



Spark API & advanced features guide

Introduction

Spark offers several advanced features and APIs. There are three types of APIs, Web APIs - available for web pages when including Spark javascript, Customer Remote APIs - provide partial Spark functionality in cases where javascript/SDK cannot be integrated, and App SDK APIs - used for Spark features on native mobile, OTT or SmartTV Apps.

This is a live document that is constantly updated with new features and information. As such, section order does not imply importance.

Do not hesitate to contact Spark support. We are available at Skype:holaspark.com or on support@holaspark.com.

Introduction	1
1. Spark Widgets	4
1.1 Embeddable Popular Videos Widget	4
1.2 Embedded Playlist Tiles Widget	7
2. Spark Web APIs	9
2.1 Feeding Watch Next recommendations via API	9
2.2 Controlling how Spark script is loaded	11
2.3 Custom video preview generation	12
2.4 Spark external events	15
2.5 Setting up a video search box	18
2.6 Manual trigger of Watch Later popup	18
2.7 Spark debug & runtime configuration options	19
3. Native apps - Spark Remote APIs	22
3.1 Generating Video Previews API	22
3.1.1 Overview	22
3.1.2 Obtaining a preview link	22
3.2 Generating Thumbnails APIs	24
3.3 Creating a Video Library file	27
3.4 Purge APIs	27

3.4.1 Spark media purge	27
3.4.2 Spark CDN Video purge	28
4. iOS Notifications with video previews	29
4.1 Local notifications - no Spark SDK	29
4.2 Remote notifications - no SDK	32
4.3 Local notifications - Integrated SparkSDK	37
4.4 Remote notifications - Integrated SparkSDK	40
5. App SDK APIs	45
5.1 Basic Spark Player	46
5.1.1 Spark Player supported API	46
5.1.2 Ad support via Google IMA	47
Android Example:	47
iOS Example:	48
5.1.3 360 Video playback	48
5.2 Spark Library	49
5.2.1 VPAID Ad Protocol	49
5.2.2 Floating Player	49
5.2.3 Video Thumbnails	49
5.1.7 Video Previews	50
5.2.4 Watch Next	50
5.2.5 TV Casting	50
5.2.6 Position Memory	51
5.2.7 My Videos Panel	51
5.3 WebView based apps	52
5.3.1 Fullscreen mode	52
6. Tracking Spark with Google Analytics	53
6.1 Enabling Google Analytics	53
6.1.1 'Use defined tracking code' checkbox	54
6.1.2 'Web property ID' edit box	54
6.1.3 'Per-domain properties'	54
6.2 Viewing analytics	54
7. Accelerated Mobile Pages	58
7.1 Spark with Brightcove AMP	58
8. Hosting the Spark on your CDN	61
8.1 Hosting the Spark JS code on your CDN	61
8.1.1 Instructions	61
8.2 Hosting Spark Video Previews on your CDN	63



9. Spark editor tools	64
9.1 Overview	64
9.2 Customizing Video Previews	65
9.2.1 Editing specific Video Previews	68
9.2.2 Blocking specific Video Previews	70
9.2.3 Purging specific Video Previews	71
9.3 Customizing Watch Next recommendations	72
9.4 Selecting Video Previews to Auto-Play	75
9.5 Downloading Video Previews	77
9.6 Editor tools without Chrome extension (Beta)	79
9.6.1 Known limitations	79
9.7 Troubleshooting	79
10. Custom Spark code for homepage	80

1. Spark Widgets

1.1 Embeddable Popular Videos Widget

The Popular videos carousel shows the top 5 videos by page views for the last 1 day, and looks like this:



In most cases, if Spark code is on your page, you can enable a demo of the widget by appending '?spark_show_previews_widget=1' to the page URL.

In order to use this widget you need to first create a `div` into which our widget will construct the content into. The `div` can be styled in any way you wish. Create the widget using:

```
spark_web.previews_widget(div, opt)
```

Where:

`div` - any DOM element to insert widget into

`opt` :

`pages_num` - number of popular videos to get. Default: 5

`site_logo` - video to show on widget initialization. You should provide video url in any [supported video format](#) (mp4, webm)



`interval` - seconds to show each video. Default: 6

`text_direction` - text direction. Provide `rtl` (right-to-left) or `ltr` (left-to-right).

Default: autodetect

`number_color` - number color. Default: `#fff`

`number_bg_color` - background color of number. Default: `#0142ad`

`title_color` - title color. Default: `#fff`

`title_bg_color` - background color of title. Default: `#000`

`progress_color` - progress bar color. Default: `#fff`

`json_url` - configure url that points to a JSON file with a static list of pages to promote

Example of use

```
spark_web.previews_widget(div,  
{  
    pages_num: 7,  
    site_logo: 'https://example.com/logo.mp4',  
    interval: 10,  
    text_direction: 'rtl',  
    progress_color: '#ccc'  
})
```

You can also configure the widget from the control panel:

The screenshot shows the Spark control panel interface. On the left is a sidebar with various settings categories: 'CDN', 'Video preview' (which is selected), 'Image preview', 'Video search', 'Welcome message', 'Floating player', 'Player thumbnails', 'YouTube-like controls', 'TV video casting', 'Spark Player', 'Watch next', 'Watch later', 'Page sharing', 'Position memory', and 'Player auto play'. The main area displays the 'Video preview' settings. At the top, there are tabs for 'Desktop', 'Mobile web', and 'Mobile App'. Below these are several sections: 'Admin settings', 'Activation rules', 'Appearance', 'Generation settings' (with a size of 480x270), 'Links selector' (customized by Spark support), 'Unite img and link', and 'Add link tags'. The 'Popular Videos widget' section is highlighted with a red border. It contains a description: 'This embeddable widget shows your site's most popular videos in a TV news headline format. See the [documentation](#).' Below this is a 'Source' dropdown set to 'Auto'. The 'Splash video' section has a text input field with the placeholder 'Short video to display first - e.g. https://example.com/logo.mp4'. The 'Colors' section includes color pickers for 'Page title' (set to #ffffff), 'Title background' (set to #000000), 'Number' (set to #ffffff), 'Number background' (set to #0142ad), and 'Progress bar' (set to #ffffff). At the bottom, there is a 'Popular videos rules' section with a 'No rules' button.



Setting 'Source' option to 'URL' will allow you to configure a url that points to a JSON file with a static list of pages to promote. The previews of the videos in these pages will be displayed.

Popular Videos widget

This embeddable widget shows your site's most popular videos in a TV news headline format. See the [documentation](#).

Source

URL

URL

<https://myvid.com/list.json>

The JSON format of this file is

```
[{page_url, poster, title},{...}]
```

Page url: link to the promoted page.

Poster: poster to use if preview is not ready, if omitted the auto extracted poster will be used

Title: Text to show over the preview, if omitted the auto extracted text will be used

Further more you can add multiple rules and JSON urls if you want different format and content in different sites or areas in your site.

Popular Videos widget

This embeddable widget shows your site's most popular videos in a TV news headline format. See the [documentation](#).

Source

URL

URL

<https://myvid.com/list.json>

Splash video

Short video to display first - e.g. <https://example.com/logo.mp4>

Colors

Page title

#ffffff

Title background

#000000

Number

#ffffff

Number background

#0142ad

Progress bar

#ffffff

Popular videos rules

#1

Match URL

all

Source

Auto

Splash video

Short video to display first - e.g. <https://example.com/logo.mp4>

Colors

Page title

#ffffff

Title background

#000000

Number

#ffffff

Number background

#0142ad

Progress bar

#ffffff

1.2 Embedded Playlist Tiles Widget

The Popular Videos Widget shows the top videos by page views for the last week/day, and looks like this:



In order to use the Popular Videos Widget you need to first create a `div` into which our widget will construct the content into. The `div` can be styled in any way you wish.

The API is called using:

```
spark_web.playlist_widget(div, opt)
```

If not providing `opt` parameter the default behavior is to try to fill the `div` with as many tiles of the most popular videos (popular in last week - each of size 190x130)

`div` - any DOM element to insert widget into

`opt` :

`type` - `popular` or `trending`. `popular` - videos most viewed in past week, `trending` - videos most viewed in past day

`rows` - hardcode how many rows you want to be inserted or choose auto fill. Default: `'auto'`

`cols` - the same as `rows` but for columns

`size` :



`width` - tile width in pixels

`height` - tile height in pixels

Example of use

```
spark_web.playlist_widget(div,  
{  
  type: 'trending',  
  rows: 4,  
  cols: 5,  
  size: {width: 220, height: 150}  
})
```



2. Spark Web APIs

Web based APIs are used within html5 web pages to configure and use Spark features in special ways that cannot be configured via the Spark control panel.

2.1 Feeding Watch Next recommendations via API

Important! Spark can use recommendations which already exist on your page by scraping your page . Simply let Spark support know which of your existing recommendations you want to appear in Spark. This method does not require any development on your part.

You can send your own content recommendations to Spark's Watch Next feature via API.

In order to create a manual playlist for watch next you can add to your player instance
`<player>.hola_playlist`

When the time comes to display next video, Spark code will search for this object on the player instance before using its default behavior.

Structure of `hola_playlist`

```
<player>.hola_playlist = {playlist: [{item1}, {item2}, ...],  
  <options>};  
  
itemX: {description, poster, video_url|page_url}
```

Set `video_url` if you want video to be played inline in current player

Set `page_url` if you want to navigate to next video page

If both are set playlist will navigate to next video page

Example:

```
{  
  description: 'Travel green, take the train',  
  poster: 'http://player.h-cdn.org/static/ve_trains_poster.png',  
  url: 'http://holaspark.com/demo/train',  
}
```



`<options>` can be:

`loop - true` to play the playlist in a continuous loop

`tile - true` - show the playlist as tiled thumbnails at end of video

`autoplay` - number of seconds to autoplay the next video



Example

```
<player>.hola_playlist = {playlist: [
{
  description: 'Travel green, take the train',
  poster: 'http://player.h-cdn.org/static/ve_trains_poster.png',
  url: 'http://holaspark.com/demo/train',
},
{
  description: 'High adrenaline sports',
  poster: 'http://player.h-cdn.org/static/ve_sports_poster.png',
  url: 'http://holaspark.com/demo/sports',
},
{
  description: 'The bicycle is now 200 years old',
  poster: 'http://player.h-cdn.org/static/ve_bikes_poster.png',
  url: 'http://holaspark.com/demo/bikes',
}], autoplay: 20, loop: true};
```

2.2 Controlling how Spark script is loaded

We know that page load times are critical for getting the best user experience, rating high on Google's performance metrics and pushing the ads to users as quickly as possible for conversion. As such Spark allows full control on its loading process and timing as follows.

When you add the one line Spark js loading line per our installation instructions you actually only load a small 2k script. The full sized script is loaded later according to selected preferences.

Control is provided by setting `window.ev_spark_load` to one of these possible values before the script load line.

Value	Description
-------	-------------



'page_load'	<u>This is the default value.</u> Full script will load when browser emits 'load' event after the initial html and all its dependencies have finished loading.
'cpu_idle'	Full script will load on browser idle callback <code>window.requestIdleCallback</code> or after 10s if hasnt happened yet.
'external'	Full script load will wait on event ' <code>external_spark_load</code> ' that should be emitted by your code at the timing of your choice. Also set <code>window.external_spark_load = true</code> So that our code will not miss the event if loaded after
<ms>	Full script will load after <ms> milliseconds

IMPORTANT: if you enabled Spark CDN which offloads video streaming from your existing CDN, then full script will load immediately regardless of the choices above in order to attach to playing video as early as possible. If you still want to delay loading of the full script contact Spark support to check that your setup is compatible with these options and they will enable it for you.



2.3 Spark external events

Spark features generate events that can be tracked. These events can be used for logging, statistics, A/B testing, etc.

Note: This feature requires Spark configuration option `enable_events` be set on init. It can be set using [Spark runtime configuraiton options](#) or by requesting support to enable it for you in your persistent configuration.

Spark events can be accessed in 4 methods most via the `spark_web` global object

1. Get a summary of event stats by calling `spark_web.event_stats()`

Console example:

```
> spark_web.event_stats()
Spark event stats
{"preview_play":16,"preview_played":13,"preview_click":1}
< undefined
```

2. Access per event details by reviewing `spark_web.events` array

Console example:

```
> spark_web.events
< (13) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}]
  0: {event: "preview_play", frame: "http://video.foxnews.com/static/orion/html/video/iframe/vod.html?v=20180629121
  1: {event: "preview_click", frame: "http://video.foxnews.com/static/orion/html/video/iframe/vod.html?v=2018062912
  2: {event: "preview_played", frame: "http://video.foxnews.com/static/orion/html/video/iframe/vod.html?v=201806291
  3: {event: "preview_play", link: "http://video.foxnews.com/v/5805362295001/president-trumps-approval-ratings-risi
  4:
    event: "preview_played"
    is_watch_next: false
    link: "http://video.foxnews.com/v/5805362295001/president-trumps-approval-ratings-rising-with-republicans"
    played_ms: 142.699951171875
    __proto__: Object
  5: {event: "preview_play", link: "http://video.foxnews.com/v/5805380040001/outnumbered-panelists-on-what-makes-an
  6: {event: "preview_played", played_ms: 1000.099853515625, link: "http://video.foxnews.com/v/5805380040001/outnun
  7: {event: "preview_play", link: "http://video.foxnews.com/v/5805380040001/outnumbered-panelists-on-what-makes-an
  8: {event: "preview_played", played_ms: 16.199951171875, link: "http://video.foxnews.com/v/5805380040001/outnumbe
  9: {event: "preview_play", link: "http://video.foxnews.com/v/5804438668001/abby-huntsman-on-whs-2-big-objectives-
  10: {event: "preview_played", played_ms: 769.800048828125, link: "http://video.foxnews.com/v/5804438668001/abby-t
```



3. By listening to spark events

Listen for events via `spark_web` object:

all: `spark_web.on('spark_any', function(e){...})`

Specific: `spark_web.on('spark_preview_play', function(data){...})`

Listen for events via global `addEventListener`:

All: `window.addEventListener('spark_any', function(e){...})`

Specific:

`window.addEventListener('spark_preview_play', function(data){...})`

Console examples:

```
> spark_web.on('any', function(i){ console.log('Spark event '+JSON.stringify(i, null, 2)); })
< undefined
Spark event {
  "event": "preview_play",
  "link": "http://video.foxnews.com/v/5804425576001/williams-lingering-resentment-from-dems-on-merrick-garland",
  "is_watch_next": false,
  "is_autoplay": false
}
Spark event {
  "event": "preview_played",
  "played_ms": 4459.199951171875,
  "link": "http://video.foxnews.com/v/5804425576001/williams-lingering-resentment-from-dems-on-merrick-garland",
  "is_watch_next": false
}
> |

> window.addEventListener('spark_any', function(e){ console.log('Global Spark event '+e.type+' info '+JSON.stringify(e.detail, null, 2)); })
< undefined
Global Spark event spark_any info {
  "event": "preview_play",
  "link": "http://video.foxnews.com/v/5804425576001/williams-lingering-resentment-from-dems-on-merrick-garland",
  "is_organic": true,
  "is_autoplay": false
}
Global Spark event spark_any info {
  "event": "preview_played",
  "played_ms": 858,
  "link": "http://video.foxnews.com/v/5804425576001/williams-lingering-resentment-from-dems-on-merrick-garland",
  "is_organic": true
}
> |
```

You can listen for spark events on the top frame and receive events from all the frames on the page or listen for events on a specific frame and receive only events from that frame. If you are listening on the top frame and receive an event from a frame then a 'frame' attribute will be added to the event data with the link of the frame where it came from.

Using `addEventListener` you can even listen to events on a specific element in the page. When using `addEventListener` the received object in the callback function is an 'event' where `event.type` is the event name and `event.detail` holds all the info as described below.

```
window.addEventListener('spark_any', function(e){ console.log('Global Spark
```



```
event '+e.type+' info '+JSON.stringify(e.detail, null, 2)); })
```

These are the events that can be listened for in Spark

Event	Description
'spark_any'	Listen for all Spark events. ' event ' attribute is added to data received to identify the specific event
'spark_preview_visible'	Called when element with preview becomes visible
'spark_preview_play'	Called when preview starts playing
'spark_preview_played'	Called when preview stops playing
'spark_preview_click'	Called when preview is clicked either by user or autoclick feature

The following data is received together with the events.

Name	Description
played_ms	duration of preview played in milliseconds in last activity
link	page link this preview refers to
is_organic	is this preview attached to an organic thumbnail element on the page or comes from Spark features (watch next, video search, preview widgets, etc)
is_autoplay	Was this preview autoplayed or triggered by user hover
is_autoclick	For 'preview_click' event - True if click was generated by autoclick feature timer running out, false if user manually clicked the preview
played	For 'preview_click' event - how many times has the preview played until now
playing	For 'preview click' event - is the preview currently playing

frame	Frame url if listening for events from top frame and event came from inside a sub frame of the page

2.4 Setting up a video search box

This API is applicable for sites who want to integrate YouTube-like Video Search into the site structure.

```
spark_web.video_search(query_text, div)
```

This call can be connected to any button click and the results will be fed into the provided `div`.

2.5 Manual trigger of Watch Later popup

This feature is deprecated!

This API allows any web element to trigger the appearance of the Watch Later pop-in box. You can create any type of button or element you choose for your site, and have it call:

```
spark_web.watch_later.show_popup();
```

This will display the dialog and return:

- "true" if the Watch Later dialog was shown
- "false" if there are no items to display, in which case the dialog will not be shown.



2.6 Spark debug & runtime configuration options

Its possible to enable/disable some Spark features via an in page configuration JSON or by adding parameters to the page url that will be read by Spark javascript when it is loaded and will behave accordingly.

Global JSON example:

```
window.spark_conf = {  
  enable_events: true,  
  debug: true,  
};
```

For using these options as url params you have to add a 'spark_' prefix to the option names.

URL param example:

http://holaspark.com/playground?spark_enable_events=true

These are the support configuration options

Option	Description
disable	Disable Spark on current page. Useful for testing, see FAQ
on	If Spark is disabled from the control panel, enabled Spark on the current page. Useful for testing.
enable_events	Start triggering events for Spark features as described in Spark external events
debug	Enable Spark debug mode. Will start printing verbose logging into console and visual debug aids on your page
ve_image_preview	Enable feature Image preview
ve_playlist	Enable feature Watch Next
ve_preview	Enable feature Video preview



video_autoplay	Enable feature Autoplay
video_search or my_videos	Enable feature Video search
persistent_video	Enable feature Floating player
show_previews_widget	Displays Popular Videos Widget for demo



How to test a feature by an url param?

- enable the feature in Control Panel to have a feature-code inside loader.js
- disable feature platforms (mobile/desktop) to have disabled feature on production

The screenshot shows the Spark Control Panel interface. On the left, the 'Spark Master Control' section is visible. Under the 'Better discovery' category, the 'Video preview' feature is enabled (toggle is on). Under the 'Better playback' category, the 'Desktop' and 'Mobile web' platforms are disabled (toggles are off). Red arrows point from the text 'Enable feature' to the 'Video preview' toggle and from 'Disable platforms' to the 'Desktop' and 'Mobile web' toggles.

- add a param to url to enable feature (i.e. ?spark_ve_preview=1)



3. Native apps - Spark Remote APIs

Note: The original Spark SDK has been discontinued, as it required the use of Spark video player which was not feasible for many customers. Support for native apps is now done only via APIs described below.

3.1 Generating Video Previews API

This API is applicable for web or native apps scenarios where our controlling javascript is not present or for server to server interactions that require Video Preview generation. The API gives native app developers complete control over how they will use Spark Video Previews in their app.

3.1.1 Overview

The process has three parts:

- Signup to Spark and obtain your customer id at https://holaspark.com/?need_signup=1
- Obtain preview link via Spark previews remote API
- Display the video preview on the image thumbnail inside your app/web page when user hovers over it (for web page) or thumbnails reaches mid screen area (for apps)

3.1.2 Obtaining a preview link

This process is done in two stages:

Obtaining the preview link for your video

Send http request to '[http://holaspark-\[customerID\].h-cdn.com/api/get_previews](http://holaspark-[customerID].h-cdn.com/api/get_previews)' with POST payload data constructed as

```
{items: [<vid1>, <vid2>, vid3>]}  
<vidX> = {type: 'video', url: <video url/manifest>}
```




Response will be in JSON format in the following structure :

```
{<video1>: <data1>, <video2>: <data2> .... }

videoX = <video url/manifest> taken from request
dataX = {url: <preview url>, cdns: [cdn1, cdns2, cdns3]}
cdnX = {host: <ip>, hostname: <domain>}
```

'data.url' = preview uri path

'data.cdns' = array of possible sources for this preview

Full preview url is built from <cdn>+<path uri>

Example Request:

```
# curl -v 'http://holaspark-exampleid.h-cdn.com/api/get_previews' -X POST
--data
'{"items":[{"type":"video","url":"https://holaspark-vod.com/movie3/playlist
.m3u8"}]}' -H 'Content-Type: application/json'
```

Example Response:

```
{"https://holaspark-vod.com/movie3/playlist.m3u8?token=mXuRH":{
  "url":"/preview.mp4?customer=exampleid&url=https%3A%2F%2Fholaspark-vod.com%
  2Fmovie3%2Fplaylist.m3u8&preview_ver=1_1",
  "cdns":[{"host":"95.141.32.92","hostname":"zagent11.h-cdn.com"}, {"host":"14
  4.217.79.16","hostname":"zagent857.h-cdn.com"}, {"host":"147.135.222.144","h
  ostname":"zagent871.h-cdn.com"}]}
```

You can choose any of the sources to obtain the preview itself. From the above example a possible full url would be

https://zagent11.h-cdn.com/preview.mp4?customer=exampleid&url=https%3A%2F%2Fholaspark-vod.com%2Fmovie3%2Fplaylist.m3u8&preview_ver=1_1



note: if you get response status 503 (busy) then retry. This is not an indication of a problem - it is normal, since Spark uses multiple servers for redundancy, you can simply ask another server

Acquiring the preview itself:

Assuming you got a preview url as described in previous step you call it to receive the preview file. Response can be as follows

Http status 200

Preview is ready and returned with content-type video/mp4

Http status 503

This source is busy, retry or switch to another source

Http status 404

Returned with type application/json and data in the format of {status: <status text>}

This means preview file is not ready yet - status text will give you a hint on the stage it is in.

'download queue full': currently preview generation queue is full, will be retried on next request

'downloading': downloading video chunks needed for preview generation

'generating': preview is being generated from data

3.2 Generating Thumbnails APIs

This API is applicable for web or native apps scenarios where our controlling javascript is not present or for server to server interactions that require thumbnails generation. It is also useful if you wish to use Spark generated Thumbnails inside your own video player.

First of all make sure you have a customer id. If not then signup to Spark and obtain your id at https://holaspark.com/?need_signup=1

Obtaining the thumbnails urls

Obtaining the thumbnails urls is done in three steps



Find the thumbnails location

`http://holaspark-<customerid>.h-cdn.com/api/get_thumb_info?customer=<id>&url=<valid video_url>`

Example:

```
# curl -v  
'https://holaspark-<customerid>.h-cdn.com/api/get_thumb_info?customer=<id>&  
url=https%3A%2F%2Fvod.cdn.net%2Fvod%2Fplaylist.m3u8'
```

Response can be

Http status 503

This source is busy, you can immediately retry

Http status 200

Successful response with content-type application/json in the following structure:

```
{"master": "zagent1674", "status": "ready"}
```

Get the thumbnails meta information

Using the source provided in previous response resend the request using this source.

```
# curl -v  
'https://zagent1674.h-cdn.com/api/get_thumb_info?customer=<id>&url=https%3A%  
2F%2Fvod.cdn.net%2Fvod%2Fplaylist.m3u8'
```

Response can be

Http status 503

This source is busy, you can immediately retry

Http status 200

Successful response with content-type application/json in the following structure:



{urls: [url1, url2..], cdns: [cdn1, cdn2..], info: <meta data>, VER: <thumbnails version>}

Example

```
{ "urls": ["/get_thumb/<customerid>/gen/vod.cdn.net/vod/playlist.m3u8hola&thumb_id=1", "/get_thumb/<customerid>/gen/vod.cdn.net/vod/playlist.m3u8&hola&thumb_id=2", "/get_thumb/ettodaynet/gen/vod.cdn.net/vod/playlist.m3u8hola&thumb_id=3", "/get_thumb/<customerid>/gen/vod.cdn.net/vod/playlist.m3u8&hola&thumb_id=4"], "cdns": [{ "host": "54.37.85.232", "hostname": "zagent1674.h-cdn.com" }, { "host": "54.36.176.166", "hostname": "zagent1666.h-cdn.com" }, { "host": "144.217.79.5", "hostname": "zagent856.h-cdn.com" } ], "info": { "group_size": 25, "width": 160, "height": 90, "count": 78, "interval": 2 }, "VER": "4" }
```

urls - array of thumbnail paths

cdns - array of sources that can obtain the thumbnails,

info - information on thumbnail generation

width/height - dimensions of thumbnails

count - number of thumbnails for entire video

interval - sec between thumbnail snapshots

group_size - max thumbnails per group url - last one usually has less

VER - internal version of thumbnails creation process

Get a thumb group by selecting the relevant url and adding the host from cdns

Example

```
# curl -v  
'http://144.217.79.5/get_thumb/ettodaynet/gen/vod-ettoday.cdn.hinet.net/ettoday-vod/_definst_/smil:mbrouput/169/114989/114989.smil/playlist.m3u8?token=Nbe_FKpyiI1omagRvEFjKA&expires=1510748506&hola&thumb_id=3'
```



3.3 Creating a Video Library file

Spark can automatically identify much of the video metadata like duration, poster, title, most popular page it appears in, category, language, and more.

However, its not always enough. If we want to add special markers, like content tags, show-season-episode categorization, age restrictions, etc or make sure that metadata is 100% correct at all times it is possible to define a complete video library.

3.4 Purge APIs

3.4.1 Spark media purge

/api/purge_media

Purge video related media: preview, thumbnails and image

Params:

- type: <preview|thumbnails> - If type is set to '*' it will purge both
- video_url: Accepts asterisk notation. If url is set to '*' it will purge all mappings and media

Example:

```
# curl -v
'https://client.h-cdn.com/api/purge_media?customer=sparkdemo&key=5fh1c4jzd11i3w48h333&type=preview&video_url=http%3A%2F%2Fvideo.h-cdn.com%2Fstatic%2Fmp4%2Fve_video_1_x3msports.mp4'
```

/api/purge_metadata

Purge map of video to page including associated poster and description

Params:

- page_url: <url>|all - using 'all' purges metadata for all pages on site
- with_media - for purging also the media itself (video previews, thumbnails, image previews)

Example:



```
# curl -v  
'https://client.h-cdn.com/api/purge_metadata?customer=sparkdemo&key=5fh1c4jz  
zd11i3w48h333&page_url=http%3A%2F%2Fholaspark.com%2F&with_media=true'
```

3.4.2 Spark CDN Video purge

/api/dmca

Purge video from all cdn servers

Params:

- url: url of video
- block: add url to blacklist so it will not be cached again
- silent: don't add changelog
- remark: comment for changelog

Example

```
# curl -v  
'https://client.h-cdn.com/api/dmca?customer=sparkdemo&key=5fh1c4jzd11i3w48h  
333&url=http%3A%2F%2Fvideo.h-cdn.com%2Fstatic%2Fmp4%2Fve_video_1_x3msports.  
mp4&block=true&remark=blocked%20copyright%20content'
```



4. iOS Notifications with video previews

Generating notifications with Spark video previews can be done in two methods:

- **Without integrating Spark SDK:** Using Spark Remote APIs to generate the previews and following the instructions below to create the special notifications with attached video previews.
- **By integrating Spark SDK** into your app and continuing to generate your regular notifications, video previews will be automatically added to notifications with links that contain videos.
-

4.1 Local notifications - no Spark SDK

For notifications generated by the App itself.

Generating video preview mapping

Without SparkSDK you need to call our WEB API to obtain your video previews.

Refer to [section 2.1](#) for details on how to use WEB API to generate and obtain video previews.

Creating notifications

NOTE: in the following code samples, we assume you already have preview URLs for your videos, and use it for notification attachments.

By default, `UNMutableNotificationContent` allows adding attachments that are located on device local storage only, so you need to extend its functionality with capability for remote attachments as well.

The following is a Swift example how to implement it in few simple steps:

- download the remote resource to local storage
- use resulting local URL to add an attachment to the notification
- execute the callback when completed



```
// Swift example
extension UNMutableNotificationContent {
    func addRemoteAttachment(_ url: URL, onComplete: @escaping ()->()) {
        let config = URLSessionConfiguration.default
        let session = URLSession(configuration: config)
        let request = URLRequest(url: url)
        let task = session.downloadTask(with: request){
            (location, response, error) in
            if let location = location, error==nil {
                if (response as? HTTPURLResponse)?.statusCode==200 {
                    let locationWithExt = location.deletingPathExtension().
                        appendingPathExtension(url.pathExtension)
                    do {
                        try FileManager.default.moveItem(at: location,
                            to: locationWithExt)
                        if let attachment = try? UNNotificationAttachment(
                            identifier: "remote", url: locationWithExt) {
                            self.attachments = [attachment]
                            onComplete()
                            return
                        }
                    }
                    catch (_){}
                }
            }
            onComplete()
        }
        task.resume()
    }
}
```




Now creating a push notification with remote content is as simple as the following:

```
// Swift example
func sendNotification(preview: URL, title: String, body: String)
{
    let content = UNMutableNotificationContent()
    content.categoryIdentifier = "spark-preview"
    content.title = "Watch"
    content.body = "Dani Alves gets kicked out after shouting at referee..."
    content.sound = UNNotificationSound.default()
    content.addRemoteAttachment(url: preview) {
        let category = UNNotificationCategory(identifier: "spark-preview", actions: [],
            intentIdentifiers: [], options: [])
        let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 10, repeats: false)
        let request = UNNotificationRequest(identifier: "demo", content: content,
            trigger: trigger)
        let center = UNUserNotificationCenter.current()
        center.setNotificationCategories([category])
        center.add(request, withCompletionHandler: { (error) in
            if let error = error {
                print("notification failed", error)
            } else {
                print("notification sent")
            }
        })
    }
}
```

Customizing notification view

For better user experience, we recommend to autoplay/loop the video preview once user opens up the notification into a “long-view”, but this is not offered by default. Instead iOS offers means to create a custom notification view, where you take the control of the notification look-and-feel and its behavior.

First, create custom notification target in your XCode project (File > New > Target > Notification Content Extension). It will add a new component folder to your project with `Info.plist`, `MainInterface.storyboard` and `NotificationViewController.swift` files in it.

Configure `Info.plist` with your personal category identifier, it will route your notifications to this custom extension. Go to Information Property List > NSExtensions > NSExtensionAttributes > `UNNotificationExtensionCategory` and change it to your identifier (e.g. `spark-preview`). Make sure to use the same value you used in creating a notification.

```
let content = UNMutableNotificationContent()
content.categoryIdentifier = "spark-preview"
```



And finally update `didReceive` method of `NotificationViewController` class:

```
// Swift example
func didReceive(_ notification: UNNotification) {
    if let attachment = notification.request.content.attachments.first {
        let asset = AVAsset(url: attachment.url)
        let item = AVPlayerItem(asset: asset)
        let player = AVQueuePlayer()
        let looper = AVPlayerLooper(player: player, templateItem: item)
        let layer = AVPlayerLayer(player: player)
        layer.videoGravity = .resizeAspect
        layer.frame = CGRect(x: 0, y: 0, width: self.view.frame.size.width,
                             height: self.view.frame.size.height)
        let observable = layer.observe(\.videoRect, options: [.new]) { (model, change) in
            if (layer.videoRect.size.width==0 ||
                layer.videoRect.size.width==layer.frame.size.width &&
                layer.videoRect.size.height==layer.frame.size.height)
            {
                return
            }
            // adjust layer dimensions based on video ratio
            layer.frame = CGRect(x: 0, y: 0, width: layer.videoRect.size.width,
                                 height: layer.videoRect.size.height)
            self.view.frame = CGRect(x: 0, y: 0, width: layer.videoRect.size.width,
                                     height: layer.videoRect.size.height)
        }
        self.view.layer.addSublayer(layer)
        player.play()
        self.looper = looper // keep the reference, otherwise looping won't work
        self.observable = observable // keep the reference for proper dealloc
    }
}
```

Note, in this example `notification.request.content` is an instance of `UNMutableNotificationContent` created in previous steps.

4.2 Remote notifications - no SDK

For notifications created remotely, not via the APP itself

Generating video preview mapping

Without SparkSDK you need to call our WEB API to obtain your video previews.

Refer to [section 2.1](#) for details on how to use WEB API to generate and obtain video previews.



Sending remote notification

NOTE: in the following code samples, we assume you already have preview URLs for your videos, and use it for notification attachments.

Use the following JSON as a notification payload to send remote notification to the app:

```
{
  "aps": {
    "alert": {
      "title": "Watch",
      "body": "Dani Alves gets kicked out after shouting at referee..."
    },
    "category": "spark-preview",
    "sound": "ping.aiff",
    "mutable-content": 1
  },
  "attachment-url": "<preview url>"
}
```

Here is an example how to do it using `node-apn` project:

<https://github.com/node-apn/node-apn>

```
var note = new apn.Notification();
note.expiry = Math.floor(Date.now() / 1000) + 3600; // expires 1 hour from now
note.sound = "ping.aiff";
note.category = "spark-preview";
note.alert = {
  title: "Watch",
  body: "Dani Alves gets kicked out after shouting at referee...",
};
note.payload = {"attachment-url": "<preview url>"};
note.topic = "<your-app-bundle-id>";
note.contentAvailable = 1;
note.mutableContent = 1;
apnProvider.send(notification, deviceToken).then(function(result) {
  console.log(result);
});
```

Note that remote notifications are not allowed to contain any attachments, but we can add arbitrary data to the payload (`attachment-url` in example above). As a next step, your app must handle this arbitrary data and add missing attachment to the notification before it is shown to the user. This is achieved using Notification Service Extension explained in the next section.

Intercepting remote notifications



Create a custom notification service extension in your XCode project (File > New > Target > Notification Service Extension). It will add a new component folder to your project with `Info.plist` and `NotificationService.swift` files in it.

By default, `UNMutableNotificationContent` allows adding attachments that are located on device local storage only, so you need to extend its functionality with capability for remote attachments as well. The following is a Swift example how to implement it in few simple steps:

- download the remote resource to local storage
- use resulting local URL to add an attachment to the notification
- execute the callback when completed

```
// Swift example
extension UNMutableNotificationContent {
    func addRemoteAttachment(_ url: URL, oncomplete: @escaping ()->()) {
        let config = URLSessionConfiguration.default
        let session = URLSession(configuration: config)
        let request = URLRequest(url: url)
        let task = session.downloadTask(with: request){
            (location, response, error) in
            if let location = location, error==nil {
                if (response as? HTTPURLResponse)?.statusCode==200 {
                    let locationWithExt = location.deletingPathExtension().
                        appendingPathExtension(url.pathExtension)
                    do {
                        try FileManager.default.moveItem(at: location,
                            to: locationWithExt)
                        if let attachment = try? UNNotificationAttachment(
                            identifier: "remote", url: locationWithExt) {
                            self.attachments = [attachment]
                            oncomplete()
                            return
                        }
                    }
                    catch (_){}
                }
            }
            oncomplete()
        }
        task.resume()
    }
}
```

Now update `didReceive` method of `NotificationService` class to add attachment sent along arbitrary data of remote notification. It will make use of custom `addRemoteAttachment` extension method introduced above:



```
// Swift example
override func didReceive(_
    request: UNNotificationRequest,
    withContentHandler contentHandler: @escaping (UNNotificationContent) -> Void)
{
    self.contentHandler = contentHandler
    self.bestAttemptContent = (request.content.mutableCopy() as?
    UNMutableNotificationContent)
    if let bestAttemptContent = self.bestAttemptContent {
        guard let url = bestAttemptContent.userInfo["attachment-url"] as? String
            else { return }
        bestAttemptContent.addRemoteAttachment(URL(string: url)!) {
            contentHandler(bestAttemptContent)
        }
    }
}
```

Customizing notification view

For better user experience, we recommend to autoplay/loop the video preview once user opens up the notification into a “long-view”, but this is not offered by default. Instead iOS offers means to create a custom notification view, where you take the control of the notification look-and-feel and its behavior.

First, create custom notification target in your XCode project (File > New > Target > Notification Content Extension). It will add a new component folder to your project with `Info.plist`, `MainInterface.storyboard` and `NotificationViewController.swift` files in it.

Configure `Info.plist` with your personal category identifier, it will route your notifications to this custom extension. Go to Information Property List > NSExtensions > NSExtensionAttributes > UNNotificationExtensionCategory and change it to your identifier (e.g. `spark-preview`). Make sure to use the same value you used in creating remote notification JSON.

```
{
  "aps": {
    "category": "spark-preview",
    ...
  },
}
```



And finally update `didReceive` method of `NotificationViewController` class:

```
// Swift example
func didReceive(_ notification: UNNotification) {
    if let attachment = notification.request.content.attachments.first {
        let asset = AVAsset(url: attachment.url)
        let item = AVPlayerItem(asset: asset)
        let player = AVQueuePlayer()
        let looper = AVPlayerLooper(player: player, templateItem: item)
        let layer = AVPlayerLayer(player: player)
        layer.videoGravity = .resizeAspect
        layer.frame = CGRect(x: 0, y: 0, width: self.view.frame.size.width,
                             height: self.view.frame.size.height)
        let observable = layer.observe(\.videoRect, options: [.new]) { (model, change) in
            if (layer.videoRect.size.width==0 ||
                layer.videoRect.size.width==layer.frame.size.width &&
                layer.videoRect.size.height==layer.frame.size.height)
            {
                return
            }
            // adjust layer dimensions based on video ratio
            layer.frame = CGRect(x: 0, y: 0, width: layer.videoRect.size.width,
                                 height: layer.videoRect.size.height)
            self.view.frame = CGRect(x: 0, y: 0, width: layer.videoRect.size.width,
                                     height: layer.videoRect.size.height)
        }
        self.view.layer.addSublayer(layer)
        player.play()
        self.looper = looper // keep the reference, or looping won't work
        self.observable = observable // keep the reference for dealloc
    }
}
```



4.3 Local notifications - Integrated SparkSDK

Installation

Using CocoaPods: add the following line to your Podfile:

```
pod 'SparkLib', '~> 1.1'
```

Manual installation:

- Download our latest library from the github: [SparkLib](#)
- Copy **SparkLib.framework** to your project's folder, e.g.:

```
<myapproot>
  <myapp>
  <myapp>.xcodeproj
  libs
    SparkLib.framework
```
- Add new files to your XCode project from Project Navigator
- Open your app configuration page
- Switch to **"Build Phases"** > **"Link Binary With Libraries"** > "+" > **"Add other"**
- Select **SparkLib.framework**

Initialization

Initialize Spark SDK with your customer id and register for notifications within iOS Notification Center:

```
// Objective-C example
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)options
{
    SparkAPI *api = [SparkAPI getAPI:@"<customer_id>"];
    [api registerForNotifications:
        UNAuthorizationOptionAlert|UNAuthorizationOptionSound
        usingRemoteNotifications:NO
        withCompletionBlock:^(NSError *error){
            if (error)
                NSLog(@"registering for notifications failed: error=%@", error);
        }];
    ...
}

// Swift example
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool
{
    let api = SparkAPI.getAPI("<customer_id>")
    api.register(forNotifications: [.alert, .sound],
        usingRemoteNotifications: false) { (error) in
```



```
        if (error)
            print("registering for notifications failed: error=", error!)
    }
    ...
}
```

NOTE: providing customer id is required only for the first provisioning call, in all subsequent calls from all around your project you can omit it and use like this:

```
// Objective-C example
- (IBAction)onSomeButtonClicked:(UIButton *)sender
{
    SparkAPI *api = [SparkAPI getAPI:nil];
    ...
}

// Swift example
@IBAction func onSomeButtonClicked(sender: UIButton)
{
    let api = SparkAPI.getAPI(nil)
    ...
}
```

Send local notifications

Spark will automatically download a preview for your video from our servers and attach it to the notification payload:

```
// Objective-C example
[api sendPreviewNotification:
 [NSURL URLWithString:@"https://yourcdn.com/path_to_video/video.m3u8"]
 withTitle:@"Watch!" withSubtitle:nil
 withBody:@"Dani Alves gets kicked out after shouting at referee..."
 withTriggerOn:[UNTimeIntervalNotificationTrigger
    triggerWithTimeInterval:10 repeats:false]
 withBeforeSendBlock:^(UNMutableNotificationContent *content,
    UNNotificationsSettings *settings)
 {
    // use this block to customize the notification payload before
    // sending it out to the notification center
    if (settings.soundSetting==UNNotificationSettingEnabled)
        content.sound = [UNNotificationSound soundNamed:@"custom_sound.aiff"];
    return YES;
 }
 withCompletionBlock:^(NSError *error){
    if (error)
        NSLog(@"local notification failed, error=%@", error);
 }
```




```
    }];

// Swift example
api.sendPreviewNotification(
    URL(string: "https://yourcdn.com/path_to_video/video.m3u8"),
    withTitle: "Watch", withSubtitle: nil,
    withBody: "Dani Alves gets kicked out after shouting at referee...",
    withTriggerOn: UNTimeIntervalNotificationTrigger(timeInterval: 10,
        repeats: false),
    withBeforeSend: { (content, settings) -> Bool in
        // use this block to customize the notification payload before
        // sending it out to the notification center
        if (settings.soundSetting == .enabled)
            content.sound = UNNotificationSound(named: "custom_sound.aiff")
        return true
    },
    withCompletionBlock: { (error) in
        if (error)
            print(@"local notification failed, error=", error!);
    })
}
```

Customize notification preview with autoplay/looping (better UX)

For better user experience, we recommend to autoplay/loop the video preview once user opens up the notification into a “long-view”, but this is not offered by default. Instead iOS offers means to create a custom notification view, where you take the control of the notification look-and-feel and its behavior.

First, create custom notification target in your XCode project (File > New > Target > Notification Content Extension). It will add a new component folder to your project with `Info.plist`, `MainInterface.storyboard` and `NotificationViewController.*` files in it.

Configure your extension `Info.plist` with the following properties:

(goto `Info.plist` > `NSExtension` > `NSExtensionAttributes` and change its attributes according to:

- **UNNotificationExtensionCategory = "spark-preview"**

it is basically a route that defines which notifications should this extension be applied to

- **UNNotificationExtensionInitialContentSizeRatio = 0.5625**

notification size gets resized automatically based on video ratio, but we assume 16:9 to avoid redundant transformation on initial show

And finally inherit `NotificationViewController` from `Spark` (replace original contents of the file)

```
// Objective-C example

// NotificationViewController.h
```



```
#import <SparkAPI.h>
@interface NotificationViewController: SparkPreviewNotificationViewController
@end

// NotificationViewController.m
#import "NotificationViewController.h"
@implementation NotificationViewController
@end

// Swift example

// NotificationViewController.swift
class NotificationViewController: SparkPreviewNotificationViewController {
}
```

NOTE: Make sure your extension is linked with SparkLib.framework (see installation section).

4.4 Remote notifications - Integrated SparkSDK

Installation

Using CocoaPods: add the following line to your Podfile:

```
pod 'SparkLib', '~> 1.1'
```

Manual installation:

- Download our latest library from the github: [SparkLib](#)
- Copy **SparkLib.framework** to your project's folder, e.g.:

```
<myapproot>
  <myapp>
    <myapp>.xcodeproj
    libs
      SparkLib.framework
```
- Add new files to your XCode project from Project Navigator
- Open your app configuration page
- Switch to **"Build Phases"** > **"Link Binary With Libraries"** > **"+"** > **"Add other"**
- Select **SparkLib.framework**

Initialization

Initialize Spark SDK with your customer id and register for notifications within iOS Notification Center:



```
// Objective-C example
- (BOOL)application:(UIApplication *)application
  didFinishLaunchingWithOptions:(NSDictionary *)options
{
    SparkAPI *api = [SparkAPI getAPI:@"<customer_id>"];
    [api registerForNotifications:
        UNAuthorizationOptionAlert|UNAuthorizationOptionSound
        usingRemoteNotifications:YES
        withCompletionBlock:^(NSError *error){
            if (error)
                NSLog(@"registering for notifications failed: error=%@", error);
        }];
    ...
}

// Swift example
func application(_ application: UIApplication, didFinishLaunchingWithOptions
  launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool
{
    let api = SparkAPI.getAPI("<customer_id>")
    api.register(forNotifications: [.alert, .sound],
        usingRemoteNotifications: true) { (error) in
        if (error)
            print("registering for notifications failed: error=", error!)
        }
    ...
}
```

NOTE: providing customer id is required only for the first provisioning call, in all subsequent calls from all around your project you can omit it and use like this:

```
// Objective-C example
- (IBAction)onSomeButtonClicked:(UIButton *)sender
{
    SparkAPI *api = [SparkAPI getAPI:nil];
    ...
}

// Swift example
@IBAction func onSomeButtonClicked(sender: UIButton)
{
    let api = SparkAPI.getAPI(nil);
    ...
}
```

Send remote notification



You need to setup remote notification server and configure it with deviceToken generated by your app. Implement `didRegisterForRemoteNotificationsWithDeviceToken` delegate method of `UIApplicationDelegate` protocol and provide your remote server with app's token:

```
// Objective-C example
- (BOOL)application:(UIApplication *)application

    didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    // provide your remote notification server with deviceToken
    // to be able to communicate to this app
}

// Swift example
func application(_ application: UIApplication,
    didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data)
{
    // provide your remote notification server with deviceToken
    // to be able to communicate to this app
}
```

Use the following JSON as a notification payload to send remote notification to the app:

```
{
  "aps": {
    "alert": {
      "title": "Watch",
      "body": "Dani Alves gets kicked out after shouting at referee..."
    },
    "category": "spark-preview",
    "sound": "ping.aiff",
    "mutable-content": 1
  },
  "spark-media-url": "<video url for which preview is to be generated>",
  "spark-customer-id": "<customer id>",
}
```

Here is an example how to do it using `node-apn` project:

<https://github.com/node-apn/node-apn>

```
var note = new apn.Notification();
note.expiry = Math.floor(Date.now() / 1000) + 3600; // expires 1 hour from now
note.sound = "ping.aiff";
note.category = "spark-preview";
note.alert = {
  title: "Watch",
  body: "Dani Alves gets kicked out after shouting at referee...",
}
```



```
};
note.payload = {"spark-media-url": "<video url>", "spark-customer-id": "<customer id>"};
note.topic = "<your-app-bundle-id>";
note.mutableContent = 1;
apnProvider.send(notification, deviceToken).then(function(result) {
    console.log(result);
});
```

Note that remote notifications are not allowed to contain any attachments, but we can add arbitrary data to the payload (`attachment-url` in example above). As a next step, your app must handle this arbitrary data and add missing attachment to the notification before it is shown to the user. This is achieved using Notification Service Extension explained in the next section.

Intercept remote notifications

Create a custom notification service extension in your XCode project (File > New > Target > Notification Service Extension). It will add a new component folder to your project with `Info.plist` and `NotificationService.*` files in it.

By default, `UNMutableNotificationContent` allows adding attachments that are located on device local storage only, so the extension must extract `spark-media-url` from notification payload, download preview generated for this video from Spark server and attach it to the notification. All of this is handled by Spark SDK, you only need to inherit your `NotificationService` of your extension from `SparkSDK` class (replace original contents of the files)

```
// Objective-C example
// NotificationService.h
#import <SparkAPI.h>
@interface NotificationService: SparkPreviewNotificationService
@end
// NotificationService.m
#import "NotificationService.h"
@implementation NotificationService
@end

// Swift example

// NotificationService.swift
class NotificationService: SparkPreviewNotificationService { }
```

Customize notification preview with autoplay/looping (better UX)

For better user experience, we recommend to autoplay/loop the video preview once user opens up the notification into a “long-view”, but this is not offered by default. Instead iOS offers means



to create a custom notification view, where you take the control of the notification look-and-feel and its behavior.

First, create custom notification target in your XCode project (File > New > Target > Notification Content Extension). It will add a new component folder to your project with `Info.plist`, `MainInterface.storyboard` and `NotificationViewController.*` files in it.

Configure your extension `Info.plist` with the following properties:

(goto `Info.plist` > `NSExtension` > `NSExtensionAttributes` and change its attributes according to:

- **UNNotificationExtensionCategory = "spark-preview"**

it is basically a route that defines which notifications should this extension be applied to

- **UNNotificationExtensionInitialContentSizeRatio = 0.5625**

notification size gets resized automatically based on video ratio, but we assume 16:9 to avoid redundant transformation on initial show

And finally inherit `NotificationViewController` from `Spark` (replace original contents of the file)

```
// Objective-C example

// NotificationViewController.h
#import <SparkAPI.h>
@interface NotificationViewController: SparkPreviewNotificationViewController
@end

// NotificationViewController.m
#import "NotificationViewController.h"
@implementation NotificationViewController
@end

// Swift example

// NotificationViewController.swift
class NotificationViewController: SparkPreviewNotificationViewController {
}
```

NOTE: Make sure your extension is linked with `SparkLib.framework` (see installation section).



5. App SDK APIs

Spark Player is a wrapper around the native player that adds generic and Spark functionality. This section details which generic API is used by both Android and iOS SDKs and the specific API that was added on top of it.

The SDK has two parts: Basic player and Spark Library.

5.1 Basic Spark Player with sources that includes

[4.1.1 Spark Player supported API](#)

[4.1.2 Ad support via Google IMA](#)

[4.1.3 360 Videos playback](#)

5.2 Spark Library with all the Spark features and full API for gathering stats and receiving events

[4.2.1 VPAID Ad Protocol](#)

[4.2.2 Floating Player](#)

[4.5 Video Thumbnails](#)

[4.6 Video Previews](#)

[4.7 Watch Next](#)

[4.8 Watch Later](#)

[4.9 TV Casting](#)

[4.10 Position Memory](#)

[4.11 My Videos Panel](#)

5.3 WebView based apps - tips and notes about using spark in a WebView based app



5.1 Basic Spark Player

5.1.1 Spark Player supported API

	Android	iOS
Native Player	ExoPlayer2	AVPlayer
Generic stats/events API	Player interface	
Set Spark customer id	<code>void set_customer(String customer)</code>	
Set video url	<code>void load(String url)</code>	
Manually add play item to the queue	<code>void queue(PlayItem item)</code> <code>PlayItem(String tag, String media, String poster)</code>	
Enable/disable fullscreen mode	<code>void fullscreen(Boolean state)</code>	
Enable/disable playback controls	<code>void set_controls_state(boolean enabled)</code>	
Get playback controls state	<code>boolean get_controls_state()</code>	
Show/hide playback controls	<code>void set_controls_visibility(boolean visible)</code>	
Get playback controls visibility	<code>boolean get_controls_visibility()</code>	
Set poster image	<code>void set_poster(String poster_url)</code>	
Get video width in pixels	<code>int get_video_width()</code>	
Get video height in pixels	<code>int get_video_height()</code>	
Uninit player	<code>void uninit()</code>	



5.1.2 Ad support via Google IMA

	Android	iOS
Available in lib	Spark Player Basic	Spark Player Basic
Configuration by	Google ExoPlayer IMA Plugin	

Android Example:

The Spark Player has integrated Google IMA inside. To create an advertisement, one need to add an ad tag to the created PlayItem and pass that PlayItem to the Spark Player . The module supports ad tags from DoubleClick for Publishers, Google AdSense or any other VAST-compliant ad server, and the IMA site provides some [sample ad tags](#) for testing.

Example of usage:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    m_spark_player = findViewById(R.id.float_player);
    m_spark_player.queue(new PlayItem(m_ad_tag,
        m_video_url, m_poster_url, m_title, m_timeout));
}
```

Api call of new PlayItem creation:

```
new PlayItem(@Nullable String ad_tag, @NonNullString video_url,
    @Nullable String poster_url, @Nullable String title, int timeout);

String ad_tag - URL of a VAST-compliant advertisement XML
String video_url - URL of a video (in case of adaptive streaming - of the
    main playlist)
String poster_url - URL of a poster to be displayed when no playback is
```



```
    active
String title - title of the video
int timeout - if greater than zero, it will add a button to a linear
               Unskippable VAST ads, allowing to skip an advertisement in <timeout> ms
```

iOS Example:

First, add Googima framework to your project:

- 1) Using CocoaPods: add googima dependency to your project's Podfile

```
pod 'GoogleAds-IMA-iOS-SDK', '~> 3.7'
```
- 2) Or manually:
 - Download googima framework from <https://developers.google.com/interactive-media-ads/docs/sdks/ios/download>
 - Unpack the archive
 - Select your app target preferences and go to "**Build Phases**" > "**Link Binary With Libraries**" > "+" > "**Add other**"
 - Select `GoogleInteractiveMediaAds.framework` from unpacked folder.

And now setup SparkPlayer with relevant Googima configuration:

```
// Swift example
let sparkPlayer = SparkPlayer(withConfig: [
    "googima": ["adTagUrl": "<your tag url>"],
    ...
])
sparkPlayer.player = AVPlayer(url: "<your video url>")
playerContainer.addSubview(sparkPlayer.view)
```

Done.

5.1.3 360 Video playback

	Android	iOS
Available in lib	Spark Player Basic	Spark Player Basic
Enable/disable 360 mode	<code>void vr_mode(Boolean state)</code>	
Check 360 mode	<code>boolean get_vr_mode()</code> Returns true if 360 mode is active	



5.2 Spark Library

5.2.1 VPAID Ad Protocol

	Android	iOS
Available in lib	Spark Library	Spark Library

5.2.2 Floating Player

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	
Manually force Enable/disable floating player	<code>void float_mode(Boolean state)</code> Note: this still requires that floating player spark feature be enabled in control panel	

5.2.3 Video Thumbnails

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	



5.1.7 Video Previews

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	

5.2.4 Watch Next

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Obtain and display automatic suggestions	
Manually set watch next videos	<code>void set_watch_next_items(PlayListItem[] items)</code> <code>PlayListItem(String video_url, String poster_url, String description)</code>	

5.2.5 TV Casting

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	



5.2.6 Position Memory

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	

5.2.7 My Videos Panel

	Android	iOS
Available in lib	Spark Library	Spark Library
Default by control panel	Enable/Disable for all player instances according to configuration	



5.3 WebView based apps

In WebView based apps you can just add our js script like any regular web page and most of the Spark functionality will just work out of the box. Below are some differences that still apply and require some attention.

5.3.1 Fullscreen mode

Android's WebView doesn't support fullscreen mode by default. To workaround this issue fullscreen emulation has been added to Spark Player. It just stretches the player element on the whole browser window, but android native ui elements (status bar navigation bar) remain visible.

In order to get the real fullscreen mode you need to customize the WebView. The easiest way to do this is to use VideoEnabledWebView class from this project:

<https://github.com/cprcrack/VideoEnabledWebView>



6. Tracking Spark with Google Analytics

Detailed Spark statistics are available in the Spark Control Panel. However, you can get basic stats also straight into your Google Analytics account as **Behavior Events**.

The analytics reports to Google account include these events:

- Video View
- Page View
- Player Autoplay
- Floating player events
- Watch Next: user click
- Watch Next: auto click
- Video Preview: user click
- Video Preview: auto click
- Image Preview: user click
- Image Preview: auto click
- Player Thumbnails: hover
- Watch Later: click
- Video Search: click
- Video History: click

6.1 Enabling Google Analytics

All you need to do to receive these stats in your Google Analytics account is to go to Spark Control panel configuration tab, click on 'General Settings' cogwheel and click on Google



analytics setup box.

Save and apply

Discard changes

There are unsaved modifications: diff

Spark

CDN

Spark Master Control

⚙️

🔴

Zone default

➤

improve discovery

Video preview

🔴

Image preview

⚪

Video search

⚪

Welcome message

⚪

Floating player

🔴

Player thumbnails

🔴

Player settings

Language

Miscellaneous

Google Analytics

👑 Enable

👑 Use defined tracking code

Web property ID: UA-XXXXXXX-Y

?

Per-domain properties +

After enabling the feature you have three methods to configure it.

6.1.1 'Use defined tracking code' checkbox

This allows our code to automatically find the google account related to that page and report the events on this account. So if you have multiple accounts for multiple domains and they are all setup correctly it will just work as is.

6.1.2 'Web property ID' edit box

Define one global google analytics ID for all pages. This will aggregate all Spark stats into one place.

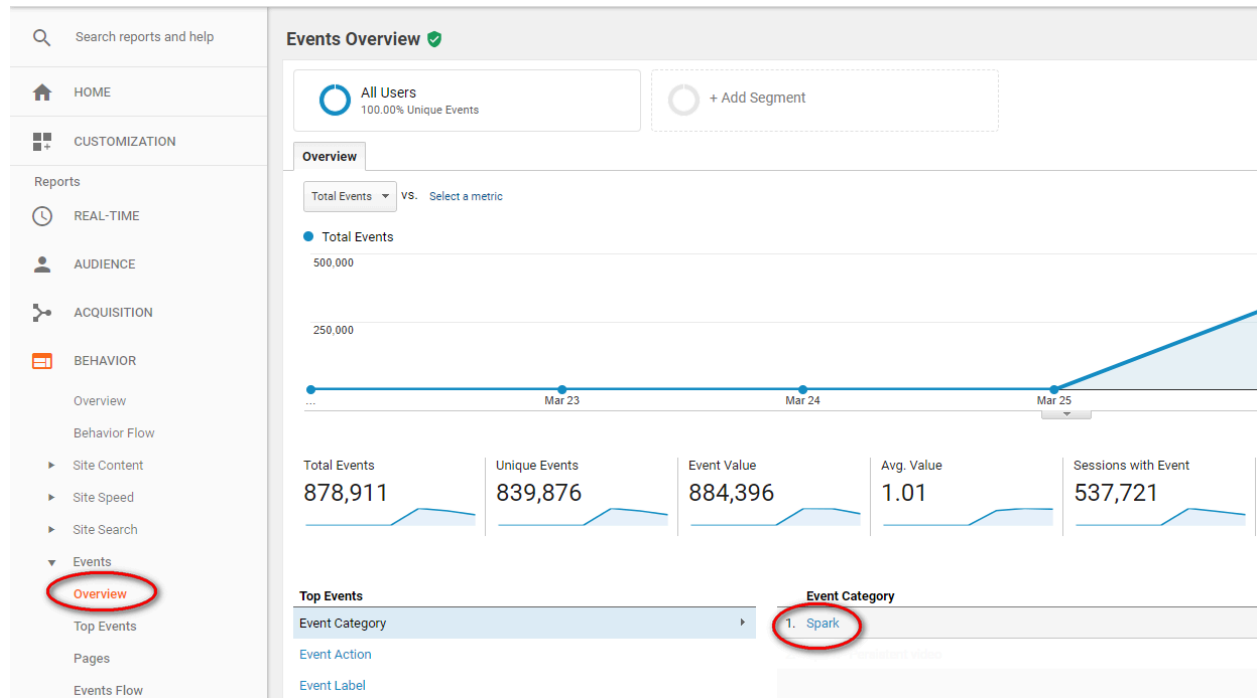
6.1.3 'Per-domain properties'

Here you can define a different google analytics ID per domain. If domain is not defined then global ID will be used. If specific domain not defined and global ID not defined stats will not be reported.

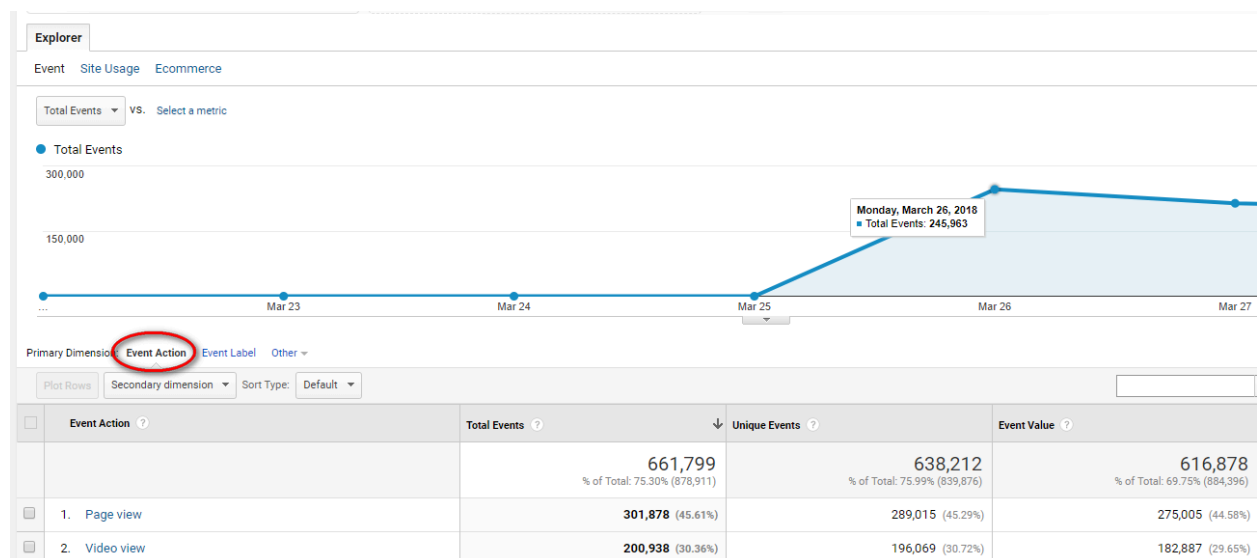
6.2 Viewing analytics

To view Spark events you should enter your google analytics control panel and select Behavior->Events->Overview

Once Overview panel loads select the 'Spark' category



Once inside click 'Event Action' as primary dimension



Then you will see all the different report Spark events. You can choose an event and then add a second dimension to analyze your traffic and clicks.

Note: Keep the primary dimension as 'Event Action'

Primary Dimension: **Event Label** **Other** ▲

Plot Rows Secondary dimension

☐ **Event Label** ?

1. <http://XXXXX.XXX>

2. <http://XXXXXX.XX>

3. <http://XXXXXX.XXX>

4. <http://XXXXX.XXX>

► Acquisition

► Behavior

▼ Events

Event Action ?

Event Category ?

Event Label ?

► Technology

► Users

☐ Display as alphabetical list

Most of the useful slices are under 'Users' subsection in 'Secondary Dimension' dropdown

Primary Dimension: **Event Label** **Event Action** ▼

Plot Rows Secondary dimension Sort Type: Default ▼

☐ **Event Action** ?

1. Page

City ?

City ID ?

Continent ?

Continent ID ?

Count of Sessions ?

Country ?

Country ISO Code ?

Data Source ?

Days Since Last ?

☐ Display as alphabetical list

	Total Events ?	Unique Events ?
	301,878 % of Total: 34.35% (878,911)	289,015 % of Total: 34.41% (839,876)
	301,878(100.00%)	289,015(100.00%)

© 2018 Google | [Analytics Home](#) | [Terms of Service](#) | [Privacy Policy](#) | [Send Feedback](#)



From there you can choose Country, Browser, Device Category and other useful slices.

Example of Country as Second Dimension

Primary Dimension: Event Label Event Action ▾				
Plot Rows Secondary dimension: Country ▾ Sort Type: Default ▾				
<input type="checkbox"/>	Event Action ?	Country ? ✕	Total Events ? ↓	Unique Events ?
			293,572 % of Total: 33.40% (878,911)	281,071 % of Total: 33.47% (839,876)
<input type="checkbox"/>	1. Page view	Taiwan	270,114 (92.01%)	258,581 (92.00%)
<input type="checkbox"/>	2. Page view	Hong Kong	6,380 (2.17%)	6,169 (2.19%)
<input type="checkbox"/>	3. Page view	China	5,043 (1.72%)	4,807 (1.71%)
<input type="checkbox"/>	4. Page view	United States	3,007 (1.02%)	2,892 (1.03%)
<input type="checkbox"/>	5. Page view	Malaysia	1,840 (0.63%)	1,757 (0.63%)
<input type="checkbox"/>	6. Page view	Japan	1,143 (0.39%)	1,093 (0.39%)
<input type="checkbox"/>	7. Page view	Singapore	937 (0.32%)	899 (0.32%)
<input type="checkbox"/>	8. Page view	Australia	895 (0.30%)	863 (0.31%)
<input type="checkbox"/>	9. Page view	Canada	726 (0.25%)	697 (0.25%)
<input type="checkbox"/>	10. Page view	Macau	668 (0.23%)	652 (0.23%)

Choosing 'Label' as Primary Dimension will give you per page statistics



7. Accelerated Mobile Pages

AMP is a Google backed open source initiative to create a high performance website publishing technology for web content and advertisements. It has initially targeted only mobile pages but has since spread to desktop web pages as well.

In general there should not be any issue including Spark JS in an AMP page in the regular manner. Since Spark JS initial load is less than 5KB it has no affect on page load, while the bulk of the Spark code is loaded after page finishes loading and will not affect AMP performance.

Below are instructions of how to add Spark in special third party integration scenarios

7.1 Spark with Brightcove AMP

<https://www.ampproject.org/docs/reference/components/amp-brightcove>

If you are doing a full Brightcove AMP implementation and embedding videos in the following manner then videos are created inside a Brightcove iframe for which you have no access and no regular method to add Spark JS.

```
<amp-brightcove
  data-account="12345"
  data-player="default"
  data-embed="default"
  data-video-id="1234"
  layout="responsive"
  width="480" height="270">
</amp-brightcove>
```

However, Brightcove APIs provide a method to solve this rather easily as follows



Solution:

1. open Brightcove Studio's Players page and choose your player
<https://studio.brightcove.com/products/videocloud/players>

If you wish to experiment you can duplicate your current player and include it on a single page or staging area for testing. When verified do these steps again on your main player.

<input type="checkbox"/>	Name	Updated	Status
<input type="checkbox"/>	Brightcove Default Player default	7/5/2018 3:44 PM	✓
<input type="checkbox"/>			

2. locate the PLUGINS section, and click the Edit button

SOCIAL SHARING Edit

Allow Sharing
No

ENDSCREEN Edit

Endscreen

PLUGINS Edit

- JavaScript0 configured
- CSS0 configured
- Name, Options (JSON)0 configured



3. in the JavaScript area, enter the link to [spark_vjs_amp_plugin.js](https://player.h-cdn.com/spark_vjs_amp_plugin.js). You can initially load it directly from Spark servers for testing but should move it to be hosted on your own servers before releasing to production

IMPORTANT: set 'customer' parameter to your Spark customer name, the same as used to load the Spark JS.

PLUGINS

CancelSave

Configure custom plugins and manage Brightcove provided solutions. For more information on plugins, visit our [Help Docs](#).

JavaScript0 configured

https://player.h-cdn.com/spark_vjs_amp_plugin.js?customer=<customer_name>

+

CSS0 configured

Name, Options (JSON)0 configured

4. in the Name, Options (JSON) area, enter spark_loader as plugin name

PLUGINS

CancelSave

Configure custom plugins and manage Brightcove provided solutions. For more information on plugins, visit our [Help Docs](#).

JavaScript1 configured

CSS0 configured

Name, Options (JSON)0 configured

spark_loader

+



5. save and publish your player
6. verify Spark was added to the player's frame by opening a relevant link and checking the browser console for this message

```
2018-07-08 06:21:26.791000 [0] cdn/loader: hola spark loader v1.101.705 tag 614 was loaded in frame: top
Video enhancements powered by HolaSpark.com v1.101.705.T614 Zone default
http://holaspark.com/?cam=wm devconsole
Enabled features: Video preview, Watch next, Viewing history

Available features: Floating player, Image preview, Player thumbnails, Watch later, Position memory, Player auto play, Video
Embedded in demo frame: https://players.brightcove.net/5805274495001/rJPDn9sf7/default/index.html?playlistId=5805274697001
```

Where 'Embedded in <customer> frame' has a `players.brightcove.net` domain

Note that you may see several such messages if there is more than one player in the page and also one reported by the script loaded at the top page.

8. Hosting the Spark on your CDN

Spark hosts all required files during the evaluation period on its CDN, but in production, different Spark elements can be served from your CDN, as it is usually best optimized to deliver JS-type content to your users in your country and/or is better tuned for scalability in case of traffic spikes.

Spark elements that can be hosted on your CDN are the Spark JS code and the Spark Video Previews.

8.1 Hosting the Spark JS code on your CDN

In order to maximize caching and ensure Spark does not slow page loading speed (see <https://holaspark.com/faq#general-gen-slowpage>), the Spark JS is loaded in 3 stages:

1. A 4kb **small Spark loader** is loaded, async, as page loads.
2. After conditions specified in the small loader are satisfied, the **full Spark code** and the most recent **Spark configuration** file are loaded. By default, both are loaded after the webpage finished loading.
3. Once the full code and configuration are downloaded, Spark is initialized using this configuration

8.1.1 Instructions

In order to serve the Spark JS from your CDN, you should keep files on your server and poll Spark servers for the latest JS code version every 5 minutes. Polling every 5 minutes ensures



you have the latest code version and that configuration updates on the Spark control panel are applied in production rapidly.

1. Small Spark loader: There is no need to host this file. It should be always loaded from Spark via

```
https://player.h-cdn.com/loader.js?customer=[customer ID]
```

2. Full Spark code

Keep a copy of the full Spark code on your CDN. **You must inform Spark Support of the location of this file.**

Important! Poll Spark servers for the latest JS code version every 5 minutes and make sure cache control `max-age` attribute is set to 5 minutes. It should look like: `Cache-Control: max-age=300`. Example for cron implementation:

```
5 * * * wget https://player.h-cdn.com/loader.js?customer=[customer ID]&no_conf=true -O /var/www/website/js/loader_code.js
```

3. Spark Configuration

Keep a copy of the full Spark code on your CDN. **You must inform Spark Support of the location of this file.**

Important! Poll Spark servers for the latest configuration every 5 minutes and make sure cache control `max-age` attribute is set to 5 minutes. It should look like: `Cache-Control: max-age=300`. Example for cron implementation:

```
5 * * * wget https://player.h-cdn.com/config.js?customer=[customer ID] -O /var/www/website/js/loader_config.js
```

Not setting `Cache-Control` means browsers will cache the JS, which will prevent upgrades and configuration changes from reaching users.

Examples of Cache-Control headers:

Apache:

```
Header set Cache-Control "max-age=300"
```

Nginx:



```
add_header 'Cache-Control' 'max-age=300';
```

Contact support@holaspark.com with any questions.

8.2 Hosting Spark Video Previews on your CDN

In some cases, you may want to host Spark Spark Video Previews on your CDN - usually because you regularly experience large traffic peaks and your own CDN is better suited to handle them than Spark's general-purpose CDN.

This process is easy - basically, your CDN will use Spark's CDN as its origin, and Spark URLs will point to your CDN. However, implementation varies for each case, so please contact support@holaspark.com for details.



9. Spark editor tools

The Spark editor tools allow content editors to customize content displayed by Spark in real-time and on production traffic without any development, using simple GUI tools.

This is a powerful option to easily customize Spark contents:

- [Edit specific Video Preview](#) to fit specific articles, important sport events etc.
- [Select Video Previews to auto-play](#) - to replace animated GIFs.
- [Block specific previews](#) from playing on your page
- [Manually purge previews](#) in case you need to re-create the preview
- [Edit recommendations](#) to promote specific content, upsell to premium etc.

These videos show editor tool in action - we recommend you watch them:

- [Creating custom video previews \(1:55\)](#)
- [Replacing animated GIFs with auto-playing previews. \(1:19\)](#)

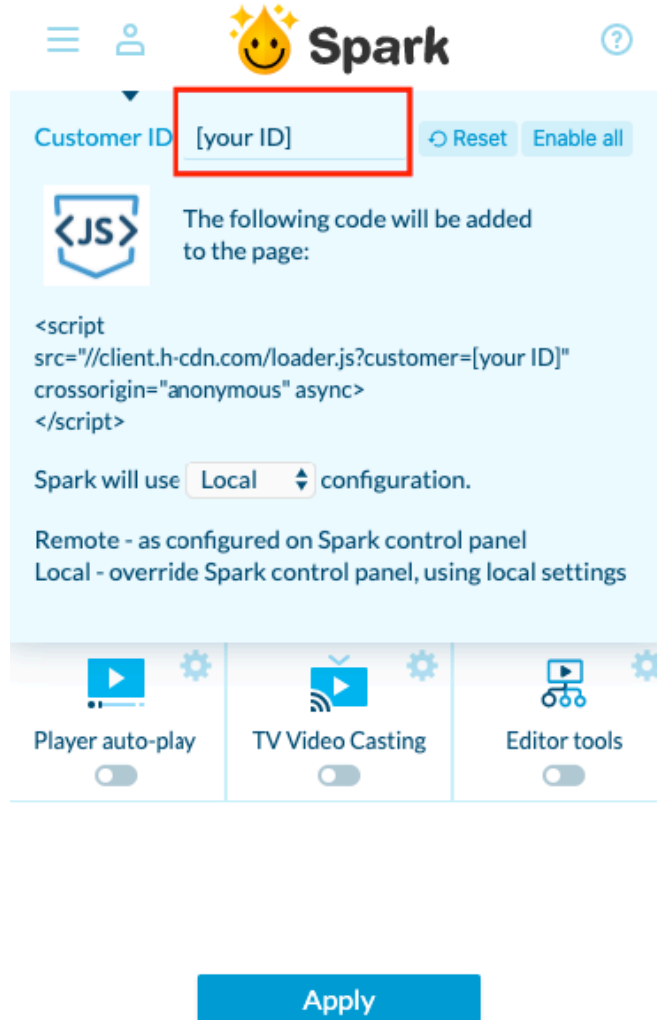
Only users who are signed in to the Spark control panel and have the appropriate permissions are allowed to edit content.

Note - if you are having issues, see the [troubleshooting area](#).

9.1 Overview

In order to use the Spark Editor Tools, you must use Chrome browser. Follow these steps:

1. Make sure you are signed in to the Spark control panel with your account.
2. Download and install the [Spark Configurator from the Chrome Webstore](#).
3. Go to your site, click the Spark Configurator icon to open the interface, then click on the user icon on the top left and enter your Spark CustomerID (reminder: Your customerID appears in the Spark Control Panel URL [https://holaspark.com/cp?cust=\[CustomerID\]](https://holaspark.com/cp?cust=[CustomerID])):



The image shows the Spark configurator interface. At the top, there is a navigation bar with a menu icon, a user icon, the Spark logo, and a help icon. Below the navigation bar, the 'Customer ID' field is highlighted with a red box and contains the text '[your ID]'. To the right of this field are 'Reset' and 'Enable all' buttons. Below the 'Customer ID' field, there is a code block containing the following JavaScript code:

```
<script  
src="//client.h-cdn.com/loader.js?customer=[your ID]"  
crossorigin="anonymous" async>  
</script>
```

Below the code block, there is a section for configuration. It states 'Spark will use' followed by a dropdown menu set to 'Local' and the word 'configuration.'. Below this, there are two lines of text: 'Remote - as configured on Spark control panel' and 'Local - override Spark control panel, using local settings'. At the bottom of the configurator, there are three toggle switches: 'Player auto-play', 'TV Video Casting', and 'Editor tools'. All three switches are currently turned off. Below the configurator, there is a large blue 'Apply' button.

4. Make sure Spark uses 'Local' configuration.
5. Click 'Apply'. Your site will refresh; note that Spark Configurator's local settings override default user settings, so some features will be disabled until you enable them in the configurator. This is normal and expected.
6. You can now edit Watch Next recommendations and/or Video Previews - see below.

9.2 Customizing Video Previews

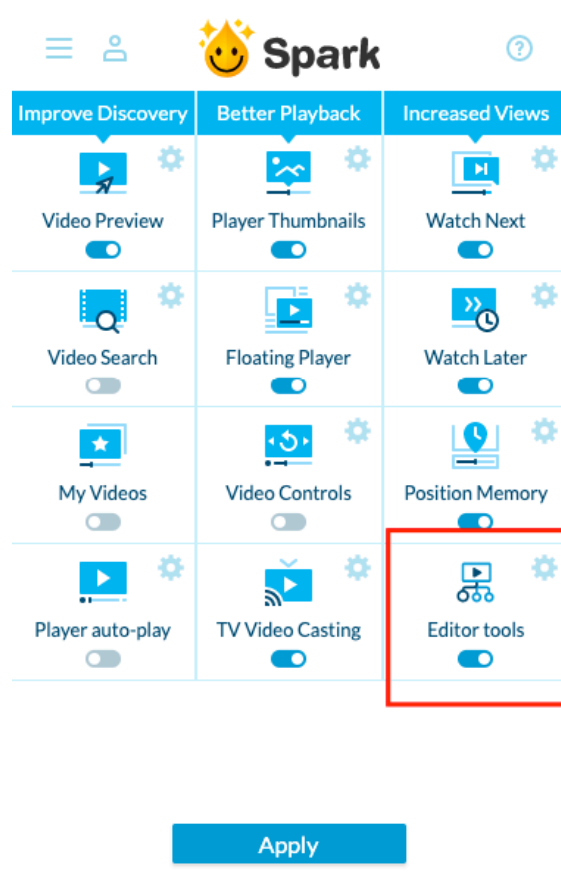
Spark Editor Tools allow you to easily edit what parts of the actual video are shown in Video Previews. Typical uses for this are:

- Highlighting interesting content for specific times (e.g. coverage of Olympics)
- Blocking Video Previews from showing for a specific video

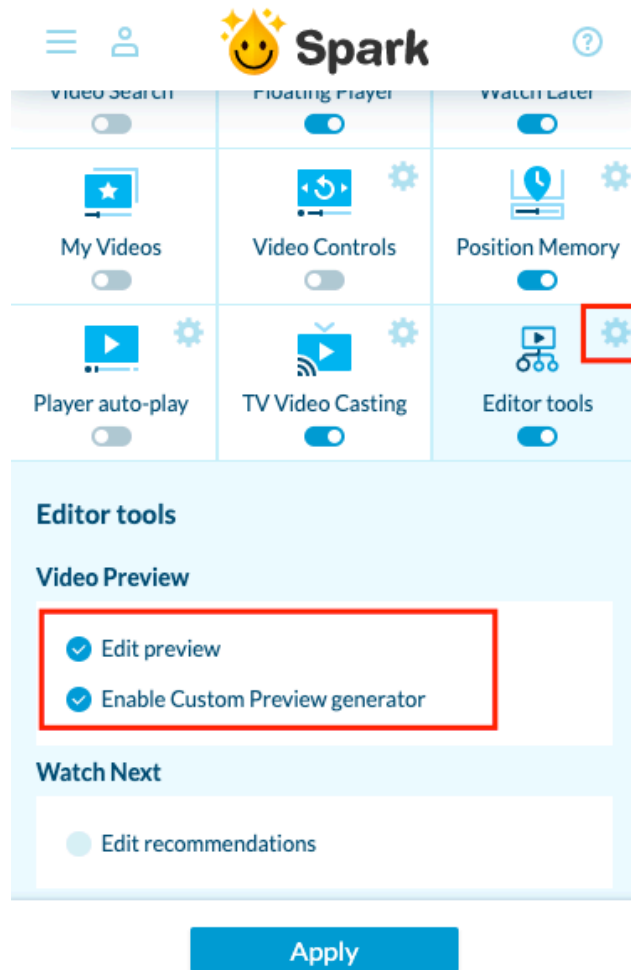
To be able to edit Video Previews:



1. Make sure you are signed in to the Spark control panel with your account.
2. Enable “Editor Tools” and click apply.



3. Click the little settings cogwheel and enable the tools you need:

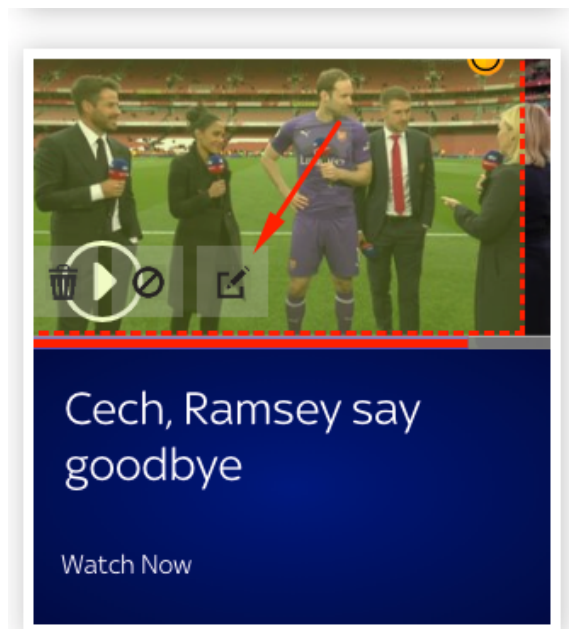


4. Click apply. The page will refresh - right click a Video Preview to block or edit it

9.2.1 Editing specific Video Previews

To edit a specific Video Preview:

1. Make sure you are signed in to the Spark control panel with your account.
2. Right click on the Video Preview you want to edit and click the edit icon



3. The video page will open and a dialog box will appear next to the video player. You can see the current Video Preview by hovering on the small thumbnail in the Spark Editor.



4. Play the video and select the parts of the video you want to include in the Video Preview by clicking "Start" and "Stop" when you need. You can pause the video at the right time and then click Start/Stop - the times will be added to the table:

Cech, Ramsey say goodbye



Petr Cech and Aaron Ramsey spoke to Kelly Cates about their time at Arsenal and the next step in their lives.

Spark Preview Editor

[Start](#) [Stop](#)

From	To	Slo-Mo
0:02.54	0:04.10	
1:57.89	1:59.47	
5:43.87	5:45.84	

Total duration: 5:12 sec

[Display preview](#)

[Reset](#) [Generate](#)

Current preview

status: not requested

[Block preview](#)

5. You can manually edit the timings in the table. The total duration of the Video Preview is displayed. It is recommended not to exceed a total of 6 seconds.
6. To see your custom Video Preview, click “Preview”. It will be displayed in the player with a “Preview” watermark:

Cech, Ramsey say goodbye



Petr Cech and Aaron Ramsey spoke to Kelly Cates about their time at Arsenal and the next step in their lives.

Spark Preview Editor

[Start](#) [Stop](#)

From	To	Slo-Mo
0:02.54	0:04.10	
1:57.89	1:59.47	
5:43.87	5:45.84	

Total duration: 5:12 sec

[Display preview](#)

[Reset](#) [Generate](#)

Current preview

status: ready

[Block preview](#)

7. Use the “Display preview” button often to try different previews. It is often useful to adjust timings in the table to get exactly the desired preview. If you want to start with a fresh table, reset the table by clicking “Reset”
8. If you are happy with the result, click “Generate”. Buttons will turn gray and Spark will get to work: purge the old preview, regenerate the new Video Preview and distribute it to multiple Spark servers. This process can take up to 2 minutes - be patient :).

Cech, Ramsey say goodbye



Petr Cech and Aaron Ramsey spoke to Kelly Cates about their time at Arsenal and the next step in their lives.

Spark Preview Editor

Start Stop


From	To	Slo-Mo
0:02.54	0:04.10	
1:57.89	1:59.47	
5:43.87	5:45.84	

Total duration: 5.12 sec

Display preview

Reset Generate

Current preview



Block preview

Important note: since the original Video Previews may be cached on your browser, you may not see your new Video Preview instantly. Clear the cache or use an incognito browser window to force the browser to load the updated preview.

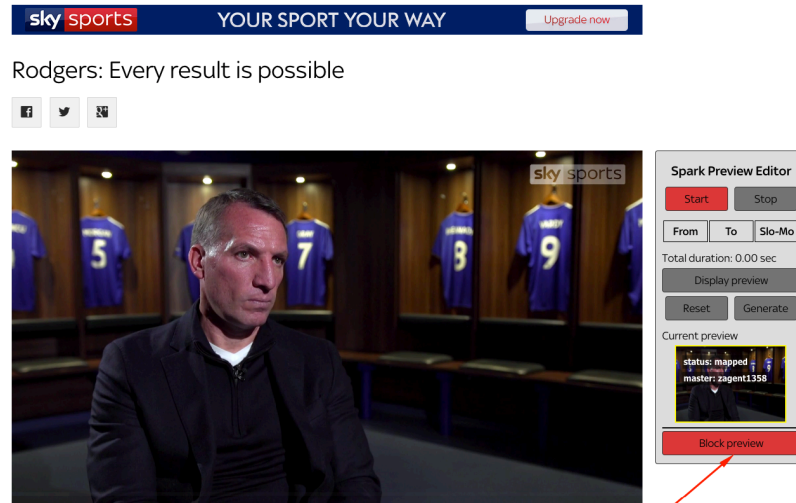
9.2.2 Blocking specific Video Previews

To block a specific Video Preview:

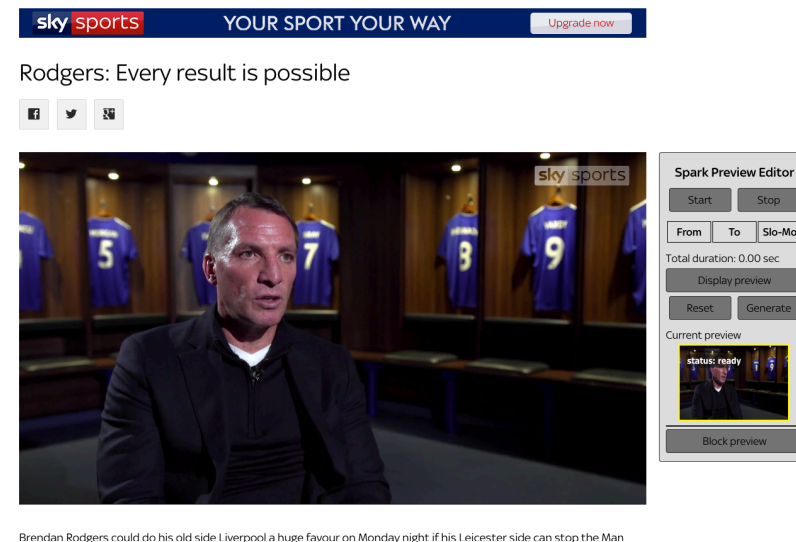
1. Make sure you are signed in to the Spark control panel with your account and the Spark configurator is set-up as described in section 9.1
2. Right click on the Video Preview you want to edit and click the block icon.
3. The Video Preview will be blocked within a few minutes.

Alternatively:

1. Click on the Video Preview you want to block in order to play the actual video
2. On the video page, after it starts playing, a dialog box will appear next to the video player. Click on "Block Preview"



3. The button will turn gray while Spark purges the Video Preview from all servers and blocks it from appearing:



4. Once the buttons become red again, the Video Preview is blocked.

9.2.3 Purging specific Video Previews

To Purge a specific Video Preview:

1. Make sure you are signed in to the Spark control panel with your account and the Spark configurator is set-up as described in section 9.1
2. Right click on the Video Preview you want to purge and click the trashcan icon.
3. The Video Preview will be purged within a few minutes and re-generated using the new video as soon as it is watched by any user.

9.3 Customizing Watch Next recommendations

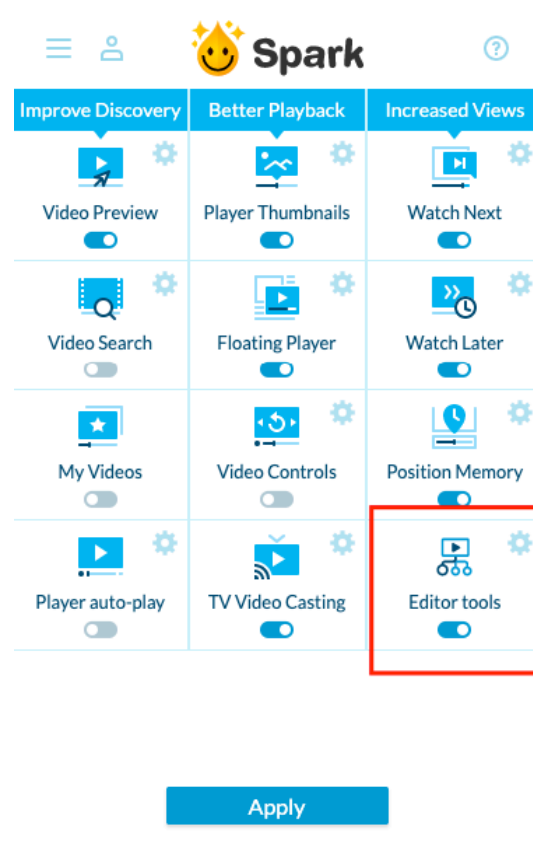
Spark Editor Tools allow you to easily add static recommendations to Watch Next. Typical uses for this are:

- Promoting premium subscriptions
- Displaying constant link to an important part of the site (e.g. live video)
- Highlighting interesting content for specific times (e.g. coverage of Olympics)

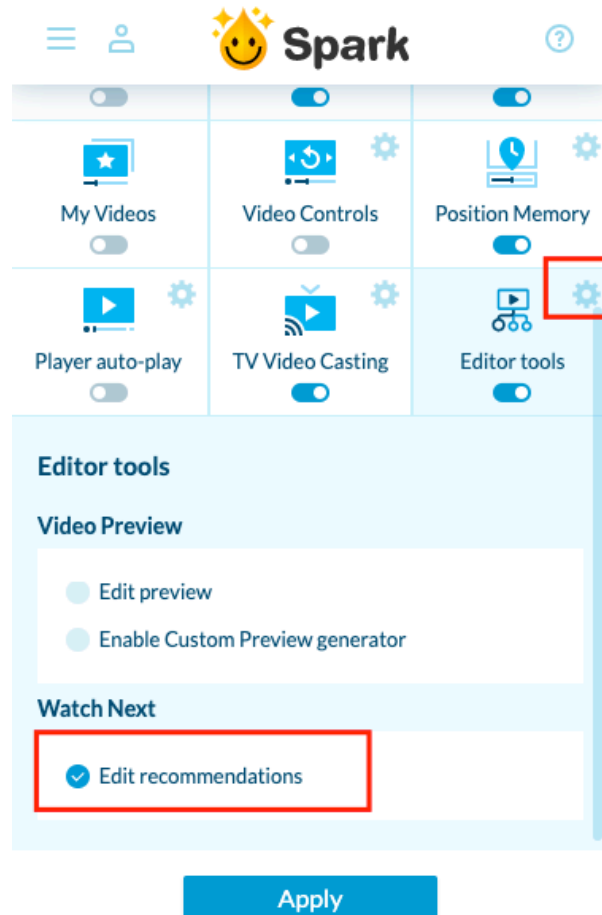
You can edit recommendations for both ‘panels’ (typically used for paused video) and ‘tiles’ (typically used for end-of-video).

To edit Watch Next recommendations:

1. Make sure you are signed in to the Spark control panel with your account.
2. Enable “Editor Tools” and click apply.



3. Click the little settings cogwheel and enable the tools you need:



4. Click apply. The page will refresh.
5. Play a video and display the recommendations - for example, pause the video to display the panels.
6. Right click the tile you want to edit. An 'edit' icon will appear:

Solskjaer: Not good enough



Click the icon and a popup will appear, allowing you to enter information.

Set static tile content

page url	poster
description	type caption

SubmitClose

Page URL: Destination URL browser will navigate to if tile is clicked

Poster: URL pointing to the image you want displayed on the tile

Description: Text line that appears at the bottom of the tile, on mouseover

Caption: Word that appears in top right of recommendation

- Changes will appear in production 2-3 minutes after you submit the dialog.
- Note that If there is already static information configured, it will appear and you can edit it. To delete a static recommendations, edit it and click reset to default

Set static tile content

/video/sports/cricket/116root-england_434.jpg	
Brilliant catch sees off F	ICYMI

SubmitClose

Reset to default

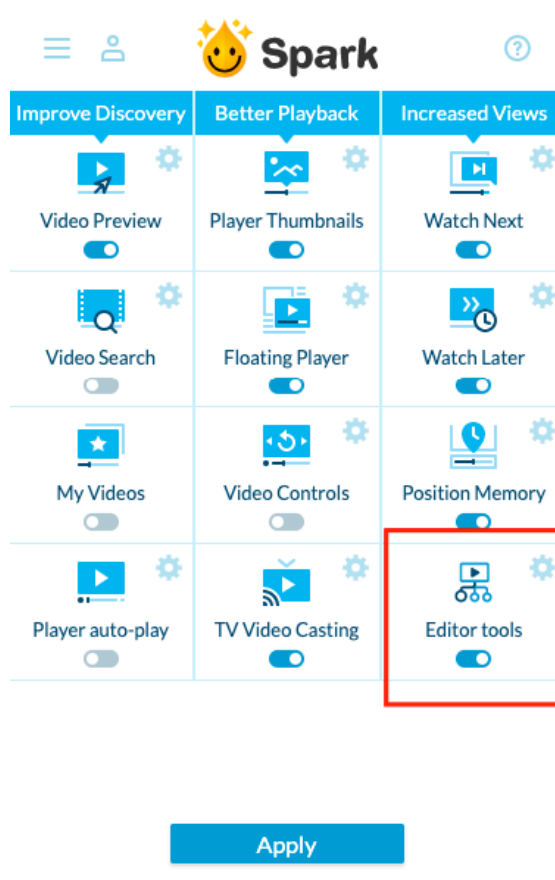
9.4 Selecting Video Previews to Auto-Play

Editors sometimes use animated GIFs to promote certain articles. They can now mark specific Spark Video Previews to auto-play when visible instead.

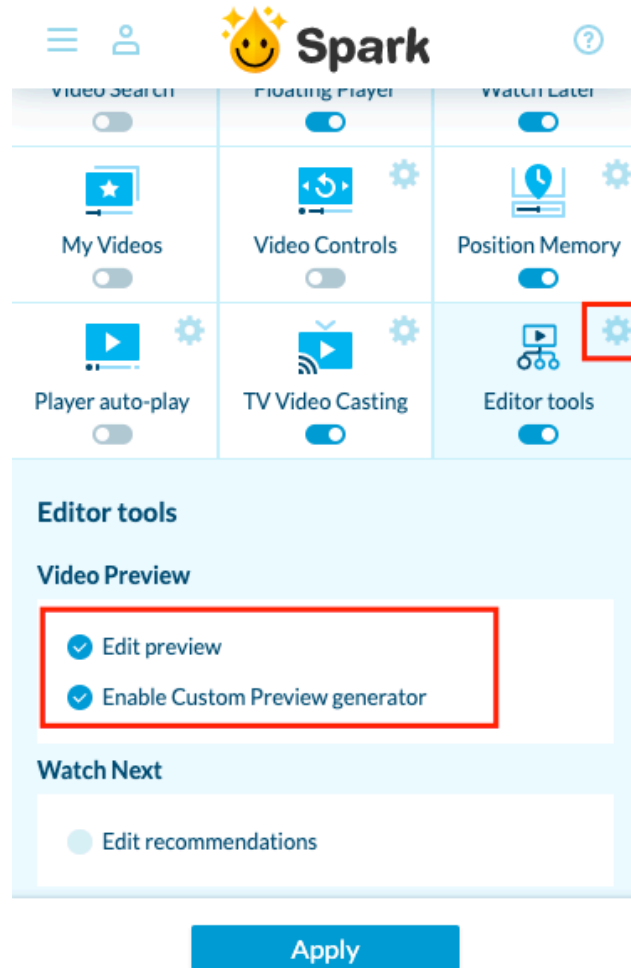
Animated GIFs have several disadvantages: They are very big (few MB), look cheap/grainy and are hard to edit. Spark Video Previews are better: They are lightweight (100-200KB), look crisp and are easy to edit.

[See this video](#) for step by step or follow these instructions:

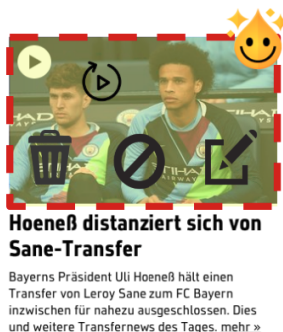
1. Make sure you are signed in to the Spark control panel with your account.
2. Enable “Editor Tools” and click apply.



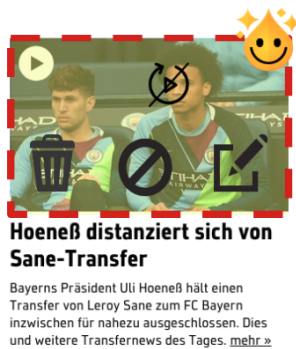
3. Click the little settings cogwheel and enable the tools you need:



4. Click apply. The page will refresh - right click a Video Preview to block or edit it
5. Right click the Video Preview you want to auto-play:



6. Click the Auto-Play icon and confirm the dialog box. Changes will be on production in 2-3 minutes.
7. Next time you visit the page, note that links that are already marked for auto-play will have a yellow background and red frame. This is of course only visible if you use the Editor Tools - normal users won't see it.

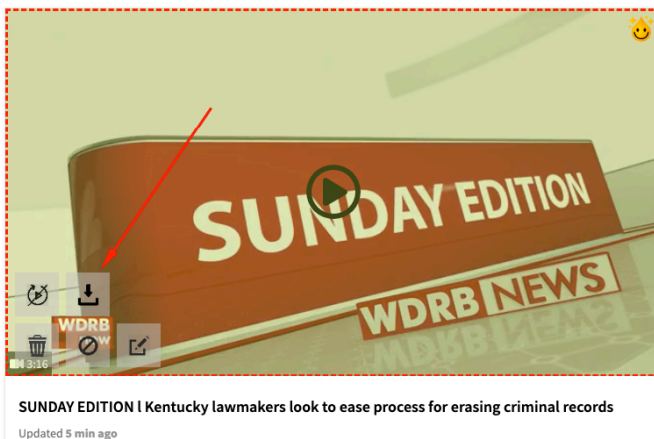


- To disable auto-play for the Video Preview, click the button to disable auto-play and confirm the dialog box. Changes will be on production in 2-3 minutes.

9.5 Downloading Video Previews

Editors sometimes want to download the Video Previews for use in social media, newsletters etc. To do so, click the “Download” icon in editor tools:

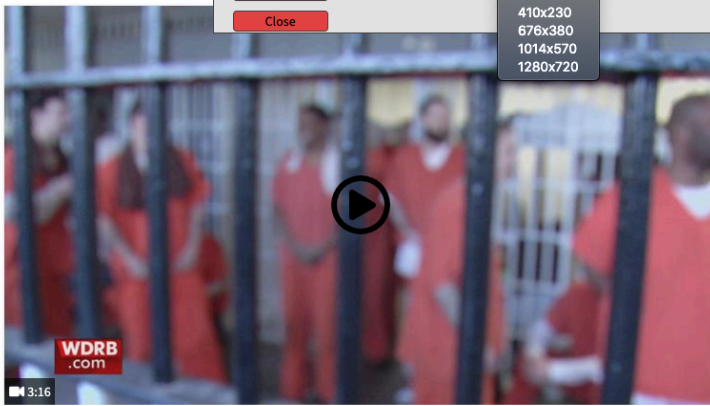
TOP VIDEOS



You can then select the desired size to download:

MORE POINT OF VIEW VIDEOS

TOP VIDEOS



Trimble Co. Church to
nearly 130 years

POLICE: Body found i
as missing Georgetov

Dairy Del, popular Lo
reopen Monday

CRAWFORD | Louisvil
sets up chance to pla

Local car enthusiast
battling brain cancer

Woman who visited I

One dead, one injure
River Road

McDonald's is giving
Monday

9.6 Editor tools without Chrome extension (Beta)

If using the Spark Configurator is not possible, authorized users who are signed-in to the Spark control panel have access to some functionality even without the Spark Configurator. Please refer to earlier sections to see step by step instructions for different editor tools operations.

 This functionality is still in beta as of Mar-2021. There are still several [limitations](#).

1. Make sure you are signed in to the Spark control panel.
2. Open the desired URL and append '?spark_mode=editor' to the end of the URL, for example:

www.example.com/page.html → www.example.com/page.html?spark_mode=editor

3. Editor tools will be activated.
 - a. You can right-click on video thumbnails to display the editor tools menu.
 - b. You can edit custom previews by using the Spark Editor which will appear next to the video player.

9.6.1 Known limitations

The recommended way of using editor tools is using the Spark Configurator. If the Spark Configurator is not used, there are some limitations:

1. Only custom preview editing is currently supported.
2. Sites that have iFrames might require a one-time configuration using the browser's developer console. This is because the '?spark_mode=editor' flag cannot be added to iFrames without the extension. In order to fix this, a one-time setup is required:
 - a. Open the web page and open the browser's developer console
 - b. Navigate to the iFrame and enter in console the following line:

```
localStorage.spark_mode='editor'
```

- c. Close the browser and refresh the page

9.7 Troubleshooting

If Editor Tools do not appear when you right-click a preview, there could be several reasons:



- You are not logged into the Spark control panel. Make sure you are logged in
- The customer ID in the configurator is incorrect. Make sure it is the same as your customerID.
- A browser extension (e.g. uBlock ad blocker) is preventing the Spark code from loading. Temporarily disable any ad blockers.
- You were randomly selected to be in a Spark “Control Group” which is used to measure the Spark improvement. This means all features are disabled for you - including Editor Tools. To fix this, you need to clear your browser’s LocalStorage area for your site - see <https://www.leadshook.com/help/how-to-clear-local-storage-in-google-chrome-browser/>

If none of this works, please contact Spark Support at support@holaspark.com.

10. Custom Spark code for homepage

Some homepages require only the Video Preview functionality, and thus do not require the full Spark code. For these cases, Spark support can create custom code which is very small (<10KB) to include on the homepage.

The full Spark code is still needed on pages with a video player for features like Thumbnails, Watch Next etc.

If you need a small script for your homepage, contact Spark support.