[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/getters-setters.js

var \_validate = \_util.extend(SCENE\_OPTIONS.validate, {  
 // validation for duration handled internally for reference to private var \_durationMethod  
 duration : function (val) {  
 if (\_util.type.String(val) && val.match(/^(\.|\d)\*\d+%$/)) {  
 // percentage value  
 var perc = parseFloat(val) / 100;  
 val = function () {  
 return \_controller ? \_controller.info("size") \* perc : 0;  
 };  
 }  
 if (\_util.type.Function(val)) {  
 // function  
 \_durationUpdateMethod = val;  
 try {  
 val = parseFloat(\_durationUpdateMethod.call(Scene));  
 } catch (e) {  
 val = -1; // will cause error below  
 }  
 }  
 // val has to be float  
 val = parseFloat(val);  
 if (!\_util.type.Number(val) || val < 0) {  
 if (\_durationUpdateMethod) {  
 \_durationUpdateMethod = undefined;  
 throw ["Invalid return value of supplied function for option \"duration\":", val];  
 } else {  
 throw ["Invalid value for option \"duration\":", val];  
 }  
 }  
 return val;  
 }  
});  
  
/\*\*  
 \* Checks the validity of a specific or all options and reset to default if neccessary.  
 \* @private  
 \*/  
var validateOption = function (check) {  
 check = arguments.length ? [check] : Object.keys(\_validate);  
 check.forEach(function (optionName, key) {  
 var value;  
 if (\_validate[optionName]) { // there is a validation method for this option  
 try { // validate value  
 value = \_validate[optionName](\_options[optionName]);  
 } catch (e) { // validation failed -> reset to default  
 value = DEFAULT\_OPTIONS[optionName];  
 // (BUILD) - REMOVE IN MINIFY - START  
 var logMSG = \_util.type.String(e) ? [e] : e;  
 if (\_util.type.Array(logMSG)) {  
 logMSG[0] = "ERROR: " + logMSG[0];  
 logMSG.unshift(1); // loglevel 1 for error msg  
 log.apply(this, logMSG);  
 } else {  
 log(1, "ERROR: Problem executing validation callback for option '" + optionName + "':", e.message);  
 }  
 // (BUILD) - REMOVE IN MINIFY - END  
 } finally {  
 \_options[optionName] = value;  
 }  
 }  
 });  
};  
  
/\*\*  
 \* Helper used by the setter/getters for scene options  
 \* @private  
 \*/  
var changeOption = function(varname, newval) {  
 var  
 changed = false,  
 oldval = \_options[varname];  
 if (\_options[varname] != newval) {  
 \_options[varname] = newval;  
 validateOption(varname); // resets to default if necessary  
 changed = oldval != \_options[varname];  
 }  
 return changed;  
};  
  
// generate getters/setters for all options  
var addSceneOption = function (optionName) {  
 if (!Scene[optionName]) {  
 Scene[optionName] = function (newVal) {  
 if (!arguments.length) { // get  
 return \_options[optionName];  
 } else {  
 if (optionName === "duration") { // new duration is set, so any previously set function must be unset  
 \_durationUpdateMethod = undefined;  
 }  
 if (changeOption(optionName, newVal)) { // set  
 Scene.trigger("change", {what: optionName, newval: \_options[optionName]});  
 if (SCENE\_OPTIONS.shifts.indexOf(optionName) > -1) {  
 Scene.trigger("shift", {reason: optionName});  
 }  
 }  
 }  
 return Scene;  
 };  
 }  
};  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the duration option value.  
 \*  
 \* As a \*\*setter\*\* it accepts three types of parameters:  
 \* 1. `number`: Sets the duration of the scene to exactly this amount of pixels.   
 \* This means the scene will last for exactly this amount of pixels scrolled. Sub-Pixels are also valid.  
 \* A value of `0` means that the scene is 'open end' and no end will be triggered. Pins will never unpin and animations will play independently of scroll progress.  
 \* 2. `string`: Always updates the duration relative to parent scroll container.   
 \* For example `"100%"` will keep the duration always exactly at the inner height of the scroll container.  
 \* When scrolling vertically the width is used for reference respectively.  
 \* 3. `function`: The supplied function will be called to return the scene duration.  
 \* This is useful in setups where the duration depends on other elements who might change size. By supplying a function you can return a value instead of updating potentially multiple scene durations.   
 \* The scene can be referenced inside the callback using `this`.  
 \* \_\*\*WARNING:\*\* This is an easy way to kill performance, as the callback will be executed every time `Scene.refresh()` is called, which happens a lot. The interval is defined by the controller (see ScrollMagic.Controller option `refreshInterval`).   
 \* It's recomended to avoid calculations within the function and use cached variables as return values.   
 \* This counts double if you use the same function for multiple scenes.\_  
 \*  
 \* @method ScrollMagic.Scene#duration  
 \* @example  
 \* // get the current duration value  
 \* var duration = scene.duration();  
 \*  
 \* // set a new duration  
 \* scene.duration(300);  
 \*  
 \* // set duration responsively to container size  
 \* scene.duration("100%");  
 \*  
 \* // use a function to randomize the duration for some reason.  
 \* var durationValueCache;  
 \* function durationCallback () {  
 \* return durationValueCache;  
 \* }  
 \* function updateDuration () {  
 \* durationValueCache = Math.random() \* 100;  
 \* }  
 \* updateDuration(); // set to initial value  
 \* scene.duration(durationCallback); // set duration callback  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @fires {@link Scene.shift}, when used as setter  
 \* @param {(number|string|function)} [newDuration] - The new duration setting for the scene.  
 \* @returns {number} `get` - Current scene duration.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the offset option value.  
 \* @method ScrollMagic.Scene#offset  
 \* @example  
 \* // get the current offset  
 \* var offset = scene.offset();  
 \*  
 \* // set a new offset  
 \* scene.offset(100);  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @fires {@link Scene.shift}, when used as setter  
 \* @param {number} [newOffset] - The new offset of the scene.  
 \* @returns {number} `get` - Current scene offset.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the triggerElement option value.  
 \* Does \*\*not\*\* fire `Scene.shift`, because changing the trigger Element doesn't necessarily mean the start position changes. This will be determined in `Scene.refresh()`, which is automatically triggered.  
 \* @method ScrollMagic.Scene#triggerElement  
 \* @example  
 \* // get the current triggerElement  
 \* var triggerElement = scene.triggerElement();  
 \*  
 \* // set a new triggerElement using a selector  
 \* scene.triggerElement("#trigger");  
 \* // set a new triggerElement using a DOM object  
 \* scene.triggerElement(document.getElementById("trigger"));  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @param {(string|object)} [newTriggerElement] - The new trigger element for the scene.  
 \* @returns {(string|object)} `get` - Current triggerElement.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the triggerHook option value.  
 \* @method ScrollMagic.Scene#triggerHook  
 \* @example  
 \* // get the current triggerHook value  
 \* var triggerHook = scene.triggerHook();  
 \*  
 \* // set a new triggerHook using a string  
 \* scene.triggerHook("onLeave");  
 \* // set a new triggerHook using a number  
 \* scene.triggerHook(0.7);  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @fires {@link Scene.shift}, when used as setter  
 \* @param {(number|string)} [newTriggerHook] - The new triggerHook of the scene. See {@link Scene} parameter description for value options.  
 \* @returns {number} `get` - Current triggerHook (ALWAYS numerical).  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the reverse option value.  
 \* @method ScrollMagic.Scene#reverse  
 \* @example  
 \* // get the current reverse option  
 \* var reverse = scene.reverse();  
 \*  
 \* // set new reverse option  
 \* scene.reverse(false);  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @param {boolean} [newReverse] - The new reverse setting of the scene.  
 \* @returns {boolean} `get` - Current reverse option value.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* or \*\*Set\*\* the loglevel option value.  
 \* @method ScrollMagic.Scene#loglevel  
 \* @example  
 \* // get the current loglevel  
 \* var loglevel = scene.loglevel();  
 \*  
 \* // set new loglevel  
 \* scene.loglevel(3);  
 \*  
 \* @fires {@link Scene.change}, when used as setter  
 \* @param {number} [newLoglevel] - The new loglevel setting of the scene. `[0-3]`  
 \* @returns {number} `get` - Current loglevel.  
 \* @returns {Scene} `set` - Parent object for chaining.  
 \*/  
  
/\*\*  
 \* \*\*Get\*\* the associated controller.  
 \* @method ScrollMagic.Scene#controller  
 \* @example  
 \* // get the controller of a scene  
 \* var controller = scene.controller();  
 \*  
 \* @returns {ScrollMagic.Controller} Parent controller or `undefined`  
 \*/  
this.controller = function () {  
 return \_controller;  
};  
  
/\*\*  
 \* \*\*Get\*\* the current state.  
 \* @method ScrollMagic.Scene#state  
 \* @example  
 \* // get the current state  
 \* var state = scene.state();  
 \*  
 \* @returns {string} `"BEFORE"`, `"DURING"` or `"AFTER"`  
 \*/  
this.state = function () {  
 return \_state;  
};  
  
/\*\*  
 \* \*\*Get\*\* the current scroll offset for the start of the scene.   
 \* Mind, that the scrollOffset is related to the size of the container, if `triggerHook` is bigger than `0` (or `"onLeave"`).   
 \* This means, that resizing the container or changing the `triggerHook` will influence the scene's start offset.  
 \* @method ScrollMagic.Scene#scrollOffset  
 \* @example  
 \* // get the current scroll offset for the start and end of the scene.  
 \* var start = scene.scrollOffset();  
 \* var end = scene.scrollOffset() + scene.duration();  
 \* console.log("the scene starts at", start, "and ends at", end);  
 \*  
 \* @returns {number} The scroll offset (of the container) at which the scene will trigger. Y value for vertical and X value for horizontal scrolls.  
 \*/  
this.scrollOffset = function () {  
 return \_scrollOffset.start;  
};  
  
/\*\*  
 \* \*\*Get\*\* the trigger position of the scene (including the value of the `offset` option).   
 \* @method ScrollMagic.Scene#triggerPosition  
 \* @example  
 \* // get the scene's trigger position  
 \* var triggerPosition = scene.triggerPosition();  
 \*  
 \* @returns {number} Start position of the scene. Top position value for vertical and left position value for horizontal scrolls.  
 \*/  
this.triggerPosition = function () {  
 var pos = \_options.offset; // the offset is the basis  
 if (\_controller) {  
 // get the trigger position  
 if (\_options.triggerElement) {  
 // Element as trigger  
 pos += \_triggerPos;  
 } else {  
 // return the height of the triggerHook to start at the beginning  
 pos += \_controller.info("size") \* Scene.triggerHook();  
 }  
 }  
 return pos;  
};

© 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).