JavaScript Reference

Namespace management

The classes and functions that form the public interface of state.js are documented here; please do not use any other methods or functions you may find in the code directly as these may be subject to internal refactoring.

Initialisation of state.js occurs in two ways: if an 'exports' object is found, the classes that form state.js are added to it; otherwise, the classes are added the global object. This (hopefully) enables state.js to work with popular package managers (especially node package manager) or in a standalone manner.

The implementation of this state machine splits the state machine model (hierarchy and transitions) from the state machine state. Classes are used to create the state machine hierarchy and transitions; any object may be used as the state machine state and it is dynamically augmented as necessary.

Classes

Class	Description
CompositeState	A state that contains child states and pseudo states.
FinalState	A state that cannot be transitioned out of.
OrthogonalState	A state than contains child regions.
PseudoState	A transient state with behaviour defined by its 'kind'.
Region	A container of states and pseudo states.
SimpleState	A leaf-level state within a state machine hierarchy.
StateMachine	The root element of a state machine hierarchy
Transition	A transition from a state or pseudo state to another state or pseudo state.
Transition.Else	A catch-all completion transition from junction or completion pseudo states used where no other transitions guards evaluate true.

Enumerations

Euneration	Description
PseudoStateKind	Defines the behaviour of an individual PseudoState.

CompositeState Class

A state that contains child states and pseudo states.

Syntax

```
new CompositeState(name, owner)
```

name

The name the composite state will be initialised with.

owner

The owning (parent) region or composite state of the new composite state.

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.
entry	An array of functions to call on entry to the composite state; create or add to this array to define state entry behaviour.
exit	An array of functions to call on leaving the composite state; create or add to this array to define state exit behaviour.

Remarks

A CompositeState is the most straightforward way to add hierarchy to a state machine. It fulfils all the semantics of SimpleState and augments it with child states and pseudo states.

FinalState Class

A state that cannot be transitioned out of.

Syntax

```
new FinalState(name, owner)
```

name

The name the final state will be initialised with.

owner

The owning (parent) region or composite state of the new final state.

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.

Remarks

A FinalState marks is owning Region or CompositeState as being ‘complete’ and as such, are a critical mechanism for determining if the lifecycle of the entire object under state management is complete.

Completion transitions from CompositeStates will only be evaluated if the CompositeStates current active child is a FinalState.

Completion transitions from OrthogonalStates will only be evaluated if the current active child state of all the OrthogonalStates child Regions are FinalStates.

OrthogonalState Class

A state that contains child Regions.

Syntax

```
new OrthogonalState(name, owner)
```

name

The name the orthogonal state will be initialised with.

owner

The owning (parent) region or composite state of the new orthogonal state

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.
entry	An array of functions to call on entry to the composite state; create or add to this array to define state entry behaviour.
exit	An array of functions to call on leaving the composite state; create or add to this array to define state exit behaviour.

Remarks

An OrthogonalState add hierarchy to a state machine through multiple child Regions. It fulfils all the semantics of the base SimpleState and augments it with child states and pseudo states.

The child Regions of an OrthogonalState are meant to be independent of each other; any messages and actions delegated to an OrthogonalState are further delegated to all child Regions as required.

Therefore, when OrthogonalStates are used, the current state of a state machine can actually have multiple values as each child Region maintains its own current state.

PseudoState Class

A transient state with behaviour defined by its ‘kind’.

Syntax

```
new PseudoState(name, kind, owner)
```

name

The name the pseudo state will be initialised with.

kind

A member of the PseudoStateKind enumeration that dictates the behaviour of the pseudo state.

owner

The owning (parent) region or composite state of the new pseudo state.

Properties

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.

Remarks

A PseudoState is a transient state in a state machine without entry or exit behaviour; upon entry, completion transitions will be evaluated to determine the next state of the state machine.

Each Region or CompositeState must have a single initial PseudoState who’s kind is Initial, DeepHistory or ShallowHistory; this is used to control the initial starting position when entering the Region or CompositeState.

Other kinds of PseudoStates are used as branching mechanisms to provide composite transitions enabling complex logic and behaviour between states.

Region Class

A container of states and pseudo states..

Syntax

```
new Region(name, owner)
```

name

The name the region will be initialised with.

owner

The owning (parent) state machine or orthogonal state of the new region. Owner may be omitted when the region is used as a top-level of the state machine hierarchy.

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.

Methods

Name	Description
initialise(state)	Initialises a Region to its initial state when used as a top-level state machine.
isComplete(state)	Determines if a Region is complete by testing if the current active child is a FinalState.
process(state, message)	Attempts to process a message. This will delegated the to the currently active child state.

Remarks

A Region is a container of states and pseudo states, usually as a child of an OrthogonalState.

A Region is recommended as the top-level of a state machine in most circumstances (only where the top level required orthogonal regions would an OrthogonalState be a better choice).

SimpleState Class

A leaf-level state within a state machine hierarchy.

Syntax

```
new SimpleState(name, owner)
```

`name`

The name the simple state will be initialised with.

`owner`

The owning (parent) region or composite state of the new simple state.

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.
entry	An array of functions to call on entry to the composite state; create or add to this array to define state entry behaviour.
exit	An array of functions to call on leaving the composite state; create or add to this array to define state exit behaviour.

Remarks

A SimpleState is the most straightforward state in a state machine. It defines a condition during the life of an object under state management. Upon entering the state, the entry functions are called and completion transitions are tested for; upon exiting the state the exit functions are called.

StateMachine Class

The root element of a state machine hierarchy.

Syntax

```
new StateMachine(name)
```

name

The name the simple state will be initialised with.

Members

Name	Description
name	The name of the composite state.
owner	The owning (parent) region or composite state.

Methods

Name	Description
initialise(state)	Initialises a StateMachine to its initial state when used as a top-level state machine.
isComplete(state)	Determines if a StateMachine is complete by testing if the child regions are all complete.
process(state, message)	Attempts to process a message. This will be delegated to all child regions.

Remarks

A StateMachine forms the root element of a state machine hierarchy. If performance critical, and no orthogonal regions are required at the top level, using a Region instead of a StateMachine is preferred.

Transition Class

A transition from a state or pseudo state to another state or pseudo state.

Syntax

```
new Transition(source, target, guard)
```

source

The source state or pseudo state.

target

The target state or pseudo state. This can be left null or undefined where an internal transition is require.

guard

An optional boolean function to act as a guard condition to control when transitions are traversed. For completion transitions, the function takes no parameters, for message based transitions, the function takes the message as as a parameter. When defining an internal transition, a guard condition must be supplied.

Members

Name	Description
effect	An array of functions to call on the traversal of a transition. For completion transitions, this is a function without parameters, for message based transitions this takes the message as a parameter.

Remarks

The transition class caters for all types of transition in state.js.

When a message is passed to a state for evaluation, transition from the source state have their guard conditions is evaluated. If a single matching transition is found it is traversed and a state transition occurs. If multiple are found, an exception is thrown as the machine is deemed to be malformed.

If the target of the transition is a PseudoState (or another SimpleState or derivative that is 'completed'), its completion transitions are evaluated and so a composite transition occurs.

A transition without a target specified is known as an internal transition, when this is traversed the state is not exited or entered and only the traversal effect is performed.

A self-transition can be created by specifying the source and target as the same state, in which case the state is exited, transition effect is performed and then the state is re-entered.

Transitions are not limited to SimpleStates and PseudoStates in the same containing Region or CompositeState; external transitions are allowed that jump across the state machine hierarchy. In these cases, the Exit operation cascades to up to, but not including the least common ancestor in the state machine hierarchy followed by a cascade of the Entry operation to the target state or pseudo state.

Transition.Else Class

A catch-all completion transition from junction or completion pseudo states used where no other transitions guards evaluate true.

Syntax

```
new Transition.Else(source, target)
```

`source`

The pseudo state.

`target`

The target state or pseudo state.

Members

Name	Description
effect	An array of functions to call on the traversal of a transition. For completion transitions, this is a function without parameters, for message based transitions this takes the message as a parameter.

Remarks

A Transition.Else is a special type of completion transition available as a catch-all for circumstances where the guard conditions of regular completion transition may not cater for every eventuality.

It is highly recommended to use Transition.Else completion transitions where you cannot guarantee 100% coverage of guard conditions.

PseudoStateKind Enumeration

Defines the behaviour of an individual PseudoState.

Members

Member Name	Description
Choice	Enables a dynamic conditional branches; within a compound transition.
DeepHistory	A type of initial pseudo state; forms the initial starting point when entering a region or composite state for the first time.
Initial	A type of initial pseudo state; forms the initial starting point when entering a region or composite state.
Junction	Enables a static conditional branches; within a compound transition.
ShallowHistory	A type of initial pseudo state; forms the initial starting point when entering a region or composite state for the first time.
Terminate	Entering a terminate PseudoState implies that the execution of this state machine by means of its context object is terminated.

Remarks

The kind of a PseudoState defines its behaviour.

Initial, ShallowHistory and DeepHistory are all 'initial' kinds, meaning they define the behaviour when entering a Region or CompositeState. Upon the first entry to the owning Region or CompositeState the initial PseudoState will be entered and its single completion transition traversed. Subsequent entry to the owning Region or Composite state will have the following behaviour:

- Initial: as per the first entry, the Initial pseudo state will be entered.
- ShallowHistory: the last known child state of the owing Region or CompositeState will be entered.
- DeepHistory: as per ShallowHistory, but all child Regions or CompositeStates will have history semantics applied as well.

Choice and Junction PseudoStates are used to break up transitions to enable complex logic and behaviour; they differ where multiple outbound guard conditions evaluate true:

- Choice will select one of the matching transitions at random.
- Junction will consider the state machine to be malformed and throw an exception.

The use of Choice PseudoStates is ill-advised where repeatable behaviour is required.