printf("Welcome");

C++11 State Machines

Written by Ludvik Jerabek on February 8, 2016 in C and C++, Programming

I recently decided to take some time to learn about the new features in C++11 while working on a project at home. C++11 has added some really nice features such as native thread support, regular expressions, safe pointer types, and variadic templates. While working on my project, I found myself needing a flexible class which would allow me to express a finite state machine (FSM) without excessive sub-classing. I also wanted to be able to select the types for states and events (transitions), which lead me down the path of developing a state machine template class. I wanted to be able to make defining a FSM as easy as the tradition C state table driven model but with better lookup times. Additionally, the class should support event callbacks should you choose to define one. The following example shows a state machine which uses std::string as the state and event. These could easily be replaced by an enum of states and enum of events.

```
1  #include "state_machine.h"
2
3  typedef state_machine<std::string,std::string> str_state_machine;
4
5  void my_enter_radio(str_state_machine::state_ptr state){
6      std::cout << __FUNCTION__ << " leaving [" << state->get_id() << "]" << std::endl;
7  }
8
9  void my_exit_radio(str_state_machine::state_ptr state){
10 std::cout << __FUNCTION__ << " leaving [" << state->get_id() << "]" << std::endl;
11 }
12
13 void my_enter_cdplayer(str_state_machine::state_ptr state){
14     std::cout << __FUNCTION__ << " entering [" << state->get_id() << "]" << std::endl;
15 }
16
17 void my_exit_cdplayer(str_state_machine::state_ptr state){
18     std::cout << __FUNCTION__ << " entering [" << state->get_id() << "]" << std::endl;
19 }
20
21 void my_event_action(str_state_machine::state::event_ptr event,
22                      str_state_machine::state_ptr current_state,
23                      str_state_machine::state_ptr next_state)
24 {
25     if( current_state == next_state )
26         std::cout << "Action: [" << event->get_id() << "] " << "No State Change" << std::endl;
27     else
28         std::cout << "Action: [" << event->get_id() << "] "  << current_state->get_id() << " --> " << next_state->get_id() << std:
29 }
30
31 int main() {
32     str_state_machine fsm;
33     // Setup state transitions for radio
34     fsm.add_state("radio")
35         ->bind_entry_action(my_enter_radio)
36         ->bind_exit_action(my_exit_radio)
37             ->add_event("switch_radio");
38     fsm.add_state("radio")->add_event("switch_cd","cdplayer")->bind_action(my_event_action);
39     fsm.add_state("radio")->add_event("next")->bind_action(my_event_action);
40     fsm.add_state("radio")->add_event("previous")->bind_action(my_event_action);
41     // Setup state transitions
42     fsm.add_state("cdplayer")
43         ->bind_entry_action(my_enter_cdplayer)
44         ->bind_exit_action(my_exit_cdplayer)
45             ->add_event("switch_cd");
46     fsm.add_state("cdplayer")->add_event("switch_radio","radio")->bind_action(my_event_action);
47     fsm.add_state("cdplayer")->add_event("next")->bind_action(my_event_action);
48     fsm.add_state("cdplayer")->add_event("previous")->bind_action(my_event_action);
49
50     fsm.dump_state_table();
51     fsm.next_state("next");
52     fsm.next_state("previous");
53
54     fsm.next_state("switch_cd");
```

```
55      fsm.next_state("next");
56      fsm.next_state("previous");
57
58      fsm.next_state("switch_radio");
59      fsm.next_state("next");
60      fsm.next_state("previous");
61  }
```

This code results in the following output:

```
STATE           ACCEPTED        TERMINAL        EVENT           ACTON           NEXT_STATE
radio           false           false           switch_radio    false           radio
radio           false           false           switch_cd       true            cdplayer
radio           false           false           previous        true            radio
radio           false           false           next            true            radio
cdplayer        false           false           switch_radio    true            radio
cdplayer        false           false           switch_cd       false           cdplayer
cdplayer        false           false           previous        true            cdplayer
cdplayer        false           false           next            true            cdplayer
--------------------------------------------------------------------------------
Action: [next] No State Change
Action: [previous] No State Change
Action: [switch_cd] radio --> cdplayer
my_exit_radio leaving [cdplayer]
my_enter_cdplayer entering [radio]
Action: [next] No State Change
Action: [previous] No State Change
Action: [switch_radio] cdplayer --> radio
my_exit_cdplayer entering [radio]
my_enter_radio leaving [cdplayer]
Action: [next] No State Change
Action: [previous] No State Change
```

This output shows the all of the states and events as well as if the state is an accepted state or a terminal state. Accepted state can be used for pattern parsing and terminal state means there are no transitions out of that state. The bottom half of the output shows the callbacks function output. The callbacks can be implemented in multiple functions or a single function depending on one's needs.

The event callback function has the following definition:

```
1 void my_event_action(state_machine<states,events>::state::event_ptr event,
2                       state_machine<states,events>::state_ptr current_state,
3                       state_machine<states,events>::state_ptr next_state);
```

The entry/exit callback function has the following definition:

```
1 void my_enter_exit_function(state_machine<states,events>::state_ptr state);
```

Lookup times are improved by relying on std::set<state> and std::set<event> which are internally nested. The only the state_machine class can create instances of the state_machine::state class and only the std_machine::state class can crate instances of the state_machine::state::event class.

```
1   template <typename state_id_type,typename event_id_type>
2   class state_machine {
3       private:
4           struct _state_comparer;
5       public:
6           typedef state_machine<state_id_type,event_id_type>* state_machine_ptr;
7           // State Class Definition
8           class state {
9               friend state_machine;
10              typedef state* state_ptr;
11              struct _event_comparer;
12          public:
13              // Transition Class Definition
14              class event {
15                  friend state;
16                  typedef event* event_ptr;
17                  // Public Member Functions
18                  public:
19                      // Returns the current event id
20                      event_id_type get_id() const;
21                      // Returns the state transition associated with the event
22                      state_ptr get_state() const;
23                      // Return true if action has been defined
24                      bool has_action() const;
25                      // Bind action to this event
26                      void bind_action(std::function<void(event_ptr,state_ptr,state_ptr)> action);
27                  // Private Member Functions
28                  private:
29                      event(event_id_type event_id, state_ptr target_state);
30                      ~event();
31                  // Private Types
32                  private:
33                      event_id_type _id;
34                      state_ptr _state;
35                      std::function<void(event_ptr,state_ptr,state_ptr)> _action;
36              };
```

```
37          typedef event* event_ptr;
38          // Private Member Type Definitions
39          private:
40              struct _event_comparer {
41                  bool operator()(const event_ptr lhs, const event_ptr rhs) const {
42                      return rhs->_id < lhs->_id;
43                  }
44              };
45              typedef std::set<event_ptr,_event_comparer> event_set;
46          // Public Member Functions
47          public:
48              state_id_type get_id() const;
49              bool is_accepted() const;
50              bool is_terminal() const;
51              // Creates a null next event
52              event_ptr add_event(event_id_type event_id);
53              event_ptr add_event(event_id_type event_id, state_id_type next_state, bool accepted = false);
54              event_ptr get_event(event_id_type event_id) const;
55              bool has_entry_action() const;
56              bool has_exit_action() const;
57              state_ptr bind_entry_action(std::function<void(state_ptr/*current_state*/)> entry_action);
58              state_ptr bind_exit_action(std::function<void(state_ptr/*next_state*/)> exit_action);
59              const event_set& get_events() const;
60          // Private Member Function
61          private:
62              state(state_machine* parent, state_id_type state_id, bool accepted);
63              ~state();
64              void _do_action(event_ptr event);
65              typename event_set::const_iterator _find_event(event_id_type event_id) const;
66              typename event_set::iterator _find_event(event_id_type event_id);
67          // Private Types
68          private:
69              state_machine* _parent; // Pointer to parent
70              state_id_type _id;
71              bool _accepted;
72              event_set _events;
73              std::function<void(state_ptr/*current_state*/)> _entry_action;
74              std::function<void(state_ptr/*next_state*/)> _exit_action;
75      };
76      typedef state* state_ptr;
77      // Private Member Type Definitions
78      private:
79          struct _state_comparer {
80              bool operator()(const state_ptr lhs, const state_ptr rhs) const {
81              return rhs->_id < lhs->_id;
82              }
83          };
84          typedef std::set<state_ptr,_state_comparer> state_set;
85      // Public Member Functions
86      public:
87          state_machine();
88          ~state_machine();
89          state_ptr add_state(state_id_type state_id, bool accepted = false);
90          state_ptr get_state(state_id_type state_id) const;
91          state_ptr get_current_state() const;
92          state_ptr get_initial_state() const;
93          state_ptr next_state(event_id_type event_id) throw(std::runtime_error);
94          state_ptr reset();
95
96          const state_set& get_states() const;
97          void dump_state_table();
98      // Private Member Functions
99      private:
100         typename state_set::const_iterator _find_state(state_id_type state_id) const;
101         typename state_set::iterator _find_state(state_id_type state_id);
102     // Private Types
103     private:
104         state_set _states;
105         state_ptr _initial;
106         state_ptr _current;
107     public:
108 };
```

Hope this simple state machine can help someone from having to recreate the wheel. You can download the source here.

Subscribe

If you enjoyed this article, subscribe now to receive more just like it.

Comments are closed.

« Prev
Next »

Enter Search Terms search

Recent Posts

- Building Boxie a Kiosk Robot
- Adventures in Science Technology Engineering and Math (STEM) – Part 1
- C++11 State Machines
- Getting Bashed by Dynamic Arrays
- XBox One – Fun with GPT Disk (Larger Drive)

Archives

- February 2017
- January 2017
- February 2016
- August 2015
- July 2014
- April 2013
- March 2013

Categories

- C and C++
- General
- Linux
- Programming
- Review
- Technology
- Tutorials

Ludvik Jerabek