



DASH Everywhere Manual

One player for all browsers

Author: Tobias Patella
E-mail: tobias.patella@castlabs.com
Status: Final
Version: 2.7
Date: 21 August 2015

CONTENT

1	DASH Everywhere Player.....	3
1.1	Features.....	3
2	Setup.....	4
2.1	Byte Range Requests	4
2.2	CORS HTTP Headers	4
2.3	Crossdomain File.....	4
2.4	HTACCESS File.....	4
2.5	Player Libraries.....	4
3	Player Integration.....	5
3.1	Integration Example.....	5
3.2	API	6
3.2.1	video.js API.....	6
3.2.2	DASH Everywhere API.....	7
3.2.3	Error Handling	9
3.2.4	Example.....	10
3.3	Plugins	10
3.3.1	video.js plugins	10
3.3.2	DASH Everywhere plugins	11
3.4	Skins.....	11
3.5	Configuration Options	12
3.5.1	Playback of unprotected DASH content	12
3.5.2	Playback of DRM-protected DASH content with DRMtoday	12
3.5.3	Playback of DRM-protected Smooth Streaming content with DRMtoday.....	13
3.5.4	ABR Configuration Recommendations.....	14
3.5.5	Full Configuration Reference	15
3.6	Chromecast Integration	23
3.6.1	Sample Receiver Application.....	23
3.6.2	Event handling on the Receiver application.....	24
3.6.3	Communication with the Sender Application.....	25
3.6.4	DRMtoday specifics.....	28
3.6.5	Receiver Application Checklist.....	29

4	Specifications.....	31
4.1	Platforms.....	31
4.2	Streaming Protocols	31
4.3	Formats	31
4.4	DRMs.....	31
5	Support.....	33

1 DASH EVERYWHERE PLAYER

The DASH Everywhere Player is a browser based, cross-platform video player that integrates several player technologies and provides a single, unified JavaScript API. It includes:

- *dash.js*: A plugin-free DASH player for HTML5 capable browsers with MSE&EME CENC (Common Encryption) support
- *dash.as*: A Flash-based player as fall back option to cover older browsers and extend the reach to all platforms.
- *smooth.js*: A plugin for dash.js to enable playback of encrypted Smooth Streaming Assets in Chrome via MSE and Widevine EME.

DASH Everywhere is provided as plugin for the open-source video.js player (<http://www.video.js.com>) and provides full access to its API, plugins and skins.

1.1 Features

The player provides the following features and capabilities:

- Playback of MPEG-DASH and Smooth Streaming content
- Adaptive bitrate streaming
- VOD and live streaming
- DRM: CENC-protected content
- Full playback control via API
- Customizable UI
- Multiple audio tracks support
- Subtitle support

Please see [4] for a complete list of specifications.

2 SETUP

The server that delivers the content needs to meet certain requirements to allow playback with DASH Everywhere.

2.1 Byte Range Requests

The server must support byte range requests as specified by

<http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.35.1>.

2.2 CORS HTTP Headers

The following CORS HTTP headers need to be present with the specified values:

- *Access-Control-Allow-Headers*: Must include at least "origin, range, Content-Type"
- *Accept-Ranges:bytes*
- *Access-Control-Allow-Origin*: Must include the domain where the player is hosted. For testing, usage of a wildcard value "*" is recommended.
- *Content-Type*: When the MP4 is delivered, this header must have the value "video/mp4". For the manifest, content-type must be "text/xml".

2.3 Crossdomain File

The server needs to host a custom *crossdomain.xml* to enable Flash-based playback. See

<https://github.com/castlabs/dashas/wiki/crossdomain>

for details.

2.4 HTACCESS File

The server needs to host a custom *.htaccess* file to enable Flash-based playback. See

<https://github.com/castlabs/dashas/wiki/htaccess>

for details.

2.5 Player Libraries

Upon purchase of DASH Everywhere, castLabs delivers the required player libraries that need to be hosted. There are no specific server requirements to host the player libraries but as they are static, usage of caching is recommended.

3 PLAYER INTEGRATION

DASH Everywhere is provided as playback technology (“tech”) for the open-source video.js player and fully supports video.js’ API, plugins and themes.

3.1 Integration Example

Integration can be done with a few lines of code. Besides integrating the video.js library, one only needs to include the cldasheverywhere.min.js file. The video.js instance is initialized with the DASH Everywhere-specific configuration object and the “dasheverywhere” playback tech. The following example shows the most basic integration to play an unencrypted DASH stream:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>castLabs DASH Everywhere</title>
<link href="http://vjs.zencdn.net/4.11.4/video-js.css" rel="stylesheet">
<script src="http://vjs.zencdn.net/4.11.4/video.js"></script>
<script type="text/javascript" src="cldasheverywhere.min.js"></script>
</head>
<body>
<video id="vid" class="video-js vjs-default-skin vjs-big-play-centered" preload="auto"
width="640px" height="480px">
</video>
<script>
var config, player;
var config = {
  "assetId": "",
  "variantId": "",
  "silverlightFile": "./dashcs/dashcs.xap",
  "flashFile": './dashas/dashas.swf',
  "techs": ['dashjs', 'dashas', 'dashcs']
};

var player = videojs('vid', {
  autoplay: true,
  controls: true,
  plugins: {
    audiotracks: {},
    texttracks: {}
  },
  techOrder: ['dasheverywhere'],
  dasheverywhere: config
});
player.loadVideo("http://demo.castlabs.com/media/bbb_abr_unenc/Manifest.mpd", {});
</script>
</body>
</html>
```

See 3.5 for a full list of all available configuration options and important notes regarding required settings for playback of DRM-protected content, Smooth Streaming content and integration examples to use DASH Everywhere together with DRMtodayas license backend.

3.2 API

DASH Everywhere and video.js provide a full API to programmatically control playback and listen to player event such as start of playback.

3.2.1 video.js API

The full list of available methods and events can be found at

<https://github.com/videojs/video.js/blob/stable/docs/api/vjs.Player.md>.

The most commonly used ones are summarized in the following two tables.

Method Header	Description	Example
controls(controls)	Shows/Hides UI controls	> player.controls(true);
currentTime(time)	Sets current playback time to value of parameter <i>time</i> [seconds]	> player.currentTime(10);
on	Attach event listener	> player.on('loadedmetadata', function(){ console.log("loadedmetadata event fired"); });
pause	Pauses playback	> player.pause();
play	Starts playback	> player.play();
volume(level)	Sets volume to <i>level</i> [0..1]	> player.volume(0.5);

Event Name	Description
durationchange	Fires when duration of content is first known or has changed
ended	Fires when player reaches end of stream
error	Fires upon error
loadedmetadata	Fires when content has been initially loaded. Playback can be started via <i>play</i> once only after this event has fired
pause	Fires when playback is paused
play	Fires when playback starts or continues
seeked	Fires when seeking is complete

seeking	Fires when seeking begins
timeupdate	Fires when playback position changed
volumechange	Fires when volume has changed

3.2.2 DASH Everywhere API

DASH Everywhere extends the video.js API with additional methods and events that are accessible via the player object.

Method Header	Description	Example
audioTracks()	Returns array of available audio tracks.	> player.audioTracks() < ["eng", "tha"]
enableABRSwitching(enable d)¹	En- or disables automatic bitrate switching.	> player.enableABRSwitching(true);
getActiveTech()	Returns name of the currently active playback technology. Can be <i>dashjs</i> , <i>dashas</i> or <i>dashcs</i> .	> player.getActiveTech(); < "dashjs"
getLastAppendedSegment¹	Returns last appended video and audio segment	> player.getLastAppendedSegment() < {audio: {...}, video: {...}}
getPlaybackStatistics()	Returns an object with metadata about current playback, e.g. active tracks or buffer lengths.	> player.getPlaybackStatistics() < { audio: { activeTrack: {...}, bandwidth: 63000, bufferLength: "8.7000" }, video: {...}, text: {...} }
loadVideo(url, config)	Stops current playback and loads a	> player.loadVideo('http://url.to/Manifest.mp

¹ dash.js only

	new asset, allowing to update previously set configuration options.	d', {'authenticationToken': '...'});
setActiveTrack(type, id)	<p>Sets the active audio or subtitle track.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • type: Type of track to set. Can be <i>audio</i> or <i>text</i>. • id: Zero-based index of track to set. 	> player.setActiveTrack('text', 1);
setPlaybackRate(speed)¹	Sets playback rate to <i>speed</i> . Valid values are [0.5-4].	> player.setPlaybackRate(2);
setQualityFor(type, index)¹	<p>Sets the representation for the active track (when ABRSwitching is disabled).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • type: Type of track to set. Can be <i>audio</i> or <i>video</i>. • id: Zero-based index of quality to set. A list of all available representations can be retrieved via the <i>getPlaybackStatistics</i> call 	> player.setQualityFor('video', 0);
subtitleTracks()	Returns array of available subtitle	> player.subtitleTracks(); < [{

	tracks.	adaptionIndex: 3, id: 0, lang: "en", mimeType: "application/ttml+xml", name: "en", subType: "subt", trackName: "en" }]
--	---------	---

Event Name	Description
firsttexttrackready	Fires when the first text track segment of a segmented text track is available (applies to Smooth Streaming content only)
texttracksready	Fires when the all text track segments of a segmented text track are available (applies to Smooth Streaming content only)
error	See below for all custom errors types thrown by DASH Everywhere

3.2.3 Error Handling

DASH Everywhere provides custom error messages. They can be detected by the *subtype* property of the error object as seen in the below example.

The available error types are:

Subtype	Description
MANIFEST_LOAD_ERROR	Manifest failed to load. Player is giving up, as all retry attempts were unsuccessful.
LICENSE_ACQUISITION_ERROR	Call to the license server failed, player couldn't acquire valid license for the content.
HDCP_OUTPUT_NOT_ALLOWED²	Playback fails due to HDCP restriction. The connected output device is not allowed (usually happens if a non-HDCP capable device like an external monitor is connected).
HDCP_OUTPUT_DOWNSCALED²	Playback has been downscaled by the video renderer due to HDCP restrictions.
LICENSE_EXPIRED²	Playback stops as current license has expired. Applies to rental

² Chrome only

scenarios.

3.2.4 Example

The following example shows how to attach event listener and handle custom error messages

```
player.on('error', function() {
  var error = this.error();
  if (error.code === 2 && error.subtype == 'MANIFEST_LOAD_ERROR') {
    console.log('Manifest load error');
  }else if (error.code === 2 && error.subtype == 'LICENSE_ACQUISITION_ERROR') {
    console.log('License acquisition error');
  }else if (error.code === 2 && error.subtype == 'HDCP_OUTPUT_NOT_ALLOWED') {
    console.log('Output device not allowed.');
```

```
  }else if (error.code === 2 && error.subtype == 'HDCP_OUTPUT_DOWNSCALED') {
    console.log('Output has been downscaled.');
```

```
  }else if (error.code === 2 && error.subtype == 'LICENSE_EXPIRED') {
    console.log('License expired.');
```

```
  }else {
    console.log('Other error');
```

```
  }
});

player.on('loadedmetadata', function(){
  //Player has finished initializing the video
  console.log('Video is ready');
```

```
});
player.on('play', function(){
  //Playback started/continued
  console.log('Playback started/resumed');
```

```
});
player.on('pause', function(){
  //Playback paused
  console.log('Playback paused');
```

```
});

//Print playback statistics every 5 seconds
player.on('timeupdate', function(){
  console.log(player.getPlaybackStatistics());
});
```

3.3 Plugins

The player can be customized with plugins to enable additional functionality.

3.3.1 video.js plugins

A list of publicly available plugins can be found at:

<https://github.com/videojs/video.js/wiki/Plugins>.

3.3.2 DASH Everywhere plugins

DASH Everywhere comes pre-integrated with two plugins to enable support for multiple audio and subtitle tracks. They don't require any specific configuration and can be referenced during player initialization:

```
var player = videojs('vid', {
  autoplay: true,
  controls: true,
  plugins: {
    audiotracks: {},
    texttracks: {}
  },
  techOrder: ['dasheverywhere'],
  dasheverywhere: config
});
```

3.4 Skins

Custom skins can be used to modify the look and feel of the player. A list of available skins can be found at

<https://github.com/videojs/video.js/wiki/Skins>.

A guide for custom skin development can be found at

<https://github.com/videojs/video.js/blob/stable/docs/guides/skins.md>.

3.5 Configuration Options

DASH Everywhere can be configured via the *dasheverywhere* configuration object as shown in the integration example:

```
...
var config, player;
var config = {
  "flashFile": path/to/dashas.swf',
  [...]
};

var player = videojs('vid', {
  autoplay: true,
  controls: true,
  plugins: {
    audiotracks: {},
    texttracks: {}
  },
  techOrder: ['dasheverywhere'],
  dasheverywhere: config
});
...
```

The following sections show typical use cases and the available configuration options. They are followed by a complete reference of all available configuration options.

3.5.1 Playback of unprotected DASH content

```
var config = {
  "flashFile": path/to/dashas.swf'
};
```

For playback of unprotected DASH content, almost all configuration options are optional. One only needs to specify the path to the Flash SWF file in order to use *dash.as*:

3.5.2 Playback of DRM-protected DASH content with DRMtoday

To enable playback of DRM-protected content using DRMtoday, a few more additional configuration options are required:

- License server URLs:
 - *accessLicenseServerURL*
 - *playReadyLicenseServerURL*
 - *widevineLicenseServerURL*
- DRMtoday-specific metadata
 - Content identifier: *assetId* and *variantId*
 - *customData* object (with *userId*, *sessionId* and *merchant*) and flag to enable sending of that data (*sendCustomData*)

When using the (optional) upfront authentication method with DRMtoday, the required token can be provided via the *authenticationToken* option.

For protected content that is missing or has invalid PSSH boxes, DASH Everywhere allows to generate them on the fly with the following options:

- *generatePSSH*: Enables client-side PSSH-box generation
- *widevineHeader*: Sets the metadata for the Widevine Modular PSSH box
- *playreadyHeader*: Sets the metadata for the PlayReady PSSH box

```
var config = {
    "flashFile": path/to/dashas.swf',
    "accessLicenseServerURL": "https://lic.
staging.drmtoday.com/flashaccess/LicenseTrigger/v1",
    "playReadyLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
    "widevineLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
widevine/cenc/",
    "assetId": "test_asset",
    "variantId": "test_variant",
    "customData": {
        "userId": "user1",
        "sessionId": "123",
        "merchant": "merchantid"
    },
    "sendCustomData": true,
    "authenticationToken": "123",
    "generatePSSH": true,
    "widevineHeader": {
        "provider": "castlabs",
        "contentId": "/KJt7bamLRtfgR9PQjT+dUeNnHUcJ9v11B9CNUFv",
        "trackType": "",
        "policy": ""
    },
    "playreadyHeader": {
        "laUrl": "http://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
        "luiUrl": "https://example.com"
    }
};
```

Using all those options together results in the following configuration object:

3.5.3 Playback of DRM-protected Smooth Streaming content with DRMtoday

When playing DRM-protected Smooth Streaming assets with DASH Everywhere, all the DRM-related setting from the previous section can be used. However, it is mandatory to set *generatePSSH* to true and set the *widevineHeader* and *playreadyHeader* accordingly.

When playing Smooth Streaming content from a Microsoft IIS Server, it is usually required to enable enhanced Smooth Streaming compatibility by setting the *enableSmoothStreamingCompatibility* flag to true.

DASH Everywhere usually automatically detects Smooth Streaming content. In rare circumstances, the detection needs to be forced to enable Smooth Streaming mode of DASH Everywhere by setting *isSmoothStreaming* to true.

Using all these settings results in the following configuration object:

```
var config = {
  "flashFile": path/to/dashas.swf',
  "accessLicenseServerURL": "https://lic.
staging.drmtoday.com/flashaccess/LicenseTrigger/v1",
  "playReadyLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
  "widevineLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
widevine/cenc/",
  "assetId": "test_asset",
  "variantId": "test_variant",
  "customData": {
    "userId": "user1",
    "sessionId": "123",
    "merchant": "merchantid"
  },
  "sendCustomData": true,
  "authenticationToken": "123",
  "generatePSSH": true,
  "widevineHeader": {
    "provider": "castlabs",
    "contentId": "/KJt7bamLRtfgR9PQjT+dUeNnHUcJ9v11B9CNUFv",
    "trackType": "",
    "policy": ""
  },
  "playreadyHeader": {
    "laUrl": "http://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
    "luiUrl": "https://example.com"
  },
  "enableSmoothStreamingCompatibility": true,
  "isSmoothStreaming": true
};
```

3.5.4 ABR Configuration Recommendations

To enable a great playback experience with minimal buffering times at the best quality possible, DASH Everywhere uses a custom downloader and ABR algorithm that features:

- Performance: Quickly adapt to changing network conditions
- Robustness: Gracefully handle network failures without affecting playback
- Efficiency: Optimize utilized bandwidth

The behavior of the algorithm can be controlled by a few configuration settings, namely the *ABRParameters* and *fragmentRequestTimeout*. Their settings correlate and optimal values depend on the content, especially the fragment duration.

- *fragmentRequestTimeout*: This value determines after how many [ms] the player should give up loading a fragment. If a request takes longer than this threshold, player aborts the requests, switches down to a lower quality and tries again.

- *minBufferLength*: The desired minimum buffer level in [s] the player tries to achieve. If the actual buffer level gets less than half of this value, player will reduce quality. This value should be higher than the *fragmentRequestTimeout* setting to have enough buffer in case downloading of a segment fails. E.g. if the *fragmentRequestTimeout* is set to 10000 (10secs), a *minBufferLength* of 15-20 is recommended which gives the player 5-10 secs after a failed request to try again downloading a segment in a lower quality.
- *maxBufferLength*: The maximum buffer level in [s] the player should keep. Useful to limit memory usage of the player and to achieve faster adaption to fluctuating network conditions. The player doesn't overwrite previously downloaded segments with a higher quality when network conditions improve.

The optimal values for those depend on the content and especially the fragment duration. For typical scenarios with a fragment duration of 2-3 seconds as it's commonly found with Smooth Streaming content and recommended for DASH content, good results can be achieved with these settings:

```
var config =
{
  [...]
  "fragmentRequestTimeout": 6000,
  "ABRParameters": {
    "minBufferLength": 15,
    "maxBufferLength": 25
  },
  [...]
};
```

As rule of thumb, the *fragmentRequestTimeout* should be between 2-3x the (constant) fragment duration. E.g. if the fragment duration is 2 seconds, the average download time for a fragment at a given quality must be less than 2 seconds to avoid buffering. Therefore, to accommodate for temporary slow downs, a timeout of 6 seconds is recommended.

3.5.5 Full Configuration Reference

The following DASH Everywhere configuration options are available.

R=Required, O=Optional, Blank=Doesn't apply (JS: HTML5 player, AS: Flash player, CS: Silverlight player)

Parameter	Description	Type	Unprotected Content			DRM-protected Content		
			JS	AS	CS	JS	AS	CS
flashFile	Path to dashas SWF file	String		R			R	

silverlightFile	Path to dashcs XAP file	String			R			R
accessLicenseServerURL	URL of Adobe Access license server	String		R			R	
playReadyLicenseServerURL	URL of PlayReady license server	String				O		R
widevineLicenseServerURL	URL of Widevine license server	String				R		
assetId	Asset Id of the content.	String					R	
variantId	Variant Id of content	String					R	
authenticationToken	Token for upfront authentication	String				O	O	O
customData	DRMtodaymetadata	JSON				R	R	R
customData.userId		String						
customData.sessionId		String						
customData.merchant		String						
sendCustomData	Enables sending of <i>customData</i>	bool				R	R	R
generatePSSH	Enable client-side PSSH box generation	bool				O		
widevineHeader	Widevine-specific header data	JSON				O		
widevineHeader.provider	Widevine provider ID. Set to <i>castLabs</i> when used with DRMtoday	String						
widevineHeader.contentId	Widevine content id. Recommended value is b64-encoded version of key id.	String						
widevineHeader.trackType	Widevine track id. Recommended to leave blank when using DRMtoday.	String						
playreadyHeader	PlayReady-specific	JSON				O		

	header data							
playreadyHeader.laUrl		String						
playreadyHeader.luiUrl		String						
enableSmoothStreamingCompatibility	Increases compatibility with Smooth Streaming assets. Should be enabled when used with IIS Smooth Streaming Server.	bool				O		
isSmoothStreaming	Forces Smooth Streaming mode	bool				O		
keyId	Allows to overwrite key id from manifest.	String				O		
drm	Forces usage of the specified DRM when used with Chromecast. Allowed values: <i>auto</i> , <i>com.widevine.alpha</i> , <i>com.youtube.playready</i>					O		
techs	Limits available playback technologies to set values. Can include <i>dashjs</i> , <i>dashas</i> and/or <i>dashcs</i> .	Array of Strings	O	O	O	O	O	O
debug	Print debug statements to browser console.	bool	O	O	O	O	O	O
ABRParameters	Parameters for adaptive bitrate switching algorithm	JSON	O	X	X	O	X	X
ABRParameters.qualitySwitchThreshold	Minimum interval in [ms] between two switch requests.	Float						
ABRParameters.bandwidthSafetyFactor	Sets bandwidth overhead that should be available before	Float						

	player switches to higher quality. A factor of e.g. 1.2 means that the player requires available bandwidth of 1200kbit for a representation of 1000kbit.							
ABRParameters.minBufferLength	Desired buffer level. Player will adapt quality to reach and keep this level.	Float						
ABRParameters.maxBufferLength	Maximum buffer level. Player will fill buffer not more than this threshold.	Float						
ABRParameters.liveDelay	Sets desired value of distance to live edge in [secs]. If not provided, default value of 8 will be used. If set to 0, value will be set to <i>2xminBufferLength</i> from Manifest.	Float						
ABRParameters.disableBufferOccupancyRule	Disables rule that allows player to switch up to higher quality if buffer is filled. Setting to true will make player base decisions only on available bandwidth. Not recommended since v 3.0.0.	bool						
ABRParameters.droppedFrameThreshold	Defines threshold after which the player should switch down one quality and never switch to the higher quality for the current playback session anymore. Useful to	Integer						

	avoid performance problems during playback on devices with weak hardware							
customHeaders	Allows providing of custom HTTP headers that are sent with various requests types							
customHeaders.manifestRequest	Additional HTTP headers to be send with manifest request	JSON	O			O		
customHeaders.licenseRequest	Additional HTTP headers to be send with license request	JSON				O		
customHeaders.fragmentRequest	Additional HTTP headers to be send with fragment request	JSON	O			O		
licenseRequestTimeout	Time in <i>[ms]</i> before a license request aborts. A value of 0 (which is the default) means there is no timeout.	Integer	O	X	X	O	X	X
manifestRequestTimeout	Time in <i>[ms]</i> before a manifest request aborts. A value of 0 (which is the default) means there is no timeout.	Integer	O	X	X	O	X	X

fragmentRequestTimeout	Time in <i>[ms]</i> before a segment request aborts. A value of 0 (which is the default) means there is no timeout.	Integer	O	X	X	O	X	X
------------------------	---	---------	---	---	---	---	---	---

See the following complete example:

```

var config = {
    "flashFile": path/to/dashas.swf',
    "silverlightFile": path/to/dashcs.xap',
    "accessLicenseServerURL": "https://lic.
staging.drmtoday.com/flashaccess/LicenseTrigger/v1",
    "playReadyLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
    "widevineLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-
widevine/cenc/",
    "assetId": "test_asset",
    "variantId": "test_variant",
    "customData":{
        "userId": "user1",
        "sessionId": "123",
        "merchant": "merchantid"
    },
    "sendCustomData": true,
    "authenticationToken": "",
    "generatePSSH": true,
    "widevineHeader": {
        "provider": "castlabs",
        "contentId": "/KJt7bamLRtfgR9PQjT+dUeNnHUcJ9v11B9CNUFv",
        "trackType": "",
        "policy": ""
    },
    "playreadyHeader": {
        "laUrl": "http://lic.staging.drmtoday.com/license-proxy-
headerauth/DRMtoday/RightsManager.asmx",
        "luiUrl": "https://example.com"
    },
    "enableSmoothStreamingCompatibility": true,
    "isSmoothStreaming": true,
    "keyId": "AAAAA-AAAAA-BBBB-CCCC-DDDD-EEEEEEEEEEEEEE"
    "drm": "auto",
    "techs": ["dashjs", "dashas", "dashcs"],
    "debug": false,
    "ABRParameters": {
        "qualitySwitchThreshold": 2000,
        "bandwidthSafetyFactor" : 1.75,
        "minBufferLength": 15,
        "maxBufferLength": 20,
        "liveDelay": 2,
        "disableBufferOccupancyRule": false
    },
    "customHeaders": {
        "manifestRequest": {
            "headerName": "headerValue"
        },
        "licenseRequest": {
            "headerName": "headerValue"
        },
        "fragmentRequest": {
            "headerName": "headerValue"
        }
    },
    "licenseRequestTimeout": 10000,
    "manifestRequestTimeout": 10000,
    "fragmentRequestTimeout": 6000
};

```


3.6 Chromecast Integration

DASH Everywhere comes with built-in Chromecast receiver functionality and can be controlled over the standard message bus *urn:x-cast:com.google.cast.media* to control media playback. There is no need to implement a separate receiver application from scratch.

For Chromecast, castLabs provides a special build of the player (*cldashjs.min.js*) without the *video.js* integration and only includes the HTML5 player. The other components are not required on Chromecast and therefore allow you to use a lighter library. The configuration options for Chromecast are the same and initialization of the player is very similar to the regular usage on the desktop.

There are two main differences:

1. No setting of *customData* in receiver application
The DRMtoday-related *customData* is – contrary to the regular usage – not set during initialization of the player in the receiver application as the receiver page is static and the same for all end users. Instead, the *customData* is provided by the sender application and sent as part of the standard *loadMedia* method of the *mediaControlChannel*. See 3.6.4 for details and examples.
2. Media is not loaded via *loadVideo* method
It is important to keep in mind that playback on Chromecast is not initiated via *loadVideo* as above, but instead media playback is controlled through commands on the *urn:x-cast:com.google.cast.media* message bus that are sent from the sender application (e.g. mobile app). See 3.6.3 for details.

3.6.1 Sample Receiver Application

The *index_chromecast.html* file, which is provided with the release bundle, can be used as Chromecast receiver application and then stylized and customized as desired:


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>castLabs Chromecast Sample Receiver</title>
<script src="cldashjs.min.js" />
</head>
<body>
<div>
<video
style="width:100%;height:100%;top:0px;left:0px;right:0px;bottom:0px;position:fixed;" src="" />
</div>
<script>//Init player
var player, config;
config = {
  [...],
  "widevineLicenseServerURL": "https://lic.staging.drmtoday.com/license-proxy-widevine/cenc/",
  [...],
};
player
= new MediaPlayer(new castLabsDashJS.classes.DRMtodayContext(config));</script>
</body>
</html>
```

To use it as receiver application, the following steps are required:

- Configure the sample application (code above) as desired
- Make the application available on a webserver, e.g.:
http://www.urltoapplication.com/index_cc.html
- Register the application and its URL using the Google Cast SDK Developer Console:
<https://cast.google.com/publish/#/applications>
and remember the assigned Application ID.

Note: It is important to not include the *cast_receiver.js* library inside the receiver application page. The player loads the library dynamically.

3.6.2 Event handling on the Receiver application

Even though the receiver application usually just displays the video and all UI elements are presented on the sender application, it can be desired to display some basic UI elements like a buffering indicator or, for example, a progress bar when playback is paused. The receiver application can listen to two group of events:

1) Events triggered by the *<video>* element

These are the regular playback-related events (e.g. *pause*, *play*, *timeupdate*) that are triggered by the *<video>* element itself and be handled by attaching event listeners directly to it:

```
var vid = document.getElementById('vid');
vid.onplay = function() {
  console.log('player play');
};
vid.onpause = function() {
  console.log('player pause');
};
vid.onloadstart = function() {
  console.log('load started');
};
vid.ontimeupdate = function() {
  console.log('timeupdate');
};
```

2) Events triggered by the Cast SDK

For most use cases, it is not required to listen to events triggered by the Cast SDK as the receiver functionality is completely handled by the castLabs dash.js player. However, it is possible to access the *CastReceiverManager* and *MediaManager* instances and add some custom event handlers.

As the receiver library is loaded asynchronously, one needs to wait for the global *chromecastReady* event to be triggered which signals that the Chromecast receiver has been initialized and is ready to use.

Important: As the Cast SDK doesn't allow the attachment of event listeners (i.e. having multiple event handlers for one event), one can only overwrite the existing event handlers. As most of them are used by castLabs dash.js, overwriting them with custom code would break functionality. Therefore, when implementing custom event handlers, it is important to keep a reference to the original event handler and execute at the end of the custom event handler. The following example shows how to achieve this:

```
window.addEventListener("chromecastReady", function(e) {
  var onReadyOrig = window.castReceiverManager.onReady;
  window.castReceiverManager.onReady = function() {
    console.log('castReceiverManager is ready.');
```

```
    onReadyOrig();
  }
  var onLoadOrig = window.mediaManager.onLoad;
  window.mediaManager.onLoad = function(data) {
    console.log('onLoad called.');
```

```
    onLoadOrig(data);
  }
}, false);
```

3.6.3 Communication with the Sender Application

The castLabs dash.js player receiver application uses two namespaces to communicate with the sender application:

1) Communication via default namespace *urn:x-cast:com.google.cast.media*

This default channel is used for playback control and for sending status messages.

For communication from Sender -> Receiver, it implements the default messages as described at <https://developers.google.com/cast/docs/reference/messages>, section "Commands from sender to receiver":

- load
- pause
- seek
- stop
- play

For communication from Receiver -> Sender, the receiver application periodically calls the *broadcastStatus* method³) to notify all connected sender applications about the current playback status. The status object contains the following properties:

- *currentTime*: Current playback position in seconds
- *playerState*: Current state of player (playing, paused, idle, buffering)
- *customData.duration*: Total duration of current title in seconds
- *customData.userId*: DRMtoday metadata
- *customData.sessionId*: DRMtoday metadata
- *customData.merchant*: DRMtoday metadata
- *customData.textTracks*: An array containing available text tracks
- *customData.audioTracks*: An array containing available audio tracks

Example:

³ <https://developers.google.com/cast/docs/reference/receiver/cast.receiver.MediaManager#broadcastStatus>

```
{ "namespace": "urn:x-cast:com.google.cast.media",
  "senderId": "*:~*",
  "data": {
    "type": "MEDIA_STATUS",
    "status": [ {
      "mediaSessionId": 1,
      "playbackRate": 1,
      "playerState": "PLAYING",
      "currentTime": 3.0661,
      "supportedMediaCommands": 15,
      "volume": {
        "level": 1,
        "muted": false
      },
      "media": {
        "contentId": "http://url.to/Manifest.mpd",
        "streamType": "BUFFERED",
        "contentType": "video/mp4",
        "metadata": {
          "type": 0,
          "metadataType": 0
        },
        "customData": {
          "userId": "123",
          "sessionId": "123",
          "merchant": "test"
        },
        "duration": 1200
      },
      "customData": {
        "duration": 1200,
        "audioTracks": [ {
          "name": "tha",
          "id": 0
        }, {
          "name": "eng",
          "id": 1
        } ],
        "textTracks": [ {
          "name": "en",
          "id": 0
        }, {
          "name": "th",
          "id": 1
        } ]
      }
    ]
  },
  "requestId": 0
}
```

2) Communication via custom namespace *urn:x-cast:com.castlabs.chromecast*

This custom namespace can be used for customized communication with the receiver application. The following message types are supported (sent as JSON):

- Track Selection:

```
message = {  
  command: 'setActiveTrack',  
  type: <typeOfTrack>,  
  textTrack: <trackId>  
}
```

where *<typeOfTrack>* can be 'text' or 'audio' and *<trackId>* is the id as returned by the status message.

- Enable Subtitles (After they've been hidden):

```
message = {  
  command: 'enableSubtitles'  
}
```

- Disable/Hide Subtitles:

```
message = {  
  command: 'disableSubtitles'  
}
```

Example:

```
{  
  "data": "{  
    \"command\": \"setActiveTrack\",  
    \"type\": \"text\",  
    \"id\": 1  
  }",  
  "namespace": "urn:x-cast:com.castlabs.chromecast",  
  "senderId": "3:client-88983"  
}
```

3.6.4 DRMtoday specifics

To start playback of a stream on the connected Chromecast device, the standard *loadMedia* method of the *mediaControlChannel* can be used. It needs to be called with a *mediaInformation* object that contains all required information about the content that should be loaded like title, stream URL or thumbnail image.

However, when the receiver application is configured to use DRMtoday as license service, a few custom parameters need to be provided upon start of playback together with the *mediaInformation* object. Those parameters contain the merchant metadata as explained at:

https://fe.staging.drmtoday.com/frontend/documentation/integration/json_objects.html#merchant-metadata

They need to be sent as part of the *customData* property of the *loadMedia* method.

Example for sender app on iOS:

```
NSMutableDictionary * customData = [[NSMutableDictionary alloc]
initWithObjectsAndKeys:
    @"sampleUserId", @"userId",
    @"sampleSessionId", @"sessionId",
    @"sampleMerchant", @"merchant",
    nil];

GCKMediaInformation * mediaInformation = [[GCKMediaInformation alloc]
initWithContentID:[url absoluteString]
streamType:GCKMediaStreamTypeNone
contentType:mimeType
metadata:metadata
streamDuration:100
customData:customData];

[self.mediaControlChannel loadMedia:mediaInformation autoplay:autoPlay
playPosition:startTime];
```

Example for sender on Chrome browser:

```
var mediaInfo = new chrome.cast.media.MediaInfo(url);
mediaInfo.customData = {
  "userId": "sampleUser",
  "sessionId": "sampleSessionId",
  "merchant": "sampleMerchant"
};
mediaInfo.metadata = new chrome.cast.media.GenericMediaMetadata();
mediaInfo.metadata.metadataType = chrome.cast.media.MetadataType.GENERIC;
mediaInfo.contentType = 'video/mp4';
var request = new chrome.cast.media.LoadRequest(mediaInfo);
this.playerState = this.PLAYER_STATE.LOADING;
this.session.loadMedia(request,
this.onMediaDiscovered.bind(this, 'loadMedia'),
this.onLoadMediaError.bind(this)
);
```

3.6.5 Receiver Application Checklist

- Configure the receiver application with all required parameters
- Upload it to a public accessible webserver
- Register the URL of the application using the Google Cast SDK Developer Console
- Remember the assigned Application ID

















































































- Whitelist all Chromecast devices that should be used for testing with their serial number using the Google Cast SDK Developer Console

4 SPECIFICATIONS

The following formats are supported by DASH Everywhere. Not all formats are available on each platform.

4.1 Platforms

DASH Everywhere is compatible with the following platforms and browsers:

	Chrome 35+	Firefox 35+	Internet Explorer 9, 10	Internet Explorer 11	Edge	Opera 29+	Safari 8+
Windows 7	 	 	 	 		 	 
Windows 8	 	 		 		 	 
Windows 8.1	 	 		 		 	 
Windows 10	 	 		 	 	 	 
OS X 10.9	 	 				 	 
OS X 10.10	 	 				 	 
Chrome OS	 						

Usage of plugin-based playback requires the respective plugins to be installed and enabled on the browser. Minimum versions: Adobe Flash 11.5, Microsoft Silverlight 4.

4.2 Streaming Protocols

- MPEG-DASH: Segment List, Segment Template, Segment Base URL, Segment Timeline
- Microsoft Smooth Streaming
- Profiles: Live and VoD

4.3 Formats

- Video: H.264 encoded content up to High Profile Level 4.1, 4.2 and 5
- Audio: HE-AAC, LC-AAC
- Subtitles: WebVTT, TTML; Embedded&Image-based TTML (Smooth Streaming only)

4.4 DRMs

- Adobe Access
- Microsoft PlayReady
- Google Widevine Modular

5 SUPPORT

Support is available through the DASH Everywhere help center:

<http://castlabs.com/support>

Please email your project manager to request access.