## Requirements

Delay

2

Updated my answer to a better one using youtube player API for more option and dynamic output. It has many option to select and you could easily control it using Javascript. I hope this could help. Credits to @wasikuss posted answer.

// Load the IFrame Player API code asynchronously.

setTimeout(function() {

player.playVideo();

}, 20000);

var tag = document.createElement('script');

tag.src = "https://www.youtube.com/player\_api";

var firstScriptTag = document.getElementsByTagName('script')[0];

firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

// Replace the 'ytplayer' element with an <iframe> and

// YouTube player after the API code downloads.

var player;

function onYouTubePlayerAPIReady() {

player = new YT.Player('ytplayer', {

height: '315',

width: '560',

videoId: 'iejOAPyooXs'

});

}

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',  
          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](https://developers.google.com/youtube/iframe_api_reference#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](https://developers.google.com/youtube/iframe_api_reference#Loading_a_Video_Player) section explains how to do so.

1. 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).
2. 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.
3. 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.
4. 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',

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.

1. 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](https://developers.google.com/youtube/player_parameters.html?playerVersion=HTML5) 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](https://developers.google.com/youtube/iframe_api_reference#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](https://developers.google.com/youtube/iframe_api_reference#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://example.com"

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](https://developers.google.com/youtube/iframe_api_reference#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](https://developers.google.com/youtube/iframe_api_reference#Getting_Started) and [Loading a video player](https://developers.google.com/youtube/iframe_api_reference#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 search results or a user's list of 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](https://developers.google.com/youtube/iframe_api_reference" \l "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});

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 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](https://developers.google.com/youtube/iframe_api_reference" \l "onPlaybackRateChange) event will fire, and your code should respond to the event rather than the fact that it called the setPlaybackRate function.  
  
The [getAvailablePlaybackRates](https://developers.google.com/youtube/iframe_api_reference" \l "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.
* 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()**](https://developers.google.com/youtube/iframe_api_reference#getVideoEmbedCode);
* event.target.playVideo();
* if (document.getElementById('embed-code')) {
* document.getElementById('embed-code').innerHTML = embedCode;
* }
* }

## Eg.

## 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](https://developers.google.com/youtube/player_parameters" \l "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 = '#FF6D00';  
  }  
  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.borderColor = color;  
    }  
  }  
  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](https://developers.google.com/youtube/iframe_api_reference" \l "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  
    }  
  });  
}

Delay

// Load the IFrame Player API code asynchronously.

setTimeout(function() {

player.playVideo();

}, 20000);

var tag = document.createElement('script');

tag.src = "https://www.youtube.com/player\_api";

var firstScriptTag = document.getElementsByTagName('script')[0];

firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

// Replace the 'ytplayer' element with an <iframe> and

// YouTube player after the API code downloads.

var player;

function onYouTubePlayerAPIReady() {

player = new YT.Player('ytplayer', {

height: '315',

width: '560',

videoId: 'iejOAPyooXs'

});

}