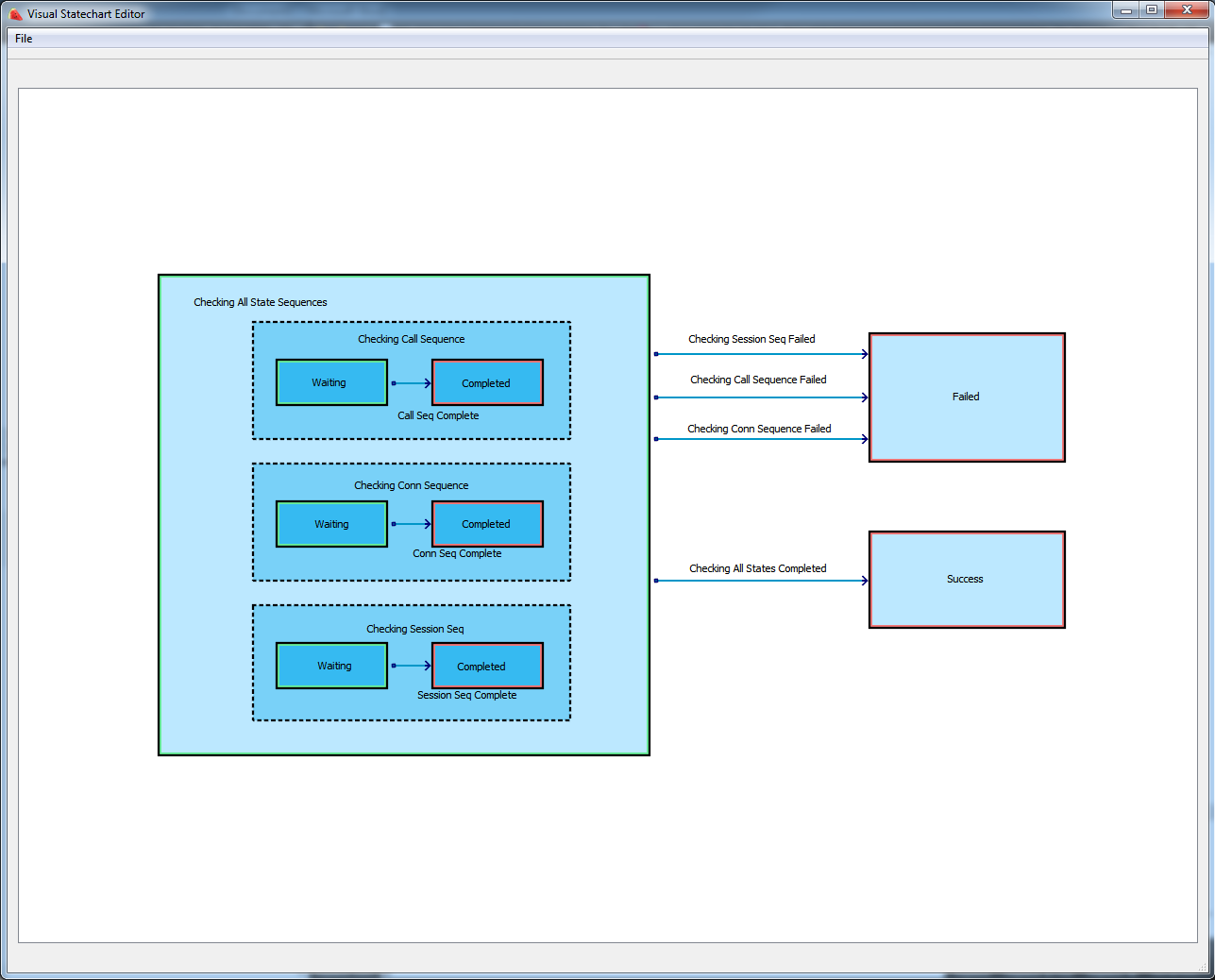
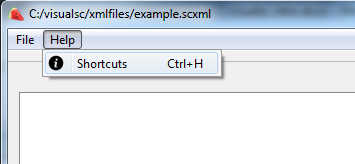
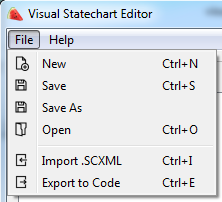
Visual Statechart Editor: Graphics View



1. **Shortcuts**

****

1. **File Menu**



New   
Clears the program and makes a new root machine

Save   
Saves the state chart to a .SCXML file. Once a file has been saved or opened, it no longer prompts for a save location—until new or save as is clicked.

Save As  
Saves the state chart into a .SCXML file. This is useful when saving a forked version of a state chart.

Open  
Opens a .SCXML file.

Import .SCXML  
Loads a state machine in a .SCXML file into the currently selected state. The root machine of the chosen .SCXML file becomes a child state of the currently selected state.

Export to Code  
Writes a .cpp and .h file for the current state chart. This will overwrite existing, generated .cpp and .h files after the first use—until new or open is pressed. The Class Name in the .cpp file is based on your Root Machine name. If one wishes to change the file name, then change the root machine name before clicking Export. Changing the root machine name will generate the Class Name AND .cpp file with the same name.

If root machine’s name is “my Root machine” then…

Class Name: MyRootMachine  
cpp file name: myrootmachine.cpp  
h file name: myrootmachine.h

See Export to Code section for more detail.

1. **States**

In VisualSc, there is always the root machine present. In the graphics view, one can think of the white space as the highest parent state and all states are descendants of this top level state.

States in the graphics view can be manipulated:

* Repositioned with a click and drag on its area
* Resized with a click and drag on its corner grabbers
* Automatically resized with a double click on its area

Automatic resizing (double click)

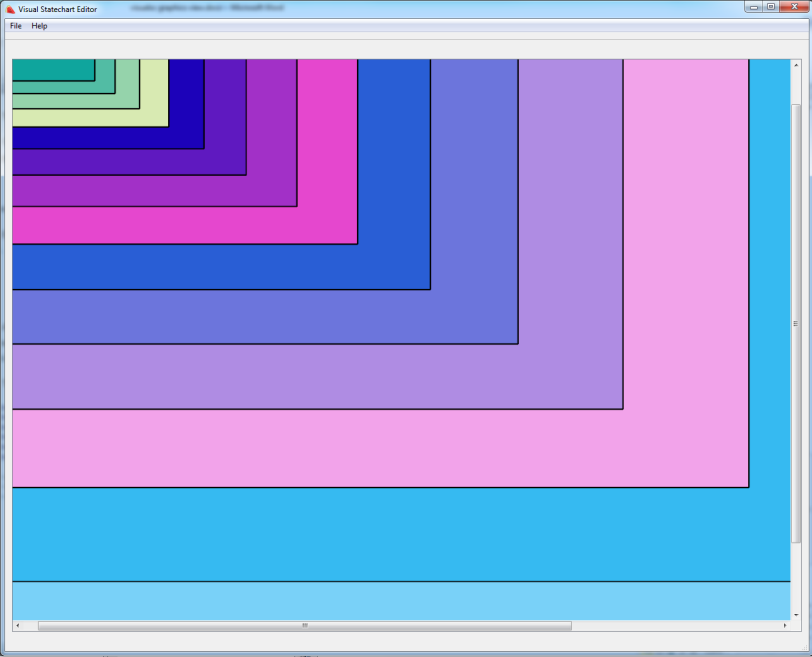
State machines will resize themselves to encapsulate all of their top level children

Childless states will resize themselves to match the dimensions of a same level state. The state chosen goes by order of first state added that is not itself.

Text blocks

Every State comes with a text block that holds its name attribute. Double click to open edit mode. Changes are updated across all locations of the attribute.

Color



A state is colored based on its level in the hierarchy. States on the same level, regardless of parent, will be the same color. The root machine is on level 0, child states have a level of one plus their parent’s level.

The algorithm for the color of states relates the level directly to a 0-255 value for R, G, and B.

#define BOX\_RED\_COLOR\_JUMP 67

#define BOX\_GREEN\_COLOR\_JUMP 23

#define BOX\_BLUE\_COLOR\_JUMP 7

#define BOX\_RED\_OFFSET 0

#define BOX\_GREEN\_OFFSET 0

#define BOX\_BLUE\_OFFSET 1

int r = 255 - (((this->getStateModel()->getLevel()-BOX\_RED\_OFFSET) \* BOX\_RED\_COLOR\_JUMP ) % 255);

int g = 255 - (((this->getStateModel()->getLevel()-BOX\_GREEN\_OFFSET) \* BOX\_GREEN\_COLOR\_JUMP ) % 255);

int b = 255 - (((this->getStateModel()->getLevel()-BOX\_BLUE\_OFFSET) \* BOX\_BLUE\_COLOR\_JUMP ) % 255);

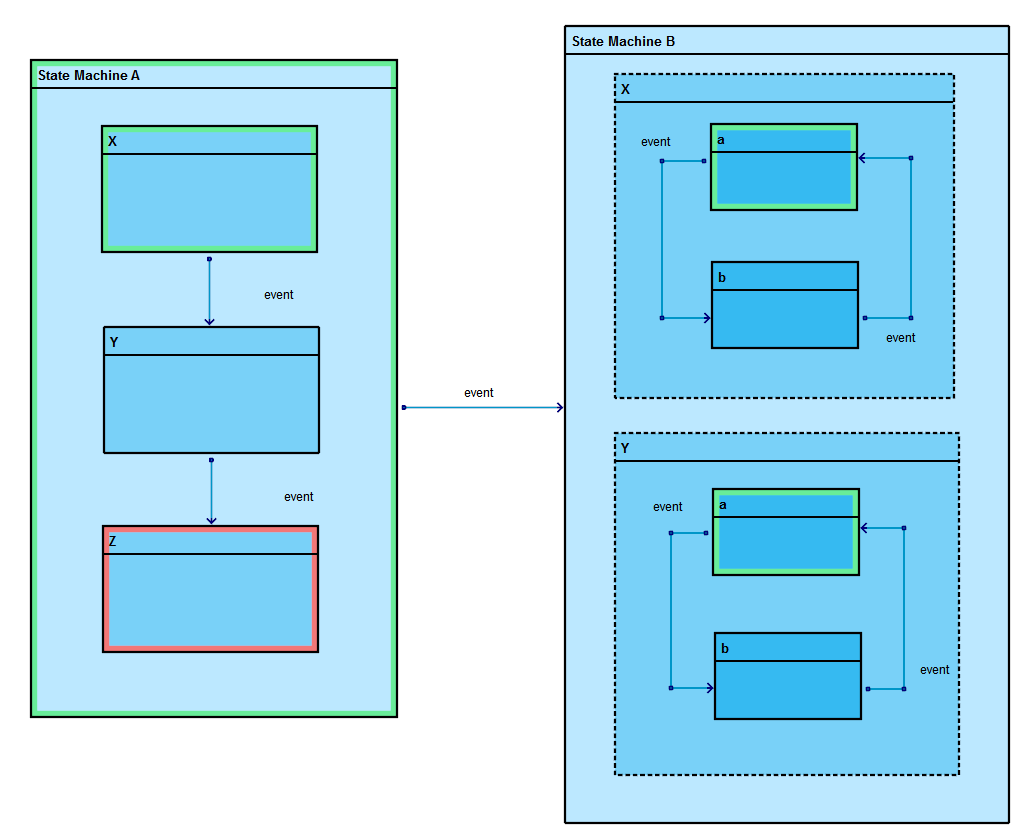
Children States and state types

A state with children states is a state machine. All children on the same hierarchy level will be the same color, regardless of parent. A feature of Qt is children graphics items have linkage to their parents. When a child item is hovered, it triggers the parent’s hover protocol by default. When a parent’s position is changed, all of its children objects are repositioned by the same amount by default. When a state is deleted, all of its children objects are also deleted.

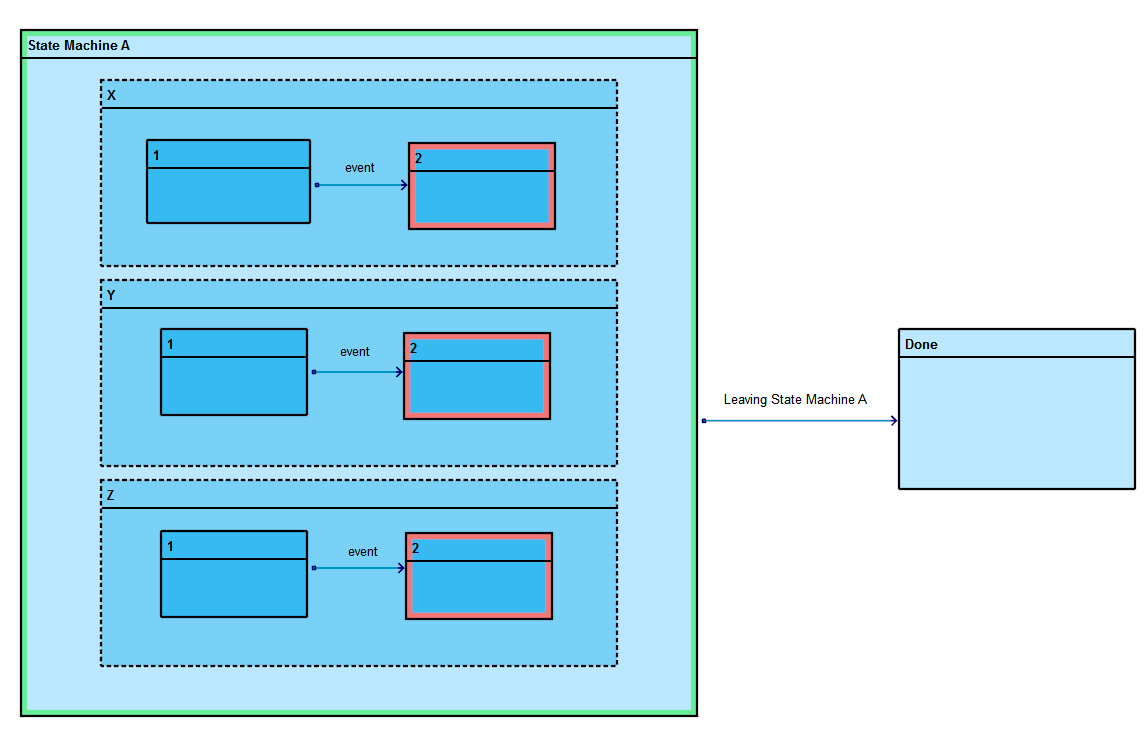
Currently, state boxes have freedom in their size and position and they are not bound to stay inside of its parent box. State text blocks and transitions, however, are restricted and attached to their respective state boxes.

Setting state types are done through the attribute table in the tree view and will be easily visualized in the graphics view. These properties must be set properly before using the code generator. One should make sure these simple rules are followed when creating states:

* Initial States: All state machines have one initial state, and this must be specified unless the state machine is parallel. This is shown with a green border.
* Final States: A state machine can have any number of final states set, including none. A final state is a way to leave a state machine entirely from a child state. This is shown with a red border.



* Every state machine needs an initial state
  + The initial state of the root machine is State Machine A
  + A parallel state machine (State Machine B) should not have an initial state
  + Note that X and Y are parallel to each other in state machine B, but X and Y are not parallel state machines themselves, so they need an initial state set
* Final States are optional
  + State Machine A will emit its finished signal when Z is entered (and can be set to trigger the transition between State Machine A to State Machine B)
  + State Machine B has no finished signal because none of its child states has a finished state.



* Parallel state machines will emit their finished signal only when all children states emit a finished signal
  + State Machine A will emit finished when states X, Y, and Z have all entered their respective state 2.
  + Leaving State Machine A can be hooked to the finished signal of State Machine A with the transition attribute “connectToFinished”

1. Transitions

Transitions always have two anchors—one at the source state and one at the target state. These elbows are snapped to the border of their state boxes.

Elbows

Elbows have a snapping to right angles when a moved elbow falls within a certain threshold. When an elbow is moved, its position is checked. If it falls within a cross of fixed width of any of its neighbor elbows, it will snap to be on a perfectly flat or vertical plane with that elbow.

New elbows can be created by hovering over the line segment and pressing “N”

<https://gyazo.com/851f6a39790ad06b61233a35aded66c1>

Elbows can be deleted by hovering an elbow and pressing “D.”

<https://gyazo.com/02b62cb9b21f591dc629d319b9f74976>

Text Blocks

Each transition has a text block for its event name. This is a child of the source elbow, so when the event text block is hovered it will also trigger the hover of the source elbow. Double click to open edit mode. Click anywhere else to leave edit mode. This will be updated across all forms.

Rules:

* Final states cannot have out transitions.
* Transitions cannot target child states of a state machine if the transition is coming from a state that is not related/a child of that state machine. These transitions must go to the high encapsulating parent first.