API Dokumentation v5

Getting started

Loadings

Loading via npm

Prerequisites

1. The installation via npm requires `webpack`, `browserify` or a similar framework for your code to be interpreted by the browser. In React, Angular, Vue or similar technologies `webpack` often is included by default.
2. Access to the npm-Registry (https://npm-registry.netrtl.com/).

How to use the correct registry

If you already have access to the npm-Registry, please keep in mind to store the correct Registry in your npm. You can check your npm-Registry by calling `npm get registry` in a shell. This call should return the registry url (https://npm-registry.netrtl.com/). If not, you can set the correct registry by calling `npm set registry https://npm-registry.netrtl.com` in a shell.

Installation

Please make sure, that the project you want to install the player in already uses npm. If not you can initialize npm in your project by calling `npm init -y`.

Install the latest version of player with the following command: `npm install @cbc/videoplayer`.

If you want to install a special version add `@x.x.x` at the end of that command. For example `npm install @cbc/videoplayer@5.0.1` installs the CBCVideoplayer in version 5.0.1. Whereas `npm install @cbc/videoplayer@5.1` installs the latest version of the CBCVideoplayer in version 5.1.x.

Please make sure that the `package.json` file should contain a new entry of the CBCVideoplayer with the specified version. A `package-lock.json` file should exist now as well. This file is important to make sure, that f.e. colleagues are able to install exactly the same versions of all packages you installed. The CBCVideoplayer included. For this reason the `package.json` **and** `package-lock.json` should both be added to your version control system.

The CBCVideoplayer is now available in your JavaScript Code. You can either import it via `import CBCVideoplayer from '@cbc/videoplayer'` or `const CBCVideoplayer = require('@cbc/videoplayer')`.

TL;DR

```
npm install @cbc/videoplayer
```

Loading via CDN

To load the CBCVideoplayer via CDN simply add the following `<script>` to your `<head>`:

```
<script src="https://bilder-a.akamaihd.net/lib/cbc/videoplayer/5/dist/videoplayer-bundle.js"></script>
```

The 5 may be replaced by any available version.

For example the following `<script>` loads the exact version 5.0.1:

```
<script src="https://bilder-a.akamaihd.net/lib/cbc/videoplayer/5.0.1/dist/videoplayer-bundle.js"></script>
```

Or the latest version of 5.1.x:

```
<script src="https://bilder-a.akamaihd.net/lib/cbc/videoplayer/5.1/dist/videoplayer-bundle.js"></script>
```

Polyfills

Please keep in mind that the `videoplayer-bundle.js` includes polyfills for unsupported browser features. If your application already loads polyfills by itself make sure to load the CBCVideoplayer bundle without polyfills included, to make sure that there are no conflicts:

```
<script src="https://bilder-a.akamaihd.net/lib/cbc/videoplayer/5/dist/videoplayer.js"></script>
```

Activation

Before using the CBCVideoplayer it is important to notice that the player contains a whitelist of URLs that are allowed to use that player. Please make sure that your test and production environments are listed in that whitelist.

A mirror of that whitelist can be found here:

Player URLs whitelist

If you need an activation feel free to reach out to the player squad, especially to Rogge, Fabian.

Interface

To integrate the player in your site it has to be initialized first. Use the `create` method on the `CBCVideoplayer` object to do that:

```
CBCVideoplayer.create(videoElementId: string, config: CbcVideoplayerConfig, videoElementId?: string)
```

Configuration

The configuration schema is the following:

```
type CbcVideoplayerConfig = {
    page: { // General settings for the page
        fullscreenElementId: string, // If id is set, player trys to toggle Fullscreen on element
        airPlay: boolean, // If airplay is allowed
        backButton: boolean, // If a back button should be rendered in the top left corner
        country: string, // Country code of the wrapping page
        settings: { // Settings for the controls
            audio: boolean, // If an audio selection should be shown
            subtitles: boolean, // If a subtitle selection should be shown
            videoQuality: [{ // A list of all available video qualities
                limit: integer, // Max bitrate
```

```
                label: string, // Description text
                sublabel: string, // Description text
                isPremium: boolean, // If this quality should only be choosable for premium users
                checked: boolean // If this quality is checked
            }]
        },
        controls: { // URLs for customized controls (see Custom Controls)
            jsUrl: string,
            cssUrl: string
        },
        addableIDsWhitelist: [string] // A whitelist of HTML-IDs that should not be cleared inside the player
container
    },
    tracking: { // Information used for tracking
        offer: string, // The tracking offer
        videoService: string, // Name of the wrapping page
        device: string, // A device descriptor
        privMode: boolean, // If the user has tracking enabled
        display: string, // Device type of the user
        heartbeat: { // Tracking options for heartbeat
            options: {
                beatInterval: integer, // Interval of the heartbeats
                clickEvent: string // The user's click event
            }
        },
        nielsen: {
            vcId: string, // Channel-ID
            clientId: string, // Mediengruppe RTL Deutschland's ID
            sfCode: string, // Use "eu-cert" for testing and "eu" in production
            prod: string, // Use "vc" to activate Beacon Measurement, "" to deactivate
            apId: string // Assigned to channels
        },
        googleAnalytics: {
            googleAnalyticsId: string, // Google Analytics ID
            gaLocation: string, // Individual parameter for the googleAnalytics measurement
            gaReferrer: string,// Individual parameter for the googleAnalytics measurement
            gaTitle: string // Individual parameter for the googleAnalytics measurement
        },
        infOnline: {
            st: string  // Individual parameter for the infOnline measurement
        },
        infOnlineLegacy: {
            st: string // Individual parameter for the infOnline measurement
        },
        chartbeat: any, // Tracking configuration for chartbeat
        nurago: any, // Tracking configuration for nurago
        tagCommander: any, // Tracking configuration for tagCommander
        facebook: any, // Tracking configuration for facebook
        googleAdWords: any, // Tracking configuration for googleAdWords
        googleFloodLight: any // Tracking configuration for googleFloodLight
    },
    features: { // General activation\deactivation of features
        concurrentStream: { // Omit to deactivate streamcheck
            startUrl: string, // URL, to start streamcheck
            heartbeatUrl: string, // URL, to update streamcheck
            stopUrl: string // URL, to stop streamcheck
        },
        bitmovinAnalytics: {
            frontendVersion: string, // The client's frontend version
            backendVersion: string, // The client's backend version
            customOffer: string // Individual offer, besides tracking.offer
        },
        logging: { // Settings for browser's logging
            level: 'TRACE' | 'DEBUG' | 'INFO' | 'WARN' | 'ERROR' // Default: 'INFO'
        },
        streamingErrors: { // Internal error measurements
            deviceId: string // The users device type
        },
        homad: { // Configuration for anti ad block
            enabled: boolean, // If anti ad block is enabled
            clientConfigUrl: string // URL to the anti ad block configuration file
        }
    },
    user: { // Information about the user
        statusCode: integer, // A magic number for the user's state, evaluating to: 'free', 'premium', etc.
        id: string, // Unique user identifier
        personalisationId: string, // The google personalisation ID
        accountPersonalisationId: string, // The accoubt personalisation ID
        hdPlayout: boolean, // If the user should receive a HD playout
        isPremium: boolean, // If the user is premium user
        ovAllowed: boolean, // If the user is allowed to choose the original version
        loggedIn: boolean, // If the user is logged in
        hashedEMail: string, // The hashed user mail
        smartDataId: string, // Tracking parameter
```

```
        activeExperimentNames: string, // Tracking parameter
        variationNames: string, // Tracking parameter
        sessionKey: string, // An ID that is unique for every login with the same account
        jwt: string // A JWT used for further authentication
    },
    isLivestream: boolean, // If the displayed source is a livestream
    unsupportedPlayerConfig: any, // An object, passed through to the Bitmovin Player
    unsupportedAdvertisingConfig: any // An object, passed through to the Advertising Module
}
```

Methods

Further methods on the CBCVideoplayer object

There are 2 more methods on the CBCVideoplayer object besides the `create` function. In the following all 3 methods will be explained in detail:

create

Method signature: `(videoElementId: string, cbcVideoplayerConfigInput: CbcVideoplayerConfig) => Player`

This method creates a new player instance. To do this 2 arguments get passed: 1. A `videoElementId`: HTML ID where the player should be rendered in 2. A `cbcVideoplayerConfigInput`: Details mentioned above

onEvent

Method signature: `onEvent (videoElementId: string, handler: Eventhandler)`

`Eventhandler` signature: `(any: Event) => void`

The `onEvent` function takes a `videoElementId` and a `handler`. The `videoElementId` equals the player container's ID, that was used to create the player. If you have more than one player instance on your page, this method may be used to bind an `Eventhandler` to a special player instance.

onEventUnfiltered

Method signature: `onEventUnfiltered (handler: Eventhandler)`

`Eventhandler` signature: `(any: Event) => void`

This function binds an `Eventhandler` on all existing player instances. If there is only one player instance on the site `onEvent` and `onEventUnfiltered` do the same.

Instance methods

The following methods describe the methods on the `Player` type that is created by a `create` method call.

The `player` type in general:

```
interface Player{
    async loadVideo (loadVideoConfig: CbcLoadVideoConfig): void
    registerPlugin (plugin: Eventhandler): void
    pause (issuer?: string): void
    async play (issuer?: string): void
    async destroy (): void
    getLogFiles (): []Log   // Log: {time: Date, logLevel:string, messages: []string}
    getCurrentTime (): number
    isSupportedPlatform (): boolean
    isSupportedDRMPlatform (): boolean
    toggleCssClass (className: string): void
    isPlaying (): boolean
    seek (): void
    getDuration (): number
    isAd (): boolean
    enterFullscreen (): void
    exitFullscreen (): void
    isFullscreen (): boolean
    setMaxVideoQuality (videoQuality: number): void
    updateMetadata(metadata: Meta): void
    updateContentInfo(contentInfo: ContentInfo): void
    disallowFullscreen (): void
    allowFullscreen (): void
    disabledActions (message?: string, actions: DisabledActions[] = [], pause: boolean = false): void
}
```

async loadVideo

Method signature: `async loadVideo (loadVideoConfig: CbcLoadVideoConfig): void`

This method loads a video into the player. This call is highly customizable why it is handled in detail in *LoadVideo call's configuration*.

Please keep in mind that this method is an `async` function. That means that the execution runs asynchronous. To wait for the resolved function result see Async-Await documentation on Mozilla Developer Network

registerPlugin

Method signature: `registerPlugin (plugin: Eventhandler): void`

`Eventhandler` signature: `(any: Event) => void`

Registers a player plugin. The eventhandler will be called on every occurring event. In some way this method is equal to a call of the `CBCVideoplayer.onEvent` method with the matching `videoElementId`.

pause

Method signature: `pause (issuer?: string): void`

Pauses the player. The `issuer` may be passed optional to display who (user or api) paused the player.

async play

Method signature: `async play (issuer?: string): void`

Starts the player. The `issuer` may be passed optional to display who (user or api) started the player.

Please keep in mind that this method is an `async` function. That means that the execution runs asynchronous. To wait for the resolved function result see Async-Await documentation on Mozilla Developer Network

async destroy

Method signature: `async destroy (): void`

Stops the player and removed it from the DOM. After this function resolved the player instance is not usable anymore.

Please keep in mind that this method is an `async` function. That means that the execution runs asynchronous. To wait for the resolved function result see Async-Await documentation on Mozilla Developer Network

getLogFiles

Method signature: `getLogFiles (): []Log`

`Log`: `{ time: Date, logLevel:string, messages: []string}`

Returns a list of log entries. This log entries will be logged independently from the configured `LogLevel`. This way the player may be configured that it will not log to the browser console, but to a retrievable log file for support cases.

getCurrentTime

Method signature: `getCurrentTime (): number`

Returns the current playback time in seconds.

isSupportedPlatform

Method signature: `isSupportedPlatform (): boolean`

Returns if the videoplayer is compatible to the combination of device and browser.

isSupportedDRMPlatform

Method signature: `isSupportedDRMPlatform (): boolean`

Returns if the playback of DRM content is compatible to the combination of device and browser.

toggleCssClass

Method signature: `toggleCssClass (className: string): void`

Adds or removes a HTML class on the player container.

isPlaying

Method signature: `isPlaying (): boolean`

Returns if the player is currently playing.

getDuration

Method signature: `getDuration (): number`

Returns the total duration (in seconds) of the currently loaded video.

isAd

Method signature: `isAd (): boolean`

Returns if the player currently plays advertising.

enterFullscreen

Method signature: `enterFullscreen (): void`

Starts the fullscreen mode programmatically.

exitFullscreen

Method signature: `exitFullscreen (): void`

Exits the fullscreen mode programmatically.

isFullscreen

Method signature: `isFullscreen (): boolean`

Returns if the player is currently in fullscreen mode.

setMaxVideoQuality

Method signature: `setMaxVideoQuality (videoQuality: number): void`

Sets the user's selected max video quality.

updateMetadata

Method signature: `updateMetadata(metadata: Meta): void`

Update Metadata for tracking.

updateContentInfo

Method signature: `updateContentInfo(contentInfo: ContentInfo): void`

Update ContentInfo for PlayerUI.

disallowFullscreen

Method signature: disallowFullscreen`(): void`

Blocks the function to enter fullscreen mode and ends the fullscreen If the user is currently using it.

allowFullscreen

Method signature: allowFullscreen`(): void`

Allowed to enter fullscreen

disabledActions

Method signature: disabledActions (message?: string, actions: DisabledActions[] = [], pause: boolean = false): void

Disabled player interaction and can shows an overlay with a message
Actions that can be deactivated (DisabledActions): playpause, seek, language, settings

LoadVideo call's configuration

The method signature for `loadVideo` looks like the following: `async loadVideo (loadVideoConfig: CbcLoadVideoConfig): void`

The following schema shows the configuration possibilities for the `loadVideoConfig`:

```
type Constraint = {
  enabled: boolean, // If the Constraint is enabled
  errorText: string // Error text to be shown if the constraint matches
}

type VideoTime = { // General type for passing time
  inSeconds: number
}

type CbcFairplay = { // Uses `user.jwt` (optional), see CbcVideoplayerConfig
  certificateUrl: string
  url: string
}

type CbcPlayready = {
  url: string  // Uses GET-Parameter 'token' in URL or `user.jwt`(optional), see CbcVideoplayerConfig
}

type CbcWidevine = { // Uses `user.jwt` (optional), see CbcVideoplayerConfig
  url: string
}


type CbcLoadVideoConfig = {
    meta: { // Meta information for the video
        id: any, // Video ID
        category: string, // The video's category
        length: VideoTime, // Video length
        title: string, // The video's title
        description: string, // The video's description text
        fsk: string, // FSK text. F.e. "ab 12
        supplier: string, // Supplier of the video content
        genre: string, // The video's genre
        format: string, // The format's name
        previewStart: string, // The preview's start. Format: YYYY-MM-DD hh:mm:ss
        startDate: string, // The video's publishing date. Format: YYYY-MM-DD hh:mm:ss
        createDate: string, // The video's creation date. Format: YYYY-MM-DD hh:mm:ss
        isPayedContent: boolean, // If the video is payed content
        isWebOnly: boolean, // If the video is only available in web
        refPlanningId: number, // individual tracking parameter
        agof: string, // individual tracking parameter
        comment: string, // individual tracking parameter
        ivw: string, // infOnline identification path of the video
        payStatusCode: number, // payStatus code of the video like 'free_justmissed' or 'pay_archive'
        startTypeCode: number, // startType code of the video like 'autoplay', 'replay' or 'userStart'
        recoStart: string // individual tracking parameter
    },
    advertising: { // The ad-playback's general configuration
        privMode: boolean, // If the private mode is enabled
        playAds: {
            preroll: boolean, // If a preroll should be played
            midroll: boolean, // If a midroll should be played
            postroll: boolean // If a postroll should be played
            nonLinear: boolean // If a nonLinear should be played
        },
        midrollOffsets: [] VideoTime, // If a midroll should be played pass the offsets where the midrolls should
play here
        limits: {
            preroll: VideoTime, // How long should a video be to play preroll ads, default is 29
            midroll: VideoTime, // How long should a video be to play midroll ads, default ist 479
            nonLinear: VideoTime, // How long should a video be to play non linear ads, default ist 479
            postroll: VideoTime, // How long should a video be to play postroll ads, default is 29
        },
        specialAds: {
            companionAds: boolean, // If companion ads should be enabled
            vpaidAds: boolean // If vpaid ads should be enabled
        },
        singlePreRoll: boolean, // If only one preroll should be played
        skippAbleAds: boolean, // If make all Ads skippAble
        clips: {
            prerollBumper: boolean, // If there should be a bumper before the preroll
            postrollBumper: boolean, // If there should be a bumper before the midroll
            stationBumper: boolean, // If there should be a bumper before the postroll
            bumperUrls: [] string, // A List of URLs the bumpers will be chosen randomly from
            closerUrls: [] string, // A List of URLs the openers will be chosen randomly from
            openerUrls: [] string // A List of URLs the bumpers will be chosen randomly from
        },
        adCall: {
            category: string, // video zone identification for the adserver
            contentPartner: string, // Individual adcall parameter
            fixParams: [] string, // key-value pairs to attach to the adcall
            tags: [] string, // key-value pairs with user targeting
            defaultTags: [] string, // default targeting as key-value pairs, if no user targeting detected
```

```
            referrerUrl: string // domain name, default: 'protocoll://hostname'
        },
    },
    styling: {
        logo: { // settings for the corner logo
            aspectRatioOld: boolean, //default: false, set true if aspect ratio is 4:3
            url: string, // The URL for the corner logo. Supported formats can be seen here: https://developer.
mozilla.org/en-US/docs/Web/Media/Formats/Image_types
            position: 'topLeft' | 'topRight' | 'bottomCenter' | 'bottomRight', // Where the logo should be
positioned
            basewidth: number // The base with from where the image should be scaled to the fitting size
        }
    },
    contentInfo: {
        productPlacement: boolean, // If an advice about product placement should be shown
        contest: boolean, // If an advice about ? should be shown
        dontCall: boolean, // If an advice 'do not call' should be shown
        fsk: string, // A text that should be shown as FSK note (f.e. 'ab 12')
        title: string, // The title that should be shown on the top left
        description: string, // The subtitle that should be shown on the top left
        format: string // The name of the format that should be shown on the top left
    },
    behavior: {
        autoplay: boolean, // If autoplay is enabled
        muted: boolean, // If the player is muted
        smoothFadeIn: boolean, // Default 'true', enabled smooth video fadeIn and FadeOut at video start
        pauseInactivePlayer : boolean, // Default 'true', enables pausing and playing video when player was left
or entered
    },
    videoSource: {
        poster: string, // URL to the poster image that is shown before the content starts (recommended format: .
jpg)
        streams: {
            dashUrl: string, // URL to the dash manifest
            hlsUrl: string, // URL to the hls manifest
            progressiveUrl: string, // URL to the progressive manifest
            dashHdUrl: string, // URL to the dash HD manifest
            hlsHdUrl: string // URL to the hls HD manifest
            preferredTech: string 'hls' | 'dash' | 'progressive', // Generates the desired stream technology for
the player
        },
        parts: { // May be omitted to disable the part player
            breakpoints: [] VideoTime // Breakpoints where the part player insert a chapter switch
        },
        startTime: VideoTime, // A time where the video should start
        eshOffset: VideoTime, // An offset from the end of the video, when the ESH-Event should be fired
        breakpoints: {
            [name: string]: VideoTime // Breakpoints to be passed // Example: //SIN: { inSeconds: 100 } // The
intro starts after 100 seconds // EIN: { inSeconds: 130 } // The intro ends after 130 seconds
        },
        drm: {
            securityLevel: number, // required securety level, default is 3, 1 is the highest
            preferredTech: string 'PlayReady' | 'Widevine' | 'FairPlay', // Generates the desired drm technology
for the player
            headers: [] string, // Custom headers for license call
            widevine: CbcWidevine | Bitmovin.WidevineDRMConfig, // https://bitmovin.com/docs/player/api-reference
/web/web-sdk-api-reference-v8#/player/web/8/docs/interfaces/drm.widevinemodulardrmconfig.html
            playready: CbcPlayready | Bitmovin.PlayReadyDRMConfig, // https://bitmovin.com/docs/player/api-
reference/web/web-sdk-api-reference-v8#/player/web/8/docs/interfaces/drm.playreadydrmconfig.html
            fairplay: CbcFairplay | Bitmovin.FairPlayDRMConfig // https://bitmovin.com/docs/player/api-reference
/web/web-sdk-api-reference-v8#/player/web/8/docs/interfaces/drm.applefairplaydrmconfig.html

        }
    },
    constraints: { // Constraints that should be enabled or disabled
        backgroundColor: string, // Set background color for error message
        concurrentStream: Constraint, // Whether the player should check for concurrent streams, if so the
errorText will be shown
        drmPlatformIssue: Constraint, // ErrorText shown if the DRM check fails on because the browser or device
does not support DRM
        drmServerIssue: Constraint, // ErrorText shown if the DRM check fails on the server
        adBlocker: Constraint, // Whether the player should check for ad blocks, if so the errorText will be shown
        downscaling: Constraint, // scaling down the size of the video, required to increase user awareness when
using adBlockers
        geoBlocking: Constraint, // Whether the player should check if the content is allowed to be played in the
playback country, if not the errorText will be shown
        platformIssue: Constraint, // Whether the player should check if the platform supports video playback, if
not the errorText will be shown
    }
}
```

Events

In the following you see a list of the most relevant events that will be passed to `Eventhandlers`: A complete list of all available events you can find in Gitlab:

CustomEvents

Event	Description
`onPlayerLoaded`	Fired after the player has been initialised
onPlayerReady	Fired when all sources have loaded additional ads and the player is ready to start playback
onVideoLoad	Fired when the content source request is started
onVideoLoaded	Fired when the content source was loaded
`onSessionStart`	Fired after first video load on this page
`onSessionEnd`	Fired after last video unload (after postroll). In the partplayer onSessionEnd only is thrown after the **last** part or last postroll
`onContentStart`	Fired on **each** content start (except advertising) even in part player mode
`onContentEnd`	Fired on **each** content end (except advertising) even in part player mode
`onRewind`	Fired if the user rewinds
`onFastForward`	Fired if the user fast forwards
`onEnterPlayer`	Fired when the player enters the visible area
`onLeavePlayer`	Fired when the player leaves the visible area
`onChapterSwitch`	Fired when a new part is loaded (partplayer)
onConCurrentStreamDetect	
onDRMServerError	
onGeoBlockingError	
onContentTimeChanged	
`onInfoShow`	Fired when some kind of information is displayed for the user
`onLoadNewContentSource`	Fired when a new content source has been loaded into the (part) player
`onESHOffset`	Fired when the ESH offset is reached (see LoadVideo call's configuration)
`onAdBlockerDetected`	
onReplay	
onUpdateContentInfo	
onUpdateMetaData	
onDestroyed	
onDisabledFullscreenChanged	
onDisabledActions	
onAdFrameChanged	

AdvertisingEvents

Event	Description
`onAdSlotStarted`	Fired when an ad slot started
`onAdSlotComplete`	Fired when an ad slot is finished
onAdClickThru	
onAdError	
onAdStopped	
onAdMuted	
onAdPaused	
onAdVideoComplete	
onAdVideoStart	
onAdPlaying	
onAdSkipped	
onAdSlotStopped	
onAdSlotStart	
onAdStarted	
onAdStart	
onAdUnmuted	
onAdVolumeChanged	
onGeneralError	

ControlsEvent

Event	Description
onSelectChapter	
onMaxVideoQualitySelected	
onBackButtonClicked	Fired when BackButton in UI was clicked
onRePlayButtonClicked	
onElementClicked	

PlayerEvent

Event	Description
onDestroy	
onReady	
onSeek	
onSeeked	
onVolumeChanged	
onVideoQualityChanged	
onVideoPlaybackQuality Changed	
onUnmuted	

onPlay	
onPlaybackFinished	
onPlaying	
onSourceLoaded	
onSourceUnloaded	
onStallEnded	
onStallStarted	
onPaused	
onMuted	
onFullscreenEnter	
onFullscreenExit	
onError	

Ad Errors

The following list shows all possible errors that can occur while playing an advertising:

ErrorCode	Description	When
100	XML parsing error.	Pre-Ad
101	VAST schema validation error.	Pre-Ad
102	VAST version of response not supported.	Pre-Ad
200	Trafficking error. The video player received an ad type that it was not expecting and/or cannot display.	Pre-Ad
201	Video player expecting different linearity.	Pre-Ad
202	Video player expecting different duration.	Pre-Ad
203	Video player expecting different size.	Pre-Ad
300	General wrapper error.	Pre-Ad
301	Timeout of VAST URI provided in wrapper element or of VAST URI provided in a subsequent wrapper element. (RI was either unavailable or reached a timeout as defined by the video player.)	Pre-Ad
302	Wrapper limit reached, as defined by the video player. Too many wrapper responses have been received with no inLine response.	Pre-Ad
303	No ads VAST response after one or more wrappers. This also includes the number of empty VAST responses from fallback.	Pre-Ad
400	General linear error. The video player is unable to display the linear ad.	Pre-Ad
401	File not found. Unable to find linear/mediaFile from URI.	Pre-Ad
402	Unable to download or timeout of MediaFile URI.	Pre-Ad
403	Could not find a media file that is supported by this video player, based on the attributes of the MediaFile element.	Post-Ad
405	Problem displaying a media file. Video player found a MediaFile with supported type but couldn't display it. MediaFile may include: unsupported codecs, different MIME type than MediaFile@type, unsupported delivery method, etc.	Post-Ad
406	A mezzanine file was required, but not provided.	Pre-Ad
407	The mezzanine file was downloaded for the first time, so the ad did not serve.	Pre-Ad
408	The ad returned in the VAST response was rejected.	Pre-Ad
409	The interactive creative defined in the InteractiveCreativeFile node was not executed.	Pre-Ad
410	The code referenced in the Verification node was not executed.	Pre-Ad
500	General NonLinearAds error.	Pre-Ad

501	Unable to display non-linear ad because creative dimensions do not align with creative display area (in other words, the creative dimension was too large).	Pre-Ad
502	Unable to fetch NonLinearAds/NonLinear resource.	Pre-Ad
503	Could not find NonLinear resource with supported type.	Pre-Ad
600	General CompanionAds error.	Pre-Ad
601	Unable to display companion because creative dimensions do not fit within the companion display area (in other words, space was not available).	Pre-Ad
602	Unable to display required companion.	Pre-Ad
603	Unable to fetch CompanionAds/Companion resource.	Pre-Ad
604	Could not find Companion resource with supported type.	Pre-Ad
900	VAST 2 error.	Pre-Ad
901	General VPAID error.	Post-Opportunity
1000	HomadPenalty	Pre-Ad

Custom Controls

As already mentioned in the *Configuration* section the controls can be customized by passing an url for the custom-control JavaScript (`CbcVideoplayerConfig.page.controls.jsUrl`) and an url for the custom-control CSS (`CbcVideoplayerConfig.page.controls.cssUrl`).

By default the bitmovin's base controls are used. The player comes with longform controls prepared.

As with the player itself the controls can be loaded via npm or CDN.

Custom Controls via NPM

The latest controls can be installed via `npm i @cbc/videoplayer-controls-longform@1`.

The JavaScript file can then be found under `node_modules/@cbc/videoplayer-controls-longform/dist/js/bitmovinplayer-ui.min.js`. The CSS file under `node_modules/@cbc/videoplayer-controls-longform/dist/css-tvnow/bitmovinplayer-ui.min.css`.

Please make sure to expose those files via your server and pass that URL to the config as described above.

Custom Controls via CDN

As with the player the controls can also be found on a CDN.

The URL for the JavaScript file is: https://bilder-a.akamaihd.net/lib/cbc/videoplayer-controls-longform/1/dist/js/bitmovinplayer-ui.min.js.

And the URL for the CSS file is: https://bilder-a.akamaihd.net/lib/cbc/videoplayer-controls-longform/1/dist/css/bitmovinplayer-ui.min.css.

Both URLs can now be passed to the configuration.

The config might look like this:

```
const config = {
  ...
  page: {
    ...
    controls: {
      jsUrl: "https://bilder-a.akamaihd.net/lib/cbc/videoplayer-controls-longform/1/dist/js/bitmovinplayer-ui.min.
js",
      cssUrl: "https://bilder-a.akamaihd.net/lib/cbc/videoplayer-controls-longform/1/dist/css/bitmovinplayer-ui.
min.css"
    },
    ...
  },
  ...
}
```