[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/animation.velocity.js

/\*!  
 \* @file ScrollMagic Velocity Animation Plugin.  
 \*  
 \* requires: velocity ~1.2  
 \* Powered by VelocityJS: http://VelocityJS.org  
 \* Velocity is published under MIT license.  
 \*/  
/\*\*  
 \* This plugin is meant to be used in conjunction with the Velocity animation framework.   
 \* It offers an easy API to \_\_trigger\_\_ Velocity animations.  
 \*  
 \* With the current version of Velocity scrollbound animations (scenes with duration) are not supported.   
 \* This feature will be added as soon as Velocity provides the appropriate API.  
 \*   
 \* To have access to this extension, please include `plugins/animation.velocity.js`.  
 \* @requires {@link http://julian.com/research/velocity/|Velocity ~1.2.0}  
 \* @mixin animation.Velocity  
 \*/  
(function (root, factory) {  
 if (typeof define === 'function' && define.amd) {  
 // AMD. Register as an anonymous module.  
 define(['ScrollMagic', 'velocity'], factory);  
 } else if (typeof exports === 'object') {  
 // CommonJS  
 factory(require('scrollmagic'), require('velocity'));  
 } else {  
 // Browser globals  
 factory(root.ScrollMagic || (root.jQuery && root.jQuery.ScrollMagic), root.Velocity || (root.jQuery && root.jQuery.Velocity));  
 }  
}(this, function(ScrollMagic, velocity) {  
 "use strict";  
 var NAMESPACE = "animation.velocity";  
  
 // (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.");  
 }  
 if (!velocity) {  
 err("(" + NAMESPACE + ") -> ERROR: Velocity could not be found. Please make sure it's loaded before ScrollMagic or use an asynchronous loader like requirejs.");  
 }  
 // (BUILD) - REMOVE IN MINIFY - END  
   
 var autoindex = 0;  
  
 ScrollMagic.Scene.extend(function () {  
 var  
 Scene = this,  
 \_util = ScrollMagic.\_util,  
 \_currentProgress = 0,  
 \_elems,  
 \_properties,  
 \_options,  
 \_dataID; // used to identify element data related to this scene, will be defined everytime a new velocity animation is added  
  
 // (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  
  
 // set listeners  
 Scene.on("progress.plugin\_velocity", function () {  
 updateAnimationProgress();  
 });  
 Scene.on("destroy.plugin\_velocity", function (e) {  
 Scene.off("\*.plugin\_velocity");  
 Scene.removeVelocity(e.reset);  
 });  
  
 var animate = function (elem, properties, options) {  
 if (\_util.type.Array(elem)) {  
 elem.forEach(function(elem) {  
 animate(elem, properties, options);  
 });  
 } else {  
 // set reverse values  
 if (!velocity.Utilities.data(elem, \_dataID)) {  
 velocity.Utilities.data(elem, \_dataID,  
 {  
 reverseProps: \_util.css(elem, Object.keys(\_properties))  
 }  
 );  
 }  
 // animate  
 velocity(elem, properties, options);  
 if (options.queue !== undefined) {  
 velocity.Utilities.dequeue(elem, options.queue);  
 }  
 }  
 };  
 var reverse = function (elem, options) {  
 if (\_util.type.Array(elem)) {  
 elem.forEach(function(elem) {  
 reverse(elem, options);  
 });  
 } else {  
 var data = velocity.Utilities.data(elem, \_dataID);  
 if (data && data.reverseProps) {  
 velocity(elem, data.reverseProps, options);  
 if (options.queue !== undefined) {  
 velocity.Utilities.dequeue(elem, options.queue);  
 }  
 }  
 }  
 };  
  
 /\*\*  
 \* Update the tween progress to current position.  
 \* @private  
 \*/  
 var updateAnimationProgress = function () {  
 if (\_elems) {  
 var progress = Scene.progress();  
 if (progress != \_currentProgress) { // do we even need to update the progress?  
 if (Scene.duration() === 0) {  
 // play the animation  
 if (progress > 0) { // play forward  
 animate(\_elems, \_properties, \_options);  
 } else { // play reverse  
 reverse(\_elems, \_options);  
 // velocity(\_elems, \_propertiesReverse, \_options);  
 // velocity("reverse");  
 }  
 } else {  
 // TODO: Scrollbound animations not supported yet...  
 }  
 \_currentProgress = progress;  
 }  
 }  
 };  
  
 /\*\*  
 \* Add a Velocity animation to the scene.   
 \* The method accepts the same parameters as Velocity, with the first parameter being the target element.  
 \*  
 \* To gain better understanding, check out the [Velocity example](../examples/basic/simple\_velocity.html).  
 \* @memberof! animation.Velocity#  
 \*  
 \* @example  
 \* // trigger a Velocity animation  
 \* scene.setVelocity("#myElement", {opacity: 0.5}, {duration: 1000, easing: "linear"});  
 \*  
 \* @param {(object|string)} elems - One or more Dom Elements or a Selector that should be used as the target of the animation.  
 \* @param {object} properties - The CSS properties that should be animated.  
 \* @param {object} options - Options for the animation, like duration or easing.  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
 Scene.setVelocity = function (elems, properties, options) {  
 if (\_elems) { // kill old ani?  
 Scene.removeVelocity();  
 }  
  
 \_elems = \_util.get.elements(elems);  
 \_properties = properties || {};  
 \_options = options || {};  
 \_dataID = "ScrollMagic." + NAMESPACE + "[" + (autoindex++) + "]";  
  
 if (\_options.queue !== undefined) {  
 // we'll use the queue to identify the animation. When defined it will always stop the previously running one.  
 // if undefined the animation will always fully run, as is expected.  
 // defining anything other than 'false' as the que doesn't make much sense, because ScrollMagic takes control over the trigger.  
 // thus it is also overwritten.  
 \_options.queue = \_dataID + "\_queue";  
 }  
  
 // (BUILD) - REMOVE IN MINIFY - START  
 var checkDuration = function () {  
 if (Scene.duration() !== 0) {  
 log(1, "ERROR: The Velocity animation plugin does not support scrollbound animations (scenes with duration) yet.");  
 }  
 };  
 Scene.on("change.plugin\_velocity", function (e) {  
 if (e.what == 'duration') {  
 checkDuration();  
 }  
 });  
 checkDuration();  
 // (BUILD) - REMOVE IN MINIFY - END  
 log(3, "added animation");  
  
 updateAnimationProgress();  
 return Scene;  
 };  
 /\*\*  
 \* Remove the animation from the scene.   
 \* This will stop the scene from triggering the animation.  
 \*  
 \* Using the reset option you can decide if the animation should remain in the current state or be rewound to set the target elements back to the state they were in before the animation was added to the scene.  
 \* @memberof! animation.Velocity#  
 \*  
 \* @example  
 \* // remove the animation from the scene without resetting it  
 \* scene.removeVelocity();  
 \*  
 \* // remove the animation from the scene and reset the elements to initial state  
 \* scene.removeVelocity(true);  
 \*  
 \* @param {boolean} [reset=false] - If `true` the animation will rewound.  
 \* @returns {Scene} Parent object for chaining.  
 \*/  
 Scene.removeVelocity = function (reset) {  
 if (\_elems) {  
 // stop running animations  
 if (\_options.queue !== undefined) {  
 velocity(\_elems, "stop", \_options.queue);  
 }  
 if (reset) {  
 reverse(\_elems, {duration: 0});  
 }  
  
 \_elems.forEach(function(elem) {  
 velocity.Utilities.removeData(elem, \_dataID);  
 });  
 \_elems = \_properties = \_options = \_dataID = undefined;  
 }  
 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).