HTML5 only

Event Type	Description
`loadstart`	Sent when loading of the media begins.
`loadeddata`	The first frame of the media has finished loading.
`loadedmetadata`	The media's metadata has finished loading; all attributes now contain as much useful information as they're going to.
`qualitychange`	The quality of playback has changed.
`canplay`	Sent when enough data is available that the media can be played, at least for a couple of frames. This corresponds to the `HAVE_ENOUGH_DATA` `readyState`.
`canplaythrough`	Sent when the ready state changes to `CAN_PLAY_THROUGH`, indicating that the entire media can be played without interruption, assuming the download rate remains at least at the current level. *Note:* Manually setting the `currentTime` will eventually fire a `canplaythrough` event in firefox. Other browsers might not fire this event.
`stalled`	Sent when the user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
`waiting`	Sent when the requested operation (such as playback) is delayed pending the completion of another operation (such as a seek).
`emptied`	he media has become empty; for example, this event is sent if the media has already been loaded (or partially loaded), and the `load()` method is called to reload it.
`cuechange`	Sent when a `TextTrack` has changed the currently displaying

Event Type	Description
	cues.
`error`	Sent when an error occurs. The element's `error` attribute contains more information.

YouTube only

Event Type	Description
`statechange`	The state of the player has changed. The code can be accessed via `event.detail.code`. Possible values are `-1`: Unstarted, `0`: Ended, `1`: Playing, `2`: Paused, `3`: Buffering, `5`: Video cued. See the YouTube Docs for more information.

Note: These events also bubble up the DOM. The event target will be the container element.

Some event details borrowed from MDN.

Embeds

YouTube and Vimeo are currently supported and function much like a HTML5 video. Similar events and API methods are available for all types. However if you wish to access the API's directly. You can do so via the `embed` property of your player object - e.g. `player.embed`. You can then use the relevant methods from the third party APIs. More info on the respective API's here:

- YouTube iframe API Reference
- Vimeo player.js Reference

Note: Not all API methods may work 100%. Your mileage may vary. It's better to use the Plyr API where possible.

Shortcuts

By default, a player will bind the following keyboard shortcuts when it has focus. If you have the `global` option to `true` and there's only one player in the document then the shortcuts will work when any element has focus, apart from an element that requires input.

Key	Action
`0` to `9`	Seek from 0 to 90% respectively
`space`	Toggle playback
`K`	Toggle playback
←	Seek backward by the `seekTime` option
→	Seek forward by the `seekTime` option
↑	Increase volume
↓	Decrease volume
`M`	Toggle mute
`F`	Toggle fullscreen
`C`	Toggle captions
`L`	Toggle loop

Preview thumbnails

It's possible to display preview thumbnails as per the demo when you hover over the scrubber or while you are scrubbing in the main video area. This can be used for all video types but is easiest with HTML5 of course. You will need to generate the sprite or images yourself. This is possible using something like AWS transcoder to generate the frames and then combine them into a sprite image. Sprites are recommended for performance reasons - they will be much faster to download and easier to compress into a small file size making them load faster.

You can see the example VTT files here and here for how the sprites are done. The coordinates are set as the `xywh` hash on the URL in the order X Offset, Y Offset, Width, Height (e.g. `240p-00001.jpg#xywh=1708,480,427,240` is offset `1708px` from the left, `480px` from the top and is `427x240px`. If you want to include images per frame, this is also possible but will be slower, resulting in a degraded experience.