ScrollTrigger

gsap.com/docs/v3/Plugins/ScrollTrigger

added in v3.3.0

▶ Quick Start

ScrollTrigger enables anyone to create jaw-dropping scroll-based animations with minimal code. Infinitely flexible. Scrub, pin, snap, or just trigger anything scroll-related, even if it has nothing to do with animation.

⓵Detailed walkthrough

Get ahead of the game by also learning about the most common ScrollTrigger mistakes.

```
gsap.to('.box',{
scrollTrigger:'.box',// start the animation when ".box" enters the viewport
(once)
x:500
});
```

```
let tl = gsap.timeline({
// yes, we can add it to an entire timeline!
scrollTrigger:{
trigger:'.container',
pin:true,// pin the trigger element while active
start:'top top',// when the top of the trigger hits the top of the viewport
end:'+=500',// end after scrolling 500px beyond the start
scrub:1,// smooth scrubbing, takes 1 second to "catch up" to the scrollbar
snap:{
snapTo:'labels',// snap to the closest label in the timeline
duration:{min:0.2,max:3},// the snap animation should be at least 0.2 seconds,
but no more than 3 seconds (determined by velocity)
delay:0.2,// wait 0.2 seconds from the last scroll event before doing the
snapping
ease:'power1.inOut'// the ease of the snap animation ("power3" by default)


// add animations and labels to the timeline
tl.addLabel('start')
.from('.box p',{scale:0.3,rotation:45,autoAlpha:0})
.addLabel('color')
.from('.box',{backgroundColor:'#28a92b'})
.addLabel('spin')
.to('.box',{rotation:360})
.addLabel('end');
```

Standalone/Custom example

You don't need to put ScrollTriggers directly into animations (though that's probably the most common use case). Use the callbacks for anything...

```
ScrollTrigger.create({
trigger:'#id',
start:'top top',
endTrigger:'#otherID',
end:'bottom 50%+=100px',
onToggle:(self)=>console.log('toggled, isActive:', self.isActive),
onUpdate:(self)=>{
console.log(
'progress:',
                        self.progress.toFixed(3),
'direction:',
                        self.direction,
'velocity',
                        self.getVelocity()


});
```

Features

💡Feature Highlights

- **Link any animation to a particular element** so that it only plays when that element is in the viewport. This improves performance and ensures that your beautiful animations actually get seen!
- ScrollTriggers can perform an actions on an animation (play, pause, resume, restart, reverse, complete, reset) when entering/leaving the defined area or link it directly to the scrollbar so that it acts like a **scrubber** (`scrub: true`).
- **Soften the link between the animation and the the scrollbar** so that takes a certain amount of time to "catch up", like `scrub: 1` would take one second to catch up.
- **Integrated with ScrollSmoother**, GreenSock's smooth-scrolling tool built on native scroll technology (members-only benefit).

▶ **read more...**

Config Object

`scrollTrigger` can be used as either a shorthand for the `trigger` (described below) or as a configuration object with any of the following properties:

- Tween | Timeline - A GSAP Tween or Timeline instance that should be controlled by the ScrollTrigger. Only one animation is controlled per ScrollTrigger, but you can wrap all your animations in a single Timeline (recommended) or create multiple ScrollTriggers if you prefer.
- Number - If you pin large sections/panels you may notice what looks like a slight delay in pinning when you scroll quickly. That's caused by the fact that most modern browsers handle scroll repaints on a separate thread, so at the moment of pinning the browser may have already painted the pre-pinned content, making it visible for perhaps 1/60th of a second. The only way to counteract that is to have ScrollTrigger monitor the scroll velocity and anticipate the pin, applying it slightly early to avoid that flash of unpinned content. A value of `anticipatePin: 1` is typically fine, but you can reduce or increase that number to control how early it does the pinning. In many cases, however, you don't need any anticipatePin (the default is 0).

- Tween | Timeline Easily trigger animations inside 'horizontally' scrolling sections that are controlled by vertical scrolling

  ▶ View More details
- String | Number | Function - Determines the ending position of the ScrollTrigger.

  ▶ View More details
- String | Element - The element (or selector text for the element) whose position in the normal document flow is used for calculating where the ScrollTrigger ends. You don't need to define an `endTrigger` unless it's DIFFERENT than the `trigger` element because that's the default.

- Boolean | Number - if `true`, it will force the current ScrollTrigger's animation to completion if you **leave** its trigger area faster than a certain velocity (default 2500px/s). This helps avoid overlapping animations when the user scrolls quickly. You can specify a number for the minimum velocity, so `fastScrollEnd: 3000` would only activate if the velocity exceeds 3000px/s. See a demo here.
- Boolean - By default, it assumes your setup uses vertical scrolling but simply set `horizontal: true` if your setup uses horizontal scrolling instead.

- String - An arbitrary unique identifier for the ScrollTrigger instance which can be used with `ScrollTrigger.getById()`. This id is also added to the markers.

- Boolean - If `true`, the animation associated with the ScrollTrigger will have its invalidate() method called whenever a refresh() occurs (typically on resize). This flushes out any internally-recorded starting values.
- Object | Boolean - Adds markers that are helpful during development/troubleshooting. `markers: true` adds them with the defaults (startColor: "green", endColor: "red", fontSize: "16px", fontWeight: "normal", indent: 0) but you can customize them by using an object like

```
markers:
{startColor:"white",endColor:"white",fontSize:"18px",fontWeight:"bold",indent:20}
```

- Boolean - If `true`, the ScrollTrigger will kill() itself as soon as the end position is reached once. This causes it to stop listening for scroll events and it becomes eligible for garbage collection. This will only call onEnter a maximum of one time as well. It does **not** kill the associated animation. It's perfect for times when you only want an animation to play once when scrolling forward and never get reset or replayed. It also sets the toggleActions to "play none none none".

- Function - A callback for when the scroll position moves forward past the "start" (typically when the trigger is scrolled into view). It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

```
onEnter:({progress, direction, isActive})=>console.log(progress, direction, isActive)
```

- Function - A callback for when the scroll position moves backward past the "end" (typically when the trigger is scrolled back into view). It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

```
onEnterBack:({progress, direction, isActive})=>console.log(progress, direction, isActive)
```

- Function - A callback for when the scroll position moves forward past the "end" (typically when the trigger is scrolled out of view). It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onLeave:({progress, direction, isActive})=>console.log(progress, direction, isActive)
  ```

- Function - A callback for when the scroll position moves backward past the "start" (typically when the trigger is scrolled all the way backward past the start). It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onLeaveBack:({progress, direction, isActive})=>console.log(progress, direction, isActive)
  ```

- Function - A callback for when the a refresh occurs (typically a resize event) which forces the ScrollTrigger to recalculate all of its positioning. It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onRefresh:({progress, direction, isActive})=>console.log(progress, direction, isActive)
  ```

- Function - A callback that gets called every time the progress of the ScrollTrigger changes (meaning the scroll position changed). If you have a numeric `scrub` applied, keep in mind that the associated animation will keep scrubbing for a little while after the scroll position stops, so if your goal is to update something whenever the animation updates, it's best to apply an `onUpdate` to the animation itself rather than the ScrollTrigger. See a demo here.

  The onUpdate callback receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`.

  Example:

  ```
  onUpdate:self=>console.log("progress", self.progress)
  ```

- Function - A callback for when a numerical scrub has completed. This is only useful when a numerical scrub (like `scrub: 1`) is applied. The callback receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onScrubComplete:({progress, direction, isActive})=>console.log(progress, direction, isActive)
  ```

- Function - A callback for when the snapping has completed. This only applies when there's a `snap` defined. A snap will be cancelled if/when the user (or anything else) interacts in any way with scrolling, so the onSnapComplete would not be triggered at all in that case. The callback receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onSnapComplete:({progress, direction, isActive})=>console.log(progress, direction, isActive)
  ```

- Function - A callback for when the ScrollTrigger toggles from inactive to active **or** the other way around. This is typically when the scroll position moves past the "start" or "end" in either direction, but if it shoots past BOTH on the same tick, like if the user scrolls extremely fast, onToggle won't fire because the state hasn't changed. You can often use this one callback in the place of onEnter, onLeave, onEnterBack, and onLeaveBack by just checking the isActive property for toggling things. It receives one parameter - the ScrollTrigger instance itself which has properties/methods like `progress`, `direction`, `isActive`, and `getVelocity()`. Example:

  ```
  onToggle:self=>console.log("toggled to active:", isActive)
  ```

- Boolean | String | Element - An element (or selector text for the element) that should be pinned during the time that the ScrollTrigger is active, meaning it will appear to "stick" in its starting position while the rest of the content continues scrolling underneath it. Only one pinned element is allowed, but it can contain as many elements as you want. Setting `pin: true` will cause it to pin the `trigger` element.

  Warning don't animate the pinned element itself because that will throw off the measurements (ScrollTrigger is highly optimized for performance and pre-calculates as much as possible). Instead, you could nest things such that you're animating only elements INSIDE the pinned element.

  Note: if you are pinning something that is nested *inside* another element that *also* gets pinned, make sure you define a `pinnedContainer` so that ScrollTrigger knows to offset the start/end positions accordingly.

  Using React? Make sure to do proper cleanup - read this article.

- Element | String - If your ScrollTrigger's `trigger`/`endTrigger` element is **INSIDE** an element that gets pinned by *another* ScrollTrigger (pretty uncommon), that would cause the start/end positions to be thrown off by however long that pin lasts, so you can set the `pinnedContainer` to that parent/container element to have ScrollTrigger calculate those offsets accordingly. Again, this is very rarely needed. **Important**: nested pinning is not supported, so this feature is only for non-pinning ScrollTriggers

  (added in 3.7.0)

- Boolean - If `true`, the pinned element will be reparented to the `<body>` while it is actively pinned so that it can escape any ancestor containing blocks. If you notice odd behavior while pinning (like the pinned element suddenly shifting and then moving with the scroll), you probably have a `transform` or `will-change` on an ancestor element which breaks`position: fixed` behavior (it's a browser thing, not ScrollTrigger). It's best to set up your project to avoid those because reparenting can be expensive, but `pinReparent: true` can bail you out if you can't avoid them. Only use this feature if you must.
  Warning: if you have CSS rules that rely on specific nesting that'd be affected by the reparenting, they'll break. For example, a CSS rule like `.section .panel p {color: white}` wouldn't apply to the nested `<p>` anymore if you pin the `.panel` element with `pinReparent: true` because during the pin, it would no longer be inside the `<section>`, so make sure you write your CSS rules to accommodate the reparenting.

- Element | String - normally ScrollTrigger creates a `<div>` internally to wrap around pinned elements but in the *extremely* rare scenario where you're loading an iframe into the pinned element, it can cause the iframe to refresh when ScrollTrigger refreshes (like on window resize), so this feature allows you to specify an element that should be used as the spacer instead of the internally-created one. That way, ScrollTrigger won't remove/add it during its refresh, keeping iframe content intact.

- Boolean | String - By default, padding will be added to the bottom (or right for `horizontal: true`) to push other elements down so that when the pinned element gets unpinned, the following content catches up perfectly. Otherwise, things may scroll UNDER the pinned element. You can tell ScrollTrigger not to add any padding by setting `pinSpacing: false`.

  ▶ View More details
- "fixed" | "transform" - by default, `position: fixed` is used for pinning only if the scroller is the `<body>`, otherwise transforms are used (because `position: fixed` won't work in various nested scenarios), but you can **force** ScrollTrigger to use `position: fixed` by setting `pinType: "fixed"`. Typically this isn't necessary or helpful. Beware that if you set the CSS property `will-change: transform`, browsers treat it just like having a transform applied, breaking `position: fixed` elements (this is unrelated to ScrollTrigger/GSAP).

- Boolean | String - this feature activates as a ScrollTrigger is about to trigger an animation; it finds preceding scrollTrigger-based animations and forces those previous animations to their end state – avoiding unsightly overlaps. if `true`, it will affect all preceding ScrollTriggers. You can use an arbitrary string to limit their effect to only others with a matching string. So `preventOverlaps: "group1"` would only affect other ScrollTriggers with `preventOverlaps: "group1"`. See a demo here.
- number - it's **VERY** unlikely that you'd need to define a `refreshPriority` as long as you create your ScrollTriggers in the order they'd happen on the page (top-to-bottom or left-to-right)...which we *strongly* recommend doing. Otherwise, use `refreshPriority` to influence the order in which ScrollTriggers get refreshed to ensure that the pinning distance gets added to the start/end values of subsequent ScrollTriggers further down the page (that's why order matters). See the sort() method for details. A ScrollTrigger with `refreshPriority: 1` will get refreshed earlier than one with `refreshPriority: 0` (the default). You're welcome to use negative numbers too, and you can assign the same number to multiple ScrollTriggers.
- String | Element - By default, the `scroller` is the **viewport** itself, but if you'd like to add a ScrollTrigger to a scrollable `<div>`, for example, just define that as the scroller. You can use selector text like "#elementID" or the element itself.

- Boolean | Number - Links the progress of the animation directly to the scrollbar so it acts like a scrubber. You can apply smoothing so that it takes a little time for the playhead to catch up with the scrollbar's position! It can be any of the following
  - **Boolean** - `scrub: true` links the animation's progress directly to the ScrollTrigger's progress.
  - **Number** - The amount of time (in seconds) that the playhead should take to "catch up", so `scrub: 0.5` would cause the animation's playhead to take 0.5 seconds to catch up with the scrollbar's position. It's great for smoothing things out.
- Number | Array | Function | Object | "labels" | "labelsDirectional" - Allows you to snap to certain progress values (between 0 and 1) after the user stops scrolling. So `snap: 0.1` would snap in increments of 0.1 (10%, 20%, 30%, etc.). `snap: [0, 0.1, 0.5, 0.8, 1]` would only let it come to rest on one of those specific progress values. It can be any of the following...

  ▶ View More details
- String | Number | Function - Determines the starting position of the ScrollTrigger.

  ▶ View More details

- String - Determines how the linked animation is controlled at the 4 distinct toggle places - **onEnter**, **onLeave**, **onEnterBack**, and **onLeaveBack**, in that order. The default is `play none none none`. So `toggleActions: "play pause resume reset"` will play the animation when entering, pause it when leaving, resume it when entering again backwards, and reset (rewind back to the beginning) when scrolling all the way back past the beginning. You can use any of the following keywords for each action: "play", "pause", "resume", "reset", "restart", "complete", "reverse", and "none".

- String | Object - Adds/removes a class to an element (or multiple elements) when the ScrollTrigger toggles active/inactive. It can be either of the following:
  - **String** - The name of the class to add to the `trigger` element, like `toggleClass: "active"`
  - **Object** - To toggle a class for elements other than just the trigger, use the object syntax like `toggleClass: {targets: ".my-selector", className: "active"}`. The "targets" can be selector text, a direct reference to an element, or an Array of elements.

  Note that `toggleActions` don't apply to `toggleClass`. To have toggle class names in a different way, use the callback functions (onEnter, onLeave, onLeaveBack, and onEnterBack).
- String | Element - The element (or selector text for the element) whose position in the normal document flow is used to calculate where the ScrollTrigger starts.

💡Looking for Smooth Scrolling?
GSAP's own ScrollSmoother tool is built on top of ScrollTrigger, so it is totally integrated and super easy to use. Built on native scroll technology, it avoids most of the accessibility issues that plague other smooth-scrolling libraries.

Properties

[read-only] The Tween or Timeline associated with the ScrollTrigger instance (if any).

[read-only] Reflects the moment-by-moment direction of scrolling where `1` is forward and `-1` is backward.

[read-only] The ScrollTrigger's ending scroll position (numeric, in pixels).

[read-only] Only `true` if the scroll position is between the start and end positions of the ScrollTrigger instance.

A way to discern the touch capabilities of the current device - `0` is mouse/pointer only (no touch), `1` is touch-only, `2` accommodates both.

[read-only] The pin element (if one was defined). If selector text was used, like ".pin", the `pin` will be the element itself (not selector text)

[read-only] The overall progress of the ScrollTrigger instance where 0 is at the start, 0.5 is in the middle, and 1 is at the end.

[read-only] The scroller element (or window) associated with the ScrollTrigger. It's the thing whose scrollbar is linked to the ScrollTrigger. By default, it's the window (viewport).

[read-only] The ScrollTrigger's starting scroll position (numeric, in pixels).

[read-only] The trigger element (if one was defined). If selector text was used, like ".trigger", the `trigger` will be the element itself (not selector text)

[read-only] The vars configuration object used to create the ScrollTrigger instance

Methods

Disables the ScrollTrigger instance, immediately unpinning and restoring any pin-related changes made to the DOM by ScrollTrigger.

Enables the ScrollTrigger instance

Returns the `scrub` tween (default) or the snapping tween (`getTween(true)`)

Gets the scroll velocity in pixels-per-second

Kills the ScrollTrigger instance, immediately unpinning and restoring any pin-related changes made to the DOM by ScrollTrigger and removing all scroll-related listeners, etc. so that the instance is eligible for garbage collection. If you only want to temporarily disable the ScrollTrigger, use the disable() method instead.

Converts a timeline label into the associated scroll position (only applicable to ScrollTriggers whose "animation" is a timeline)

Returns the next ScrollTrigger in the refresh order.

Returns the previous ScrollTrigger in the refresh order.

Forces the ScrollTrigger instance to re-calculate its start and end values (the scroll positions where it'll be activated).
Gets/Sets the scroll position of the associated scroller (numeric).
Add a listener for any of the following events: "scrollStart", "scrollEnd", "refreshInit", "revert", "matchMedia", or"refresh" which get dispatched globally when **any** such ScrollTrigger-related event occurs (it is not tied to a particular instance).
Creates a coordinated group of ScrollTriggers (one for each target element) that batch their callbacks (onEnter, onLeave, etc.) within a certain interval, delivering a neat Array so that you can easily do something like create a staggered animation of all the elements that enter the viewport around the same time.
Clears any recorded scroll positions in ScrollTrigger so that no scroll positions get restored after a refresh(). Normally, this isn't necessary but in some frameworks that handle routing in unconventional ways, it can be useful.
Allows you to configure certain global behaviors of ScrollTrigger like `limitCallbacks`
Creates a standalone ScrollTrigger instance
Allows you to set the default values that apply to every ScrollTrigger upon creation, like `toggleActions`, `markers`, etc.
Returns an Array of all ScrollTrigger instances
Returns the ScrollTrigger that was assigned the corresponding `id`
Returns `true` if the element is in the viewport. You can optionally specify a minimum proportion, like `ScrollTrigger.isInViewport(element, 0.2)` would only return `true` if at least 20% of the element is in the viewport.
Indicates whether or not any ScrollTrigger-related scroller is in the process of scrolling.
Immediately calls `kill()` on **all** ScrollTriggers (except the main ScrollSmoother one if it exists).
[DEPRECATED] Allows you to set up ScrollTriggers that only apply to certain viewport sizes (using media queries).

A utility function for getting the maximum scroll value for a particular element/scroller. For example, if the element/scroller is 500px tall and contains 800px of content, maxScroll() would return 300.

Forces scrolling to be done on the JavaScript thread, ensuring screen updates are synchronized and the address bar doesn't show/hide on [most] mobile devices.

Super-flexible, unified way to sense meaningful events across all (touch/mouse/pointer) devices without wrestling with all the implementation details. Trigger simple callbacks like onUp, onDown, onLeft, onRight, onChange, onHover, onDrag, etc. Functionally identical to Observer.create()

Returns a normalized value representing the element's position in relation to the viewport where 0 is at the top of the viewport, 0.5 is in the center, and 1 is at the bottom. So, for example, if the top of the element is 80% down from the top of the viewport, the following code would return 0.8:
`ScrollTrigger.positionInViewport(element, "top");`

Recalculates the positioning of all of the ScrollTriggers on the page; this typically happens automatically when the window/scroller resizes but you can force it by calling `ScrollTrigger.refresh()`

Removes an event listener

Internally records the current inline CSS styles for the given elements so that when ScrollTrigger reverts (typically for a refresh() or matchMedia() change) those elements will be reverted accordingly even if they had animations that added/changed inline styles. Think of it like taking a snapshot of the inline CSS and telling ScrollTrigger "re-apply these inline styles only and dump all others when you revert internally".

Allows you to hijack the `scrollTop` and/or `scrollLeft` getters/setters for a particular scroller element so that you can implement things like smooth scrolling or other custom effects.

Returns a snapping function to which you can feed any value to snap, along with a direction where `1` is forward (greater than) and `-1` is backward (less than).

Sorts the internal Array of ScrollTrigger instances to control the order in which they refresh() (calculate their start/end values).

Checks where the scrollbar is and updates all ScrollTrigger instances' `progress` and `direction` values accordingly, controls the animation (if necessary) and fires the appropriate callbacks.

FAQs

▶

▶

▶

▶

How do I include undefined in my project?

▶

Is this included in the GSAP core?

▶

It works fine during development, but suddenly stops working in the production build! What do I do?

▶

Is it bad to register a plugin multiple times?

▶

How do I include ScrollTrigger in my project?

▶

Is this included in the GSAP core?

▶

It works fine during development, but suddenly stops working in the production build! What do I do?

▶

Is it bad to register a plugin multiple times?

Demos

Check out the full collection of Scroll animation demos on CodePen.

ScrollTrigger Demos