[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: plugins/debug.addIndicators.js

/\*!  
 \* @file Debug Extension for ScrollMagic.  
 \*/  
/\*\*  
 \* This plugin was formerly known as the ScrollMagic debug extension.  
 \*  
 \* It enables you to add visual indicators to your page, to be able to see exactly when a scene is triggered.  
 \*  
 \* To have access to this extension, please include `plugins/debug.addIndicators.js`.  
 \* @mixin debug.addIndicators  
 \*/  
(function (root, factory) {  
 if (typeof define === 'function' && define.amd) {  
 // AMD. Register as an anonymous module.  
 define(['ScrollMagic'], factory);  
 } else if (typeof exports === 'object') {  
 // CommonJS  
 factory(require('scrollmagic'));  
 } else {  
 // no browser global export needed, just execute  
 factory(root.ScrollMagic || (root.jQuery && root.jQuery.ScrollMagic));  
 }  
}(this, function(ScrollMagic) {  
 "use strict";  
 var NAMESPACE = "debug.addIndicators";  
  
 // (BUILD) - REMOVE IN MINIFY - START  
 var  
 console = window.console || {},  
 err = Function.prototype.bind.call(console.error || console.log || function() {}, console);  
 if (!ScrollMagic) {  
 err("(" + NAMESPACE + ") -> ERROR: The ScrollMagic main module could not be found. Please make sure it's loaded before this plugin or use an asynchronous loader like requirejs.");  
 }  
 // (BUILD) - REMOVE IN MINIFY - END  
  
 // plugin settings  
 var  
 FONT\_SIZE = "0.85em",  
 ZINDEX = "9999",  
 EDGE\_OFFSET = 15; // minimum edge distance, added to indentation  
  
 // overall vars  
 var  
 \_util = ScrollMagic.\_util,  
 \_autoindex = 0;  
  
  
  
 ScrollMagic.Scene.extend(function () {  
 var  
 Scene = this,  
 \_indicator;  
  
 // (BUILD) - REMOVE IN MINIFY - START  
 var log = function () {  
 if (Scene.\_log) { // not available, when main source minified  
 Array.prototype.splice.call(arguments, 1, 0, "(" + NAMESPACE + ")", "->");  
 Scene.\_log.apply(this, arguments);  
 }  
 };  
 // (BUILD) - REMOVE IN MINIFY - END  
  
 /\*\*  
 \* Add visual indicators for a ScrollMagic.Scene.   
 \* @memberof! debug.addIndicators#  
 \*  
 \* @example  
 \* // add basic indicators  
 \* scene.addIndicators()  
 \*  
 \* // passing options  
 \* scene.addIndicators({name: "pin scene", colorEnd: "#FFFFFF"});  
 \*  
 \* @param {object} [options] - An object containing one or more options for the indicators.  
 \* @param {(string|object)} [options.parent] - A selector, DOM Object or a jQuery object that the indicators should be added to.   
 If undefined, the controller's container will be used.  
 \* @param {number} [options.name=""] - This string will be displayed at the start and end indicators of the scene for identification purposes. If no name is supplied an automatic index will be used.  
 \* @param {number} [options.indent=0] - Additional position offset for the indicators (useful, when having multiple scenes starting at the same position).  
 \* @param {string} [options.colorStart=green] - CSS color definition for the start indicator.  
 \* @param {string} [options.colorEnd=red] - CSS color definition for the end indicator.  
 \* @param {string} [options.colorTrigger=blue] - CSS color definition for the trigger indicator.  
 \*/  
 Scene.addIndicators = function(options) {  
 if (!\_indicator) {  
 var  
 DEFAULT\_OPTIONS = {  
 name: "",  
 indent: 0,  
 parent: undefined,  
 colorStart: "green",  
 colorEnd: "red",  
 colorTrigger: "blue",  
 };  
   
 options = \_util.extend({}, DEFAULT\_OPTIONS, options);  
  
 \_autoindex++;  
 \_indicator = new Indicator(Scene, options);  
  
 Scene.on("add.plugin\_addIndicators", \_indicator.add);  
 Scene.on("remove.plugin\_addIndicators", \_indicator.remove);  
 Scene.on("destroy.plugin\_addIndicators", Scene.removeIndicators);  
  
 // it the scene already has a controller we can start right away.  
 if (Scene.controller()) {  
 \_indicator.add();  
 }  
 }  
 return Scene;  
 };  
  
 /\*\*  
 \* Removes visual indicators from a ScrollMagic.Scene.  
 \* @memberof! debug.addIndicators#  
 \*  
 \* @example  
 \* // remove previously added indicators  
 \* scene.removeIndicators()  
 \*  
 \*/  
 Scene.removeIndicators = function() {  
 if (\_indicator) {  
 \_indicator.remove();  
 this.off("\*.plugin\_addIndicators");  
 \_indicator = undefined;  
 }  
 return Scene;  
 };  
  
 });  
  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* Extension for controller to store and update related indicators  
 \* ----------------------------------------------------------------  
 \*/  
 // add option to globally auto-add indicators to scenes  
 /\*\*  
 \* Every ScrollMagic.Controller instance now accepts an additional option.   
 \* See {@link ScrollMagic.Controller} for a complete list of the standard options.  
 \* @memberof! debug.addIndicators#  
 \* @method new ScrollMagic.Controller(options)  
 \* @example  
 \* // make a controller and add indicators to all scenes attached  
 \* var controller = new ScrollMagic.Controller({addIndicators: true});  
 \* // this scene will automatically have indicators added to it  
 \* new ScrollMagic.Scene()  
 \* .addTo(controller);  
 \*  
 \* @param {object} [options] - Options for the Controller.  
 \* @param {boolean} [options.addIndicators=false] - If set to `true` every scene that is added to the controller will automatically get indicators added to it.  
 \*/  
 ScrollMagic.Controller.addOption("addIndicators", false);  
 // extend Controller  
 ScrollMagic.Controller.extend(function () {  
 var  
 Controller = this,  
 \_info = Controller.info(),  
 \_container = \_info.container,  
 \_isDocument = \_info.isDocument,  
 \_vertical = \_info.vertical,  
 \_indicators = { // container for all indicators and methods  
 groups: []  
 };  
  
 // (BUILD) - REMOVE IN MINIFY - START  
 var log = function () {  
 if (Controller.\_log) { // not available, when main source minified  
 Array.prototype.splice.call(arguments, 1, 0, "(" + NAMESPACE + ")", "->");  
 Controller.\_log.apply(this, arguments);  
 }  
 };  
 if (Controller.\_indicators) {  
 log(2, "WARNING: Scene already has a property '\_indicators', which will be overwritten by plugin.");  
 }  
 // (BUILD) - REMOVE IN MINIFY - END  
   
 // add indicators container  
 this.\_indicators = \_indicators;  
 /\*  
 needed updates:  
 +++++++++++++++  
 start/end position on scene shift (handled in Indicator class)  
 trigger parameters on triggerHook value change (handled in Indicator class)  
 bounds position on container scroll or resize (to keep alignment to bottom/right)  
 trigger position on container resize, window resize (if container isn't document) and window scroll (if container isn't document)  
 \*/  
   
 // event handler for when associated bounds markers need to be repositioned  
 var handleBoundsPositionChange = function () {  
 \_indicators.updateBoundsPositions();  
 };  
  
 // event handler for when associated trigger groups need to be repositioned  
 var handleTriggerPositionChange = function () {  
 \_indicators.updateTriggerGroupPositions();  
 };  
  
 \_container.addEventListener("resize", handleTriggerPositionChange);  
 if (!\_isDocument) {  
 window.addEventListener("resize", handleTriggerPositionChange);  
 window.addEventListener("scroll", handleTriggerPositionChange);  
 }  
 // update all related bounds containers  
 \_container.addEventListener("resize", handleBoundsPositionChange);  
 \_container.addEventListener("scroll", handleBoundsPositionChange);  
  
  
 // updates the position of the bounds container to aligned to the right for vertical containers and to the bottom for horizontal  
 this.\_indicators.updateBoundsPositions = function (specificIndicator) {  
 var // constant for all bounds  
 groups = specificIndicator ?  
 [\_util.extend({}, specificIndicator.triggerGroup, {members: [specificIndicator]})]: // create a group with only one element  
 \_indicators.groups, // use all  
 g = groups.length,  
 css = {},  
 paramPos = \_vertical ? "left" : "top",  
 paramDimension = \_vertical ? "width" : "height",  
 edge = \_vertical ?  
 \_util.get.scrollLeft(\_container) + \_util.get.width(\_container) - EDGE\_OFFSET:  
 \_util.get.scrollTop(\_container) + \_util.get.height(\_container) - EDGE\_OFFSET,  
 b, triggerSize, group;  
 while (g--) { // group loop  
 group = groups[g];  
 b = group.members.length;  
 triggerSize = \_util.get[paramDimension](group.element.firstChild);  
 while (b--) { // indicators loop  
 css[paramPos] = edge - triggerSize;  
 \_util.css(group.members[b].bounds, css);  
 }  
 }  
 };  
  
 // updates the positions of all trigger groups attached to a controller or a specific one, if provided  
 this.\_indicators.updateTriggerGroupPositions = function (specificGroup) {  
 var // constant vars  
 groups = specificGroup ? [specificGroup] : \_indicators.groups,  
 i = groups.length,  
 container = \_isDocument ? document.body : \_container,  
 containerOffset = \_isDocument ? {top: 0, left: 0} : \_util.get.offset(container, true),  
 edge = \_vertical ?  
 \_util.get.width(\_container) - EDGE\_OFFSET :  
 \_util.get.height(\_container) - EDGE\_OFFSET,  
 paramDimension = \_vertical ? "width" : "height",  
 paramTransform = \_vertical ? "Y" : "X";  
 var // changing vars  
 group,  
 elem,  
 pos,  
 elemSize,  
 transform;  
 while (i--) {  
 group = groups[i];  
 elem = group.element;  
 pos = group.triggerHook \* Controller.info("size");  
 elemSize = \_util.get[paramDimension](elem.firstChild.firstChild);  
 transform = pos > elemSize ? "translate" + paramTransform + "(-100%)" : "";  
  
 \_util.css(elem, {  
 top: containerOffset.top + (\_vertical ? pos : edge - group.members[0].options.indent),  
 left: containerOffset.left + (\_vertical ? edge - group.members[0].options.indent : pos)  
 });  
 \_util.css(elem.firstChild.firstChild, {  
 "-ms-transform" : transform,  
 "-webkit-transform" : transform,  
 "transform" : transform  
 });  
 }  
 };  
  
 // updates the label for the group to contain the name, if it only has one member  
 this.\_indicators.updateTriggerGroupLabel = function (group) {  
 var  
 text = "trigger" + (group.members.length > 1 ? "" : " " + group.members[0].options.name),  
 elem = group.element.firstChild.firstChild,  
 doUpdate = elem.textContent !== text;  
 if (doUpdate) {  
 elem.textContent = text;  
 if (\_vertical) { // bounds position is dependent on text length, so update  
 \_indicators.updateBoundsPositions();  
 }  
 }  
 };  
  
 // add indicators if global option is set  
 this.addScene = function (newScene) {  
  
 if (this.\_options.addIndicators && newScene instanceof ScrollMagic.Scene && newScene.controller() === Controller) {  
 newScene.addIndicators();  
 }  
 // call original destroy method  
 this.$super.addScene.apply(this, arguments);  
 };  
  
 // remove all previously set listeners on destroy  
 this.destroy = function () {  
 \_container.removeEventListener("resize", handleTriggerPositionChange);  
 if (!\_isDocument) {  
 window.removeEventListener("resize", handleTriggerPositionChange);  
 window.removeEventListener("scroll", handleTriggerPositionChange);  
 }  
 \_container.removeEventListener("resize", handleBoundsPositionChange);  
 \_container.removeEventListener("scroll", handleBoundsPositionChange);  
 // call original destroy method  
 this.$super.destroy.apply(this, arguments);  
 };  
 return Controller;  
  
 });  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* Internal class for the construction of Indicators  
 \* ----------------------------------------------------------------  
 \*/  
 var Indicator = function (Scene, options) {  
 var  
 Indicator = this,  
 \_elemBounds = TPL.bounds(),  
 \_elemStart = TPL.start(options.colorStart),  
 \_elemEnd = TPL.end(options.colorEnd),  
 \_boundsContainer = options.parent && \_util.get.elements(options.parent)[0],  
 \_vertical,  
 \_ctrl;  
  
 // (BUILD) - REMOVE IN MINIFY - START  
 var log = function () {  
 if (Scene.\_log) { // not available, when main source minified  
 Array.prototype.splice.call(arguments, 1, 0, "(" + NAMESPACE + ")", "->");  
 Scene.\_log.apply(this, arguments);  
 }  
 };  
 // (BUILD) - REMOVE IN MINIFY - END  
  
 options.name = options.name || \_autoindex;  
  
 // prepare bounds elements  
 \_elemStart.firstChild.textContent += " " + options.name;  
 \_elemEnd.textContent += " " + options.name;  
 \_elemBounds.appendChild(\_elemStart);  
 \_elemBounds.appendChild(\_elemEnd);  
  
 // set public variables  
 Indicator.options = options;  
 Indicator.bounds = \_elemBounds;  
 // will be set later  
 Indicator.triggerGroup = undefined;  
  
 // add indicators to DOM  
 this.add = function () {  
 \_ctrl = Scene.controller();  
 \_vertical = \_ctrl.info("vertical");  
  
 var isDocument = \_ctrl.info("isDocument");  
  
 if (!\_boundsContainer) {  
 // no parent supplied or doesnt exist  
 \_boundsContainer = isDocument ? document.body : \_ctrl.info("container"); // check if window/document (then use body)  
 }  
 if (!isDocument && \_util.css(\_boundsContainer, "position") === 'static') {  
 // position mode needed for correct positioning of indicators  
 \_util.css(\_boundsContainer, {position: "relative"});  
 }  
  
 // add listeners for updates  
 Scene.on("change.plugin\_addIndicators", handleTriggerParamsChange);  
 Scene.on("shift.plugin\_addIndicators", handleBoundsParamsChange);  
  
 // updates trigger & bounds (will add elements if needed)  
 updateTriggerGroup();  
 updateBounds();  
  
 setTimeout(function () { // do after all execution is finished otherwise sometimes size calculations are off  
 \_ctrl.\_indicators.updateBoundsPositions(Indicator);  
 }, 0);  
  
 log(3, "added indicators");  
 };  
  
 // remove indicators from DOM  
 this.remove = function () {  
 if (Indicator.triggerGroup) { // if not set there's nothing to remove  
 Scene.off("change.plugin\_addIndicators", handleTriggerParamsChange);  
 Scene.off("shift.plugin\_addIndicators", handleBoundsParamsChange);  
  
 if (Indicator.triggerGroup.members.length > 1) {  
 // just remove from memberlist of old group  
 var group = Indicator.triggerGroup;  
 group.members.splice(group.members.indexOf(Indicator), 1);  
 \_ctrl.\_indicators.updateTriggerGroupLabel(group);  
 \_ctrl.\_indicators.updateTriggerGroupPositions(group);  
 Indicator.triggerGroup = undefined;  
 } else {  
 // remove complete group  
 removeTriggerGroup();  
 }  
 removeBounds();  
   
 log(3, "removed indicators");  
 }  
 };  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* internal Event Handlers  
 \* ----------------------------------------------------------------  
 \*/  
  
 // event handler for when bounds params change  
 var handleBoundsParamsChange = function () {  
 updateBounds();  
 };  
  
 // event handler for when trigger params change  
 var handleTriggerParamsChange = function (e) {  
 if (e.what === "triggerHook") {  
 updateTriggerGroup();  
 }  
 };  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* Bounds (start / stop) management  
 \* ----------------------------------------------------------------  
 \*/  
  
 // adds an new bounds elements to the array and to the DOM  
 var addBounds = function () {  
 var v = \_ctrl.info("vertical");  
 // apply stuff we didn't know before...  
 \_util.css(\_elemStart.firstChild, {  
 "border-bottom-width" : v ? 1 : 0,  
 "border-right-width" : v ? 0 : 1,  
 "bottom": v ? -1 : options.indent,  
 "right": v ? options.indent : -1,  
 "padding": v ? "0 8px" : "2px 4px",  
 });  
 \_util.css(\_elemEnd, {  
 "border-top-width" : v ? 1 : 0,  
 "border-left-width" : v ? 0 : 1,  
 "top": v ? "100%" : "",  
 "right": v ? options.indent : "",  
 "bottom": v ? "" : options.indent,  
 "left": v ? "" : "100%",  
 "padding": v ? "0 8px" : "2px 4px"  
 });  
 // append  
 \_boundsContainer.appendChild(\_elemBounds);  
 };  
  
 // remove bounds from list and DOM  
 var removeBounds = function () {  
 \_elemBounds.parentNode.removeChild(\_elemBounds);  
 };  
  
 // update the start and end positions of the scene  
 var updateBounds = function () {  
 if (\_elemBounds.parentNode !== \_boundsContainer) {  
 addBounds(); // Add Bounds elements (start/end)  
 }  
 var css = {};  
 css[\_vertical ? "top" : "left"] = Scene.triggerPosition();  
 css[\_vertical ? "height" : "width"] = Scene.duration();  
 \_util.css(\_elemBounds, css);  
 \_util.css(\_elemEnd, {  
 display: Scene.duration() > 0 ? "" : "none"  
 });  
 };  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* trigger and trigger group management  
 \* ----------------------------------------------------------------  
 \*/  
  
 // adds an new trigger group to the array and to the DOM  
 var addTriggerGroup = function () {  
 var triggerElem = TPL.trigger(options.colorTrigger); // new trigger element  
 var css = {};  
 css[\_vertical ? "right" : "bottom"] = 0;  
 css[\_vertical ? "border-top-width" : "border-left-width"] = 1;  
 \_util.css(triggerElem.firstChild, css);  
 \_util.css(triggerElem.firstChild.firstChild, {  
 padding: \_vertical ? "0 8px 3px 8px" : "3px 4px"  
 });  
 document.body.appendChild(triggerElem); // directly add to body  
 var newGroup = {  
 triggerHook: Scene.triggerHook(),  
 element: triggerElem,  
 members: [Indicator]  
 };  
 \_ctrl.\_indicators.groups.push(newGroup);  
 Indicator.triggerGroup = newGroup;  
 // update right away  
 \_ctrl.\_indicators.updateTriggerGroupLabel(newGroup);  
 \_ctrl.\_indicators.updateTriggerGroupPositions(newGroup);  
 };  
  
 var removeTriggerGroup = function () {  
 \_ctrl.\_indicators.groups.splice(\_ctrl.\_indicators.groups.indexOf(Indicator.triggerGroup), 1);  
 Indicator.triggerGroup.element.parentNode.removeChild(Indicator.triggerGroup.element);  
 Indicator.triggerGroup = undefined;  
 };  
  
 // updates the trigger group -> either join existing or add new one  
 /\*   
 \* Logic:  
 \* 1 if a trigger group exist, check if it's in sync with Scene settings – if so, nothing else needs to happen  
 \* 2 try to find an existing one that matches Scene parameters  
 \* 2.1 If a match is found check if already assigned to an existing group  
 \* If so:  
 \* A: it was the last member of existing group -> kill whole group  
 \* B: the existing group has other members -> just remove from member list  
 \* 2.2 Assign to matching group  
 \* 3 if no new match could be found, check if assigned to existing group  
 \* A: yes, and it's the only member -> just update parameters and positions and keep using this group  
 \* B: yes but there are other members -> remove from member list and create a new one  
 \* C: no, so create a new one  
 \*/  
 var updateTriggerGroup = function () {  
 var  
 triggerHook = Scene.triggerHook(),  
 closeEnough = 0.0001;  
  
 // Have a group, check if it still matches  
 if (Indicator.triggerGroup) {  
 if (Math.abs(Indicator.triggerGroup.triggerHook - triggerHook) < closeEnough) {  
 // \_util.log(0, "trigger", options.name, "->", "no need to change, still in sync");  
 return; // all good  
 }  
 }  
 // Don't have a group, check if a matching one exists  
 // \_util.log(0, "trigger", options.name, "->", "out of sync!");  
 var  
 groups = \_ctrl.\_indicators.groups,  
 group,  
 i = groups.length;  
 while (i--) {  
 group = groups[i];  
 if (Math.abs(group.triggerHook - triggerHook) < closeEnough) {  
 // found a match!  
 // \_util.log(0, "trigger", options.name, "->", "found match");  
 if (Indicator.triggerGroup) { // do I have an old group that is out of sync?  
 if (Indicator.triggerGroup.members.length === 1) { // is it the only remaining group?  
 // \_util.log(0, "trigger", options.name, "->", "kill");  
 // was the last member, remove the whole group  
 removeTriggerGroup();  
 } else {  
 Indicator.triggerGroup.members.splice(Indicator.triggerGroup.members.indexOf(Indicator), 1); // just remove from memberlist of old group  
 \_ctrl.\_indicators.updateTriggerGroupLabel(Indicator.triggerGroup);  
 \_ctrl.\_indicators.updateTriggerGroupPositions(Indicator.triggerGroup);  
 // \_util.log(0, "trigger", options.name, "->", "removing from previous member list");  
 }  
 }  
 // join new group  
 group.members.push(Indicator);  
 Indicator.triggerGroup = group;  
 \_ctrl.\_indicators.updateTriggerGroupLabel(group);  
 return;  
 }  
 }  
  
 // at this point I am obviously out of sync and don't match any other group  
 if (Indicator.triggerGroup) {  
 if (Indicator.triggerGroup.members.length === 1) {  
 // \_util.log(0, "trigger", options.name, "->", "updating existing");  
 // out of sync but i'm the only member => just change and update  
 Indicator.triggerGroup.triggerHook = triggerHook;  
 \_ctrl.\_indicators.updateTriggerGroupPositions(Indicator.triggerGroup);  
 return;  
 } else {  
 // \_util.log(0, "trigger", options.name, "->", "removing from previous member list");  
 Indicator.triggerGroup.members.splice(Indicator.triggerGroup.members.indexOf(Indicator), 1); // just remove from memberlist of old group  
 \_ctrl.\_indicators.updateTriggerGroupLabel(Indicator.triggerGroup);  
 \_ctrl.\_indicators.updateTriggerGroupPositions(Indicator.triggerGroup);  
 Indicator.triggerGroup = undefined; // need a brand new group...  
 }  
 }  
 // \_util.log(0, "trigger", options.name, "->", "add a new one");  
 // did not find any match, make new trigger group  
 addTriggerGroup();  
 };  
 };  
  
 /\*  
 \* ----------------------------------------------------------------  
 \* Templates for the indicators  
 \* ----------------------------------------------------------------  
 \*/  
 var TPL = {  
 start: function (color) {  
 // inner element (for bottom offset -1, while keeping top position 0)  
 var inner = document.createElement("div");  
 inner.textContent = "start";  
 \_util.css(inner, {  
 position: "absolute",  
 overflow: "visible",  
 "border-width" : 0,  
 "border-style" : "solid",  
 color: color,  
 "border-color" : color  
 });  
 var e = document.createElement('div');  
 // wrapper  
 \_util.css(e, {  
 position: "absolute",  
 overflow: "visible",  
 width: 0,  
 height: 0  
 });  
 e.appendChild(inner);  
 return e;  
 },  
 end: function (color) {  
 var e = document.createElement('div');  
 e.textContent = "end";  
 \_util.css(e, {  
 position: "absolute",  
 overflow: "visible",  
 "border-width" : 0,  
 "border-style" : "solid",  
 color: color,  
 "border-color" : color  
 });  
 return e;  
 },  
 bounds: function () {  
 var e = document.createElement('div');  
 \_util.css(e, {  
 position: "absolute",  
 overflow: "visible",  
 "white-space": "nowrap",  
 "pointer-events" : "none",  
 "font-size": FONT\_SIZE  
 });  
 e.style.zIndex = ZINDEX;  
 return e;  
 },  
 trigger: function (color) {  
 // inner to be above or below line but keep position  
 var inner = document.createElement('div');  
 inner.textContent = "trigger";  
 \_util.css(inner, {  
 position: "relative",  
 });  
 // inner wrapper for right: 0 and main element has no size  
 var w = document.createElement('div');   
 \_util.css(w, {  
 position: "absolute",  
 overflow: "visible",  
 "border-width" : 0,  
 "border-style" : "solid",  
 color: color,  
 "border-color" : color  
 });  
 w.appendChild(inner);  
 // wrapper  
 var e = document.createElement('div');  
 \_util.css(e, {  
 position: "fixed",  
 overflow: "visible",  
 "white-space": "nowrap",  
 "pointer-events" : "none",  
 "font-size": FONT\_SIZE  
 });  
 e.style.zIndex = ZINDEX;  
 e.appendChild(w);  
 return e;  
 },  
 };  
  
}));

© 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).