**Code Exporter**

**Modeling State Machines**

The hierarchical state machine is modeled using the Qt framework with QStates. The code exporter will create a .cpp and .h file based on the states and transitions in your state machine.

The class name and file name generated are based on the name of the root machine. If one wishes to change the name of the .cpp and .h, they should do so through the root machine name.

For example, a root machine named “my Root machine”

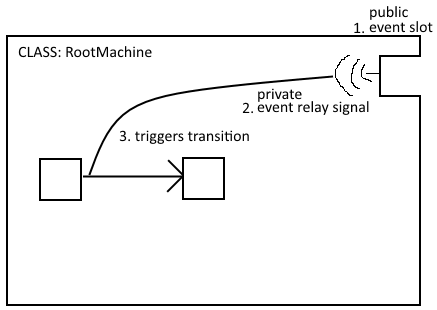
* Class Name: MyRootMachine
* .cpp file name: myrootmachine.cpp
* .h file name: myrootmachine.h

**Generator Intro**

The code generator will create a skeleton class for your state machine. This class has a member QState object for every state (QState, QStateMachine, or QFinalState). There will be some public signals and slots and some private signals and slots created for every state.

Public Slots Created

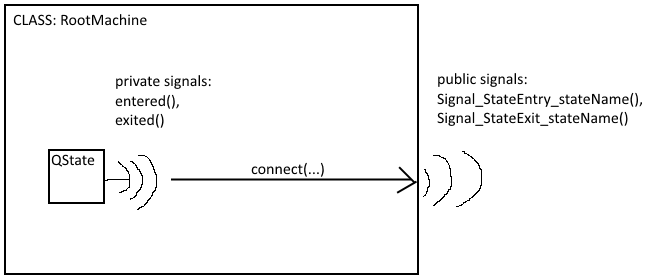
* Every transition will have its own public slot
* Slot Name: Event\_eventName\_uid();
* Each event slot emits a private signal corresponding to the event
  + Signal Name: Relay\_Event\_eventName\_uid();
  + addTransition() uses this signal



Public Signals Created

Each state will have publicly accessible signals corresponding to the QState private signals.

* QStates
  + have two public signals for their state entry and state exit which correspond to the private signals entered() and exited() signals of the QState
  + Public Signal Names:
    - Signal\_StateEntry\_stateName\_uid()
    - Signal\_StateExit\_stateName\_uid()



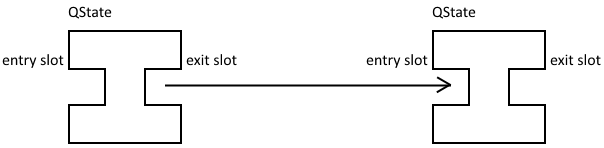
* QStateMachine
  + The root machine only has one public signal for its QStateMachine started() signal
  + Public Signal Name:
    - Signal\_StateReady\_stateName\_uid()
* entryAction for states
  + each element in a comma separated list for the entryAction attribute for a state will have its own public signal
    - An empty list results in no additional signals
  + Public Signal Name: EntryAction\_elementName()
  + These signals are emitted in the state’s private entry slot
* exitAction for states
  + each element in the comma separated list for the exitAction attribute for a state will have its own public signal
    - An empty list results in no additional signals
  + Public Signal Name: ExitAction\_elementName()
  + These signals are emitted in the state’s private exit slot

Private Slots Created

Each state will have private slots corresponding to QStates Entry and Exit. When a transition is triggered, then two slots will be called.

1. The exit slot of the source state
2. The entry slot of the target state

This is setup in the constructor with a connect of the entered() and exited() signals to these slots



Private Signals Created

* Every transition has a private Event Signal, which triggers the corresponding transition
  + Private Signal Name: Relay\_Event\_eventName\_uid()
  + addTransition() uses this signal
  + emit called inside corresponding public event slot

Private QState members

Each state will have a generated member QState variable associated with it. Depending on the state type, it will be one of three:

* QStateMachine
* QState
* QFinalState

There is only one QStateMachine, the root machine. QStates can also be state machines by initializing QStates with a parent QState, but this is how it is modeled in Qt. QFinalStates are for any final states.

All state machines need an initial state set (done through form view), unless the state machine is a parallel state—where all children are concurrent and essentially all initial states (and in this case do not set any initial states). A final state is optional for state machines.

When a final state is entered, it emits its finished() signal. This finished signal is relayed to its parent state machine, so when any of a state machine’s children states emits finished(), the state machine also emits its finished() signal.

**State Translation**

The attributes of a state and its parent and children will determine how it is used in the code generator. QStates will help in modeling state relationships and transitions. Any state with children states is classified as a state machine. The constructor of the generated code will set up all parent-child relationships between states, call addTransition for every transition, and additional things depending on the attributes for states and transitions.

Each state attribute affects the code generator:

* Name
  + member QState variable name is a camel case conversion of the name
  + Member name: \_nameInCamelCase\_uid;
* entryAction
  + each element in a comma separated list for the entryAction attribute for a state will have its own public signal
    - An empty list results in no additional signals
  + Private Signal Name: EntryAction\_elementName()
  + These signals are emitted in the state’s private entry slot
* exitAction for states
  + each element in the comma separated list for the exitAction attribute for a state will have its own public signal
    - An empty list results in no additional signals
  + Private Signal Name: ExitAction\_elementName()
  + These signals are emitted in the state’s private exit slot
* finalState
  + true/false
  + the QState member object will be created as a QFinalState
  + final states are optional
  + a state machine can have any number of final states set, including none
* initialState
  + true/false
  + constructor calls setInitialState for this state’s parent
  + every state machine needs an initial state set (unless parallel)
* parallelState
  + true/false
  + initializes QState as a parallel state
  + top level children run concurrently
  + only emits a finished() signal once all children emit their finished signal

**Transitions**

Every transition will be added to the state machine model by calling addTransition in the constructor. Each transition also gets a public slot. This slot emits a relay signal that triggers the transition.

Each transition attribute affects the code generator:

* event
  + name of transition
  + Public Slot Name: Event\_eventName\_uid()
  + Private Relay Signal Name: Relay\_Event\_eventName\_uid()
* Comments
  + Creates a comment in the constructor for the addTransition line
* connectToFinished
  + true/false
  + adds another addTransition Line to connect the source state’s finished() signal to the target state