file format:	select “Export”. For more details, see Section 2.9, “Exporting”
To print the current FSM:	select “Print”.
To open a new window with a separate working area where a different FSM can be edited:	select “New Window”. Note that you can copy, cut and paste states and transitions from/to different FSMs.
To close the current FSM:	select “Close”.
To quit Qfsm:	select “Quit”.

2.1.2. Edit

To undo the last action:	select “Undo”.
To cut the currently selected states and transitions to the clipboard:	select “Cut”.
To copy the currently selected states and transitions to the clipboard:	select “Copy”.
To paste the clipboard into the current FSM:	select “Paste”.
To delete the currently selected states and transitions:	select “Delete”.
To switch to the select mode:	select “Select”. For more details, see Section 2.3.1, “Using the Select mode”
To select all states and transitions of the current FSM:	select “Select All”.
To deselect all objects:	select “Deselect All”.
To open the options dialog:	select “Options”. For more details, see Section 2.11, “Setting the options”

2.1.3. View

To show/hide the state codes inside the states:	select “State Codes”. Each state has a unique identifier, called state code, which is an integer that is automatically determined by Qfsm.
To show/hide the Moore outputs inside the states:	select “Moore Outputs”. Each state defines its Moore outputs which are the values that are sent to the outputs of the FSM when the respective state is reached.
To show/hide the Mealy input conditions on the transitions:	select “Mealy Inputs”. Mealy inputs are asynchronous inputs to the FSM. They can trigger transitions from one state to another if the condition of the respective transition is satisfied.
To show/hide the Mealy outputs on the transitions:	select “Mealy Outputs”. Mealy outputs are outputs of the FSM that were sent when a transition is triggered. Thus, each transition can define the Moore outputs that are sent when it is triggered.
To show/hide the shadows of the states:	select “Shadows”.
To show/hide the grid on the working area:	select “Grid”.
To show/hide the window displaying the names of the inputs and outputs of a finite state machine:	select “IO View”.
To switch to the pan mode:	select “Pan View”. For more details, see Section 2.3.2, “Using the Pan mode”
To switch to the zoom mode:	select “Zoom”. For more details, see Section 2.3.3, “Using the Zoom mode”
To zoom the view in:	select “zoom in”. The current zoom value is shown in the leftmost part of the status bar.
To zoom the view out:	select “zoom out”. The current zoom value is shown in the leftmost part of the status bar.
To set the zoom to the original value (100%):	select “Zoom 100%”. The current zoom value is shown in the leftmost part of the status bar.

2.1.4. Machine

To modify the properties of the current FSM:	select “Edit”. It opens a dialog in which you can modify the properties of the FSM. For more details, see Section 2.2, “Creating and modifying a Finite State Machine”
To correct automatically the codes (or identifiers) of all states such that each state code is unique:	select “Auto correct State Codes”
To switch to the simulation mode:	select “Simulate”. For more details, see Section 2.3.6, “Using the Simulate mode” and Section 2.8, “Simulating a FSM”
To perform an integrity check on the current FSM:	select “Integrity Check”. For more details, see Section 2.7, “Checking the integrity of a FSM”

2.1.5. State

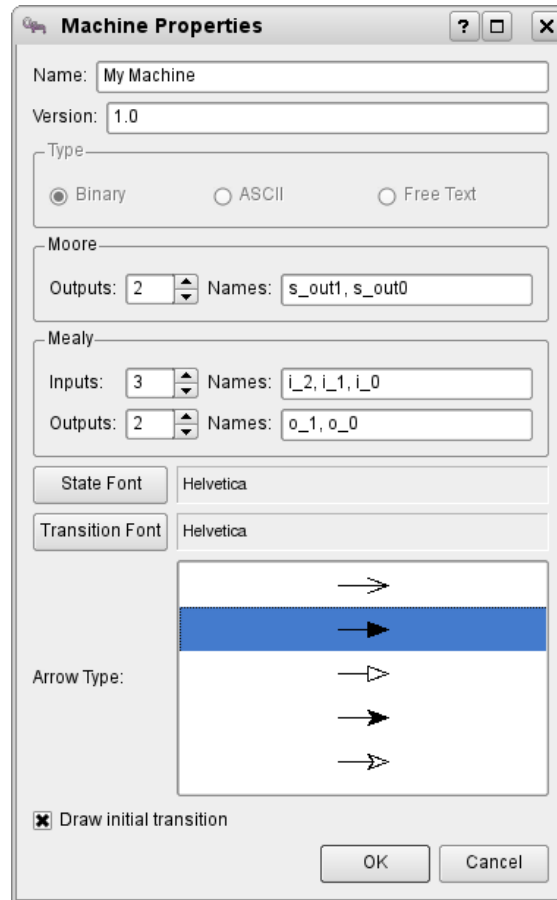
To switch to the "add state" mode and create a new state:	select "New". For more details, see Section 2.3.4, "Using the Add State mode" and Section 2.4, "Adding and modifying states"
To modify the properties of the currently selected state:	select "Edit". It opens a dialog in which you can modify the properties of the state. For more details, see Section 2.4, "Adding and modifying states"
To define the currently selected state as the initial state of the FSM:	select "Set Initial State".
To define the currently selected state as a final or non-final state:	select "Toggle Final State".

2.1.6. Transition

To switch to the "add transition" mode and create a new transition:	select "New". For more details, see Section 2.3.5, "Using the Add Transition mode" and Section 2.5, "Adding and modifying transitions"
To modify the properties of the currently selected transition:	select "Edit". It opens a dialog in which you can modify the properties of the transition. For more details, see Section 2.5, "Adding and modifying transitions"
To straighten the currently selected transition:	select "Straighten".

2.2. Creating and modifying a Finite State Machine

To create a new Finite State Machine (FSM), choose the menu item *File->New*. The "Machine properties" dialog opens and lets you specify the properties of the FSM. Here is an explanation of each item of the dialog.



The image shows a 'Machine Properties' dialog box with the following fields and options:

- Name:** My Machine
- Version:** 1.0
- Type:**
 - ☒ Binary
 - ☐ ASCII
 - ☐ Free Text
- Moore:**
 - Outputs:** 2
 - Names:** s_out1, s_out0
- Mealy:**
 - Inputs:** 3
 - Names:** i_2, i_1, i_0
 - Outputs:** 2
 - Names:** o_1, o_0
- State Font:** Helvetica
- Transition Font:** Helvetica
- Arrow Type:** A selection box showing five arrow styles. The second style, a solid black arrow, is highlighted with a blue background.
- ☒ Draw initial transition
- Buttons:** OK, Cancel

Table 2.1. Attributes of a Finite State Machine

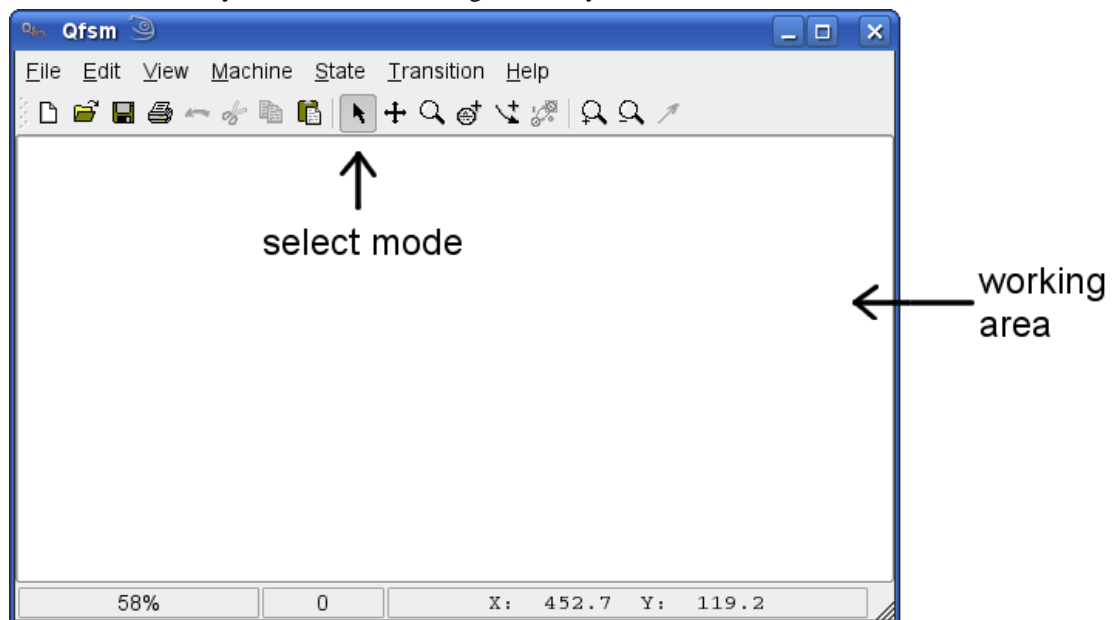
Name	Enter a name for the FSM. The name is used for example when exporting to the following formats: VHDL, Verilog HDL or Ragel.
Version	You can enter the version of the FSM. This is a free character string that is only used when printing the diagram.
Type	<p>Select the type. The type attribute determines which type of information is processed by the FSM, i.e. the inputs, the outputs etc. Binary FSMs process zeros and ones at the inputs or outputs. This is the main type used for hardware design.</p> <p>ASCII FSMs process characters (i.e. letters, digits etc.). These characters are coded in ASCII format using 8 bits. This type of FSM can be used either for hardware design or to create string parsers.</p> <p>The "Free Text" type allows to specify inputs and outputs using any kind of character string of variable length. This type of FSM cannot be simulated afterwards because the input conditions won't be interpreted.</p>

Moore outputs and Mealy inputs/outputs	If you are creating a "binary" FSM you can specify the number of bits of the moore output and mealy input/output and their respective names. The names are lists of character strings separated by commas. If you do not want to choose the names you can leave these fields blank and they will be automatically set.
Fonts	You can specify the font to use for the state names and for the input conditions and outputs displayed on the transitions.
Arrow Type	You can choose the type of arrow to use for drawing transitions.
Draw initial transition	You can tick or not this option if you want to draw or not the initial transition or if you want to reset or start the transition.

When you want to modify the properties of an existing FSM you can select *Machine->Edit* from the main menu and the same dialog box displays. As soon as you click OK the changes take effect.

2.3. Using the Working Area

The working area denotes the area of the Qfsm window that shows the state diagram. Once you have created a new FSM you see a blank working area and you are in the *selectmode*.



There are six different modes you can be in and which determine what happens when you click or drag the mouse inside the working area of Qfsm.

1. Select
2. Pan
3. Zoom
4. Add State
5. Add Transition

6. Simulate

Only one mode can be activated at a time. To change the mode, click on one of the icons in the middle of the toolbar.

When a diagram is larger the working area, you have the possibility to move the view of a diagram. To move the view of a diagram, click on the middle mouse button and drag the mouse pointer. As soon as you release the middle mouse button the application reverts to the selected mode.



Alternatively, you can select the respective menu entry or press the respective short cut. The active mode is indicated by a highlighted toolbar button. In some modes, the form of the mouse cursor also changes, e.g. a magnifier for the zoom mode.

2.3.1. Using the Select mode

In this mode you can select, move or modify graphical objects.

To **select a state or a transition**, click on it with the left mouse button.

To **select several states or transitions at the same time**, hold down the shift key. You can then apply further actions on selected graphical objects, i.e. copy or edit, by using the menu.

To **unselect all the selected graphical objects**, click on the background.

To **show the context menu for a state or a transition**, click with the right mouse button on the state or transition.

To **modify the properties of a state or a transition**, double-click on the state or transition. It opens a dialog in which you can modify the properties.

To **select multiple graphical objects**, draw a rectangle holding the left mouse button around the graphical objects you want to select.

To **move state or transitions**, select them and drag them to the desired position.

To **move the transition control points**, select the respective transition and then drag one of the control points. The transition control points are indicated by small red and green points when a transition is selected. The red points control the form of the transition, i.e. the bend. The green ones are used to attach them to a starting and end state.

2.3.2. Using the Pan mode



In this mode, you can move the view to a different part of your diagram when your diagram is larger than the working area of the window.

To move the view, drag the mouse pointer.

2.3.3. Using the Zoom mode



In this mode you can zoom in the view.

To zoom in the view, click with the left mouse button on the working area.

To zoom out, keep the CTRL key pressed at the same time you click.

2.3.4. Using the Add State mode



In this mode you can add states and specify their properties.

To create a new state, click with the left mouse button and fill the dialog with the properties. For more details, see Section 2.4, “Adding and modifying states”

2.3.5. Using the Add Transition mode



In this mode you can add transitions and specify their properties.

To add a transition:

- 1) click on a state and drag the mouse pointer to another state.
- 2) fill the dialog with the properties of the transition.

For more details, see Section 2.5, “Adding and modifying transitions”

2.3.6. Using the Simulate mode



In the simulate mode, you can test the behaviour of your state machine with respect to external input.

When you enter this state, the simulator dialog opens and all interaction with the state diagram is disabled until you close the dialog. For more details, see Section 2.8, “Simulating a FSM”

2.4. Adding and modifying states

To create a state:

- 1) first you need to:

- create a new FSM or;
- open an existing file.

To create a new FSM, select *File->New* (see Section 2.2, “Creating and modifying a Finite State Machine”)

To open an existing file, select *File->Open* and choose the desired file.

- 2) Switch to the "add state" mode (see Section 2.3.4, "Using the Add State mode").
- 3) Click at the position of the working area where you want the new state to be.

The following dialog opens.

To specify or modify the properties of the state, fill the "State properties" dialog:

Table 2.2. Attributes of a state

Name	Enter a name for the state.
Code	Normally you do not need to fill this field. This is a unique identifier of the state you create, it is filled automatically.
Moore Outputs	Enter the outputs sent by the FSM. In "binary" FSMs this is a string of zeros and ones and in "ASCII" FSMs this is just one character.
Entry actions	Enter the actions (functions, e.g. "doThis()") that shall be executed when entering the state. Used only for "free text" machines and the SMC export.
Exit actions	Enter the actions (functions, e.g. "doThat()") that shall be executed when exiting the state. Used only for "free text" machines and the SMC export.
Radius	Enter the radius of the drawn circle of the state, in pixels.
Line width	Enter the line width of the outline of the state.
Colour	If you want to change the colour of the outline of the state, click on the "Colour" button.
Description	You can enter a description of the state. This is only for documentation purposes.

To modify an existing state:

1) switch to the "select mode" (see Section 2.3.1, "Using the Select mode").

2) double-click on the respective state

Or

2) select one state and select *State->Edit* from the main menu.

2.5. Adding and modifying transitions

To create a new transition:

1) you need first to:

- create a FSM or

- open an existing file.

To create a FSM, select *File->New* (see Section 2.2, "Creating and modifying a Finite State Machine")

To open an existing file, select *File->Open* from the main menu and choose the desired file.

2) create at least one state (see Section 2.4, "Adding and modifying states").

3) switch to the "add transition" mode (see Section 2.3.5, "Using the Add Transition mode").

To create a transition from state A to state B, press and hold the left mouse button on state A and release it on state B.

To draw loops, i.e. transitions that go from one state to itself, press and release the mouse button on the same state.

The "Transition properties" dialog opens. To specify or modify the properties of the state, fill the dialog.

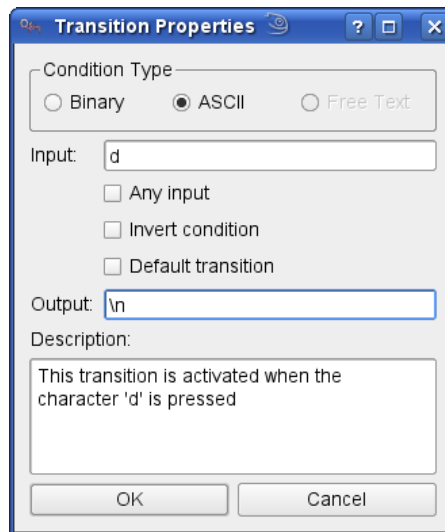


Table 2.3. Attributes of a transition

Condition Type	Choose the type of the condition. The type of the condition determines the format in which you enter the input condition in the next field.
----------------	---

Input	<p>If the condition type is binary, enter a string of zeros and ones representing the Mealy input that should trigger this transition. You can also use the character 'x' meaning: "don't care". You can also specify several alternative sets of inputs linked with " " (OR).</p> <p>If the condition type is ASCII enter a character or specify an expression in a specific format which is explained in detail in Section 2.6, "Using input ASCII conditions".</p> <p>For "free text" conditions enter any input character string. However, it has no logical meaning and won't be interpreted, for example when simulating the machine. For SMC export, put function names here (e.g. "suspend()").</p>
Output	<p>Enter the Mealy output sent from the FSM when the transition is activated, i.e. the input condition is satisfied. Depending on the type of the condition the format is either a string of zeros and ones (binary) a character (ASCII) or any character string (free text). Note that in case of an ASCII character it can also be an escape sequence. For more details on escape sequences, see Section 2.6, "Using input ASCII conditions" For SMC export, put function names here (e.g. "suspend()").</p>
Description	<p>You can enter a description of the transition. This is only for documentation purposes.</p>

To modify the properties of an existing transition:

- 1) switch to the "select mode" (see Section 2.3.1, "Using the Select mode").
- 2) double-click on the respective transition

Or

- 2) select one transition and select *Transition->Edit* from the main menu.

To change the bend of the transition as well as its start state and end state:

- 1) switch to the "Select mode"
- 2) click on the transition.

Four control points appear. You can drag them around with the left mouse button.

The green ones allow you to change the start and end state. With the red ones you can change the bend of the transition.

2.6. Using input ASCII conditions

When you create a transition of FSM that processes ASCII characters you have to enter an input condition. This condition can be a simple character, e.g. 'a', or several characters that are expressed by a special notation explained in the following.

2.6.1. Single character

This is the most simple form of condition. It contains one ASCII character, e.g. 'a' or 'z'.

Note that for special characters, e.g. '-' (minus sign) or the space character you need to use an escape sequence (see Section 2.6.3, “Escape sequences”).

2.6.2. Multiple characters

If you want the condition to contain multiple characters, i.e. 'a' or 'f' or '+' you just enter the string: 'af+'. Clearly, the order is not important.

Note that it *is not* possible to use a concatenation of characters as input condition, for example 'print' in order to recognize the word "print". To do this, you have to create a transition and a state for each character and build a chain with the respective characters.

2.6.3. Escape sequences

Special characters like the newline character need to be escaped, i.e. backslash + some character. The following table shows the recognized escape sequences.

Table 2.4. Recognized escape sequences

escape sequence	meaning
\t	tab
\n	newline
\r	carriage return
\s	space
\-	minus
\d	digit (0-9)

Note that the last escape sequence '\d' actually represents 10 characters.

Characters that are neither printable nor in the above table can be specified by '\0' (backslash zero) followed by their hexadecimal code. For example, '\0CF' would represent the ASCII character 207 (decimal).

2.6.4. Ranges

You can further specify ranges by using the minus sign. Thus, 'a-z' means one of the characters between 'a' and 'z' (including). Any character, even escaped ones, can be used as start or end point of a range.

2.6.5. Mixed formats

You can combine several conditions, each of them in one of the above mentioned notations, into one long condition by just concatenating them. Note that you must not separate them by any character, like white space or comma.

Here are some examples:

- A-F0-9
- +\-\d
- \n\r\tXYZ
- xyz0-3\010A-Z

2.7. Checking the integrity of a FSM

To check the integrity of a FSM, select *Machine->Integrity Check* from the main menu.

Warning

This may take a long time for larger FSMs, be careful not to interrupt the procedure.

The following tests are performed:

Unambiguous Conditions	Checks if the FSM has transitions with conditions that are ambiguous, i.e. transitions that are activated simultaneously by the same input (in the same state). Note that ambiguous transitions are only allowed in <i>non-deterministic</i> FSMs, which are currently not supported by Qfsm.
Initial state	Checks if the FSM has an initial state.
Final state	Checks if the FSM has a final state.
No dead locks	Checks if the FSM has states where it can get of out, i.e. states with no transitions going out.
Completeness	Checks if for every possible input in every state there exists a transition that is activated.
States reachable	Checks if all the states of the FSM are reachable.
Final states reachable	Checks if all the final states of the FSM are reachable.
Transitions connected	Checks if all the transitions of the diagram are actually connected to a start end an end state. Note that sometimes a transition looks as if it is connected to a state but in fact the connection point is slightly away from it.

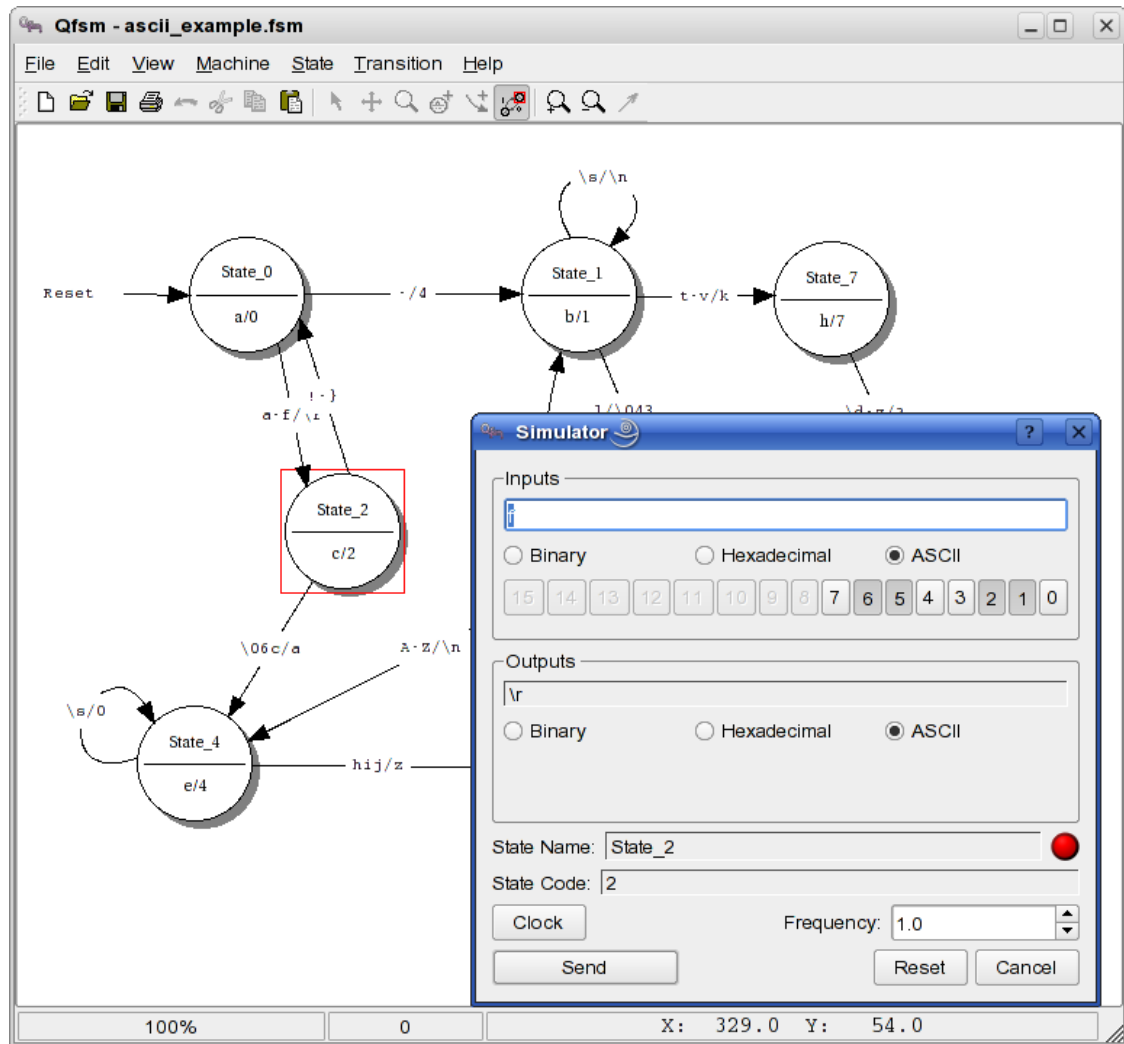
A log of the checks that have been performed is printed in the protocol field at the bottom of the window.

2.8. Simulating a FSM

Once you have created a FSM with some states and transitions you can simulate its behaviour with respect to varying input signals.

To start the simulation of an FSM, select *Machine->Simulate* from the main menu.

The "simulator" dialog opens allowing you to input data to the machine while displaying its current state and output.



Input	<ol style="list-style-type: none"> 1) Enter the input data in the text field 2) select one of the formats: binary, hexadecimal or ASCII. 3) you can set or unset input bits using the buttons 0 to 15. <p>When you choose the ASCII format you can also use escape sequences as detailed in Section 2.6.3, "Escape sequences". However, you can only enter a single character. Thus, ' \d ' or ranges, for example, are not allowed.</p>
Output	Choose the format of the current FSM, i.e. binary, hexadecimal or ASCII.
State Name and State Code fields	display the current state of the FSM. The red or green point next to it indicates if the FSM is in a final state or not, i.e. green for final state and red otherwise.

There are two modes to send input data:

- click on the *Send* button or, press the *Enter* key)
- click on the *Clock* button.

In the clock mode, Qfsm periodically sends the data in the input text field on the top of the dialog to the FSM.

You can specify the frequency of the clock in the input field at the bottom right. For ASCII input, the input text field is cleared after each clock signal, which allows to simulate the behaviour of ASCII FSMs in real-time by pressing different keys.

To exit the *Clock* Mode, click once again on the Clock button.

To reset the FSM, click on the *Reset* Button.

This sets the FSM to its initial state.

2.9. Exporting

There are several export functions in Qfsm.

To export, select *File->Export* in the main menu.

They can be divided into the following categories:

1. state diagrams,
2. hardware description languages,
3. state tables,
4. code generation languages,
5. hardware test code,
6. I/O description,
7. vvvv Automata code
8. State Chart XML (SCXML)

Here are some explanations about the different export functions:

2.9.1. State diagrams

A state diagram as it is seen in the working area can be exported in various graphics formats. The formats that are supported are:

- Encapsulated Postscript (EPS)
- Scalable Vector Graphics (SVG)
- Portable Network Graphics (PNG)

2.9.2. Hardware description languages

Hardware description languages are high level descriptions that can be synthesized into integrated circuits like FPGAs using special software. The languages supported are the following:

1. AHDL

2. VHDL
3. Verilog HDL
4. KISS

Some of the export functions open a dialog allowing you to specify additional export options. However, they should be self-contained for the users having experience with hardware description languages.

2.9.3. State Tables

State tables can be exported in the formats: ASCII (plain text), Latex or HTML. State tables show for each possible state and input (here called *event*) the respective resulting states.

To change the options concerning the layout of the state table, a dialog allows you to change some options.

Include asynchronous output	Tick this option if you want the asynchronous outputs (Mealy outputs) to be printed in the table cells together with the resulting states.
Resolve inverted conditions	<p>Tick this option if you want the inverted conditions to be printed using the inversion descriptor, e.g. 'NOT a', or without it, i.e. printing every character (or binary string) except the ones in the condition.</p> <p>In the case of 'NOT a' this would be the two ranges '\000-`' and 'b-\0FF'.</p> <p>Which one is clearest depends on the respective FSM.</p>
Orientation	Tick the different options in order to determine the orientation of the table, i.e. if current states represent the different rows of the table and the events the columns or vice-versa.

2.9.4. Code generation languages

There are two different types of files that can be used by other code-generating software:

1. Ragel (.rl)
2. State Machine Compiler (SMC) (.sm)

2.9.4.1. Ragel file export

For the *ragel* (.rl) file format only ASCII FSMs can be exported. The resulting file serves as an input for the ragel state machine compiler. The ragel state machine compiler is a compiler that generates code from a high-level state machine description language. In this way, you can create string parsers for example. For details refer to the ragel homepage [<http://www.cs.queensu.ca/~thurston/ragel/>].

A dialog allows you to create a so-called *action file*. That means, the ragel state machine specification is divided into two files. One that contains the state machine logic (which I will call FSM file here) and an action file that contains the action definitions and a framework calling the state machine. Thus, the action file actually includes the FSM file. The name of the action file is determined automatically by appending '_action' at the end of the file name.

The idea is that you create the action file only once at the beginning, and then edit this file manually to define the specific actions. You can then modify your FSM logic in Qfsm as many times as you want and regenerate the FSM file without losing your own action definitions.

Example: suppose you have created an ASCII FSM and you export it under the name `myFSM.rl`. If you check the option '*Create action file*' the action file will be created under the name `myFSM_actions.rl`.

Using `ragel` you can compile the action file: **`ragel -C -o myFSM.c myFSM_actions.rl`**

This will create a file, called `myFSM.c` with the C code of the FSM. It will contain a function: `int parse(char* string)` that parses an input string and returns 1 if the FSM accepts it, i.e. finishes in a final state, and 0 otherwise.

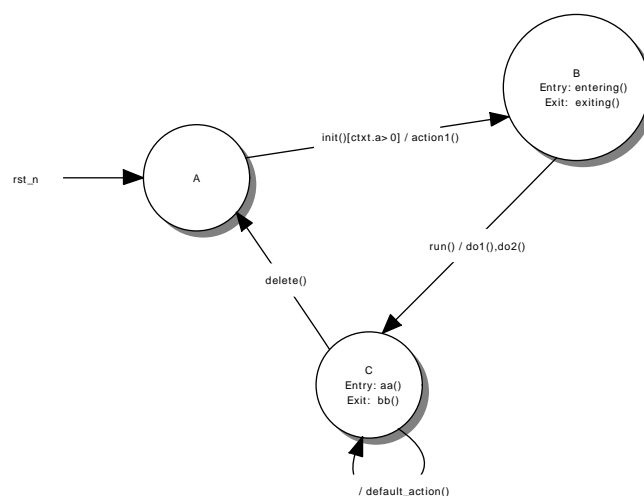
2.9.4.2. SMC file export

Similarly to `ragel`, the State Machine Compiler (SMC) transforms a textual state machine description (a `.sm` file) into code in one of the following languages: C, C++, C#, Groovy, Java, JavaScript, Lua, Objective-C, Perl, PHP, Python, Ruby, Scala, TCL, VB.net. Again, there is a separation into two parts of code, i.e. two classes. One class defines the FSM and is generated by SMC from the `.sm` file. The other is called the application class, or "AppClass", which you have to write yourself and which should contain an instance of the FSM class. In the AppClass, you can issue events by calling the respective methods of your FSM class, i.e. the events/actions that you have defined as inputs in the FSM. And you have to define the actions/events (i.e. methods) that the FSM will call, i.e. the FSM transition outputs and entry/exit actions. For details refer to the SMC homepage [<http://smc.sourceforge.net/>].

Qfsm can generate the `.sm` file from your graphical state diagram. This only works for "free text" FSMs, and the inputs and outputs of the machine are not simple ASCII symbols or binary bits but *method names*, for example `"run()"` or `"suspend()"`. Several output actions (or events) can be specified by a comma-separated list.

Qfsm also supports state entry and exit actions, as well as default transitions (but not default states). Transition guards can also be added after the input events, for example `"doThis() [ctxt.isValid()]"`. The object "ctxt" refers to your AppClass.

Here is a screenshot of a FSM for the purpose of exporting to SMC.



Once you have exported your FSM to a `.sm` file, let's say `FSM1.sm`, you can compile it for example to Java code with the command:

java -jar /usr/local/bin/smc/bin/Smc.jar -java FSM1.sm

replacing the path of Smc.jar corresponding to your smc installation. This creates the file FSM1Context.java - your FSM class file.

Here is an example of how to write a corresponding application class (AppClass). Note that the name of the class ("FSM1") has to be the same as the FSM name.

```
public class FSM1
{
    private final FSM1Context fsm;
    public int a;

    FSM1()
    {
        fsm = new FSM1Context(this);
    }

    public void action1() { System.out.println("executing action1()"); }
    public void do1() { System.out.println("executing do1()"); }
    public void do2() { System.out.println("executing do2()"); }
    public void entering() { System.out.println("entering State"); }
    public void exiting() { System.out.println("exiting State"); }
    public void aa() { System.out.println("executing aa()"); }
    public void bb() { System.out.println("executing bb()"); }
    public void default_action()
    { System.out.println("executing default_action()"); }

    public void testit()
    {
        a=1;
        fsm.init();
        fsm.run();
        fsm.init();
    }

    public static void main(String[] args)
    {
        FSM1 f = new FSM1();
        f.testit();
    }
}
```

You can then compile everything with the java compiler:

javac *.java -cp /usr/local/bin/smc/lib/statemap.jar

making sure to put statemap.jar in your class path.

2.9.5. Hardware test code

The hardware test code is used together with the output of a hardware description export. The supported language for test code is VHDL. The VHDL testbench export will create a set of files that enables a hardware simulation and debugging software to analyse the VHDL code exported by Qfsm.

The following files are generated:

t_fsmname.vhd	The VHDL testbench instantiating the FSM module, a clock generation process, an initial reset and a stimuli_observer process.
p_fsmname.vhd	A VHDL package declaration and body with all the necessary subprograms used in the testbench.
fsmname_stimuli.vec	The file includes all the stimuli and expected results necessary for a complete verification of a finite state machine.
fsmname_result.log	Errors are documented in this file.

Per default, these files can be written to different directories within a clearly structured design directory. Therefore, in the export options a 'Base Directory' input string is asked, first:

```

Base_directory
|
+-src
|   fsmname.fsm
|   e_fsmname.vhd
|   a_fsmname_architecturename.vhd
|   t_fsmname.vhd
|   p_fsmname.vhd
|
+-stimuli
|   fsmname_stimuli.vec
|
+-log
    fsmname_result.log

```

2.9.6. I/O description

When exporting the I/O description of the state diagram a file containing the names of all the inputs and outputs of the FSM is created. The description is stored in a comma separated value (CSV) format, where the separator is a semicolon and the values are inside double quotes. The first value of each line is a textual description of the values in that line, i.e. "Inputs", "Mealy Outputs" and so on.

The following lines are written:

- "Inputs": the names of the (Mealy) inputs
- "Mealy Outputs": the names of the Mealy outputs
- "State/Output": the names of the Moore outputs
- Finally a line for each state containing its name and its Moore outputs

2.9.7. vvvv Automata code

A "free text" FSM can be exported to vvvv Automata code. This code can be used in conjunction with the tool vvvv [<http://vvvv.org/tiki-index.php>], a toolkit for real-time video synthesis. This textual format is a list of quadrupels, where each item/line describes a transition and is composed of:

- start state,
- event,
- end state,
- action.

When selecting this export function from the menu a non-modal dialog box is opened that displays the resulting code. This is automatically updated as the diagram is modified. An additional reset transition can optionally be added for each state by checking the box at the bottom of the dialog. There you can also specify the name of the reset event and the name of the respective action to be triggered.

2.9.8. State Chart XML

A "free text" FSM can be exported in the State Chart XML format as proposed by the W3C. The specification of SCXML can be found here [<http://www.w3.org/TR/scxml/>]. Note that each transition is triggered by an event. And the text that is entered as transition input is used as the name of the event. The text that is entered as the transition output is interpreted as the name of an event that is to be sent when the transition is triggered. Thus, if the output linked to a transition is non-empty a corresponding `<send>` tag will be written.

2.10. Importing

Qfsm can import files written in the DOT language [<http://www.graphviz.org/content/dot-language>] using Graphviz [<http://www.graphviz.org>].

To import a DOT file (with extension .gv), select *File->Import->Graphviz...* in the main menu.

Qfsm uses Graphviz to transform the textual description of the FSM into a graphical layout of the state diagram.

When writing a DOT file make sure you respect the following syntax rules:

- the type of the diagram needs to be a directed graph (*digraph*),
- the rankdir must be "LR" (left-right) or "TB" (top-bottom).

Here is an example of a dot file describing a binary FSM. The file is included in the Qfsm package in the *examples* folder.

```
digraph binary_finite_state_machine {
  rankdir=LR;
  size="8,5"
  node [shape = doublecircle]; LR_0 LR_3;
  node [shape = circle];
  LR_0 [ label = "State_1" moore = "01" ];
  LR_0 -> LR_1 [ label = "0000|0001###11" ];
  LR_1 -> LR_2 [ label = "NOT 0000###00" ];
  LR_1 -> LR_3 [ label = "DEF###01" ];
  LR_0 -> LR_2 [ label = "ANY###10" ];
  LR_2 -> LR_3 [ label = "1111###11" ];
}
```

Qfsm automatically determines the type of FSM: binary, ASCII, or Free Text. In the above example, "LR_0", "LR_1", ... denote the states. It can be given any name by specifying the `label` attribute. If you don't do this, the node name (here: "LR_1", ...) is used as the state name. The `moore` attribute is used to specify the Moore outputs of a state.

Transitions are written as

```
State1 -> State2;
```

or

```
State1 -> State2 [ label = "00###11" ];
```

where the `label` attribute specifies the Mealy inputs, and optionally the Moore outputs separated by the string `###`. NOT denotes negation, DEF a default transition, and ANY transitions that accept any input.

To define end states, use the `shape = doublecircle` node attribute as illustrated in the above example.

2.11. Setting the options

To display the options dialog select *Edit->Options* from the main menu.

2.11.1. Changing language

To change the language:

- 1) select Edit>Options>General
- 2) select the language
- 3) click on OK
- 4) restart the application

The language change is now effective.

2.11.2. Changing the display

To change the display, select Edit>Options>Display from the main menu

Grid	You can choose the colour and size of the grid displayed on the working area. You can activate the grid via the main menu entry <i>View->Grid</i> .
Shadows	You can determine if shadows are to be drawn and the their colour.
Transitions	You can determine the appearance of input conditions and outputs drawn on top of the transitions.
Tooltips	You can determine if tooltips should be shown or not when moving the mouse pointer over a state or a transition.
Initial transition descriptor	Enter the text that is displayed next to the initial transition. Default: "Reset".

Inversion descriptor	Enter the text that is displayed before inverted transition conditions. Default: "NOT".
"Any input" descriptor	Enter the text that is displayed for transitions that are activated by any input. Default: "any".
Default transition descriptor	Enter the text that is displayed for default transitions. Default: "default".

2.11.3. Printing

To change the printing options, select Edit>Options>Printing from the main menu.

Print header	Tick this option if you want to print a header with the FSM name and version at the top of the diagram.
--------------	---

Chapter 3. Contact

If you have questions or suggestions concerning Qfsm feel free to contact me at:

`<qfsm(at)duffner(dash)net(dot)de>`

I'm also glad about any contribution you want to make to the project, e.g. code, bug fixes, documentation, packaging, testing etc.

Index

A

- Add state mode, 6
- Add transition mode, 6
- AHDL, 17
- AIX, 2
- Arrow Type, 8
- ASCII, 7, 7, 11, 13, 13, 13, 14, 16, 16, 18, 19
 - State table, 18

B

- Behaviour
 - testing, 10
- Binary, 7, 8, 11, 13, 16, 16

C

- C++, 1
- Clock
 - Button, 17
 - Mode, 17
- Close, 4
- CMake, 2
- Code
 - State, 11
- Code generation languages, 18
- Colour
 - Button, 11
 - State, 11
- Comma Separated Values (CSV), 21
- Completeness, 15
- Condition
 - ASCII, 13
 - Type, 12
- Copy, 4
- Correct automatically, 5
- Create, 4
- Cut, 4

D

- Delete, 4
- Description
 - state, 11
 - Transition, 13
- Deselect all, 4
- Display
 - "Any input" descriptor, 24
 - Changing the display, 23
 - Default transition descriptor, 24
 - Grid, 23
 - Initial transition descriptor, 23

- Inversion descriptor, 24
- Shadows, 23
- Tooltips, 23
- Transitions, 23
- DOT, 22

E

- Edit, 4 (see Copy)
 - Cut, 4
 - Delete, 4
 - Deselect all, 4
 - Options, 4
 - Paste, 4
 - Select, 4
 - Select all, 4
 - Undo, 4
- Entry actions
 - State, 11
- Escape sequences, 14, 14
- Event, 18, 22, 22
- Exit actions
 - State, 11
- Export, 4, 17

F

- File, 4
 - Close, 4
 - Export, 4
 - New, 4
 - New window, 4
 - Open, 4, 10
 - Open Recent, 4
 - Print, 4
 - Quit, 4
 - Save, 4
 - Save as, 4
- Final state, 15
- Final states reachable , 15
- Finite state machine, 1
 - Creating, 6, 10
 - Integrity check, 15
 - Modifying, 6
 - Properties, 7
- Fonts, 8
- Free text, 7, 13, 13
- FreeBSD, 2

G

- Graphviz, 2, 22
- Grid, 5

H

- Hardware description languages, 17

Hardware test code, 20

Hexadecimal, 16

HP-UX, 2

HTML

State table, 18

I

I/O description, 21

IDE project file, 3

Import, 22

Include asynchronous output, 18

Initial state, 6, 15

Input

Transition property, 13

Installation, 2

binary, 2

from source, 2

Linux, 2

Mac OS X, 2

Windows, 2

Integrity check, 5, 15

IO view, 5

IRIX, 2

K

KISS, 18

L

Language

Changing language, 23

Latex

State table, 18

Line width

State, 11

Linux, 2

building under, 2

Loop

Draw loops, 12

M

Mac OS X, 2

Machine, 5

Auto correct State Codes, 5

Edit, 5

Integrity check, 5

Simulate, 5

Mealy input , 5, 8, 13

Mealy output, 8, 13

Mixed formats, 14

Mode

Add state, 8, 10

Add transition, 8, 10

Pan, 8, 9

Select, 8, 9, 12

Simulate, 9, 10

Zoom, 8, 9

Moore output, 5, 8

State, 11

Multiple characters, 14

N

Name

Finite state machine, 7

State, 11

New window, 4

No dead locks, 15

O

Open, 4

Open recent, 4

openSUSE, 2

Options, 4, 23

Orientation, 18

Output

Transition property, 13

P

Pan view, 5

Paste, 4

Print, 4

Printing, 24

Print header, 24

Properties

Finite state machine, 5

State, 11

Transition, 12

Protocol, 15

Q

Qt, 1, 2, 2

Quadrupels, 21

Quit, 4

R

Radius

State, 11

Ragel, 7, 18

Action file, 18

Ranges, 14

Requirements, 2

Reset

Button, 17

Transition, 22, 23

Resolve inverted conditions, 18

RPM, 2

S

- Save, 4
- Save as, 4
- SCXML, 22
- Select, 4
- Select all, 4
- Select mode, 13
- Send button, 17
- Shadows, 5
- Simulation, 5, 15
 - VHDL, 20
- Simulator dialog
 - Input, 16
 - Output, 16
 - State name and state code fields, 16
- Single character, 14
- SMC, 19
- Solaris, 2
- State, 6, 9, 9, 9, 10, 10
 - Adding, 10
 - Creating, 10
 - Edit, 6
 - Final, 6, 15
 - Initial, 6, 15
 - Modifying, 6, 10, 11
 - New, 6
 - Non final state, 6
 - Toggle final state, 6
- State Chart XML, 22
- State codes, 5
- State diagrams, 17
- State properties, 11
- State table, 18
- States reachable, 15
- Stimuli, 21
- Straighten, 6

T

- Testbench
 - VHDL, 21
- Transition, 6, 8, 8, 8, 9, 9, 9, 10, 10
 - Adding, 12
 - Creating, 12
 - Edit, 6
 - Initial, 8, 23
 - Modifying, 6, 12, 13
 - New, 6
 - Straighten, 6
- Transition control point, 9
- Transition properties, 12
- Transitions connected, 15
- Type
 - Condition, 12

- Finite state machine, 7

U

- Unambiguous Conditions, 15
- Undo, 4
- Unix, 2
 - building under, 2

V

- Verilog HDL, 7, 18
- Version
 - Finite state machine, 7
- VHDL, 7, 18
 - text code, 20
- View, 5
 - Grid, 5
 - IO view, 5
 - Mealy input , 5
 - Moore outputs, 5
 - Pan view, 5
 - Shadows, 5
 - State codes, 5
 - Zoom, 5
 - Zoom 100%, 5
 - Zoom in, 5
 - Zoom out, 5
- vvvv Automata code, 21

W

- Windows, 2
 - building under, 3
- Working area, 8, 9

Z

- Zoom, 5
- Zoom 100%, 5
- Zoom in, 5, 10, 10
- Zoom out, 5, 10