General UML Statemachine Implementation

0000

Generated by Doxygen 1.8.11

Contents

1 Main Page

UML state machines (main source wikipedia)

The Unified Modeling Language has a notation for describing state machines. UML state machines overcome the limitations of traditional finite state machines while retaining their main benefits.

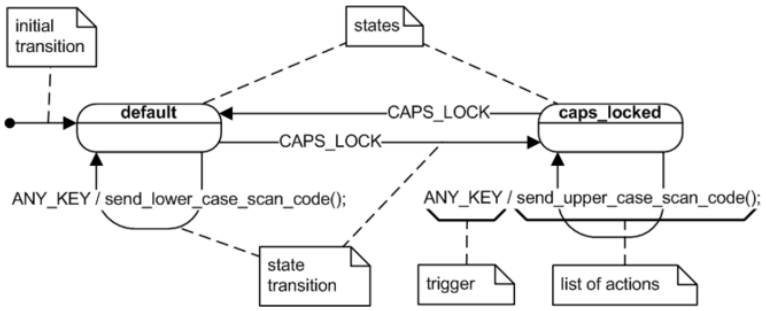


Figure 1 Basic UML state chart

UML state machines introduce the new concepts of hierarchically nested states and orthogonal regions, while extending the notion of actions. UML state machines have the characteristics of both Mealy machines and Moore machines. They support actions that depend on both the state of the system and the triggering event, as in Mealy machines, as well as entry and exit actions, which are associated with states rather than transitions, as in Moore machines.

Supported features in this implementation (main source wikipedia)

Events

An event is something that happens that affects the system. Strictly speaking, in the UML specification, the term event refers to the type of occurrence rather than to any concrete instance of that occurrence.

An event can have associated parameters, allowing the event instance to convey not only the occurrence of some interesting incident but also quantitative information regarding that occurrence.

An event instance outlives the instantaneous occurrence that generated it and might convey this occurrence to one or more state machines. Once generated, the event instance goes through a processing life cycle that can consist of up to three stages. First, the event instance is received when it is accepted and waiting for processing (e.g., it is placed on the event queue). Later, the event instance is dispatched to the state machine, at which point it becomes the current event. Finally, it is consumed when the state machine finishes processing the event instance. A consumed event instance is no longer available for processing.

States

A state captures the relevant aspects of the system's history very efficiently. To relate this concept to programming, this means that instead of recording the event history in a multitude of variables, flags, and convoluted logic, you rely mainly on just one state variable that can assume only a limited number of a priori determined values. The value of the state variable crisply defines the current state of the system at any given time. The concept of state reduces the problem of identifying the execution context in the code to testing just the state variable instead of many variables, thus eliminating a lot of conditional logic. Moreover, switching between different states

Guard conditions

Guard conditions (or simply guards) are Boolean expressions evaluated dynamically based on the value of extended state variables and event parameters. Guard conditions affect the behavior of a state machine by enabling actions or transitions only when they evaluate to TRUE and disabling them when they evaluate to FALSE. In the UML notation, guard conditions are shown in square brackets.

The need for guards is the immediate consequence of adding memory extended state variables to the state machine formalism. Used sparingly, extended state variables and guards make up a powerful mechanism that can simplify designs. But don not let the fancy name ("guard") and the concise UML notation fool you. When you actually code an extended state machine, the guards become the same IFs and ELSEs that you wanted to eliminate by using the state machine in the first place. Too many of them, and you will find yourself back in square one ("spaghetti code"), where the guards effectively take over handling of all the relevant conditions in the system.

Actions and transitions

When an event instance is dispatched, the state machine responds by performing actions, such as changing a variable, performing I/O, invoking a function, generating another event instance, or changing to another state. Any parameter values associated with the current event are available to all actions directly caused by that event. Switching from one state to another is called state transition, and the event that causes it is called the triggering event, or simply the trigger.

Entry and exit actions

Every state in a UML statechart can have optional entry actions, which are executed upon entry to a state, as well as optional exit actions, which are executed upon exit from a state. Entry and exit actions are associated with states, not transitions.

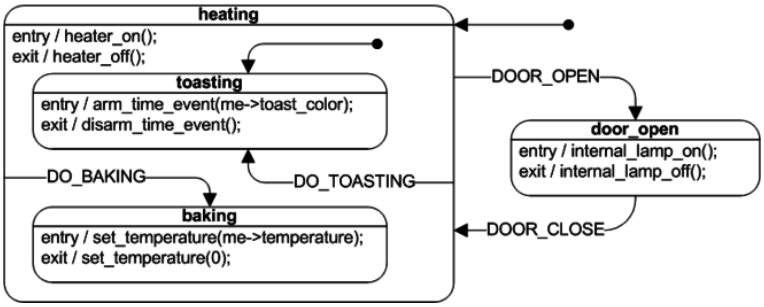


Figure 2 Entry/Exit Actions

Regardless of how a state is entered or exited, all its entry and exit actions will be executed. Because of this characteristic, statecharts behave like Moore machines. The UML notation for state entry and exit actions is to place the reserved word "entry" (or "exit") in the state right below the name compartment, followed by the forward slash and the list of arbitrary actions.

Internal transitions

Very commonly, an event causes only some internal actions to execute but does not lead to a change of state (state transition). In this case, all actions executed comprise the internal transition. In the absence of entry and exit actions, internal transitions would be identical to self-transitions (transitions in which the target state is the same as the source state).

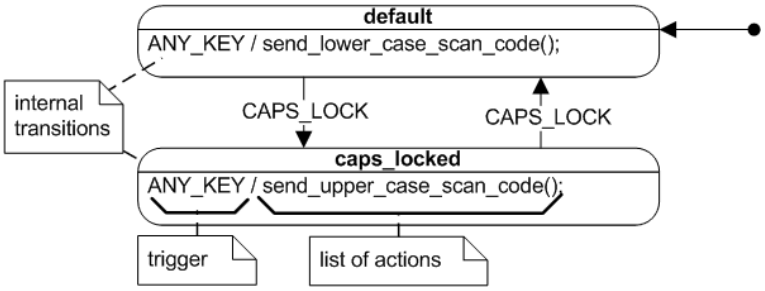


Figure 3 Internal Transitions

In fact, in a classical Mealy machine, actions are associated exclusively with state transitions, so the only way to execute actions without changing state is through a self-transition. However, in the presence of entry and exit actions, as in UML statecharts, a self-transition involves the execution of exit and entry actions and therefore it is distinctively different from an internal transition.

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

4 File Index

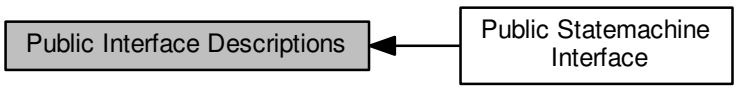
4.1 File List

Here is a list of all files with brief descriptions:

5 Module Documentation

5.1 Public Interface Descriptions

Collaboration diagram for Public Interface Descriptions:



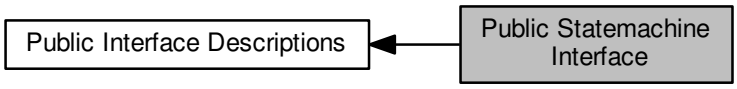
Modules

- Public Statemachine Interface

5.1.1 Detailed Description

5.2 Public Statemachine Interface

Collaboration diagram for Public Statemachine Interface:



Functions

- sm_state_t * sm_init (sm_t *sm, const sm_state_t *state)

  Initialize a statemachine with a provided initial state.
- void sm_terminate (sm_t *sm)

  Terminates the statemachine.
- sm_state_t * sm_send (sm_t *sm, event_t event, void *data)

  Sends an event to a statemachine.

5.2.1 Detailed Description

5.2.2 Function Documentation

5.2.2.1 sm_state_t* sm_init (sm_t * *sm*, const sm_state_t * *state*)

Initialize a statemachine with a provided initial state.

Initializes the statemachine with a provided state. The Initial State from the UML Statechart Diagram is the state of an object before any transitions. The Initial State from the UML Statechart Diagram marks the entry point and the initial statemachine state. The notation for the Initial State is a small solid filled circle. There can only be one Initial State on a diagram. Invoking this function transits from initial pseudo state to the initial statemachine state.

Parameters

in,out	*sm*	State machine instance
in	*state*	State to initialize the state machine with. Has to be a valid state of the state machines state table.

Returns

State of the state machine after initial transition. NULL if initialization failed

Definition at line 116 of file sm.c.

Here is the caller graph for this function:

5.2.2.2 sm_state_t* sm_send (sm_t * *sm*, event_t *event*, void * *data*)

Sends an event to a statemachine.

Parameters

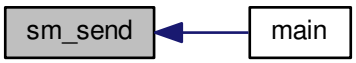
in,out	*sm*	State machine instance
in	*event*	Event to be sent to the state machine
in,out	*data*	Data associated with the event

Returns

State of the state machine after transition. NULL if transition failed.

Definition at line 166 of file sm.c.

Here is the caller graph for this function:



5.2.2.3 void sm_terminate (sm_t * *sm*)

Terminates the statemachine.

This function is meant to be used to terminate a statemachine. Usually a statemachine terminates itself by entering a pseudo final state.

Parameters

in,out	*sm*	State machine instance

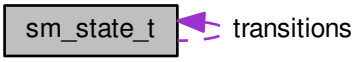
Definition at line 140 of file sm.c.

6 Data Structure Documentation

6.1 sm_state_t Struct Reference

`#include <sm.h>`

Collaboration diagram for sm_state_t:



Data Fields

- sm_entry_action_fp entry_action
- sm_transitions_fp transitions
- sm_exit_action_fp exit_action

6.1.1 Detailed Description

A state captures the relevant aspects of the system's history very efficiently. The state of an object is always determined by its attributes and associations. States in state chart diagrams represent a set of those value combinations, in which an object behaves the same in response to events.

Definition at line 165 of file sm.h.

6.1.2 Field Documentation

6.1.2.1 sm_entry_action_fp entry_action

entry action to be performed if the state gets entered

Definition at line 166 of file sm.h.

6.1.2.2 sm_exit_action_fp exit_action

exit action to be performed if the state gets exited

Definition at line 168 of file sm.h.

6.1.2.3 sm_transitions_fp transitions

external,internal and self transitions of the state

Definition at line 167 of file sm.h.

The documentation for this struct was generated from the following file:

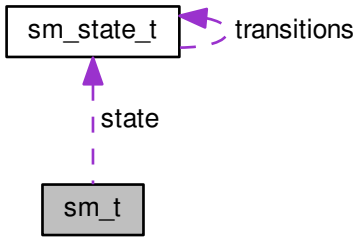
- src/sm.h

6.2 sm_t Struct Reference

Statemachine declaration.

`#include <sm.h>`

Collaboration diagram for sm_t:



Data Fields

- sm_state_t * state
- void * data

6.2.1 Detailed Description

Statemachine declaration.

Holds the current state and data associated with the statemachine.

Definition at line 175 of file sm.h.

6.2.2 Field Documentation

6.2.2.1 void* data

(Object-) Data associated with the statemachine

Definition at line 177 of file sm.h.

6.2.2.2 sm_state_t* state

Current state of the statemachine

Definition at line 176 of file sm.h.

The documentation for this struct was generated from the following file:

- src/sm.h

7 File Documentation

7.1 doxy/mainpage.dox File Reference

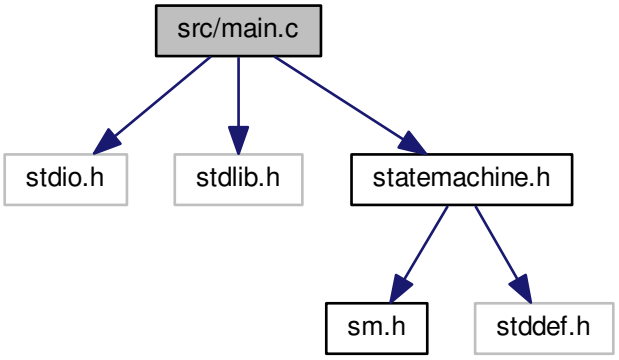
7.2 doxy/modules.dox File Reference

7.3 src/main.c File Reference

Main entry point of application.

```
#include <stdio.h>
#include <stdlib.h>
#include "statemachine.h"
```

Include dependency graph for main.c:



Functions

- int main (void)

  Main entry point.

7.3.1 Detailed Description

Main entry point of application.

Copyright

Copyright (c) 2013, marco@bacchi.at All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR “AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DI↩ RECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PR↩ OFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Version

0000

Authors

marco@bacchi.at

Date

2013

Test Application for testing purposes during development. Has to be replaced with a unittest

Changelist

Revision	Date	Name	Change
0000	00.00.00	bacmar	Detailed Change Text

7.3.2 Function Documentation

7.3.2.1 int main (void)

Main entry point.

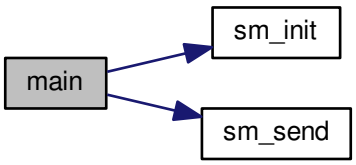
Initializes a designed and implemented statemachine with its initial state. Gets input from keyboard and forwards it as event to the statemachine. Valid events are from a to e.

Returns

EXIT_SUCCESS in case of success. Otherwise EXIT_FAILURE.

Definition at line 62 of file main.c.

Here is the call graph for this function:

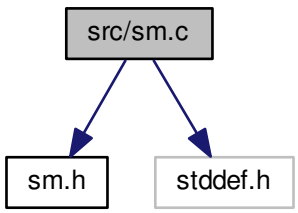


7.4 src/sm.c File Reference

General UML state machine implementation.

```
#include "sm.h"
#include <stddef.h>
```

Include dependency graph for sm.c:



Functions

- sm_state_t ∗ sm_init (sm_t ∗sm, const sm_state_t ∗state)

  Initialize a statemachine with a provided initial state.
- void sm_terminate (sm_t ∗sm)

  Terminates the statemachine.
- sm_state_t ∗ sm_send (sm_t ∗sm, event_t event, void ∗data)

  Sends an event to a statemachine.

Variables

- sm_transition_effect_fp sm_transition_effect = NULL

  Action to carry out in case of a specific transition.

7.4.1 Detailed Description

General UML state machine implementation.

Copyright

Version

0000

Authors

marco@bacchi.at

Date

2013

UML state machine, also known as UML state chart, is a significantly enhanced realization of the mathematical concept of a finite automaton in computer science applications as expressed in the Unified Modeling Language (UML) notation. (wikipedia)

The concepts behind it are about organizing the way a device, computer program, or other (often technical) process works such that an entity or each of its sub-entities is always in exactly one of a number of possible states and where there are well-defined conditional transitions between these states. (wikipedia)

This module and its interface provides the general implementation part for any state machine written for this implementation.

This implementation supports:

- finite flat state machines
- guard conditions for transitions
- state entry and exit actions
- do actions by starting/stopping (proto)threads in entry/exit action respectively
- transition effects (actions directly associated with a specific transition)

Limitations of this implementation:

- not interrupt/thread safe
- hierarchical state machines not supported
- execution time overhead due to generalization in comparison to quick and dirty FSMs
- transition fork/join not supported
- history states not supported
- sending events in entry/exit/transition/effect functions not allowed (due to recursion)

References: ISBN 3-8273-1486-0 Das UML-Benutzerhandbuch

Changelog

Revision	Date	Name	Change
0000	00.00.00	bacmar	Detailed Change Text

7.4.2 Variable Documentation

7.4.2.1 sm_transition_effect_fp sm_transition_effect = NULL

Action to carry out in case of a specific transition.

Callback function type for transition effects.

You can associate a transition with an effect, which is an optional activity performed when a specific transition fires. The associated function may only be set within a transition function. It will be reset to NULL again after state transition. Transition effects will not work for internal state transitions.

Remarks

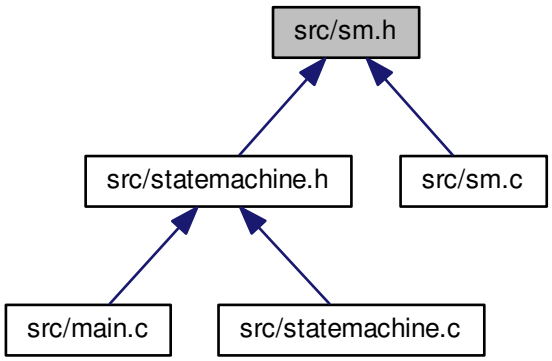
Move to sm instance in future (for future thread safe version) or to transition function as parameter

Definition at line 93 of file sm.c.

7.5 src/sm.h File Reference

General UML state machine interface.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct sm_state_t
- struct sm_t

  Statemachine declaration.

Macros

- #define SM_EVENT_INIT (~((event_t)0))
- #define SM_EVENT_EXIT (SM_EVENT_INIT-1)
- #define sm_state_id(base_ptr, state_ptr) ((state_ptr)-(base_ptr))

  Preprocessor macro to retrieve state id of a state.

Typedefs

- typedef unsigned int event_t
- typedef struct sm_state_t sm_state_t
- typedef void(∗ sm_entry_action_fp) (event_t event, void ∗data)

  Callback function type for entry actions.
- typedef void(∗ sm_exit_action_fp) (event_t event, void ∗data)

  Callback function type for exit actions.
- typedef const sm_state_t ∗(∗ sm_transitions_fp) (event_t event, void ∗data)

  Callback function type for transitions.
- typedef void(∗ sm_transition_effect_fp) (event_t event, void ∗data)

  Callback function type for transition effects.

Functions

- sm_state_t ∗ sm_init (sm_t ∗sm, const sm_state_t ∗state)

  Initialize a statemachine with a provided initial state.
- void sm_terminate (sm_t ∗sm)

  Terminates the statemachine.
- sm_state_t ∗ sm_send (sm_t ∗sm, event_t event, void ∗data)

  Sends an event to a statemachine.

Variables

- sm_transition_effect_fp sm_transition_effect

  Callback function type for transition effects.

7.5.1 Detailed Description

General UML state machine interface.

Copyright

Version

0000

Authors

marco@bacchi.at

Date

2013

UML state machine, also known as UML state chart, is a significantly enhanced realization of the mathematical concept of a finite automaton in computer science applications as expressed in the Unified Modeling Language (UML) notation. (wikipedia)

The concepts behind it are about organizing the way a device, computer program, or other (often technical) process works such that an entity or each of its sub-entities is always in exactly one of a number of possible states and where there are well-defined conditional transitions between these states. (wikipedia)

This module and its interface provides the general implementation part for any state machine written for this implementation.

This implementation supports:

- finite flat state machines
- guard conditions for transitions
- state entry and exit actions
- do actions by starting/stopping (proto)threads in entry/exit action respectively
- transition effects (actions directly associated with a specific transition)

Limitations of this implementation:

- not interrupt/thread safe
- hierarchical state machines not supported
- execution time overhead due to generalization in comparison to quick and dirty FSMs
- transition fork/join not supported
- history states not supported
- sending events in entry/exit/transition/effect functions not allowed (due to recursion)

Changelog

Revision	Date	Name	Change
0000	00.00.00	bacmar	Detailed Change Text

References: ISBN 3-8273-1486-0 Das UML-Benutzerhandbuch

7.5.2 Macro Definition Documentation

7.5.2.1 #define SM_EVENT_EXIT (SM_EVENT_INIT-1)

Definition of event id to terminate a statemachine, must not be sent by the user

Definition at line 83 of file sm.h.

7.5.2.2 #define SM_EVENT_INIT (~((event_t)0))

Definition of event id to initialize a statemachine, must not be sent by the user

Definition at line 81 of file sm.h.

7.5.2.3 #define sm_state_id(*base_ptr, state_ptr*) ((state_ptr)-(base_ptr))

Preprocessor macro to retrieve state id of a state.

Preprocessor macro to retrieve state id of a state.

Parameters

in	*base_ptr*	Pointer to the state table
in	*state_ptr*	Pointer to a specific state of the state table

Returns

id of the specific state

Definition at line 190 of file sm.h.

7.5.3 Typedef Documentation

7.5.3.1 typedef unsigned int event_t

An event is something that happens that affects the system. An event can have associated parameters, allowing the event instance to convey not only the occurrence of some interesting incident but also quantitative information regarding that occurrence. The associated parameters can be forwarded to the state machine via the data pointer of the sm_send function

Definition at line 92 of file sm.h.

7.5.3.2 typedef void(* sm_entry_action_fp) (event_t event, void *data)

Callback function type for entry actions.

Every state in a UML state chart can have an optional entry action, which is executed upon entry to a state. Entry actions are associated with states, not transitions. Regardless of how a state is entered, its entry action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 110 of file sm.h.

7.5.3.3 typedef void(* sm_exit_action_fp) (event_t event, void *data)

Callback function type for exit actions.

Every state in a UML state chart can have an optional exit action, which is executed upon exit from a state. Exit actions are associated with states, not transitions. Regardless of how a state is left, its exit action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 122 of file sm.h.

7.5.3.4 typedef struct sm_state_t sm_state_t

Forward declaration of sm_state structure

Definition at line 97 of file sm.h.

7.5.3.5 typedef void(* sm_transition_effect_fp) (event_t event, void *data)

Callback function type for transition effects.

Switching from one state to another is called state transition. An effect specifies an optional behavior to be performed when the transition fires. The transition effect has to be set in the transition function and is invoked after exit of the source state and before entry of the target state.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 149 of file sm.h.

7.5.3.6 typedef const sm_state_t*(* sm_transitions_fp) (event_t event, void *data)

Callback function type for transitions.

Switching from one state to another is called state transition, and the event that causes it is called the triggering event, or simply the trigger. This function has to be implemented by the state machine designer for each state of the state machine.

Parameters

event	Triggering event
data	Data associated with the event

Returns

New state in case of external transition. Same state as before in case of self transition. NULL in case of internal or no transition.

Definition at line 137 of file sm.h.

7.5.4 Variable Documentation

7.5.4.1 sm_transition_effect_fp sm_transition_effect

Callback function type for transition effects.

Specifies an optional behavior to be performed when the transition fires.global variable for the transition effect

Callback function type for transition effects.

You can associate a transition with an effect, which is an optional activity performed when a specific transition fires. The associated function may only be set within a transition function. It will be reset to NULL again after state transition. Transition effects will not work for internal state transitions.

Remarks

Move to sm instance in future (for future thread safe version) or to transition function as parameter

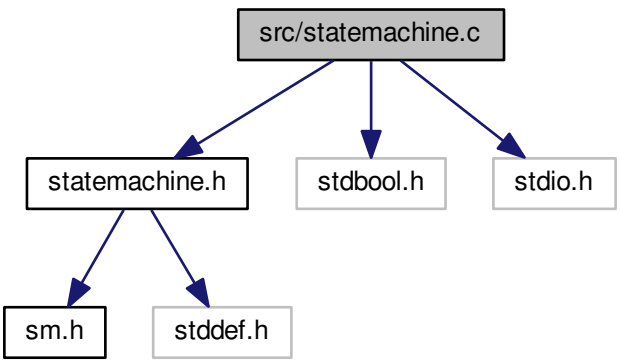
Definition at line 93 of file sm.c.

7.6 src/statemachine.c File Reference

Statemachine for testing purposes - implementation.

```
#include "statemachine.h"
#include <stdbool.h>
#include <stdio.h>
```

Include dependency graph for statemachine.c:



Macros

- #define DEBUG 1
- #define DBG(...) printf(__VA_ARGS__)

Functions

- void A_entry (event_t event, void *data)

  State A entry function.
- const sm_state_t * A_transitions (event_t event, void *data)

  State A transitions function.
- void A_exit (event_t event, void *data)

  State A exit function.
- void BeC_transition_effect (event_t event, void *data)
- const sm_state_t * B_transitions (event_t event, void *data)

  State B transitions function.
- void B_exit (event_t event, void *data)

  State B exit function.
- void C_entry (event_t event, void *data)

  State C entry function.
- const sm_state_t * C_transitions (event_t event, void *data)

  State C transitions function.

Variables

- bool guard = false
- const sm_state_t statemachine_states [ ]
- const size_t statemachine_state_count = sizeof(statemachine_states)/sizeof(statemachine_states[0])

7.6.1 Detailed Description

Statemachine for testing purposes - implementation.

Copyright

Version

0000

Authors

marco@bacchi.at

Date

2013

This statemachine is for testing purposes. It defines three states and some testing transitions between the states. It also makes use of a guard condition and covers exit/entry actions, transition effects and internal transitions.

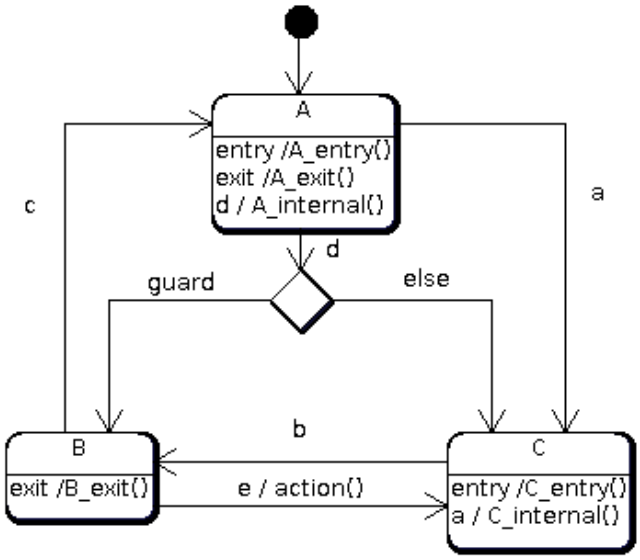


Figure 4 Statemachine for testing purposes

Changelist

Revision	Date	Name	Change
0000	00.00.00	bacmar	Detailed Change Text

7.6.2 Macro Definition Documentation

7.6.2.1 #define DBG(...) printf(__VA_ARGS__)

Definition at line 58 of file statemachine.c.

7.6.2.2 #define DEBUG 1

Definition at line 54 of file statemachine.c.

7.6.3 Function Documentation

7.6.3.1 void A_entry (event_t *event*, void * *data*)

State A entry function.

Every state in a UML state chart can have an optional entry action, which is executed upon entry to a state. Entry actions are associated with states, not transitions. Regardless of how a state is entered, its entry action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 75 of file statemachine.c.

7.6.3.2 void A_exit (event_t *event*, void * *data*)

State A exit function.

Every state in a UML state chart can have an optional exit action, which is executed upon exit from a state. Exit actions are associated with states, not transitions. Regardless of how a state is left, its exit action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 123 of file statemachine.c.

7.6.3.3 const sm_state_t* A_transitions (event_t *event*, void * *data*)

State A transitions function.

Switching from one state to another is called state transition, and the event that causes it is called the triggering event, or simply the trigger. This function has to be implemented by the state machine designer for each state of the state machine.

Parameters

event	Triggering event
data	Data associated with the event

Returns

New state in case of external transition. Same state as before in case of self transition. NULL in case of internal or no transition.

Definition at line 92 of file statemachine.c.

7.6.3.4 void B_exit (event_t *event*, void * *data*)

State B exit function.

Every state in a UML state chart can have an optional exit action, which is executed upon exit from a state. Exit actions are associated with states, not transitions. Regardless of how a state is left, its exit action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 176 of file statemachine.c.

7.6.3.5 const sm_state_t* B_transitions (event_t *event*, void * *data*)

State B transitions function.

Switching from one state to another is called state transition, and the event that causes it is called the triggering event, or simply the trigger. This function has to be implemented by the state machine designer for each state of the state machine.

Parameters

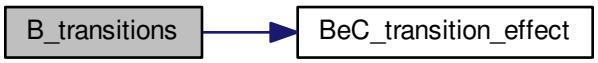
event	Triggering event
data	Data associated with the event

Returns

New state in case of external transition. Same state as before in case of self transition. NULL in case of internal or no transition.

Definition at line 149 of file statemachine.c.

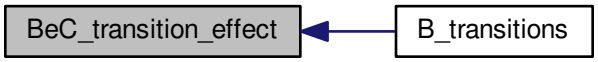
Here is the call graph for this function:



7.6.3.6 void BeC_transition_effect (event_t *event*, void * *data*)

Definition at line 131 of file statemachine.c.

Here is the caller graph for this function:



7.6.3.7 void C_entry (event_t *event,* void * *data*)

State C entry function.

Every state in a UML state chart can have an optional entry action, which is executed upon entry to a state. Entry actions are associated with states, not transitions. Regardless of how a state is entered, its entry action will be executed.

Parameters

event	Triggering event
data	Data associated with the event

Definition at line 193 of file statemachine.c.

7.6.3.8 const sm_state_t* C_transitions (event_t *event,* void * *data*)

State C transitions function.

Switching from one state to another is called state transition, and the event that causes it is called the triggering event, or simply the trigger. This function has to be implemented by the state machine designer for each state of the state machine.

Parameters

event	Triggering event
data	Data associated with the event

Returns

New state in case of external transition. Same state as before in case of self transition. NULL in case of internal or no transition.

Definition at line 211 of file statemachine.c.

7.6.4 Variable Documentation

7.6.4.1 bool guard = false

Guard condition variable as used in UML state chart

Definition at line 63 of file statemachine.c.

7.6.4.2 const size_t statemachine_state_count = sizeof(statemachine_states)/sizeof(statemachine_states[0])

Number of state entries in the state table

Definition at line 239 of file statemachine.c.

7.6.4.3 const sm_state_t statemachine_states[ ]

Initial value:

```
= {
      {A_entry,      A_transitions,       A_exit},
      {NULL,         B_transitions,       B_exit},
      {C_entry,      C_transitions,       NULL}
}
```

Statemachine states table

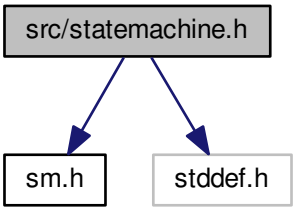
Definition at line 232 of file statemachine.c.

7.7 src/statemachine.h File Reference

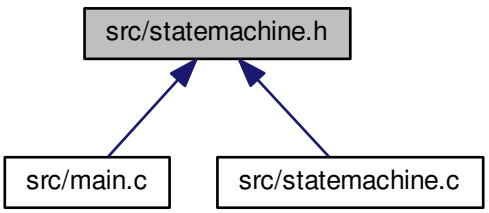
Statemachine for testing purposes - interface.

```
#include "sm.h"
#include "stddef.h"
```

Include dependency graph for statemachine.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum statemachine_event_t {
  a, b, c, d,
  e }

  Enumeration of events.
- enum statemachine_states_t { A, B, C }

  Enumeration of states.

Variables

- const sm_state_t statemachine_states [ ]
- const size_t statemachine_state_count

7.7.1 Detailed Description

Statemachine for testing purposes - interface.

Copyright

Copyright (c) 2013, marco@bacchi.at All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR “AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DI↩ RECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PR↩ OFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Version

0000

Authors

marco@bacchi.at

Date

2013

This statemachine is for testing purposes. It defines three states and some testing transitions between the states. It also makes use of a guard condition and covers exit/entry actions, transition effects and internal transitions.

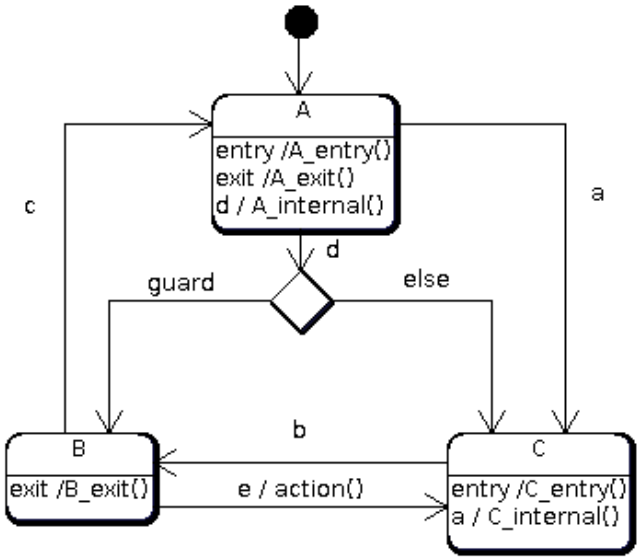


Figure 5 Statemachine for testing purposes

Changelist

Revision	Date	Name	Change
0000	00.00.00	bacmar	Detailed Change Text

7.7.2 Enumeration Type Documentation

7.7.2.1 enum statemachine_event_t

Enumeration of events.

Enumerator

a Special event used for transiting from state A to state C

b Same as event a, but this time for transition from state C to state B

c Event used for transition from state B to A

d Event used to demonstrate a trnsition with an external guard condition

e Event used to demonstrate an extrnal transition with a transition effect

Definition at line 61 of file statemachine.h.

7.7.2.2 enum statemachine_states_t

Enumeration of states.

Enumeration of states of the state machine. This enumeration has to match with the entries in the state table!

Enumerator

- ***A*** State with entry/exit action, an internal transition, and external transition and an external contition with guard condition
- ***B*** State with exit action, an external tranistion and another external transition with a transition effect
- ***C*** State with entry action, internal tranition and eternal transition

Definition at line 74 of file statemachine.h.

7.7.3 Variable Documentation

7.7.3.1 const size_t statemachine_state_count

Number of state entries in the state table

Definition at line 239 of file statemachine.c.

7.7.3.2 const sm_state_t statemachine_states[ ]

Statemachine states table

Definition at line 232 of file statemachine.c.

Index