HTML Video

ON THIS PAGE

Note

The documentation below applies to firmware versions 7.1.x and later.

You can use `<video>` elements to play streaming video (HLS, UDP, RTP, RTSP) and local video files. You can also display HDMI input on the XD1230, XD1132, 4K1142, or XT1143 (and RF Input on the XD1230).

Streaming Video

Streaming video functions similar to any standard HTML page. The `Load()` and `Play()` methods should be called on the video element to ensure it plays; this is especially true if the same `<video>` element is used to display different content.

Note that pause/resume commands currently work for HLS streams only.

STREAMING PARAMETERS

You can configure streaming video using a number of unique BrightSign attributes. These attributes can be overridden by parameters that are included in the streaming URL.

Example (HTML)

```
<video src="udp://239.192.1.1:5004" x-bs-stream-timeout="0">
```

Example (JavaScript)

```
videoPlayer.setAttribute("x-bs-stream-low-latency", "1");
```

The following streaming parameters are supported:

- `[int] x-bs-stream-timeout`: The length of time (in milliseconds) to wait for a stream to start before abandoning playback. Passing zero indicates that the video player should wait indefinitely. The default timeout value is 5000ms for HTTP streams and 3000ms for all other streams.
- `[int] x-bs-stream-latency`: The amount of deviation (in milliseconds) from the default latency value: For example, a value of -500 will reduce the latency by half a second; a 500 value will increase the latency by half a second; and a 0 value will specify the default latency. Specifying a negative value will not change the buffer size; instead, it will give the buffer less time to fill up before playback begins. Usable values extend to approximately -750, though this value may differ depending on the network environment. Reducing the latency too much will result in obvious playback stutter.
- `[int] x-bs-stream-fade-in`: The fade-in length (in milliseconds) for streams. By default, non-HTTP streams have a fade value of 1500ms if they contain video and 4000ms if they contain audio only. HTTP streams have a default fade value of 0ms.
- `[int or boolean] x-bs-stream-low-latency`: Low-latency mode for RTSP streams. Setting this parameter to `true` (1) will achieve the lowest possible latency for a stream, but at a reduced maximum bitrate–which is often useful for IP camera streams. This parameter is set to `false` (0) by default.
- `[int] x-bs-intrinsic-width`: The intrinsic width of the source video
- `[int] x-bs-intrinsic-height`: The intrinsic height of the source video

Intrinsic Video Size

The following order of precedence is used to determine the intrinsic size of a video:

1. The `x-bs-intrinsic-width` and `x-bs-intrinsic-height` attributes
2. The size of the image specified by the `poster` attribute
3. The `width` and `height` attributes of the `<video>` element
4. The size as reported by the WebMediaPlayer
5. The CSS default size, as defined by W3C: 300x150

If only the `width` and `height` attributes of a `<video>` element are specified, the rendering engine assumes that the intrinsic size of the source video is the same as the `<video>` element size, so it will scale the video to fill the video element, ignoring the aspect ratio of the source video if necessary. Additionally, if the `width` and `height` attributes of a `<video>` are not specified and the size is not available from the sources (for example, if the stream is still loading), the video will initially default to a resolution of 300x150.

If you know the size or aspect ratio of a source video beforehand, you can use the `x-bs-intrinsic-width`/`x-bs-intrinsic-height` attributes to ensure the aspect ratio of the source video is maintained when scaled to fit a `<video>` element: For example, if the video window is oriented as portrait HD (`width=1080, height=1920`), specifying an intrinsic width and height of 1920x1080 ensures that landscape HD videos will scale to fit while maintaining their aspect ratio and letter-boxing appropriately. Likewise, videos will scale to the intrinsic width/height immediately if the `<video>` element lacks `width` and `height` parameters, rather than possibly defaulting to 300x150 for a short time.

YOUTUBE

BrightSign players support YouTube videos and playlists.

MEDIA SOURCE EXTENSIONS

BrightSign players currently do not support Media Source Extensions (MSE).

HLS LIVE STREAMING

BrightSign players support HLS live streaming, but large playlists (which usually result from the server delivering a DVR playlist rather than a LIVE playlist) will cause performance issues.

HDMI Input

You must input the video source using the following URI: “tv:brightsign.biz/hdmi”. Note that you *cannot* substitute another host URL for “brightsign.biz”.

Example

```
<video width="320" height="240">
       <source src="tv:brightsign.biz/hdmi">
</video>
```

RF Input

You must input the video source using the following URI: “tv:brightsign.biz/vc/n”, where “n” is the virtual channel number attained via channel scan. Note that you *cannot* substitute another host URL for “brightsign.biz”. Before initializing a `<video>` element with RF input, you must perform a channel scan to initialize the virtual channel database in the registry.

Example

```
<video width="320" height="240"
       <source src="tv:brightsign.biz/vc/23.1">
</video>
```

Multiple Video Elements

By default, XT, 4K, and XD players support a maximum of two active `<video>` elements at any time (HD/LS models only support one). You can have more than this amount of `<video>` elements on a page as long as additional `<video>` elements have their `src` attribute set to an empty string. By modifying the `src` string, you can enable and disable `<video>` elements on the page.

You can enable Mosiac Mode (using the `SetDecoderMode()` JavaScript method) to increase the minimum allowed number of `<video>` element s.

Small Videos

BrightSign players do not support videos that are less than 64 pixels in width or height. However, a video can be scaled down beyond this limit by making the `<video>` element smaller than 64x64. To get the desired downscaling behavior, ensure that the `<video>` element does not have the `viewmode="scale-to-fill-and-crop"` attribute.

View Mode

In firmware versions 7.0.x and later, HWZ video scales to fit the aspect ratio of a `<video>` element (i.e. `"scale-to-fill"`). In earlier versions of firmware, `<video>` elements maintain the aspect ratio of the source video (i.e. `"scale-to-fit"`). This default behavior can be modified using the `viewmode` attribute; however, we now recommend using standard CSS methods (e.g. `object-fit`) to modify the view mode behavior of a `<video>` element.

- `viewmode="scale-to-fill"`: Scales the video to fill the window. The aspect ratio of the source video is ignored, so the video may appear stretched.
- `viewmode="scale-to-fit"`: Letterboxes the video.
- `viewmode="scale-to-fill-and-crop"`: Scales the video to fill the window. The aspect ratio of the source video is maintained, so the video may be cropped.

HWZ Video

HTML `<video>` elements can have an optional “hwz” attribute. When “hwz” is enabled, the video is rendered as a video element, ensuring the highest possible frame rate and video quality. When "hwz" is disabled, the video is rendered as a graphics element.

The "hwz" attribute can equal either “off” or “on”.

Example

```
<video src="example_movie.mp4" hwz="on">
```

The "hwz" attribute is disabled by default, so it must be enabled on individual `<video>` elements as shown above. Alternatively, you can enable "hwz" for all videos in an HTML widget using one of the following methods:

- **BrightAuthor**: Check the **Enable native video plane playback** box in the HTML5 state.
- **BrightScript**: Call `SetHWZDefault("on")` on the *roHtmlWidget* instance.

HWZ WITH CSS

Videos that have “hwz” enabled are not compatible with all CSS transforms. We recommend testing a manipulated "hwz" video thoroughly before using it in a production environment.

Z-ORDERING HWZ VIDEO

In firmware versions 5.0.x and later, enabling the "hwz" attribute places the `<video>` element in front of all text and images. This is the case regardless of whether the graphics are part of the HTML page or part of another zone in BrightAuthor.

If you want to customize the z-ordering of an "hwz" `<video>` element, you can specify one of the following:

- `hwz="z-index:2"`: Places the video in front of all graphics, as well as a second video element.
- `hwz="z-index:1"`: Places the video in front of all graphics (the default setting).
- `hwz="z-index:0"`: Disables "hwz" mode completely.
- `hwz="z-index:-1"`: Places the video behind all graphics.
- `hwz="z-index:-2"`: Places the video behind all graphics, as well as a second video element.

TRANSFORMING HWZ VIDEO

In firmware versions 5.1.x and later, you can add the optional "transform" parameter to the "hwz" attribute to rotate or mirror the `<video>` element . The `z-index:` parameter must also be specified for the transform to work. The "transform" parameter can be assigned the following values:

- `identity`: No transformation (default behavior)
- `rot90`: 90 degree clockwise rotation
- `rot180`: 180 degree rotation
- `rot270`: 270 degree clockwise rotation
- `mirror`: Horizontal mirror transformation
- `mirror_rot90`: Mirrored 90 degree clockwise rotation
- `mirror_rot180`: Mirrored 180 degree clockwise rotation
- `mirror_rot270`: Mirrored 270 degree clockwise rotation

Note

Multiple HWZ video tag extensions are separated with a semicolon. A semicolon should also be appended to the final parameter.

Example

```
 // Video rotated 180 degrees and behind graphics layer.
<video src="example_movie.mp4" hwz="z-index:-1; transform:rot180;">
```

FADING HWZ VIDEO STREAMS

The "fade" parameter allows you to control the fading behavior between streaming videos in a `<video>` element (this setting does not apply to local video). The "fade" parameter accepts the following values:

- `auto`: If the video player is currently not showing anything (i.e. it hasn't played anything yet or the previous loaded video was cleared), the next video will fade in. If the video player is currently playing video, is paused, or is stopped without being cleared, the next video will not fade in. This is the default behavior.
- `always` : When a video ends, the video window will go black. The new video will then fade in.
- `never`: Videos transition without fade effects.

HWZ VIDEO TRANSPARENCY EXTENSIONS

If "hwz" is enabled for a `<video>` element, the video window can also support luma and chroma keys for video transparency. The `z-index:` parameter must also be specified for transparency to work. The luma and chroma keys are specified as follows:

- `luma-key:[HEX_VALUE]`
- `cr-key:[HEX_VALUE]`
- `cb-key:[HEX_VALUE]`

Example

```
 // Video on video layer, in front of graphics layer, with luma keyed
video.
<video src="example_movie.mp4" hwz="z-index:1; luma-key:#ff0020;">
```

Video Decryption

The HTML `<video>` tag can be used to decrypt video files and IP streams. Streaming decryption is currently only supported with the UDP protocol and the HTTP protocol (when HTTP is paired with an MPEG2 transport stream). If using a UDP multicast MPEG2 transport stream, one of the elemental streams should provide the PCR to the player.

The `<video>` tag supports the `EncryptionAlgorithm` and `EncryptionKey` methods:

- `EncryptionAlgorithm`: The file-encryption algorithm. The following are the current options:
  - "AesCtr": The AES algorithm in CTR mode
  - "AesCtrHmac": The AES algorithm in CTR mode with HMAC
  - "TsAesEcb": The AES algorithm in ECB mode (e.g. with a Harmonic Prostream). This algorithm is currently used for streaming encryption/decryption.
  - "TsAesCbcRbt": The AES algorithm in CBC mode with residual block termination. This algorithm is used for streaming encryption /decryption.
- `EncryptionKey`: A byte array consisting of 128 bits of key. If the encryption algorithm is AES-CTR or AES-CTR-HMAC, this is followed by 128 bits of IV.

Example (HTML)

```
<video src="udp://239.192.1.59:5000" EncryptionAlgorithm="TsAesEcb"
EncryptionKey="01030507090b0d0f00020406080a0c0e">
```

Important

The video player no longer accepts "{A|A}" AES encryption keys (i.e. where the top and bottom 64 bits of the key are identical).

Example (JavaScript)

```
var player = document.getElementById("my_video_player");
player.setAttribute("EncryptionAlgorithm", "TsAesEcb");
player.setAttribute("EncryptionKey", "01030507090b0d0f00020406080a0c0e");
```

Video Stream Parsing

The following optional attributes can be included in an HTML `<video>` tag: "preferredvideo", "preferredaudio", and "preferredcaptions". If multiple video, audio, or data streams are encapsulated in the video input, these attributes allow you to determine which stream to use. For example, if a video may contain English and Spanish audio tracks, you can use the "preferredaudio" attribute to specify that the Spanish track should be played if it exists, with the video defaulting to English otherwise.

Preferred streams are chosen by matching the supplied text patterns against the textual description of the stream:

1. Each attribute ("preferredvideo", "preferredaudio", or "preferredcaptions") is a semicolon-separated list of templates.
2. Each template is a comma-separated list of patterns.
3. Each pattern is a `[field_name]=[field_value]` pair that is matched directly against the stream description.

VIDEO STREAMS

Each template in a "preferredvideo" attribute can contain the following patterns:

- `pid=[integer]`: The packet identifier (PID) of the video stream you wish to display
- `program=[integer]`: The program number of the video stream
- `codec=[video_codec]`: The preferred video codec, which can be any of the following:
  - `MPEG1`
  - `MPEG2`
  - `MPEG4Part2`
  - `H263`
  - `H264`
  - `VC1`
  - `H265`
- `width=[integer]`: The preferred video width
- `height=[integer]`: The preferred video height
- `aspect=[float(x.yy)]`: The preferred aspect ratio of the video stream as a floating-point number with two fractional digits.
- `colordepth=[integer]`: The preferred color depth of the video.

Example

```
preferredvideo="pid=7680, codec=H264, width=1280, height=720, aspect=1.78,
colordepth=8;;"
```

AUDIO STREAMS

Each template in a "preferredaudio" attribute can contain the following patterns:

- `pid=[integer]`: The packet identifier (PID) of the audio stream you wish to play
- `program=[integer]`: The program number of the audio stream
- `codec=[audio_codec]`: The preferred audio codec, which can be any of the following:
  - `MPEG`
  - `MP3`
  - `AAC`
  - `AAC-PLUS`
  - `AC3`
  - `AC3-PLUS`
  - `DTS`
  - `PCM`
  - `FLAC`
  - `Vorbis`
- `channels=[integer]`: The preferred number of audio channels (from 1 to 8)
- `freq=[frequency]`: The preferred sample frequency of the audio track, which can be any of the following:
  - `32000`
  - `44100`
  - `48000`

- `lang=[language]`: A code that determines the preferred language of the audio track (e.g. `eng`, `spa`). The language codes are specified in the ISO 639-2 standard.
- `type=[audio_type]`: The preferred audio type, which can be one of the following:
  - `Main audio`
  - `Clean effects`
  - `Hearing impaired`
  - `Visual impaired commentary`

Example

```
preferredaudio="pid=4192, codec=AC3, channels=5, freq=48000, lang=eng,
type=Main audio;;"
```

SUBTITLE AND CAPTION STREAMS

Each template in a "preferredcaption" attribute can contain the following patterns:

- `pid=[integer]`: The packet identifier (PID) of the caption stream you wish to play
- `type=[subtitle_type]`: The encoding standard of the subtitles. This value can be one of the following:
  - `CEA708`: If the CEA-708 standard is not present, the subtitle_type will default to CEA-608 (if it is present).
  - `CEA608`
  - `DVB`
- `lang=[language]`: A code that determines the preferred language of the subtitles (e.g. `eng`, `spa`). The language codes are specified in the ISO 639-2 standard.
- `service=[integer]`: The preferred service number of the caption stream

Example

```
preferredcaptions="pid=0, type=Cea708, lang=eng service=1;;"
```

PATTERN MATCHING BEHAVIOR

Note the following when matching templates to stream descriptions:

- For a template to match a stream description, every pattern within the template must match.
- The first listed template to match the stream description (if any) will be used.
- An empty template string will match any stream description.
- All value comparisons are case-insensitive.
- Numerical values must match the stream description exactly (without leading zeroes). For example, the pattern `pid=016` will never match the stream PID value of 16.
- To indicate logical negation, apply the "!" exclamation mark to the beginning of a pattern. For example, specifying `preferredvideo="!codec=H265"` will match only streams that are not encoded using H.265.
- Apply the ">" greater-than symbol before an integer to indicate that, for a successful match, the value in the stream description must be *greater than* the value following the symbol. For example, specifying `preferredvideo="width=<1921,height=<1081"` will match only videos that are no larger than full-HD.
- Apply the "<" less-than symbol before an integer to indicate that, for a successful match, the value in the stream description must be *less than* the value following the symbol.

FURTHER EXAMPLES

The following template list contains three patterns: `lang=eng`, `lang=spa`, and an empty string. The first pattern specifies an English language channel; if the English channel does not exist, the second pattern specifies a Spanish language channel. The third pattern specifies any other channel if the first two don't exist (the empty string matches anything).

```
preferredaudio="lang=eng;lang=spa;;"
```

Since the following template list is empty, no captions are specified. This can be used to disable captions altogether.

```
preferredcaptions=""
```

The following template list contains an empty string template. Since an empty template matches anything, the first video stream encountered will be played. This is the default behavior of all attributes.

```
preferredvideo=";"
```

The following template list specifies a 48KHz audio stream if there is one; otherwise, no audio stream will be played. Observe that the list is not terminated with a semicolon; in this case, the semi-colon is implicitly supplied.

```
preferredaudio="freq=48000"
```

The following template list contains two templates. Note that all patterns within a template must match the stream description for the entire template to match. In this example, an AAC-encoded English track is preferred; an MP3-encoded English track is designated as the second option; and any track will be chosen if neither template is matched.

```
preferredaudio="codec=aac,lang=eng;codec=mp3,lang=eng;;"
```

Video Track Switching

The BrightSign media playback framework does not support dynamic switching among tracks via HTML5 video/audio. However, it does support querying track information using the Tracklist API. The following steps outline how to select tracks using this API.

1. Create a video element in JavaScript that is not attached to the DOM (i.e. that is invisible).
2. Set the `onloadedmetadata` event listener.
3. Set the `src` of the media URL.
4. Call `load()` on the HTML5 video element. This will trigger the `loadedmetadata` event.
5. Read the track information from the event.
6. Create a new HTML5 video player and use the track information to set the `preferredAudio`, `preferredVideo`, and `preferredSubtitle` parameters (described above).
7. Call `load()/play()` on the video element.
8. Append the video element to the DOM.

Example

```
<!DOCTYPE html>
<head>
  <script>
var video;

function playTrackNext()
{
  // Switch to audio/video 1
```

```
    var pid = video.videoTracks[1].id;
    var audiopid = video.audioTracks[1].id;
    video.src = "";                            // release the video element.
    video = document.createElement('video');   // create a new video element.
    video.src = "http://brightsign.biz/example.ts";
    video.hwz = "on";
    video.preferredvideo = "pid=" + pid;       // use the params we gathered
  from previous video load.
    video.preferredaudio = "pid=" + audiopid;
    video.load();
    video.play();
    document.getElementById("videoarea").appendChild(video);
  }


  function runTest()
  {
    // Initialize testing bs/javascript bridge.
    bs.start();
    // Initial load of media to gather track information
    video = document.createElement('video');
    video.onloadedmetadata = playTrackNext; // play track 0
    video.src = "http://brightsign.biz/example.ts";
    video.hwz = "on";
    video.load(); // will trigger loadedmetadata event once the video is
  loaded.
  }
    </script>
  </head>
  <html>
    <body bgcolor="#E6E6FA" onload="runTest()">
      <h1>Video Audio Tracks Test Page</h1>
      <div id="videoarea"></div>
  </html>
```

Audio Routing <audio>/<video> Elements

BrightSign players have unique audio attributes for `<audio>`/`<video>` elements. These allow you to control how the audio is routed through the device outputs:

- `Pcmaudio`: PCM audio
- `Compaudio`: Compressed audio
- `Multiaudio`: Multi-channel audio

Each attribute can be passed the following values, which determine where the audio will be routed:

- `“none”`
- `“hdmi”`
- `“usb”`
- `“spdif”`
- `"analog"`
- `“analog:N”` (N specifies the port enumeration, which is useful for models with multiple analog-audio ports; you can also use `"analog:1"` to specify the analog output on a model with a single analog-audio port)

Note

If you don't assign any audio attributes to an `<audio>`/`<video>` element, then the audio will be routed to all audio outputs, along with any other audio that is currently playing.

Example 1

```
<video src="example_movie.mp4" width="512" height="400" pcmaudio="hdmi"
autoplay>
  Your browser does not support the video tag.
</video>
```

Example 2

```
<video src="example_movie.mp4" width="512" height="400" compaudio="hdmi;
usb"autoplay>
  Your browser does not support the video tag.
</video>
```