[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/core.js

// set event listeners  
Scene  
 .on("change.internal", function (e) {  
 if (e.what !== "loglevel" && e.what !== "tweenChanges") { // no need for a scene update scene with these options...  
 if (e.what === "triggerElement") {  
 updateTriggerElementPosition();  
 } else if (e.what === "reverse") { // the only property left that may have an impact on the current scene state. Everything else is handled by the shift event.  
 Scene.update();  
 }  
 }  
 })  
 .on("shift.internal", function (e) {  
 updateScrollOffset();  
 Scene.update(); // update scene to reflect new position  
 });  
   
// (BUILD) - REMOVE IN MINIFY - START  
/\*\*  
 \* Send a debug message to the console.  
 \* @private  
 \* but provided publicly with \_log for plugins  
 \*  
 \* @param {number} loglevel - The loglevel required to initiate output for the message.  
 \* @param {...mixed} output - One or more variables that should be passed to the console.  
 \*/  
var log = this.\_log = function (loglevel, output) {  
 if (\_options.loglevel >= loglevel) {  
 Array.prototype.splice.call(arguments, 1, 0, "(" + NAMESPACE + ") ->");  
 \_util.log.apply(window, arguments);  
 }  
};  
// (BUILD) - REMOVE IN MINIFY - END  
  
/\*\*  
 \* Add the scene to a controller.   
 \* This is the equivalent to `Controller.addScene(scene)`.  
 \* @method ScrollMagic.Scene#addTo  
 \*  
 \* @example  
 \* // add a scene to a ScrollMagic Controller  
 \* scene.addTo(controller);  
 \*  
 \* @param {ScrollMagic.Controller} controller - The controller to which the scene should be added.  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.addTo = function (controller) {  
 if (!(controller instanceof ScrollMagic.Controller)) {  
 log(1, "ERROR: supplied argument of 'addTo()' is not a valid ScrollMagic Controller");  
 } else if (\_controller != controller) {  
 // new controller  
 if (\_controller) { // was associated to a different controller before, so remove it...  
 \_controller.removeScene(Scene);  
 }  
 \_controller = controller;  
 validateOption();  
 updateDuration(true);  
 updateTriggerElementPosition(true);  
 updateScrollOffset();  
 \_controller.info("container").addEventListener('resize', onContainerResize);  
 controller.addScene(Scene);  
 Scene.trigger("add", {controller: \_controller});  
 log(3, "added " + NAMESPACE + " to controller");  
 Scene.update();  
 }  
 return Scene;  
};  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the current enabled state of the scene.   
 \* This can be used to disable this scene without removing or destroying it.  
 \* @method ScrollMagic.Scene#enabled  
 \*  
 \* @example  
 \* // get the current value  
 \* var enabled = scene.enabled();  
 \*  
 \* // disable the scene  
 \* scene.enabled(false);  
 \*  
 \* @param {boolean} [newState] - The new enabled state of the scene `true` or `false`.  
 \* @returns {(boolean|Scene)} Current enabled state or parent object for chaining.  
 \*/  
this.enabled = function (newState) {  
 if (!arguments.length) { // get  
 return \_enabled;  
 } else if (\_enabled != newState) { // set  
 \_enabled = !!newState;  
 Scene.update(true);  
 }  
 return Scene;  
};  
  
/\*\*  
 \* Remove the scene from the controller.   
 \* This is the equivalent to `Controller.removeScene(scene)`.  
 \* The scene will not be updated anymore until you readd it to a controller.  
 \* To remove the pin or the tween you need to call removeTween() or removePin() respectively.  
 \* @method ScrollMagic.Scene#remove  
 \* @example  
 \* // remove the scene from its controller  
 \* scene.remove();  
 \*  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.remove = function () {  
 if (\_controller) {  
 \_controller.info("container").removeEventListener('resize', onContainerResize);  
 var tmpParent = \_controller;  
 \_controller = undefined;  
 tmpParent.removeScene(Scene);  
 Scene.trigger("remove");  
 log(3, "removed " + NAMESPACE + " from controller");  
 }  
 return Scene;  
};  
  
/\*\*  
 \* Destroy the scene and everything.  
 \* @method ScrollMagic.Scene#destroy  
 \* @example  
 \* // destroy the scene without resetting the pin and tween to their initial positions  
 \* scene = scene.destroy();  
 \*  
 \* // destroy the scene and reset the pin and tween  
 \* scene = scene.destroy(true);  
 \*  
 \* @param {boolean} [reset=false] - If `true` the pin and tween (if existent) will be reset.  
 \* @returns {null} Null to unset handler variables.  
 \*/  
this.destroy = function (reset) {  
 Scene.trigger("destroy", {reset: reset});  
 Scene.remove();  
 Scene.off("\*.\*");  
 log(3, "destroyed " + NAMESPACE + " (reset: " + (reset ? "true" : "false") + ")");  
 return null;  
};  
  
  
/\*\*  
 \* Updates the Scene to reflect the current state.   
 \* This is the equivalent to `Controller.updateScene(scene, immediately)`.   
 \* The update method calculates the scene's start and end position (based on the trigger element, trigger hook, duration and offset) and checks it against the current scroll position of the container.   
 \* It then updates the current scene state accordingly (or does nothing, if the state is already correct) – Pins will be set to their correct position and tweens will be updated to their correct progress.  
 \* This means an update doesn't necessarily result in a progress change. The `progress` event will be fired if the progress has indeed changed between this update and the last.   
 \* \_\*\*NOTE:\*\* This method gets called constantly whenever ScrollMagic detects a change. The only application for you is if you change something outside of the realm of ScrollMagic, like moving the trigger or changing tween parameters.\_  
 \* @method ScrollMagic.Scene#update  
 \* @example  
 \* // update the scene on next tick  
 \* scene.update();  
 \*  
 \* // update the scene immediately  
 \* scene.update(true);  
 \*  
 \* @fires Scene.update  
 \*  
 \* @param {boolean} [immediately=false] - If `true` the update will be instant, if `false` it will wait until next update cycle (better performance).  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.update = function (immediately) {  
 if (\_controller) {  
 if (immediately) {  
 if (\_controller.enabled() && \_enabled) {  
 var  
 scrollPos = \_controller.info("scrollPos"),  
 newProgress;  
  
 if (\_options.duration > 0) {  
 newProgress = (scrollPos - \_scrollOffset.start)/(\_scrollOffset.end - \_scrollOffset.start);  
 } else {  
 newProgress = scrollPos >= \_scrollOffset.start ? 1 : 0;  
 }  
  
 Scene.trigger("update", {startPos: \_scrollOffset.start, endPos: \_scrollOffset.end, scrollPos: scrollPos});  
  
 Scene.progress(newProgress);  
 } else if (\_pin && \_state === SCENE\_STATE\_DURING) {  
 updatePinState(true); // unpin in position  
 }  
 } else {  
 \_controller.updateScene(Scene, false);  
 }  
 }  
 return Scene;  
};  
  
/\*\*  
 \* Updates dynamic scene variables like the trigger element position or the duration.  
 \* This method is automatically called in regular intervals from the controller. See {@link ScrollMagic.Controller} option `refreshInterval`.  
 \*   
 \* You can call it to minimize lag, for example when you intentionally change the position of the triggerElement.  
 \* If you don't it will simply be updated in the next refresh interval of the container, which is usually sufficient.  
 \*  
 \* @method ScrollMagic.Scene#refresh  
 \* @since 1.1.0  
 \* @example  
 \* scene = new ScrollMagic.Scene({triggerElement: "#trigger"});  
 \*   
 \* // change the position of the trigger  
 \* $("#trigger").css("top", 500);  
 \* // immediately let the scene know of this change  
 \* scene.refresh();  
 \*  
 \* @fires {@link Scene.shift}, if the trigger element position or the duration changed  
 \* @fires {@link Scene.change}, if the duration changed  
 \*  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.refresh = function () {  
 updateDuration();  
 updateTriggerElementPosition();  
 // update trigger element position  
 return Scene;  
};  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the scene's progress.   
 \* Usually it shouldn't be necessary to use this as a setter, as it is set automatically by scene.update().   
 \* The order in which the events are fired depends on the duration of the scene:  
 \* 1. Scenes with `duration == 0`:   
 \* Scenes that have no duration by definition have no ending. Thus the `end` event will never be fired.   
 \* When the trigger position of the scene is passed the events are always fired in this order:   
 \* `enter`, `start`, `progress` when scrolling forward   
 \* and   
 \* `progress`, `start`, `leave` when scrolling in reverse  
 \* 2. Scenes with `duration > 0`:   
 \* Scenes with a set duration have a defined start and end point.   
 \* When scrolling past the start position of the scene it will fire these events in this order:   
 \* `enter`, `start`, `progress`   
 \* When continuing to scroll and passing the end point it will fire these events:   
 \* `progress`, `end`, `leave`   
 \* When reversing through the end point these events are fired:   
 \* `enter`, `end`, `progress`   
 \* And when continuing to scroll past the start position in reverse it will fire:   
 \* `progress`, `start`, `leave`   
 \* In between start and end the `progress` event will be called constantly, whenever the progress changes.  
 \*   
 \* In short:   
 \* `enter` events will always trigger \*\*before\*\* the progress update and `leave` envents will trigger \*\*after\*\* the progress update.   
 \* `start` and `end` will always trigger at their respective position.  
 \*   
 \* Please review the event descriptions for details on the events and the event object that is passed to the callback.  
 \*   
 \* @method ScrollMagic.Scene#progress  
 \* @example  
 \* // get the current scene progress  
 \* var progress = scene.progress();  
 \*  
 \* // set new scene progress  
 \* scene.progress(0.3);  
 \*  
 \* @fires {@link Scene.enter}, when used as setter  
 \* @fires {@link Scene.start}, when used as setter  
 \* @fires {@link Scene.progress}, when used as setter  
 \* @fires {@link Scene.end}, when used as setter  
 \* @fires {@link Scene.leave}, when used as setter  
 \*  
 \* @param {number} [progress] - The new progress value of the scene `[0-1]`.  
 \* @returns {number} `get` - Current scene progress.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
this.progress = function (progress) {  
 if (!arguments.length) { // get  
 return \_progress;  
 } else { // set  
 var  
 doUpdate = false,  
 oldState = \_state,  
 scrollDirection = \_controller ? \_controller.info("scrollDirection") : 'PAUSED',  
 reverseOrForward = \_options.reverse || progress >= \_progress;  
 if (\_options.duration === 0) {  
 // zero duration scenes  
 doUpdate = \_progress != progress;  
 \_progress = progress < 1 && reverseOrForward ? 0 : 1;  
 \_state = \_progress === 0 ? SCENE\_STATE\_BEFORE : SCENE\_STATE\_DURING;  
 } else {  
 // scenes with start and end  
 if (progress < 0 && \_state !== SCENE\_STATE\_BEFORE && reverseOrForward) {  
 // go back to initial state  
 \_progress = 0;  
 \_state = SCENE\_STATE\_BEFORE;  
 doUpdate = true;  
 } else if (progress >= 0 && progress < 1 && reverseOrForward) {  
 \_progress = progress;  
 \_state = SCENE\_STATE\_DURING;  
 doUpdate = true;  
 } else if (progress >= 1 && \_state !== SCENE\_STATE\_AFTER) {  
 \_progress = 1;  
 \_state = SCENE\_STATE\_AFTER;  
 doUpdate = true;  
 } else if (\_state === SCENE\_STATE\_DURING && !reverseOrForward) {  
 updatePinState(); // in case we scrolled backwards mid-scene and reverse is disabled => update the pin position, so it doesn't move back as well.  
 }  
 }  
 if (doUpdate) {  
 // fire events  
 var  
 eventVars = {progress: \_progress, state: \_state, scrollDirection: scrollDirection},  
 stateChanged = \_state != oldState;  
  
 var trigger = function (eventName) { // tmp helper to simplify code  
 Scene.trigger(eventName, eventVars);  
 };  
  
 if (stateChanged) { // enter events  
 if (oldState !== SCENE\_STATE\_DURING) {  
 trigger("enter");  
 trigger(oldState === SCENE\_STATE\_BEFORE ? "start" : "end");  
 }  
 }  
 trigger("progress");  
 if (stateChanged) { // leave events  
 if (\_state !== SCENE\_STATE\_DURING) {  
 trigger(\_state === SCENE\_STATE\_BEFORE ? "start" : "end");  
 trigger("leave");  
 }  
 }  
 }  
  
 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).