[ScrollMagic](http://docs.google.com/index.html)

* [Classes](http://docs.google.com/classes.list.html)
  + [Controller](http://docs.google.com/ScrollMagic.Controller.html)
  + [Scene](http://docs.google.com/ScrollMagic.Scene.html)
* [Events](http://docs.google.com/events.list.html)
  + [add](http://docs.google.com/ScrollMagic.Scene.html#event:add)
  + [change](http://docs.google.com/ScrollMagic.Scene.html#event:change)
  + [destroy](http://docs.google.com/ScrollMagic.Scene.html#event:destroy)
  + [end](http://docs.google.com/ScrollMagic.Scene.html#event:end)
  + [enter](http://docs.google.com/ScrollMagic.Scene.html#event:enter)
  + [leave](http://docs.google.com/ScrollMagic.Scene.html#event:leave)
  + [progress](http://docs.google.com/ScrollMagic.Scene.html#event:progress)
  + [remove](http://docs.google.com/ScrollMagic.Scene.html#event:remove)
  + [shift](http://docs.google.com/ScrollMagic.Scene.html#event:shift)
  + [start](http://docs.google.com/ScrollMagic.Scene.html#event:start)
  + [update](http://docs.google.com/ScrollMagic.Scene.html#event:update)
* [Plugins](http://docs.google.com/mixins.list.html)
  + [GSAP](http://docs.google.com/animation.GSAP.html)
  + [Velocity](http://docs.google.com/animation.Velocity.html)
  + [addIndicators](http://docs.google.com/debug.addIndicators.html)
  + [jQuery](http://docs.google.com/framework.jQuery.html)

Source: ScrollMagic/Scene/event-management.js

/\*  
 \* ----------------------------------------------------------------  
 \* Event Management  
 \* ----------------------------------------------------------------  
 \*/  
  
var \_listeners = {};  
/\*\*  
 \* Scene start event.   
 \* Fires whenever the scroll position its the starting point of the scene.   
 \* It will also fire when scrolling back up going over the start position of the scene. If you want something to happen only when scrolling down/right, use the scrollDirection parameter passed to the callback.  
 \*  
 \* For details on this event and the order in which it is fired, please review the {@link Scene.progress} method.  
 \*  
 \* @event ScrollMagic.Scene#start  
 \*  
 \* @example  
 \* scene.on("start", function (event) {  
 \* console.log("Hit start point of scene.");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.progress - Reflects the current progress of the scene  
 \* @property {string} event.state - The current state of the scene `"BEFORE"` or `"DURING"`  
 \* @property {string} event.scrollDirection - Indicates which way we are scrolling `"PAUSED"`, `"FORWARD"` or `"REVERSE"`  
 \*/  
/\*\*  
 \* Scene end event.   
 \* Fires whenever the scroll position its the ending point of the scene.   
 \* It will also fire when scrolling back up from after the scene and going over its end position. If you want something to happen only when scrolling down/right, use the scrollDirection parameter passed to the callback.  
 \*  
 \* For details on this event and the order in which it is fired, please review the {@link Scene.progress} method.  
 \*  
 \* @event ScrollMagic.Scene#end  
 \*  
 \* @example  
 \* scene.on("end", function (event) {  
 \* console.log("Hit end point of scene.");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.progress - Reflects the current progress of the scene  
 \* @property {string} event.state - The current state of the scene `"DURING"` or `"AFTER"`  
 \* @property {string} event.scrollDirection - Indicates which way we are scrolling `"PAUSED"`, `"FORWARD"` or `"REVERSE"`  
 \*/  
/\*\*  
 \* Scene enter event.   
 \* Fires whenever the scene enters the "DURING" state.   
 \* Keep in mind that it doesn't matter if the scene plays forward or backward: This event always fires when the scene enters its active scroll timeframe, regardless of the scroll-direction.  
 \*  
 \* For details on this event and the order in which it is fired, please review the {@link Scene.progress} method.  
 \*  
 \* @event ScrollMagic.Scene#enter  
 \*  
 \* @example  
 \* scene.on("enter", function (event) {  
 \* console.log("Scene entered.");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.progress - Reflects the current progress of the scene  
 \* @property {string} event.state - The current state of the scene - always `"DURING"`  
 \* @property {string} event.scrollDirection - Indicates which way we are scrolling `"PAUSED"`, `"FORWARD"` or `"REVERSE"`  
 \*/  
/\*\*  
 \* Scene leave event.   
 \* Fires whenever the scene's state goes from "DURING" to either "BEFORE" or "AFTER".   
 \* Keep in mind that it doesn't matter if the scene plays forward or backward: This event always fires when the scene leaves its active scroll timeframe, regardless of the scroll-direction.  
 \*  
 \* For details on this event and the order in which it is fired, please review the {@link Scene.progress} method.  
 \*  
 \* @event ScrollMagic.Scene#leave  
 \*  
 \* @example  
 \* scene.on("leave", function (event) {  
 \* console.log("Scene left.");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.progress - Reflects the current progress of the scene  
 \* @property {string} event.state - The current state of the scene `"BEFORE"` or `"AFTER"`  
 \* @property {string} event.scrollDirection - Indicates which way we are scrolling `"PAUSED"`, `"FORWARD"` or `"REVERSE"`  
 \*/  
/\*\*  
 \* Scene update event.   
 \* Fires whenever the scene is updated (but not necessarily changes the progress).  
 \*  
 \* @event ScrollMagic.Scene#update  
 \*  
 \* @example  
 \* scene.on("update", function (event) {  
 \* console.log("Scene updated.");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.startPos - The starting position of the scene (in relation to the conainer)  
 \* @property {number} event.endPos - The ending position of the scene (in relation to the conainer)  
 \* @property {number} event.scrollPos - The current scroll position of the container  
 \*/  
/\*\*  
 \* Scene progress event.   
 \* Fires whenever the progress of the scene changes.  
 \*  
 \* For details on this event and the order in which it is fired, please review the {@link Scene.progress} method.  
 \*  
 \* @event ScrollMagic.Scene#progress  
 \*  
 \* @example  
 \* scene.on("progress", function (event) {  
 \* console.log("Scene progress changed to " + event.progress);  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {number} event.progress - Reflects the current progress of the scene  
 \* @property {string} event.state - The current state of the scene `"BEFORE"`, `"DURING"` or `"AFTER"`  
 \* @property {string} event.scrollDirection - Indicates which way we are scrolling `"PAUSED"`, `"FORWARD"` or `"REVERSE"`  
 \*/  
/\*\*  
 \* Scene change event.   
 \* Fires whenvever a property of the scene is changed.  
 \*  
 \* @event ScrollMagic.Scene#change  
 \*  
 \* @example  
 \* scene.on("change", function (event) {  
 \* console.log("Scene Property \"" + event.what + "\" changed to " + event.newval);  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {string} event.what - Indicates what value has been changed  
 \* @property {mixed} event.newval - The new value of the changed property  
 \*/  
/\*\*  
 \* Scene shift event.   
 \* Fires whenvever the start or end \*\*scroll offset\*\* of the scene change.  
 \* This happens explicitely, when one of these values change: `offset`, `duration` or `triggerHook`.  
 \* It will fire implicitly when the `triggerElement` changes, if the new element has a different position (most cases).  
 \* It will also fire implicitly when the size of the container changes and the triggerHook is anything other than `onLeave`.  
 \*  
 \* @event ScrollMagic.Scene#shift  
 \* @since 1.1.0  
 \*  
 \* @example  
 \* scene.on("shift", function (event) {  
 \* console.log("Scene moved, because the " + event.reason + " has changed.)");  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {string} event.reason - Indicates why the scene has shifted  
 \*/  
/\*\*  
 \* Scene destroy event.   
 \* Fires whenvever the scene is destroyed.  
 \* This can be used to tidy up custom behaviour used in events.  
 \*  
 \* @event ScrollMagic.Scene#destroy  
 \* @since 1.1.0  
 \*  
 \* @example  
 \* scene.on("enter", function (event) {  
 \* // add custom action  
 \* $("#my-elem").left("200");  
 \* })  
 \* .on("destroy", function (event) {  
 \* // reset my element to start position  
 \* if (event.reset) {  
 \* $("#my-elem").left("0");  
 \* }  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {boolean} event.reset - Indicates if the destroy method was called with reset `true` or `false`.  
 \*/  
/\*\*  
 \* Scene add event.   
 \* Fires when the scene is added to a controller.  
 \* This is mostly used by plugins to know that change might be due.  
 \*  
 \* @event ScrollMagic.Scene#add  
 \* @since 2.0.0  
 \*  
 \* @example  
 \* scene.on("add", function (event) {  
 \* console.log('Scene was added to a new controller.');  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \* @property {boolean} event.controller - The controller object the scene was added to.  
 \*/  
/\*\*  
 \* Scene remove event.   
 \* Fires when the scene is removed from a controller.  
 \* This is mostly used by plugins to know that change might be due.  
 \*  
 \* @event ScrollMagic.Scene#remove  
 \* @since 2.0.0  
 \*  
 \* @example  
 \* scene.on("remove", function (event) {  
 \* console.log('Scene was removed from its controller.');  
 \* });  
 \*  
 \* @property {object} event - The event Object passed to each callback  
 \* @property {string} event.type - The name of the event  
 \* @property {Scene} event.target - The Scene object that triggered this event  
 \*/  
  
/\*\*  
 \* Add one ore more event listener.   
 \* The callback function will be fired at the respective event, and an object containing relevant data will be passed to the callback.  
 \* @method ScrollMagic.Scene#on  
 \*  
 \* @example  
 \* function callback (event) {  
 \* console.log("Event fired! (" + event.type + ")");  
 \* }  
 \* // add listeners  
 \* scene.on("change update progress start end enter leave", callback);  
 \*  
 \* @param {string} names - The name or names of the event the callback should be attached to.  
 \* @param {function} callback - A function that should be executed, when the event is dispatched. An event object will be passed to the callback.  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.on = function (names, callback) {  
 if (\_util.type.Function(callback)) {  
 names = names.trim().split(' ');  
 names.forEach(function (fullname) {  
 var  
 nameparts = fullname.split('.'),  
 eventname = nameparts[0],  
 namespace = nameparts[1];  
 if (eventname != "\*") { // disallow wildcards  
 if (!\_listeners[eventname]) {  
 \_listeners[eventname] = [];  
 }  
 \_listeners[eventname].push({  
 namespace: namespace || '',  
 callback: callback  
 });  
 }  
 });  
 } else {  
 log(1, "ERROR when calling '.on()': Supplied callback for '" + names + "' is not a valid function!");  
 }  
 return Scene;  
};  
  
/\*\*  
 \* Remove one or more event listener.  
 \* @method ScrollMagic.Scene#off  
 \*  
 \* @example  
 \* function callback (event) {  
 \* console.log("Event fired! (" + event.type + ")");  
 \* }  
 \* // add listeners  
 \* scene.on("change update", callback);  
 \* // remove listeners  
 \* scene.off("change update", callback);  
 \*  
 \* @param {string} names - The name or names of the event that should be removed.  
 \* @param {function} [callback] - A specific callback function that should be removed. If none is passed all callbacks to the event listener will be removed.  
 \* @returns {Scene} Parent object for chaining.  
\*/  
this.off = function (names, callback) {  
 if (!names) {  
 log(1, "ERROR: Invalid event name supplied.");  
 return Scene;  
 }  
 names = names.trim().split(' ');  
 names.forEach(function (fullname, key) {  
 var  
 nameparts = fullname.split('.'),  
 eventname = nameparts[0],  
 namespace = nameparts[1] || '',  
 removeList = eventname === '\*' ? Object.keys(\_listeners) : [eventname];  
 removeList.forEach(function (remove){  
 var  
 list = \_listeners[remove] || [],  
 i = list.length;  
 while(i--) {  
 var listener = list[i];  
 if (listener && (namespace === listener.namespace || namespace === '\*') && (!callback || callback == listener.callback)) {  
 list.splice(i, 1);  
 }  
 }  
 if (!list.length) {  
 delete \_listeners[remove];  
 }  
 });  
 });  
 return Scene;  
};  
  
 /\*\*  
 \* Trigger an event.  
 \* @method ScrollMagic.Scene#trigger  
 \*  
 \* @example  
 \* this.trigger("change");  
 \*  
 \* @param {string} name - The name of the event that should be triggered.  
 \* @param {object} [vars] - An object containing info that should be passed to the callback.  
 \* @returns {Scene} Parent object for chaining.  
\*/  
this.trigger = function (name, vars) {  
 if (name) {  
 var  
 nameparts = name.trim().split('.'),  
 eventname = nameparts[0],  
 namespace = nameparts[1],  
 listeners = \_listeners[eventname];  
 log(3, 'event fired:', eventname, vars ? "->" : '', vars || '');  
 if (listeners) {  
 listeners.forEach(function (listener, key) {  
 if (!namespace || namespace === listener.namespace) {  
 listener.callback.call(Scene, new ScrollMagic.Event(eventname, listener.namespace, Scene, vars));  
 }  
 });  
 }  
 } else {  
 log(1, "ERROR: Invalid event name supplied.");  
 }  
 return Scene;  
};

© Jan Paepke 2015

Documentation generated by [JSDoc 3.5.5](https://github.com/jsdoc3/jsdoc) using a customized version of the [DocStrap template](https://github.com/terryweiss/docstrap).