[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/feature-pinning.js

var  
 \_pin,  
 \_pinOptions;  
  
Scene  
 .on("shift.internal", function (e) {  
 var durationChanged = e.reason === "duration";  
 if ((\_state === SCENE\_STATE\_AFTER && durationChanged) || (\_state === SCENE\_STATE\_DURING && \_options.duration === 0)) {  
 // if [duration changed after a scene (inside scene progress updates pin position)] or [duration is 0, we are in pin phase and some other value changed].  
 updatePinState();  
 }  
 if (durationChanged) {  
 updatePinDimensions();  
 }  
 })  
 .on("progress.internal", function (e) {  
 updatePinState();  
 })  
 .on("add.internal", function (e) {  
 updatePinDimensions();  
 })  
 .on("destroy.internal", function (e) {  
 Scene.removePin(e.reset);  
 });  
/\*\*  
 \* Update the pin state.  
 \* @private  
 \*/  
var updatePinState = function (forceUnpin) {  
 if (\_pin && \_controller) {  
 var   
 containerInfo = \_controller.info(),  
 pinTarget = \_pinOptions.spacer.firstChild; // may be pin element or another spacer, if cascading pins  
  
 if (!forceUnpin && \_state === SCENE\_STATE\_DURING) { // during scene or if duration is 0 and we are past the trigger  
 // pinned state  
 if (\_util.css(pinTarget, "position") != "fixed") {  
 // change state before updating pin spacer (position changes due to fixed collapsing might occur.)  
 \_util.css(pinTarget, {"position": "fixed"});  
 // update pin spacer  
 updatePinDimensions();  
 }  
  
 var  
 fixedPos = \_util.get.offset(\_pinOptions.spacer, true), // get viewport position of spacer  
 scrollDistance = \_options.reverse || \_options.duration === 0 ?  
 containerInfo.scrollPos - \_scrollOffset.start // quicker  
 : Math.round(\_progress \* \_options.duration \* 10)/10; // if no reverse and during pin the position needs to be recalculated using the progress  
   
 // add scrollDistance  
 fixedPos[containerInfo.vertical ? "top" : "left"] += scrollDistance;  
  
 // set new values  
 \_util.css(\_pinOptions.spacer.firstChild, {  
 top: fixedPos.top,  
 left: fixedPos.left  
 });  
 } else {  
 // unpinned state  
 var  
 newCSS = {  
 position: \_pinOptions.inFlow ? "relative" : "absolute",  
 top: 0,  
 left: 0  
 },  
 change = \_util.css(pinTarget, "position") != newCSS.position;  
   
 if (!\_pinOptions.pushFollowers) {  
 newCSS[containerInfo.vertical ? "top" : "left"] = \_options.duration \* \_progress;  
 } else if (\_options.duration > 0) { // only concerns scenes with duration  
 if (\_state === SCENE\_STATE\_AFTER && parseFloat(\_util.css(\_pinOptions.spacer, "padding-top")) === 0) {  
 change = true; // if in after state but havent updated spacer yet (jumped past pin)  
 } else if (\_state === SCENE\_STATE\_BEFORE && parseFloat(\_util.css(\_pinOptions.spacer, "padding-bottom")) === 0) { // before  
 change = true; // jumped past fixed state upward direction  
 }  
 }  
 // set new values  
 \_util.css(pinTarget, newCSS);  
 if (change) {  
 // update pin spacer if state changed  
 updatePinDimensions();  
 }  
 }  
 }  
};  
  
/\*\*  
 \* Update the pin spacer and/or element size.  
 \* The size of the spacer needs to be updated whenever the duration of the scene changes, if it is to push down following elements.  
 \* @private  
 \*/  
var updatePinDimensions = function () {  
 if (\_pin && \_controller && \_pinOptions.inFlow) { // no spacerresize, if original position is absolute  
 var  
 after = (\_state === SCENE\_STATE\_AFTER),  
 before = (\_state === SCENE\_STATE\_BEFORE),  
 during = (\_state === SCENE\_STATE\_DURING),  
 vertical = \_controller.info("vertical"),  
 pinTarget = \_pinOptions.spacer.firstChild, // usually the pined element but can also be another spacer (cascaded pins)  
 marginCollapse = \_util.isMarginCollapseType(\_util.css(\_pinOptions.spacer, "display")),  
 css = {};  
  
 // set new size  
 // if relsize: spacer -> pin | else: pin -> spacer  
 if (\_pinOptions.relSize.width || \_pinOptions.relSize.autoFullWidth) {  
 if (during) {  
 \_util.css(\_pin, {"width": \_util.get.width(\_pinOptions.spacer)});  
 } else {  
 \_util.css(\_pin, {"width": "100%"});  
 }  
 } else {  
 // minwidth is needed for cascaded pins.  
 css["min-width"] = \_util.get.width(vertical ? \_pin : pinTarget, true, true);  
 css.width = during ? css["min-width"] : "auto";  
 }  
 if (\_pinOptions.relSize.height) {  
 if (during) {  
 // the only padding the spacer should ever include is the duration (if pushFollowers = true), so we need to substract that.  
 \_util.css(\_pin, {"height": \_util.get.height(\_pinOptions.spacer) - (\_pinOptions.pushFollowers ? \_options.duration : 0)});  
 } else {  
 \_util.css(\_pin, {"height": "100%"});  
 }  
 } else {  
 // margin is only included if it's a cascaded pin to resolve an IE9 bug  
 css["min-height"] = \_util.get.height(vertical ? pinTarget : \_pin, true , !marginCollapse); // needed for cascading pins  
 css.height = during ? css["min-height"] : "auto";  
 }  
  
 // add space for duration if pushFollowers is true  
 if (\_pinOptions.pushFollowers) {  
 css["padding" + (vertical ? "Top" : "Left")] = \_options.duration \* \_progress;  
 css["padding" + (vertical ? "Bottom" : "Right")] = \_options.duration \* (1 - \_progress);  
 }  
 \_util.css(\_pinOptions.spacer, css);  
 }  
};  
  
/\*\*  
 \* Updates the Pin state (in certain scenarios)  
 \* If the controller container is not the document and we are mid-pin-phase scrolling or resizing the main document can result to wrong pin positions.  
 \* So this function is called on resize and scroll of the document.  
 \* @private  
 \*/  
var updatePinInContainer = function () {  
 if (\_controller && \_pin && \_state === SCENE\_STATE\_DURING && !\_controller.info("isDocument")) {  
 updatePinState();  
 }  
};  
  
/\*\*  
 \* Updates the Pin spacer size state (in certain scenarios)  
 \* If container is resized during pin and relatively sized the size of the pin might need to be updated...  
 \* So this function is called on resize of the container.  
 \* @private  
 \*/  
var updateRelativePinSpacer = function () {  
 if ( \_controller && \_pin && // well, duh  
 \_state === SCENE\_STATE\_DURING && // element in pinned state?  
 ( // is width or height relatively sized, but not in relation to body? then we need to recalc.  
 ((\_pinOptions.relSize.width || \_pinOptions.relSize.autoFullWidth) && \_util.get.width(window) != \_util.get.width(\_pinOptions.spacer.parentNode)) ||  
 (\_pinOptions.relSize.height && \_util.get.height(window) != \_util.get.height(\_pinOptions.spacer.parentNode))  
 )  
 ) {  
 updatePinDimensions();  
 }  
};  
  
/\*\*  
 \* Is called, when the mousewhel is used while over a pinned element inside a div container.  
 \* If the scene is in fixed state scroll events would be counted towards the body. This forwards the event to the scroll container.  
 \* @private  
 \*/  
var onMousewheelOverPin = function (e) {  
 if (\_controller && \_pin && \_state === SCENE\_STATE\_DURING && !\_controller.info("isDocument")) { // in pin state  
 e.preventDefault();  
 \_controller.\_setScrollPos(\_controller.info("scrollPos") - ((e.wheelDelta || e[\_controller.info("vertical") ? "wheelDeltaY" : "wheelDeltaX"])/3 || -e.detail\*30));  
 }  
};  
  
/\*\*  
 \* Pin an element for the duration of the scene.  
 \* If the scene duration is 0 the element will only be unpinned, if the user scrolls back past the start position.   
 \* Make sure only one pin is applied to an element at the same time.  
 \* An element can be pinned multiple times, but only successively.  
 \* \_\*\*NOTE:\*\* The option `pushFollowers` has no effect, when the scene duration is 0.\_  
 \* @method ScrollMagic.Scene#setPin  
 \* @example  
 \* // pin element and push all following elements down by the amount of the pin duration.  
 \* scene.setPin("#pin");  
 \*  
 \* // pin element and keeping all following elements in their place. The pinned element will move past them.  
 \* scene.setPin("#pin", {pushFollowers: false});  
 \*  
 \* @param {(string|object)} element - A Selector targeting an element or a DOM object that is supposed to be pinned.  
 \* @param {object} [settings] - settings for the pin  
 \* @param {boolean} [settings.pushFollowers=true] - If `true` following elements will be "pushed" down for the duration of the pin, if `false` the pinned element will just scroll past them.   
 Ignored, when duration is `0`.  
 \* @param {string} [settings.spacerClass="scrollmagic-pin-spacer"] - Classname of the pin spacer element, which is used to replace the element.  
 \*  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.setPin = function (element, settings) {  
 var  
 defaultSettings = {  
 pushFollowers: true,  
 spacerClass: "scrollmagic-pin-spacer"  
 };  
 // (BUILD) - REMOVE IN MINIFY - START  
 var pushFollowersActivelySet = settings && settings.hasOwnProperty('pushFollowers');  
 // (BUILD) - REMOVE IN MINIFY - END  
 settings = \_util.extend({}, defaultSettings, settings);  
  
 // validate Element  
 element = \_util.get.elements(element)[0];  
 if (!element) {  
 log(1, "ERROR calling method 'setPin()': Invalid pin element supplied.");  
 return Scene; // cancel  
 } else if (\_util.css(element, "position") === "fixed") {  
 log(1, "ERROR calling method 'setPin()': Pin does not work with elements that are positioned 'fixed'.");  
 return Scene; // cancel  
 }  
  
 if (\_pin) { // preexisting pin?  
 if (\_pin === element) {  
 // same pin we already have -> do nothing  
 return Scene; // cancel  
 } else {  
 // kill old pin  
 Scene.removePin();  
 }  
   
 }  
 \_pin = element;  
   
 var  
 parentDisplay = \_pin.parentNode.style.display,  
 boundsParams = ["top", "left", "bottom", "right", "margin", "marginLeft", "marginRight", "marginTop", "marginBottom"];  
  
 \_pin.parentNode.style.display = 'none'; // hack start to force css to return stylesheet values instead of calculated px values.  
 var  
 inFlow = \_util.css(\_pin, "position") != "absolute",  
 pinCSS = \_util.css(\_pin, boundsParams.concat(["display"])),  
 sizeCSS = \_util.css(\_pin, ["width", "height"]);  
 \_pin.parentNode.style.display = parentDisplay; // hack end.  
  
 if (!inFlow && settings.pushFollowers) {  
 log(2, "WARNING: If the pinned element is positioned absolutely pushFollowers will be disabled.");  
 settings.pushFollowers = false;  
 }  
 // (BUILD) - REMOVE IN MINIFY - START  
 window.setTimeout(function () { // wait until all finished, because with responsive duration it will only be set after scene is added to controller  
 if (\_pin && \_options.duration === 0 && pushFollowersActivelySet && settings.pushFollowers) {  
 log(2, "WARNING: pushFollowers =", true, "has no effect, when scene duration is 0.");  
 }  
 }, 0);  
 // (BUILD) - REMOVE IN MINIFY - END  
  
 // create spacer and insert  
 var  
 spacer = \_pin.parentNode.insertBefore(document.createElement('div'), \_pin),  
 spacerCSS = \_util.extend(pinCSS, {  
 position: inFlow ? "relative" : "absolute",  
 boxSizing: "content-box",  
 mozBoxSizing: "content-box",  
 webkitBoxSizing: "content-box"  
 });  
  
 if (!inFlow) { // copy size if positioned absolutely, to work for bottom/right positioned elements.  
 \_util.extend(spacerCSS, \_util.css(\_pin, ["width", "height"]));  
 }  
  
 \_util.css(spacer, spacerCSS);  
 spacer.setAttribute(PIN\_SPACER\_ATTRIBUTE, "");  
 \_util.addClass(spacer, settings.spacerClass);  
  
 // set the pin Options  
 \_pinOptions = {  
 spacer: spacer,  
 relSize: { // save if size is defined using % values. if so, handle spacer resize differently...  
 width: sizeCSS.width.slice(-1) === "%",  
 height: sizeCSS.height.slice(-1) === "%",  
 autoFullWidth: sizeCSS.width === "auto" && inFlow && \_util.isMarginCollapseType(pinCSS.display)  
 },  
 pushFollowers: settings.pushFollowers,  
 inFlow: inFlow, // stores if the element takes up space in the document flow  
 };  
   
 if (!\_pin.\_\_\_origStyle) {  
 \_pin.\_\_\_origStyle = {};  
 var  
 pinInlineCSS = \_pin.style,  
 copyStyles = boundsParams.concat(["width", "height", "position", "boxSizing", "mozBoxSizing", "webkitBoxSizing"]);  
 copyStyles.forEach(function (val) {  
 \_pin.\_\_\_origStyle[val] = pinInlineCSS[val] || "";  
 });  
 }  
  
 // if relative size, transfer it to spacer and make pin calculate it...  
 if (\_pinOptions.relSize.width) {  
 \_util.css(spacer, {width: sizeCSS.width});  
 }  
 if (\_pinOptions.relSize.height) {  
 \_util.css(spacer, {height: sizeCSS.height});  
 }  
  
 // now place the pin element inside the spacer   
 spacer.appendChild(\_pin);  
 // and set new css  
 \_util.css(\_pin, {  
 position: inFlow ? "relative" : "absolute",  
 margin: "auto",  
 top: "auto",  
 left: "auto",  
 bottom: "auto",  
 right: "auto"  
 });  
   
 if (\_pinOptions.relSize.width || \_pinOptions.relSize.autoFullWidth) {  
 \_util.css(\_pin, {  
 boxSizing : "border-box",  
 mozBoxSizing : "border-box",  
 webkitBoxSizing : "border-box"  
 });  
 }  
  
 // add listener to document to update pin position in case controller is not the document.  
 window.addEventListener('scroll', updatePinInContainer);  
 window.addEventListener('resize', updatePinInContainer);  
 window.addEventListener('resize', updateRelativePinSpacer);  
 // add mousewheel listener to catch scrolls over fixed elements  
 \_pin.addEventListener("mousewheel", onMousewheelOverPin);  
 \_pin.addEventListener("DOMMouseScroll", onMousewheelOverPin);  
  
 log(3, "added pin");  
  
 // finally update the pin to init  
 updatePinState();  
  
 return Scene;  
};  
  
/\*\*  
 \* Remove the pin from the scene.  
 \* @method ScrollMagic.Scene#removePin  
 \* @example  
 \* // remove the pin from the scene without resetting it (the spacer is not removed)  
 \* scene.removePin();  
 \*  
 \* // remove the pin from the scene and reset the pin element to its initial position (spacer is removed)  
 \* scene.removePin(true);  
 \*  
 \* @param {boolean} [reset=false] - If `false` the spacer will not be removed and the element's position will not be reset.  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
this.removePin = function (reset) {  
 if (\_pin) {  
 if (\_state === SCENE\_STATE\_DURING) {  
 updatePinState(true); // force unpin at position  
 }  
 if (reset || !\_controller) { // if there's no controller no progress was made anyway...  
 var pinTarget = \_pinOptions.spacer.firstChild; // usually the pin element, but may be another spacer (cascaded pins)...  
 if (pinTarget.hasAttribute(PIN\_SPACER\_ATTRIBUTE)) { // copy margins to child spacer  
 var  
 style = \_pinOptions.spacer.style,  
 values = ["margin", "marginLeft", "marginRight", "marginTop", "marginBottom"],  
 margins = {};  
 values.forEach(function (val) {  
 margins[val] = style[val] || "";  
 });  
 \_util.css(pinTarget, margins);  
 }  
 \_pinOptions.spacer.parentNode.insertBefore(pinTarget, \_pinOptions.spacer);  
 \_pinOptions.spacer.parentNode.removeChild(\_pinOptions.spacer);  
 if (!\_pin.parentNode.hasAttribute(PIN\_SPACER\_ATTRIBUTE)) { // if it's the last pin for this element -> restore inline styles  
 // TODO: only correctly set for first pin (when cascading) - how to fix?  
 \_util.css(\_pin, \_pin.\_\_\_origStyle);  
 delete \_pin.\_\_\_origStyle;  
 }  
 }  
 window.removeEventListener('scroll', updatePinInContainer);  
 window.removeEventListener('resize', updatePinInContainer);  
 window.removeEventListener('resize', updateRelativePinSpacer);  
 \_pin.removeEventListener("mousewheel", onMousewheelOverPin);  
 \_pin.removeEventListener("DOMMouseScroll", onMousewheelOverPin);  
 \_pin = undefined;  
 log(3, "removed pin (reset: " + (reset ? "true" : "false") + ")");  
 }  
 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).