YouTube Player API Reference for iframe Embeds

The IFrame player API lets you embed a YouTube video player on your website and control the player using JavaScript.

Using the API's JavaScript functions, you can queue videos for playback; play, pause, or stop those videos; adjust the player volume; or retrieve information about the video being played. You can also add event listeners that will execute in response to certain player events, such as a player state change.

This guide explains how to use the IFrame API. It identifies the different types of events that the API can send and explains how to write event listeners to respond to those events. It also details the different JavaScript functions that you can call to control the video player as well as the player parameters you can use to further customize the player.

Requirements

The user's browser must support the HTML5 `postMessage` feature. Most modern browsers support `postMessage`.

Embedded players must have a viewport that is at least 200px by 200px. If the player displays controls, it must be large enough to fully display the controls without shrinking the viewport below the minimum size. We recommend 16:9 players be at least 480 pixels wide and 270 pixels tall.

Any web page that uses the IFrame API must also implement the following JavaScript function:

- `onYouTubeIframeAPIReady` – The API will call this function when the page has finished downloading the JavaScript for the player API, which enables you to then use

the API on your page. Thus, this function might create the player objects that you want to display when the page loads.

Getting started

The sample HTML page below creates an embedded player that will load a video, play it for six seconds, and then stop the playback. The numbered comments in the HTML are explained in the list below the example.

```
<!DOCTYPE html>
<html>
  <body>
    <!-- 1. The <iframe> (and video player) will replace this <div> tag. -->
    <div id="player"></div>

    <script>
      // 2. This code loads the IFrame Player API code asynchronously.
      var tag = document.createElement('script');

      tag.src = "https://www.youtube.com/iframe_api";
      var firstScriptTag = document.getElementsByTagName('script')[0];
      firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

      // 3. This function creates an <iframe> (and YouTube player)
      //    after the API code downloads.
      var player;
      function onYouTubeIframeAPIReady() {
        player = new YT.Player('player', {
          height: '390',
          width: '640',
          videoId: 'M7lc1UVf-VE',
          playerVars: {
            'playsinline': 1
          },
          events: {
            'onReady': onPlayerReady,
            'onStateChange': onPlayerStateChange
          }
        });
      }

      // 4. The API will call this function when the video player is ready.
      function onPlayerReady(event) {
        event.target.playVideo();
```

```
      }

      // 5. The API calls this function when the player's state changes.
      //    The function indicates that when playing a video (state=1),
      //    the player should play for six seconds and then stop.
      var done = false;
      function onPlayerStateChange(event) {
        if (event.data == YT.PlayerState.PLAYING && !done) {
          setTimeout(stopVideo, 6000);
          done = true;
        }
      }
      function stopVideo() {
        player.stopVideo();
      }
    </script>
  </body>
</html>
```

The following list provides more details about the sample above:

1. The `<div>` tag in this section identifies the location on the page where the IFrame API will place the video player. The constructor for the player object, which is described in the Loading a video player (#Loading_a_Video_Player) section, identifies the `<div>` tag by its `id` to ensure that the API places the `<iframe>` in the proper location. Specifically, the IFrame API will replace the `<div>` tag with the `<iframe>` tag.

   As an alternative, you could also put the `<iframe>` element directly on the page. The Loading a video player (#Loading_a_Video_Player) section explains how to do so.

2. The code in this section loads the IFrame Player API JavaScript code. The example uses DOM modification to download the API code to ensure that the code is retrieved asynchronously. (The `<script>` tag's `async` attribute, which also enables asynchronous downloads, is not yet supported in all modern browsers as discussed in this Stack Overflow answer (http://stackoverflow.com/a/1834129).

3. The `onYouTubeIframeAPIReady` function will execute as soon as the player API code downloads. This portion of the code defines a global variable, `player`, which refers to the video player you are embedding, and the function then constructs the video player object.

4. The `onPlayerReady` function will execute when the `onReady` event fires. In this example, the function indicates that when the video player is ready, it should begin to play.

5. The API will call the `onPlayerStateChange` function when the player's state changes, which may indicate that the player is playing, paused, finished, and so forth. The function indicates that when the player state is `1` (playing), the player should play for six seconds and then call the `stopVideo` function to stop the video.

Loading a video player

After the API's JavaScript code loads, the API will call the `onYouTubeIframeAPIReady` function, at which point you can construct a `YT.Player` object to insert a video player on your page. The HTML excerpt below shows the `onYouTubeIframeAPIReady` function from the example above:

```
var player;
function onYouTubeIframeAPIReady() {
  player = new YT.Player('player', {
    height: '390',
    width: '640',
    videoId: 'M7lc1UVf-VE',
    playerVars: {
      'playsinline': 1
    },
    events: {
      'onReady': onPlayerReady,
      'onStateChange': onPlayerStateChange
    }
  });
}
```

The constructor for the video player specifies the following parameters:

1. The first parameter specifies either the DOM element or the `id` of the HTML element where the API will insert the `<iframe>` tag containing the player.

   The IFrame API will replace the specified element with the `<iframe>` element containing the player. This could affect the layout of your page if the element being replaced has a different display style than the inserted `<iframe>` element. By default, an `<iframe>` displays as an `inline-block` element.

2. The second parameter is an object that specifies player options. The object contains the following properties:

   - `width` (number) – The width of the video player. The default value is `640`.

- `height` (number) – The height of the video player. The default value is `390`.
- `videoId` (string) – The YouTube video ID that identifies the video that the player will load.
- `playerVars` (object) – The object's properties identify player parameters (/youtube/player_parameters) that can be used to customize the player.
- `events` (object) – The object's properties identify the events that the API fires and the functions (event listeners) that the API will call when those events occur. In the example, the constructor indicates that the `onPlayerReady` function will execute when the `onReady` event fires and that the `onPlayerStateChange` function will execute when the `onStateChange` event fires.

As mentioned in the Getting started (#Getting_Started) section, instead of writing an empty `<div>` element on your page, which the player API's JavaScript code will then replace with an `<iframe>` element, you could create the `<iframe>` tag yourself. The first example in the Examples (#Examples) section shows how to do this.

```
<iframe id="player" type="text/html" width="640" height="390"
  src="http://www.youtube.com/embed/M7lc1UVf-VE?enablejsapi=1&origin=http://e
  frameborder="0"></iframe>
```

Note that if you do write the `<iframe>` tag, then when you construct the `YT.Player` object, you do not need to specify values for the `width` and `height`, which are specified as attributes of the `<iframe>` tag, or the `videoId` and player parameters, which are are specified in the `src` URL. As an extra security measure, you should also include the `origin` parameter to the URL, specifying the URL scheme (`http://` or `https://`) and full domain of your host page as the parameter value. While `origin` is optional, including it protects against malicious third-party JavaScript being injected into your page and hijacking control of your YouTube player.

The Examples (#Examples) section also shows a couple other examples for constructing video player objects.

Operations

To call the player API methods, you must first get a reference to the player object you wish to control. You obtain the reference by creating a `YT.Player` object as discussed in the

Getting started (#Getting_Started) and Loading a video player (#Loading_a_Video_Player) sections of this document.

Functions

Queueing functions

Queueing functions allow you to load and play a video, a playlist, or another list of videos. If you are using the object syntax described below to call these functions, then you can also queue or load a list of a user's uploaded videos.

The API supports two different syntaxes for calling the queueing functions.

- The argument syntax requires function arguments to be listed in a prescribed order.
- The object syntax lets you pass an object as a single parameter and to define object properties for the function arguments that you wish to set. In addition, the API may support additional functionality that the argument syntax does not support.

For example, the **`loadVideoById`** (#loadVideoById) function can be called in either of the following ways. Note that the object syntax supports the **`endSeconds`** property, which the argument syntax does not support.

- **Argument syntax**

```
loadVideoById("bHQqvYy5KYo", 5, "large")
```

- **Object syntax**

```
loadVideoById({'videoId': 'bHQqvYy5KYo',
               'startSeconds': 5,
               'endSeconds': 60});
```

Queueing functions for videos

`cueVideoById`

- **Argument syntax**

```
player.cueVideoById(videoId:String,
                    startSeconds:Number):Void
```

- **Object syntax**

```
player.cueVideoById({videoId:String,
                     startSeconds:Number,
                     endSeconds:Number}):Void
```

This function loads the specified video's thumbnail and prepares the player to play the video. The player does not request the FLV until `playVideo()` (#playVideo) or `seekTo()` (#seekTo) is called.

- The required `videoId` parameter specifies the YouTube Video ID of the video to be played. In the YouTube Data API, a `video` resource's `id` (/youtube/v3/docs/videos#id) property specifies the ID.
- The optional `startSeconds` parameter accepts a float/integer and specifies the time from which the video should start playing when `playVideo()` (#playVideo) is called. If you specify a `startSeconds` value and then call `seekTo()` (#seekTo), then the player plays from the time specified in the `seekTo()` (#seekTo) call. When the video is cued and ready to play, the player will broadcast a `video cued` event (#getPlayerState) (`5`).
- The optional `endSeconds` parameter, which is only supported in object syntax, accepts a float/integer and specifies the time when the video should stop playing when `playVideo()` (#playVideo) is called. If you specify an `endSeconds` value and then call `seekTo()` (#seekTo), the `endSeconds` value will no longer be in effect.

`loadVideoById`

- **Argument syntax**

```
player.loadVideoById(videoId:String,
                     startSeconds:Number):Void
```

- **Object syntax**

```
player.loadVideoById({videoId:String,
                      startSeconds:Number,
                      endSeconds:Number}):Void
```

This function loads and plays the specified video.

- The required `videoId` parameter specifies the YouTube Video ID of the video to be played. In the YouTube Data API, a `video` resource's `id` (/youtube/v3/docs/videos#id) property specifies the ID.
- The optional `startSeconds` parameter accepts a float/integer. If it is specified, then the video will start from the closest keyframe to the specified time.
- The optional `endSeconds` parameter accepts a float/integer. If it is specified, then the video will stop playing at the specified time.

`cueVideoByUrl`

- **Argument syntax**

```
player.cueVideoByUrl(mediaContentUrl:String,
                     startSeconds:Number):Void
```

- **Object syntax**

```
player.cueVideoByUrl({mediaContentUrl:String,
                      startSeconds:Number,
                      endSeconds:Number}):Void
```

This function loads the specified video's thumbnail and prepares the player to play the video. The player does not request the FLV until `playVideo()` (#playVideo) or `seekTo()` (#seekTo) is called.

- The required `mediaContentUrl` parameter specifies a fully qualified YouTube player URL in the format `http://www.youtube.com/v/VIDEO_ID?version=3`.
- The optional `startSeconds` parameter accepts a float/integer and specifies the time from which the video should start playing when `playVideo()`

(#playVideo) is called. If you specify `startSeconds` and then call `seekTo()` (#seekTo), then the player plays from the time specified in the `seekTo()` (#seekTo) call. When the video is cued and ready to play, the player will broadcast a `video cued` event (#getPlayerState) (5).

- The optional `endSeconds` parameter, which is only supported in object syntax, accepts a float/integer and specifies the time when the video should stop playing when `playVideo()` (#playVideo) is called. If you specify an `endSeconds` value and then call `seekTo()` (#seekTo), the `endSeconds` value will no longer be in effect.

`loadVideoByUrl`

- **Argument syntax**

```
player.loadVideoByUrl(mediaContentUrl:String,
                      startSeconds:Number):Void
```

- **Object syntax**

```
player.loadVideoByUrl({mediaContentUrl:String,
                       startSeconds:Number,
                       endSeconds:Number}):Void
```

This function loads and plays the specified video.

- The required `mediaContentUrl` parameter specifies a fully qualified YouTube player URL in the format `http://www.youtube.com/v/VIDEO_ID?version=3`.
- The optional `startSeconds` parameter accepts a float/integer and specifies the time from which the video should start playing. If `startSeconds` (number can be a float) is specified, the video will start from the closest keyframe to the specified time.
- The optional `endSeconds` parameter, which is only supported in object syntax, accepts a float/integer and specifies the time when the video should stop playing.

Queueing functions for lists

The `cuePlaylist` and `loadPlaylist` functions allow you to load and play a playlist. If you are using object syntax to call these functions, you can also queue (or load) a list of a user's uploaded videos.

Since the functions work differently depending on whether they are called using the argument syntax or the object syntax, both calling methods are documented below.

`cuePlaylist`

- **Argument syntax**

```
player.cuePlaylist(playlist:String|Array,
                   index:Number,
                   startSeconds:Number):Void
```

Queues the specified playlist. When the playlist is cued and ready to play, the player will broadcast a `video cued` event (#getPlayerState) (`5`).

- The required `playlist` parameter specifies an array of YouTube video IDs. In the YouTube Data API, the `video` resource's `id` (/youtube/v3/docs/videos#id) property identifies that video's ID.
- The optional `index` parameter specifies the index of the first video in the playlist that will play. The parameter uses a zero-based index, and the default parameter value is `0`, so the default behavior is to load and play the first video in the playlist.
- The optional `startSeconds` parameter accepts a float/integer and specifies the time from which the first video in the playlist should start playing when the `playVideo()` (#playVideo) function is called. If you specify a `startSeconds` value and then call `seekTo()` (#seekTo), then the player plays from the time specified in the `seekTo()` (#seekTo) call. If you cue a playlist and then call the `playVideoAt()` (#playVideoAt) function, the player will start playing at the beginning of the specified video.

- **Object syntax**

```
player.cuePlaylist({listType:String,
                    list:String,
                    index:Number,
                    startSeconds:Number}):Void
```

Queues the specified list of videos. The list can be a playlist or a user's uploaded videos feed. The ability to queue a list of search results is deprecated (#release_notes_10_13_2020) and will no longer be supported as of 15 November 2020.

When the list is cued and ready to play, the player will broadcast a `video cued` event (#getPlayerState) (`5`).

- The optional `listType` property specifies the type of results feed that you are retrieving. Valid values are `playlist` and `user_uploads`. A deprecated value, `search`, will no longer be supported as of 15 November 2020. The default value is `playlist`.
- The required `list` property contains a key that identifies the particular list of videos that YouTube should return.
  - If the `listType` property value is `playlist`, then the `list` property specifies the playlist ID or an array of video IDs. In the YouTube Data API, the `playlist` resource's `id` (/youtube/v3/docs/playlists#id) property identifies a playlist's ID, and the `video` resource's `id` (/youtube/v3/docs/videos#id) property specifies a video ID.
  - If the `listType` property value is `user_uploads`, then the `list` property identifies the user whose uploaded videos will be returned.
  - If the `listType` property value is `search`, then the `list` property specifies the search query. **Note:** This functionality is deprecated (#release_notes_10_13_2020) and will no longer be supported as of 15 November 2020.
- The optional `index` property specifies the index of the first video in the list that will play. The parameter uses a zero-based index, and the default parameter value is `0`, so the default behavior is to load and play the first video in the list.
- The optional `startSeconds` property accepts a float/integer and specifies the time from which the first video in the list should start playing when the `playVideo()` (#playVideo) function is called. If you specify a `startSeconds` value and then call `seekTo()` (#seekTo), then the player plays from the time specified in the `seekTo()` (#seekTo) call. If you cue a list and then call the `playVideoAt()` (#playVideoAt) function, the player will start playing at the beginning of the specified video.

`loadPlaylist`

- **Argument syntax**

```
player.loadPlaylist(playlist:String|Array,
                    index:Number,
                    startSeconds:Number):Void
```

This function loads the specified playlist and plays it.

- The required `playlist` parameter specifies an array of YouTube video IDs. In the YouTube Data API, the `video` resource's `id` (/youtube/v3/docs/videos#id) property specifies a video ID.
- The optional `index` parameter specifies the index of the first video in the playlist that will play. The parameter uses a zero-based index, and the default parameter value is `0`, so the default behavior is to load and play the first video in the playlist.
- The optional `startSeconds` parameter accepts a float/integer and specifies the time from which the first video in the playlist should start playing.

- **Object syntax**

```
player.loadPlaylist({list:String,
                     listType:String,
                     index:Number,
                     startSeconds:Number}):Void
```

This function loads the specified list and plays it. The list can be a playlist or a user's uploaded videos feed. The ability to load a list of search results is deprecated (#release_notes_10_13_2020) and will no longer be supported as of 15 November 2020.

- The optional `listType` property specifies the type of results feed that you are retrieving. Valid values are `playlist` and `user_uploads`. A deprecated value, `search`, will no longer be supported as of 15 November 2020. The default value is `playlist`.
- The required `list` property contains a key that identifies the particular list of videos that YouTube should return.

- If the `listType` property value is `playlist`, then the `list` property specifies a playlist ID or an array of video IDs. In the YouTube Data API, the `playlist` resource's `id` (/youtube/v3/docs/playlists#id) property specifies a playlist's ID, and the `video` resource's `id` (/youtube/v3/docs/videos#id) property specifies a video ID.
- If the `listType` property value is `user_uploads`, then the `list` property identifies the user whose uploaded videos will be returned.
- If the `listType` property value is `search`, then the `list` property specifies the search query. **Note:** This functionality is deprecated (#release_notes_10_13_2020) and will no longer be supported as of 15 November 2020.

- The optional `index` property specifies the index of the first video in the list that will play. The parameter uses a zero-based index, and the default parameter value is `0`, so the default behavior is to load and play the first video in the list.
- The optional `startSeconds` property accepts a float/integer and specifies the time from which the first video in the list should start playing.

Playback controls and player settings

Playing a video

`player.playVideo():Void`

Plays the currently cued/loaded video. The final player state after this function executes will be `playing` (1).

Note: A playback only counts toward a video's official view count if it is initiated via a native play button in the player.

`player.pauseVideo():Void`

Pauses the currently playing video. The final player state after this function executes will be `paused` (2) unless the player is in the `ended` (`0`) state when the function is called, in which case the player state will not change.

`player.stopVideo():Void`

Stops and cancels loading of the current video. This function should be reserved for rare situations when you know that the user will not be watching additional video in

the player. If your intent is to pause the video, you should just call the `pauseVideo` (#pauseVideo) function. If you want to change the video that the player is playing, you can call one of the queueing functions without calling `stopVideo` first.

Important: Unlike the `pauseVideo` (#pauseVideo) function, which leaves the player in the `paused` (`2`) state, the `stopVideo` function could put the player into any not-playing state, including `ended` (`0`), `paused` (`2`), `video cued` (`5`) or `unstarted` (`-1`).

`player.seekTo(seconds:Number, allowSeekAhead:Boolean):Void`

Seeks to a specified time in the video. If the player is paused when the function is called, it will remain paused. If the function is called from another state (`playing`, `video cued`, etc.), the player will play the video.

- The `seconds` parameter identifies the time to which the player should advance.

  The player will advance to the closest keyframe before that time unless the player has already downloaded the portion of the video to which the user is seeking.

- The `allowSeekAhead` parameter determines whether the player will make a new request to the server if the `seconds` parameter specifies a time outside of the currently buffered video data.

  We recommend that you set this parameter to `false` while the user drags the mouse along a video progress bar and then set it to `true` when the user releases the mouse. This approach lets a user scroll to different points of a video without requesting new video streams by scrolling past unbuffered points in the video. When the user releases the mouse button, the player advances to the desired point in the video and requests a new video stream if necessary.

Controlling playback of 360° videos

The 360° video playback experience has limited support on mobile devices. On unsupported devices, 360° appear distorted and there is no supported way to change the viewing perspective at all, including throug sing orientation sensors, or responding to touch/drag actions on the device's screen.

`player.getSphericalProperties():Object`

Retrieves properties that describe the viewer's current perspective, or view, for a video playback. In addition:

- This object is only populated for 360° videos, which are also called spherical videos.
- If the current video is not a 360° video or if the function is called from a non-supported device, then the function returns an empty object.
- On supported mobile devices, if the `enableOrientationSensor` (#enableOrientationSensor) property is set to `true`, then this function returns an object in which the `fov` property contains the correct value and the other properties are set to `0`.

The object contains the following properties:

Properties	
`yaw`	A number in the range [0, 360) that represents the horizontal angle of the view in degrees, which reflects the extent to which the user turns the view to face further left or right. The neutral position, facing the center of the video in its equirectangular projection, represents 0°, and this value increases as the viewer turns left.
`pitch`	A number in the range [-90, 90] that represents the vertical angle of the view in degrees, which reflects the extent to which the user adjusts the view to look up or down. The neutral position, facing the center of the video in its equirectangular projection, represents 0°, and this value increases as the viewer looks up.
`roll`	A number in the range [-180, 180] that represents the clockwise or counterclockwise rotational angle of the view in degrees. The neutral position, with the horizontal axis in the equirectangular projection being parallel to the horizontal axis of the view, represents 0°. The value increases as the view rotates clockwise and decreases as the view rotates counterclockwise. Note that the embedded player does not present a user interface for adjusting the roll of the view. The roll can be adjusted in either of these mutually exclusive ways: 1. Use the orientation sensor in a mobile browser to provide roll for the view. If the orientation sensor (#enableOrientationSensor) is enabled, then the `getSphericalProperties` function always returns `0` as the value of the `roll` property.

	2. If the orientation sensor is disabled, set the roll to a nonzero value using this API.
`fov`	A number in the range [30, 120] that represents the field-of-view of the view in degrees as measured along the longer edge of the viewport. The shorter edge is automatically adjusted to be proportional to the aspect ratio of the view. The default value is 100 degrees. Decreasing the value is like zooming in on the video content, and increasing the value is like zooming out. This value can be adjusted either by using the API or by using the mousewheel when the video is in fullscreen mode.

`player.setSphericalProperties(properties:Object):Void`

Sets the video orientation for playback of a 360° video. (If the current video is not spherical, the method is a no-op regardless of the input.)

The player view responds to calls to this method by updating to reflect the values of any known properties in the `properties` object. The view persists values for any other known properties not included in that object.

In addition:

- If the object contains unknown and/or unexpected properties, the player ignores them.
- As noted at the beginning of this section, the 360° video playback experience is not supported on all mobile devices.
- By default, on supported mobile devices, this function sets only sets the `fov` property and does not affect the `yaw`, `pitch`, and `roll` properties for 360° video playbacks. See the `enableOrientationSensor` property below for more detail.

The `properties` object passed to the function contains the following properties:

Properties	
`yaw`	See definition (#spherical-property-yaw) above.
`pitch`	See definition (#spherical-property-pitch) above.
`roll`	See definition (#spherical-property-roll) above.
`fov`	See definition (#spherical-property-fov) above.

`enableOrientationSensor` ★

Note: This property affects the 360° viewing experience on supported devices only.

A boolean value that indicates whether the IFrame embed should respond to events that signal changes in a supported device's orientation, such as a mobile browser's `DeviceOrientationEvent`. The default parameter value is `true`.

Supported mobile devices

- When the value is `true`, an embedded player relies *only* on the device's movement to adjust the `yaw`, `pitch`, and `roll` properties for 360° video playbacks. However, the `fov` property can still be changed via the API, and the API is, in fact, the only way to change the `fov` property on a mobile device. This is the default behavior.
- When the value is `false`, then the device's movement does not affect the 360° viewing experience, and the `yaw`, `pitch`, `roll`, and `fov` properties must all be set via the API.

Unsupported mobile devices
The `enableOrientationSensor` property value does not have any effect on the playback experience.

Playing a video in a playlist

`player.nextVideo():Void`

This function loads and plays the next video in the playlist.

- If `player.nextVideo()` is called while the last video in the playlist is being watched, and the playlist is set to play continuously (`loop` (#setLoop)), then the player will load and play the first video in the list.
- If `player.nextVideo()` is called while the last video in the playlist is being watched, and the playlist is not set to play continuously, then playback will end.

`player.previousVideo():Void`

This function loads and plays the previous video in the playlist.

- If `player.previousVideo()` is called while the first video in the playlist is being watched, and the playlist is set to play continuously (`loop` (#setLoop)), then the player will load and play the last video in the list.

- If `player.previousVideo()` is called while the first video in the playlist is being watched, and the playlist is not set to play continuously, then the player will restart the first playlist video from the beginning.

`player.playVideoAt(index:Number):Void`

This function loads and plays the specified video in the playlist.

- The required `index` parameter specifies the index of the video that you want to play in the playlist. The parameter uses a zero-based index, so a value of `0` identifies the first video in the list. If you have shuffled (#setShuffle) the playlist, this function will play the video at the specified position in the shuffled playlist.

Changing the player volume

`player.mute():Void`

Mutes the player.

`player.unMute():Void`

Unmutes the player.

`player.isMuted():Boolean`

Returns `true` if the player is muted, `false` if not.

`player.setVolume(volume:Number):Void`

Sets the volume. Accepts an integer between `0` and `100`.

`player.getVolume():Number`

Returns the player's current volume, an integer between `0` and `100`. Note that `getVolume()` will return the volume even if the player is muted.

Setting the player size

`player.setSize(width:Number, height:Number):Object`

Sets the size in pixels of the `<iframe>` that contains the player.

Setting the playback rate

`player.getPlaybackRate():Number`

This function retrieves the playback rate of the currently playing video. The default playback rate is `1`, which indicates that the video is playing at normal speed. Playback rates may include values like `0.25`, `0.5`, `1`, `1.5`, and `2`.

`player.setPlaybackRate(suggestedRate:Number):Void`

This function sets the suggested playback rate for the current video. If the playback rate changes, it will only change for the video that is already cued or being played. If you set the playback rate for a cued video, that rate will still be in effect when the `playVideo` function is called or the user initiates playback directly through the player controls. In addition, calling functions to cue or load videos or playlists (`cueVideoById`, `loadVideoById`, etc.) will reset the playback rate to `1`.

Calling this function does not guarantee that the playback rate will actually change. However, if the playback rate does change, the `onPlaybackRateChange` (#onPlaybackRateChange) event will fire, and your code should respond to the event rather than the fact that it called the `setPlaybackRate` function.

The `getAvailablePlaybackRates` (#getAvailablePlaybackRates) method will return the possible playback rates for the currently playing video. However, if you set the `suggestedRate` parameter to a non-supported integer or float value, the player will round that value down to the nearest supported value in the direction of `1`.

`player.getAvailablePlaybackRates():Array`

This function returns the set of playback rates in which the current video is available. The default value is `1`, which indicates that the video is playing in normal speed.

The function returns an array of numbers ordered from slowest to fastest playback speed. Even if the player does not support variable playback speeds, the array should always contain at least one value (`1`).

Setting playback behavior for playlists

`player.setLoop(loopPlaylists:Boolean):Void`

This function indicates whether the video player should continuously play a playlist or if it should stop playing after the last video in the playlist ends. The default behavior is that playlists do not loop.

This setting will persist even if you load or cue a different playlist, which means that if you load a playlist, call the `setLoop` function with a value of `true`, and then load a

second playlist, the second playlist will also loop.

The required `loopPlaylists` parameter identifies the looping behavior.

- If the parameter value is `true`, then the video player will continuously play playlists. After playing the last video in a playlist, the video player will go back to the beginning of the playlist and play it again.
- If the parameter value is `false`, then playbacks will end after the video player plays the last video in a playlist.

`player.setShuffle(shufflePlaylist:Boolean):Void`

This function indicates whether a playlist's videos should be shuffled so that they play back in an order different from the one that the playlist creator designated. If you shuffle a playlist after it has already started playing, the list will be reordered while the video that is playing continues to play. The next video that plays will then be selected based on the reordered list.

This setting will not persist if you load or cue a different playlist, which means that if you load a playlist, call the `setShuffle` function, and then load a second playlist, the second playlist will not be shuffled.

The required `shufflePlaylist` parameter indicates whether YouTube should shuffle the playlist.

- If the parameter value is `true`, then YouTube will shuffle the playlist order. If you instruct the function to shuffle a playlist that has already been shuffled, YouTube will shuffle the order again.
- If the parameter value is `false`, then YouTube will change the playlist order back to its original order.

Playback status

`player.getVideoLoadedFraction():Float`

Returns a number between `0` and `1` that specifies the percentage of the video that the player shows as buffered. This method returns a more reliable number than the now-deprecated `getVideoBytesLoaded` (#getVideoBytesLoaded) and `getVideoBytesTotal` (#getVideoBytesTotal) methods.

`player.getPlayerState():Number`

Returns the state of the player. Possible values are:

- `-1` – unstarted
- `0` – ended
- `1` – playing
- `2` – paused
- `3` – buffering
- `5` – video cued

`player.getCurrentTime():Number`

Returns the elapsed time in seconds since the video started playing.

`player.getVideoStartBytes():Number`

Deprecated as of October 31, 2012. Returns the number of bytes the video file started loading from. (This method now always returns a value of `0`.) Example scenario: the user seeks ahead to a point that hasn't loaded yet, and the player makes a new request to play a segment of the video that hasn't loaded yet.

`player.getVideoBytesLoaded():Number`

Deprecated as of July 18, 2012. Instead, use the `getVideoLoadedFraction` (#getVideoLoadedFraction) method to determine the percentage of the video that has buffered.

This method returns a value between `0` and `1000` that approximates the amount of the video that has been loaded. You could calculate the fraction of the video that has been loaded by dividing the `getVideoBytesLoaded` value by the `getVideoBytesTotal` value.

`player.getVideoBytesTotal():Number`

Deprecated as of July 18, 2012. Instead, use the `getVideoLoadedFraction` (#getVideoLoadedFraction) method to determine the percentage of the video that has buffered.

Returns the size in bytes of the currently loaded/playing video or an approximation of the video's size.

This method always returns a value of `1000`. You could calculate the fraction of the

video that has been loaded by dividing the `getVideoBytesLoaded` value by the `getVideoBytesTotal` value.

Retrieving video information

`player.getDuration():Number`

Returns the duration in seconds of the currently playing video. Note that `getDuration()` will return `0` until the video's metadata is loaded, which normally happens just after the video starts playing.

If the currently playing video is a live event (/youtube/2.0/developers_guide_protocol_retrieving_live_events), the `getDuration()` function will return the elapsed time since the live video stream began. Specifically, this is the amount of time that the video has streamed without being reset or interrupted. In addition, this duration is commonly longer than the actual event time since streaming may begin before the event's start time.

`player.getVideoUrl():String`

Returns the YouTube.com URL for the currently loaded/playing video.

`player.getVideoEmbedCode():String`

Returns the embed code for the currently loaded/playing video.

Retrieving playlist information

`player.getPlaylist():Array`

This function returns an array of the video IDs in the playlist as they are currently ordered. By default, this function will return video IDs in the order designated by the playlist owner. However, if you have called the `setShuffle` (#setShuffle) function to shuffle the playlist order, then the `getPlaylist()` function's return value will reflect the shuffled order.

`player.getPlaylistIndex():Number`

This function returns the index of the playlist video that is currently playing.

- If you have not shuffled the playlist, the return value will identify the position where the playlist creator placed the video. The return value uses a zero-based index, so a value of `0` identifies the first video in the playlist.

- If you have shuffled the playlist, the return value will identify the video's order within the shuffled playlist.

Adding or removing an event listener

`player.addEventListener(event:String, listener:String):Void`

Adds a listener function for the specified `event`. The Events (#Events) section below identifies the different events that the player might fire. The listener is a string that specifies the function that will execute when the specified event fires.

`player.removeEventListener(event:String, listener:String):Void`

Removes a listener function for the specified `event`. The `listener` is a string that identifies the function that will no longer execute when the specified event fires.

Accessing and modifying DOM nodes

`player.getIframe():Object`

This method returns the DOM node for the embedded `<iframe>`.

`player.destroy():Void`

Removes the `<iframe>` containing the player.

Events

The API fires events to notify your application of changes to the embedded player. As noted in the previous section, you can subscribe to events by adding an event listener when constructing the `YT.Player` object (#Loading_a_Video_Player), and you can also use the `addEventListener` (#addEventListener) function.

The API will pass an event object as the sole argument to each of those functions. The event object has the following properties:

- The event's `target` identifies the video player that corresponds to the event.
- The event's `data` specifies a value relevant to the event. Note that the `onReady` event does not specify a `data` property.

The following list defines the events that the API fires:

onReady

This event fires whenever a player has finished loading and is ready to begin receiving API calls. Your application should implement this function if you want to automatically execute certain operations, such as playing the video or displaying information about the video, as soon as the player is ready.

The example below shows a sample function for handling this event. The event object that the API passes to the function has a `target` property, which identifies the player. The function retrieves the embed code for the currently loaded video, starts to play the video, and displays the embed code in the page element that has an `id` value of `embed-code`.

```
function onPlayerReady(event) {
  var embedCode = event.target.getVideoEmbedCode() (#getVideoEmbedCode);
  event.target.playVideo();
  if (document.getElementById('embed-code')) {
    document.getElementById('embed-code').innerHTML = embedCode;
  }
}
```

onStateChange

This event fires whenever the player's state changes. The `data` property of the event object that the API passes to your event listener function will specify an integer that corresponds to the new player state. Possible values are:

- `-1` (unstarted)
- `0` (ended)
- `1` (playing)
- `2` (paused)
- `3` (buffering)
- `5` (video cued).

When the player first loads a video, it will broadcast an `unstarted` (`-1`) event. When a video is cued and ready to play, the player will broadcast a `video cued` (`5`) event. In your code, you can specify the integer values or you can use one of the following namespaced variables:

- `YT.PlayerState.ENDED`
- `YT.PlayerState.PLAYING`
- `YT.PlayerState.PAUSED`
- `YT.PlayerState.BUFFERING`
- `YT.PlayerState.CUED`

`onPlaybackQualityChange`

This event fires whenever the video playback quality changes. It might signal a change in the viewer's playback environment. See the YouTube Help Center (https://support.google.com/youtube/answer/91449) for more information about factors that affect playback conditions or that might cause the event to fire.

The `data` property value of the event object that the API passes to the event listener function will be a string that identifies the new playback quality. Possible values are:

- `small`
- `medium`
- `large`
- `hd720`
- `hd1080`
- `highres`

`onPlaybackRateChange`

This event fires whenever the video playback rate changes. For example, if you call the `setPlaybackRate(suggestedRate)` (#setPlaybackRate) function, this event will fire if the playback rate actually changes. Your application should respond to the event and should not assume that the playback rate will automatically change when the `setPlaybackRate(suggestedRate)` (#setPlaybackRate) function is called. Similarly, your code should not assume that the video playback rate will only change as a result of an explicit call to `setPlaybackRate`.

The `data` property value of the event object that the API passes to the event listener function will be a number that identifies the new playback rate. The `getAvailablePlaybackRates` (#getAvailablePlaybackRates) method returns a list of the valid playback rates for the currently cued or playing video.

`onError`

This event fires if an error occurs in the player. The API will pass an `event` object to the event listener function. That object's `data` property will specify an integer that identifies the type of error that occurred. Possible values are:

- `2` – The request contains an invalid parameter value. For example, this error occurs if you specify a video ID that does not have 11 characters, or if the video ID contains invalid characters, such as exclamation points or asterisks.
- `5` – The requested content cannot be played in an HTML5 player or another error related to the HTML5 player has occurred.
- `100` – The video requested was not found. This error occurs when a video has been removed (for any reason) or has been marked as private.
- `101` – The owner of the requested video does not allow it to be played in embedded players.
- `150` – This error is the same as `101`. It's just a `101` error in disguise!

`onApiChange`

This event is fired to indicate that the player has loaded (or unloaded) a module with exposed API methods. Your application can listen for this event and then poll the player to determine which options are exposed for the recently loaded module. Your application can then retrieve or update the existing settings for those options.

The following command retrieves an array of module names for which you can set player options:

```
player.getOptions();
```

Currently, the only module that you can set options for is the `captions` module, which handles closed captioning in the player. Upon receiving an `onApiChange` event, your application can use the following command to determine which options can be set for the `captions` module:

```
player.getOptions('captions');
```

By polling the player with this command, you can confirm that the options you want to access are, indeed, accessible. The following commands retrieve and update module options:

```
Retrieving an option:
player.getOption(module, option);

Setting an option
player.setOption(module, option, value);
```

The table below lists the options that the API supports:

Module	Option	Description
captions	fontSize	This option adjusts the font size of the captions displayed in the player. Valid values are `-1`, `0`, `1`, `2`, and `3`. The default size is `0`, and the smallest size is `-1`. Setting this option to an integer below `-1` will cause the smallest caption size to display, while setting this option to an integer above `3` will cause the largest caption size to display.
captions	reload	This option reloads the closed caption data for the video that is playing. The value will be `null` if you retrieve the option's value. Set the value to `true` to reload the closed caption data.

Mobile Considerations

Autoplay and Scripted Playback

The HTML5 `<video>` element, in certain mobile browsers (such as Chrome and Safari), only allows playback to take place if it's initiated by a user interaction (such as tapping on the player). Here's an excerpt from Apple's documentation (https://developer.apple.com/library/safari/#documentation/AudioVideo/Conceptual/Using_HTML5_Audio_Video/AudioandVideoTagBasics/AudioandVideoTagBasics.html):

ng: To prevent unsolicited downloads over cellular networks at the user’s expense, embedded media canr automatically in Safari on iOS — the user always initiates playback."

Due to this restriction, functions and parameters such as `autoplay`, `playVideo()`, `loadVideoById()` won't work in all mobile environments.

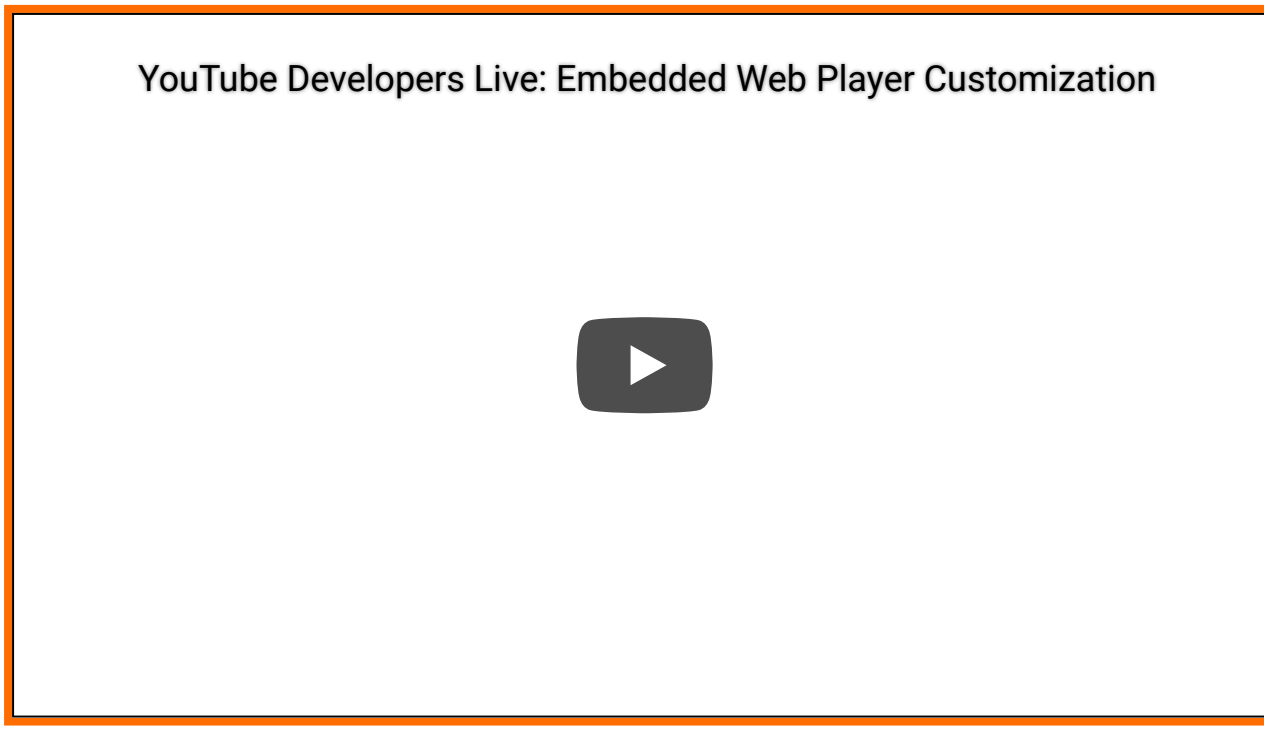
Examples

Creating `YT.Player` objects

- **Example 1: Use API with existing <iframe>**

  In this example, an `<iframe>` element on the page already defines the player with which the API will be used. Note that either the player's `src` URL must set the `enablejsapi` (/youtube/player_parameters#enablejsapi) parameter to `1` or the `<iframe>` element's `enablejsapi` attribute must be set to `true`.

  The `onPlayerReady` function changes the color of the border around the player to orange when the player is ready. The `onPlayerStateChange` function then changes the color of the border around the player based on the current player status. For example, the color is green when the player is playing, red when paused, blue when buffering, and so forth.



  This example uses the following code:

```
<iframe id="existing-iframe-example"
        width="640" height="360"
        src="https://www.youtube.com/embed/M7lc1UVf-VE?enablejsapi=1"
```

```
        frameborder="0"
        style="border: solid 4px #37474F"
></iframe>

<script type="text/javascript">
  var tag = document.createElement('script');
  tag.id = 'iframe-demo';
  tag.src = 'https://www.youtube.com/iframe_api';
  var firstScriptTag = document.getElementsByTagName('script')[0];
  firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

  var player;
  function onYouTubeIframeAPIReady() {
    player = new YT.Player('existing-iframe-example', {
        events: {
          'onReady': onPlayerReady,
          'onStateChange': onPlayerStateChange
        }
    });
  }
  function onPlayerReady(event) {
    document.getElementById('existing-iframe-example').style.borderColor
  }
  function changeBorderColor(playerStatus) {
    var color;
    if (playerStatus == -1) {
      color = "#37474F"; // unstarted = gray
    } else if (playerStatus == 0) {
      color = "#FFFF00"; // ended = yellow
    } else if (playerStatus == 1) {
      color = "#33691E"; // playing = green
    } else if (playerStatus == 2) {
      color = "#DD2C00"; // paused = red
    } else if (playerStatus == 3) {
      color = "#AA00FF"; // buffering = purple
    } else if (playerStatus == 5) {
      color = "#FF6DOO"; // video cued = orange
    }
    if (color) {
      document.getElementById('existing-iframe-example').style.borderColo
    }
  }
  function onPlayerStateChange(event) {
    changeBorderColor(event.data);
  }
</script>
```

- **Example 2: Loud playback**

  This example creates a 1280px by 720px video player. The event listener for the `onReady` event then calls the `setVolume` (#setVolume) function to adjust the volume to the highest setting.

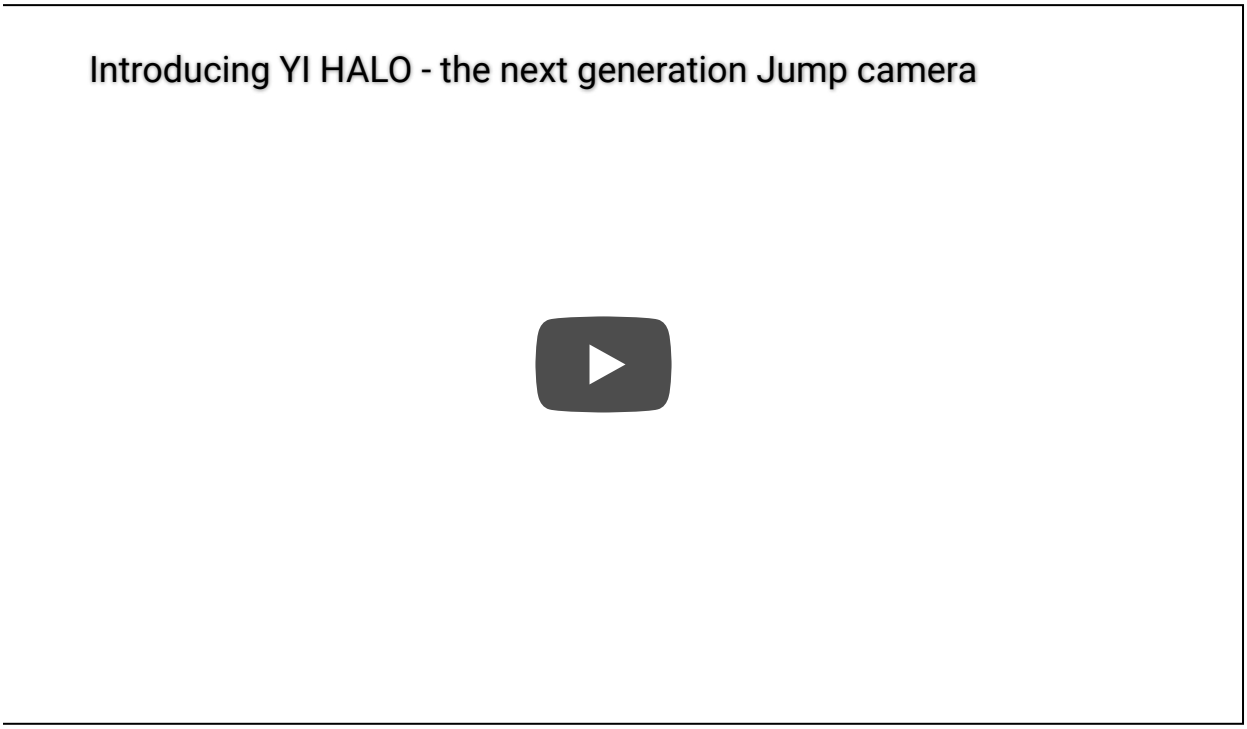
```
function onYouTubeIframeAPIReady() {
  var player;
  player = new YT.Player('player', {
    width: 1280,
    height: 720,
    videoId: 'M7lc1UVf-VE',
    events: {
      'onReady': onPlayerReady,
      'onStateChange': onPlayerStateChange,
      'onError': onPlayerError
    }
  });
}

function onPlayerReady(event) {
  event.target.setVolume(100);
  event.target.playVideo();
}
```

- **Example 3:** This example sets player parameters to automatically play the video when it loads and to hide the video player's controls. It also adds event listeners for several events that the API broadcasts.

```
function onYouTubeIframeAPIReady() {
  var player;
  player = new YT.Player('player', {
    videoId: 'M7lc1UVf-VE',
    playerVars: { 'autoplay': 1, 'controls': 0 },
    events: {
      'onReady': onPlayerReady,
      'onStateChange': onPlayerStateChange,
      'onError': onPlayerError
    }
  });
}
```

Controlling 360° videos



: pitch: roll: fov: Update properties

This example uses the following code:

```
<style>
  .current-values {
    color: #666;
    font-size: 12px;
  }
</style>
<!-- The player is inserted in the following div element -->
<div id="spherical-video-player"></div>

<!-- Display spherical property values and enable user to update them. -->
<table style="border: 0; width: 640px;">
  <tr style="background: #fff;">
    <td>
      <label for="yaw-property">yaw: </label>
      <input type="text" id="yaw-property" style="width: 80px"><br>
      <div id="yaw-current-value" class="current-values"> </div>
    </td>
    <td>
      <label for="pitch-property">pitch: </label>
```

```
      <input type="text" id="pitch-property" style="width: 80px"><br>
      <div id="pitch-current-value" class="current-values"> </div>
    </td>
    <td>
      <label for="roll-property">roll: </label>
      <input type="text" id="roll-property" style="width: 80px"><br>
      <div id="roll-current-value" class="current-values"> </div>
    </td>
    <td>
      <label for="fov-property">fov: </label>
      <input type="text" id="fov-property" style="width: 80px"><br>
      <div id="fov-current-value" class="current-values"> </div>
    </td>
    <td style="vertical-align: bottom;">
      <button id="spherical-properties-button">Update properties</button>
    </td>
  </tr>
</table>

<script type="text/javascript">
  var tag = document.createElement('script');
  tag.id = 'iframe-demo';
  tag.src = 'https://www.youtube.com/iframe_api';
  var firstScriptTag = document.getElementsByTagName('script')[0];
  firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

  var PROPERTIES = ['yaw', 'pitch', 'roll', 'fov'];
  var updateButton = document.getElementById('spherical-properties-button');

  // Create the YouTube Player.
  var ytplayer;
  function onYouTubeIframeAPIReady() {
    ytplayer = new YT.Player('spherical-video-player', {
        height: '360',
        width: '640',
        videoId: 'FAtdv94yzp4',
    });
  }

  // Don't display current spherical settings because there aren't any.
  function hideCurrentSettings() {
    for (var p = 0; p < PROPERTIES.length; p++) {
      document.getElementById(PROPERTIES[p] + '-current-value').innerHTML = '
    }
  }

  // Retrieve current spherical property values from the API and display them
  function updateSetting() {
```

```
    if (!ytplayer || !ytplayer.getSphericalProperties) {
      hideCurrentSettings();
    } else {
      let newSettings = ytplayer.getSphericalProperties();
      if (Object.keys(newSettings).length === 0) {
        hideCurrentSettings();
      } else {
        for (var p = 0; p < PROPERTIES.length; p++) {
          if (newSettings.hasOwnProperty(PROPERTIES[p])) {
            currentValueNode = document.getElementById(PROPERTIES[p] +
                                                       '-current-value');
            currentValueNode.innerHTML = ('current: ' +
                newSettings[PROPERTIES[p]].toFixed(4));
          }
        }
      }
    }
    requestAnimationFrame(updateSetting);
  }
  updateSetting();

  // Call the API to update spherical property values.
  updateButton.onclick = function() {
    var sphericalProperties = {};
    for (var p = 0; p < PROPERTIES.length; p++) {
      var propertyInput = document.getElementById(PROPERTIES[p] + '-property')
      sphericalProperties[PROPERTIES[p]] = parseFloat(propertyInput.value);
    }
    ytplayer.setSphericalProperties(sphericalProperties);
  }
</script>
```

Revision history

April 27, 2021

The Getting Started (#Getting_Started) and Loading a Video Player (#Loading_a_Video_Player) sections have been updated to include examples of using a **`playerVars`** object to customize the player.

October 13, 2020

Note: This is a deprecation announcement for the embedded player functionality that lets you configure the player to load search results. This announcement affects the IFrame Player API's queueing functions for lists (#Playlist_Queueing_Functions), **`cuePlaylist`** (#cuePlaylist) and **`loadPlaylist`** (#loadPlaylist).

This change will become effective on or after 15 November 2020. After that time, calls to the `cuePlaylist` or `loadPlaylist` functions that set the `listType` property to `search` will generate a `4xx` response code, such as `404` (`Not Found`) or `410` (`Gone`). This change also affects the `list` property for those functions as that property no longer supports the ability to specify a search query.

As an alternative, you can use the YouTube Data API's `search.list` (/youtube/v3/docs/search/list) method to retrieve search results and then load selected videos in the player.

October 24, 2019

The documentation has been updated to reflect the fact that the API no longer supports functions for setting or retrieving playback quality. As explained in this YouTube Help Center article (https://support.google.com/youtube/answer/91449), to give you the best viewing experience, YouTube adjusts the quality of your video stream based on your viewing conditions.

The changes explained below have been in effect for more than one year. This update merely aligns the documentation with current functionality:

- The `getPlaybackQuality`, `setPlaybackQuality`, and `getAvailableQualityLevels` functions are no longer supported. In particular, calls to `setPlaybackQuality` will be no-op functions, meaning they will not actually have any impact on the viewer's playback experience.
- The queueing functions for videos and playlists -- `cueVideoById` (#cueVideoById), `loadVideoById` (#loadVideoById), etc. -- no longer support the `suggestedQuality` argument. Similarly, if you call those functions using object syntax, the `suggestedQuality` field is no longer supported. If `suggestedQuality` is specified, it will be ignored when the request is handled. It will not generate any warnings or errors.
- The `onPlaybackQualityChange` (#onPlaybackQualityChange) event is still supported and might signal a change in the viewer's playback environment. See the Help Center article (https://support.google.com/youtube/answer/91449) referenced above for more

information about factors that affect playback conditions or that might cause the event to fire.

May 16, 2018

The API now supports features that allow users (or embedders) to control the viewing perspective for 360° videos (#Spherical_Video_Controls):

- The `getSphericalProperties` (#getSphericalProperties) function retrieves the current orientation for the video playback. The orientation includes the following data:
  - **yaw** - represents the horizontal angle of the view in degrees, which reflects the extent to which the user turns the view to face further left or right
  - **pitch** - represents the vertical angle of the view in degrees, which reflects the extent to which the user adjusts the view to look up or down
  - **roll** - represents the rotational angle (clockwise or counterclockwise) of the view in degrees.
  - **fov** - represents the field-of-view of the view in degrees, which reflects the extent to which the user zooms in or out on the video.
- The `setSphericalProperties` (#setSphericalProperties) function modifies the view to match the submitted property values. In addition to the orientation values described above, this function supports a Boolean field that indicates whether the IFrame embed should respond to `DeviceOrientationEvents` on supported mobile devices.

This example (#Example_Control_Spherical_Videos) demonstrates and lets you test these new features.

June 19, 2017

This update contains the following changes:

- Documentation for the YouTube Flash Player API and YouTube JavaScript Player API has been removed and redirected to this document. The deprecation announcement (http://youtube-eng.blogspot.com/2015/01/youtube-now-defaults-to-html5_27.html) for the Flash and JavaScript players was made on January 27, 2015. If you haven't done so already, please migrate your applications to use IFrame embeds and the IFrame Player API.

August 11, 2016

This update contains the following changes:

- The newly published YouTube API Services Terms of Service ("the Updated Terms"), discussed in detail on the YouTube Engineering and Developers Blog (http://youtube-eng.blogspot.com/), provides a rich set of updates to the current Terms of Service. In addition to the Updated Terms (/youtube/terms/api-services-terms-of-service), which will go into effect as of February 10, 2017, this update includes several supporting documents to help explain the policies that developers must follow.

  The full set of new documents is described in the revision history for the Updated Terms (/youtube/terms/revision-history). In addition, future changes to the Updated Terms or to those supporting documents will also be explained in that revision history. You can subscribe to an RSS feed listing changes in that revision history from a link in that document.

June 29, 2016

This update contains the following changes:

- The documentation has been corrected to note that the `onApiChange` (#onApiChange) method provides access to the `captions` module and not the `cc` module.

June 24, 2016

The Examples (#Examples) section has been updated to include an example that demonstrates how to use the API with an existing `<iframe>` element.

January 6, 2016

The `clearVideo` function has been deprecated and removed from the documentation. The function no longer has any effect in the YouTube player.

December 18, 2015

European Union (EU) laws require that certain disclosures must be given to and consents obtained from end users in the EU. Therefore, for end users in the European Union, you must comply with the EU User Consent Policy (http://www.google.com/about/company/user-consent-policy.html). We have added a notice of this requirement in our YouTube API Terms of Service (/youtube/terms#notices-to-users).

April 28, 2014

This update contains the following changes:

- The new removeEventListener (#removeEventListener) function lets you remove a listener for a specified event.

March 25, 2014

This update contains the following changes:

- The Requirements (#Requirements) section has been updated to note that embedded players must have a viewport that is at least 200px by 200px. If a player displays controls, it must be large enough to fully display the controls without shrinking the viewport below the minimum size. We recommend 16:9 players be at least 480 pixels wide and 270 pixels tall.

July 23, 2013

This update contains the following changes:

- The Overview (#Overview) now includes a video of a 2011 Google I/O presentation that discusses the iframe player.

October 31, 2012

This update contains the following changes:

- The Queueing functions (#Queueing_Functions) section has been updated to explain that you can use either argument syntax or object syntax to call all of those functions. Note that the API may support additional functionality in object syntax that the argument syntax does not support.

  In addition, the descriptions and examples for each of the video queueing functions (#Video_Queueing_Functions) have been updated to reflect the newly added support for object syntax. (The API's playlist queueing functions (#Playlist_Queueing_Functions) already supported object syntax.)

- When called using object syntax, each of the video queueing functions (#Video_Queueing_Functions) supports an **endSeconds** property, which accepts a float/integer and specifies the time when the video should stop playing when `playVideo()` (#playVideo) is called.

- The `getVideoStartBytes` (#getVideoStartBytes) method has been deprecated. The method now always returns a value of `0`.

August 22, 2012

This update contains the following changes:

- The example in the Loading a video player (#Loading_a_Video_Player) section that demonstrates how to manually create the `<iframe>` tag has been updated to include a closing `</iframe>` tag since the `onYouTubeIframeAPIReady` function is only called if the closing `</iframe>` element is present.

August 6, 2012

This update contains the following changes:

- The Operations (#Operations) section has been expanded to list all of the supported API functions rather than linking to the JavaScript Player API Reference (/youtube/js_api_reference) for that list.
- The API supports several new functions and one new event that can be used to control the video playback speed:
  - **Functions**
    - `getAvailablePlaybackRates` (#getAvailablePlaybackRates) – Retrieve the supported playback rates for the cued or playing video. Note that variable playback rates are currently only supported in the HTML5 player.
    - `getPlaybackRate` (#getPlaybackRate) – Retrieve the playback rate for the cued or playing video.
    - `setPlaybackRate` (#setPlaybackRate) – Set the playback rate for the cued or playing video.
  - **Events**
    - `onPlaybackRateChange` (#onPlaybackRateChange) – This event fires when the video's playback rate changes.

July 19, 2012

This update contains the following changes:

- The new `getVideoLoadedFraction` (#getVideoLoadedFraction) method replaces the now-deprecated `getVideoBytesLoaded` (#getVideoBytesLoaded) and `getVideoBytesTotal` (#getVideoBytesTotal) methods. The new method returns the percentage of the video that the player shows as buffered.
- The `onError` (#onError) event may now return an error code of `5`, which indicates that the requested content cannot be played in an HTML5 player or another error related to the HTML5 player has occurred.
- The Requirements (#Requirements) section has been updated to indicate that any web page using the IFrame API must also implement the `onYouTubeIframeAPIReady` function. Previously, the section indicated that the required function was named `onYouTubePlayerAPIReady`. Code samples throughout the document have also been updated to use the new name.

  Note: To ensure that this change does not break existing implementations, both names will work. If, for some reason, your page has an `onYouTubeIframeAPIReady` function and an `onYouTubePlayerAPIReady` function, both functions will be called, and the `onYouTubeIframeAPIReady` function will be called first.
- The code sample in the Getting started (#Getting_Started) section has been updated to reflect that the URL for the IFrame Player API code has changed to `http://www.youtube.com/iframe_api`. To ensure that this change does not affect existing implementations, the old URL (`http://www.youtube.com/player_api`) will continue to work.

July 16, 2012

This update contains the following changes:

- The Operations (#Operations) section now explains that the API supports the `setSize()` and `destroy()` methods. The `setSize()` method sets the size in pixels of the `<iframe>` that contains the player and the `destroy()` method removes the `<iframe>`.

June 6, 2012

This update contains the following changes:

- We have removed the `experimental` status from the IFrame Player API.
- The Loading a video player (#Loading_a_Video_Player) section has been updated to point out that when inserting the `<iframe>` element that will contain the YouTube player, the

IFrame API replaces the element specified in the constructor for the YouTube player. This documentation change does not reflect a change in the API and is intended solely to clarify existing behavior.

In addition, that section now notes that the insertion of the `<iframe>` element could affect the layout of your page if the element being replaced has a different display style than the inserted `<iframe>` element. By default, an `<iframe>` displays as an `inline-block` element.

March 30, 2012

This update contains the following changes:

- The Operations (#Operations) section has been updated to explain that the IFrame API supports a new method, `getIframe()`, which returns the DOM node for the IFrame embed.

March 26, 2012

This update contains the following changes:

- The Requirements (#Requirements) section has been updated to note the minimum player size.

Last updated 2021-04-27 UTC.