BrightSign®

HTML5: DEVELOPER'S GUIDE

BrightAuthor Software Version: 4.4.0.x
BrightSign Firmware Version: 6.1.x

BrightSign, LLC. 16780 Lark Ave., Suite B Los Gatos, CA 95032 | 408-852-9263 | www.brightsign.biz

TABLE OF CONTENTS

INTRODUCTION

The BrightSign 4Kx42, XDx32, XDx30, and HDx22 models allow you to publish content using HTML5. You can use a single, full-screen page of HTML5 content as a presentation, or you can display HTML5 within a BrightAuthor zone, along with other multimedia content. This feature greatly increases your creative options when using BrightSign players.

These are not general-purpose instructions for writing HTML5 code or using HTML5 publishing software, but you can use this guide to get the most out of your HTML5 presentations and ensure that your HTML5 content works seamlessly with your BrightSign players.

BEST PRACTICES

This section is intended for anyone who wants to use HTML5 content in BrightAuthor presentations; there is also an advanced section at the end of this guide for users who wish to write custom BrightScript code.

Content Restrictions

The following list outlines content restrictions associated with HTML5 pages:

- A BrightSign player is *not* intended for use as a general-purpose web browser. It is best to think of BrightSign units as HTML5 players with interactive capabilities, not web-surfing tools.
- BrightSign players do not support Flash content. Any HTML5 pages that have embedded flash content will not display correctly. Most Flash authoring applications, including the Adobe Creative Suite, have tools that allow you to export flash content as HTML5.
- BrightSign players do no support Media Streaming Extensions (MSE).
- 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 a `viewmode="scale-to-fill-and-crop"` attribute.
- 4Kx42 models can display HTML pages using 4K video modes (3840x2160 or 4096x2160), but these players do not support a native 4K HTML graphics plane: Pages must be specified as 1920x1080 (or 2048x1080 for DCI 4K); they can then be upscaled to 4K.
- Images larger than 2048x1080x32bpp will not be displayed by default. If a 4K video mode is used, the player will upscale images accordingly. This default limit can be increased using the *roVideoMode.SetImageSizeThreshold()*.

Creating HTML5 Pages

Follow these steps when creating HTML5 pages:

1. Make sure the HTML5 page has the same aspect ratio as your signage display. If you are using HTML5 content in a BrightAuthor zone that is smaller than the screen, fit the page to the same aspect ratio as the zone.
2. Use a master Div aligned to 0,0 when building an HTML page. This will ensure correct alignment.

3. When creating an HTML5 site, make sure that all webpage assets (image files, video files, etc.) are contained within the same folder on your local disk. This folder is a "site folder," meaning that all assets in this folder and its subfolders will be used in the production of the webpage. If these assets are not in the folder, they will not display when the project is published.
4. You can test the layout and appearance of an HTML5 page locally by opening it with Google Chrome, which uses a similar browser-rendering engine to BrightSign players.
5. If you want to publish resource-intensive presentations (e.g. `<video>` elements or multiple transforms) using HTML5, we recommend using a Class 10 (10Mb/s) SD card.

BrightSign Extensions

The BrightSign implementation of the Chromium engine includes several platform-specific extensions. Extensions for <video> elements are covered in the HTML Video chapter.

GPU Rasterization

You can use GPU rasterization to improve HTML graphics performance in most cases, though this method increases GPU memory usage substantially. You can use either of the following methods to enable GPU rasterization:

- **HTML**: Add the following meta tag to your HTML page(s): `<meta name="viewport" content="width=device-width, minimum-scale=1.0">`
- **BrightScript**: Call `ForceGPURasterization(true)` on the *roHtmlWidget* instance used to display the page.

Note: *In either case, the rendering engine may not enable GPU rasterization if it determines that the page is not compatible.*

Optimized Image Rendering

The `image-rendering` CSS property can be assigned the `optimizeSpeedBS` value. Using this value ensures that Chromium uses lower-quality but faster bilinear filtering when scaling images to 50% or less. We recommend using this value with pages that scale a lot of images at runtime.

Animations and Add-on Libraries

This section outlines support for animations and add-on libraries for the Chromium engine on BrightSign players.

JavaScript Animations

Animations that use JavaScript timers, including the JQuery .animate() library, do not make efficient use of GPU resources and are not accurate enough to achieve smooth animations. For this reason, we recommend using CSS animations whenever possible. The JQuery Transit library uses CSS animations and provides an API similar to the .animate() library.

BrightSign players support the OpenGL API for JavaScript (i.e. WebGL).

Vector Animations

The SVG protocol should be used to specify vector animations.

Canvas Animations

Bitmap animations display smoothly when they are 1/3 or less of a 1080p HTML canvas. Setting the canvas size to 720p allows for larger high-quality animations to occupy the screen.

CSS Transforms

All CSS transforms should be specified as WebKit transforms. When performing a transform on a `<div>` or graphics element, you should not specify the transform in-line.

Animations that use the "top" and "left" properties are rendered using the CPU. We recommend using the `translate()` and `translate3d()` methods instead to offload work onto the GPU, ensuring smoother animations.

The following code shows an example of an effective CSS transform for a BrightSign player:

```
<style>

                .flipme{
                -webkit-animation-name:flipon;
                -webkit-animation-fill-mode:forwards;
                -webkit-animation-iteration-count:1;
                -webkit-animation-duration:2s;
                }


@-webkit-keyframes flipon
{
0%   {-webkit-transform:rotateY(0deg);}
30%   {-webkit-transform:rotateY(-90deg);}
100% {-webkit-transform: rotateY(360deg);}
}

</style>
```

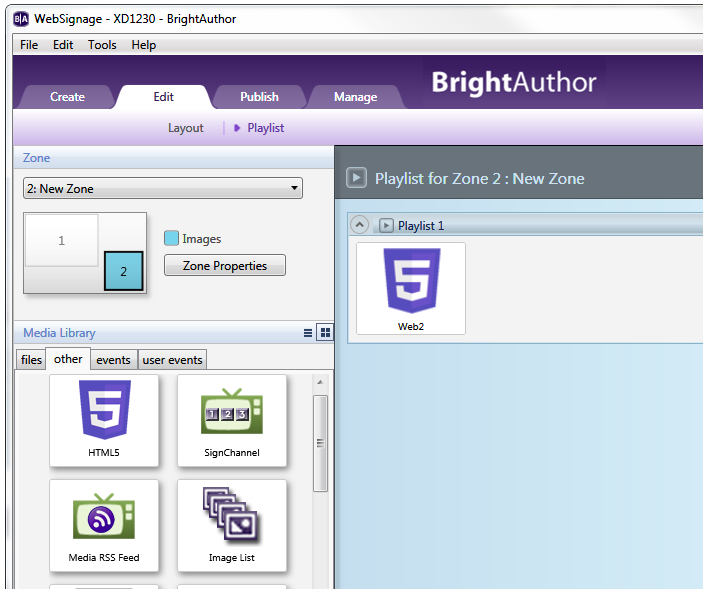
Push Technology

The WebSocket protocol and long polling technique have been tested and proven to work on BrightSign players.

Using HTML5 Pages in BrightAuthor

These are the general rules for using HTML5 content in BrightAuthor:

Zones



- You can have multiple HTML5 zones in a BrightAuthor presentation.
- HTML5 content can be inserted into a Video or Images zone or an Images Zone. You cannot use HTML5 content in a Video Only zone.
- The dimensions of the HTML background/page must match the size of the zone in BrightAuthor: You cannot use background image scaling to fit zones of different sizes.

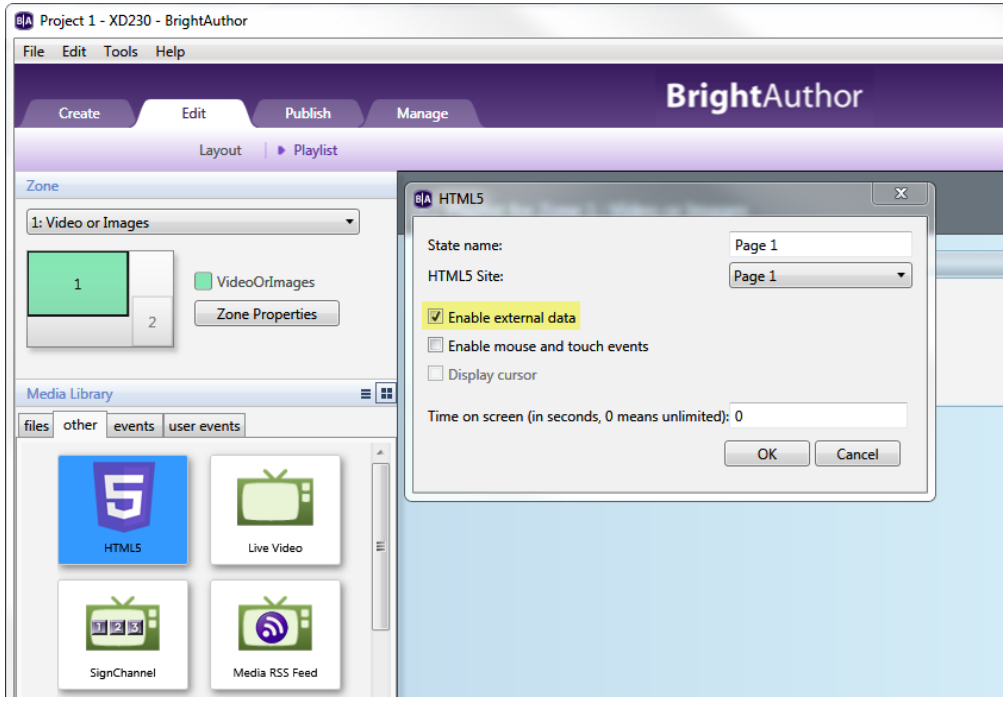
Z-Ordering

- HTML5 content will show at the highest Z-level of graphics zones, meaning that an HTML5 zone will cover all other zones that contain images and text. This behavior does not extend to touch screen events.
- HTML5 content can be placed in front of or behind zones containing video content, depending on the **Graphics plane z position** setting of the zone containing the HTML content (configurable in the **Edit > Layout** tab). If the HTML5 page contains video and the **Enable native video plane playback** option is enabled in the HTML5 state, the HTML5 page will *always* display over other video zones.

Content Sourcing

- HTML5 content can originate from a remote server, a local server, or the local storage (SD card) of the player. HTML5 content can also be downloaded onto the local storage from the BrightSign Network.

- If your HTML5 content relies on assets from multiple locations, make sure to check the **Enable external data** box when creating or editing an HTML5 state.



Using Custom Fonts

- When creating an HTML5 state in BrightAuthor, click the **Add Font** button to add custom True Type Font (*.ttf*) files to the HTML page. This feature works for both local and remote HTML content.
- When the presentation is published, the font file(s) will act as though they are located in the same file directory as the *index.html* file (i.e. they can be accessed with the standard `font-family` attribute in HTML/CSS).

Exporting an HTML5 Presentation

- Exporting a presentation that contains multiple HTML5 pages also exports in full all asset folders associated with those pages. If your pages share common asset folders, the entire contents will be duplicated multiple times. This

can become problematic if your asset folders contain large content files, so you may need to prune and/or rearrange asset folders that are duplicated after export.

Caching and Storage

BrightAuthor allows you to configure common browser caching and storage functions on the player. To enable these functions, open your BrightAuthor HTML presentation and navigate to **Edit > Preferences > Storage** and check **Limit storage space by function**. You can allocate space for the following functions:

- **HTML data**: The amount of space dedicated to the HTML application cache
- **HTML local storage**: The amount of space dedicated to JavaScript variables and data

Note that, if you select **Specify absolute size**, it is possible to specify a combined set of segments that is larger or smaller than the absolute size of your storage device. If you select **Specify percentages**, you will need to ensure that the percentages add up to 100% (which is equivalent to the absolute size of the local storage on the player).

Note: *The **HTML local storage** feature is only available in BrightAuthor versions 4.1.1.12 and later. A similar feature is available in the **File > Presentation Properties > HTML** section in earlier versions.*

Scrollbars

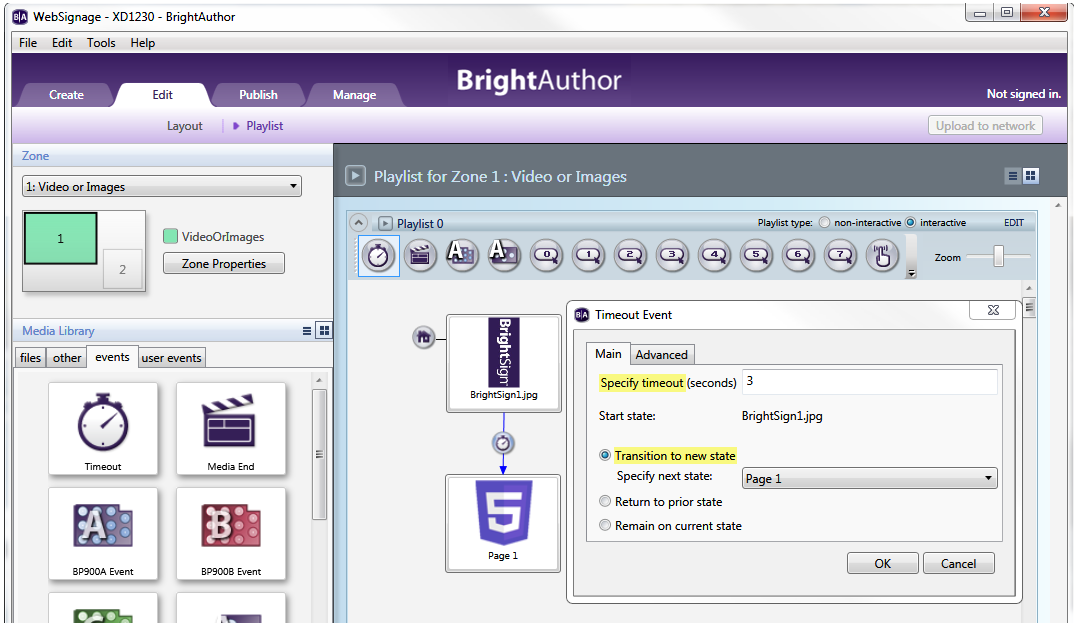
Browser scrollbars are disabled by default. They can be enabled by calling `EnableScrollbars(true)` on an *roHtmlWidget* instance in BrightScript.

Disguising Network Latency

When the BrightSign player loads HTML content from a URL, there may be a delay based on network latency. You can add a preload image to sidestep this issue.

Note: *This solution is not necessary if all of your HTML5 assets are located on the local storage (SD card) of the player.*

1. Drag and drop an image file from your media library.
2. Check the **Set as initial state** box while editing the image state.
3. Add a Timeout event to this image.
4. Specify the timeout for three seconds (or more if desired).
5. Set the Timeout event to transition to the HTML5 state.



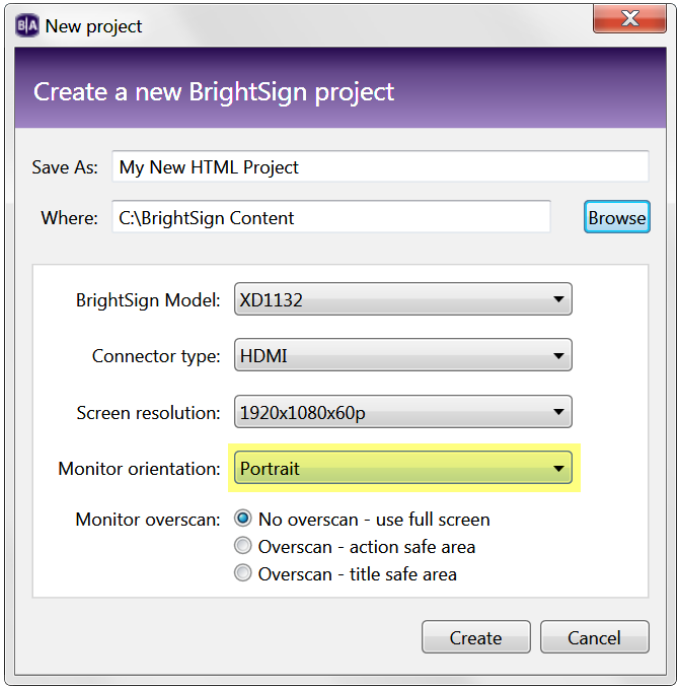
Creating Pages with Portrait Orientation

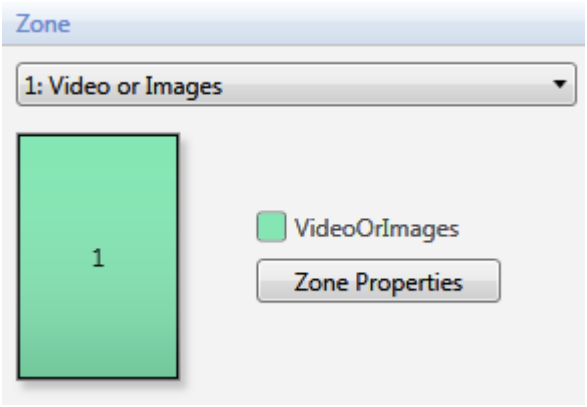
Follow these steps to create a digital-signage canvas that is portrait oriented:

Note: *These instructions apply to BrightAuthor versions 4.3.0.x and later.*

1. Edit the aspect ratio in your web authoring software (e.g. Dreamweaver) so that it is the reverse of your monitor/television resolution. For example, if you plan on displaying the portrait content in 1080p, set the resolution to 1080x1920 rather than 1920x1080.
2. Create a new presentation for your player in BrightAuthor.

3. Set the **Monitor Orientation** to **Portrait**. Click **Create**.
4. When prompted to select a template, choose **Full screen**.
5. Add your HTML5 content to the playlist.

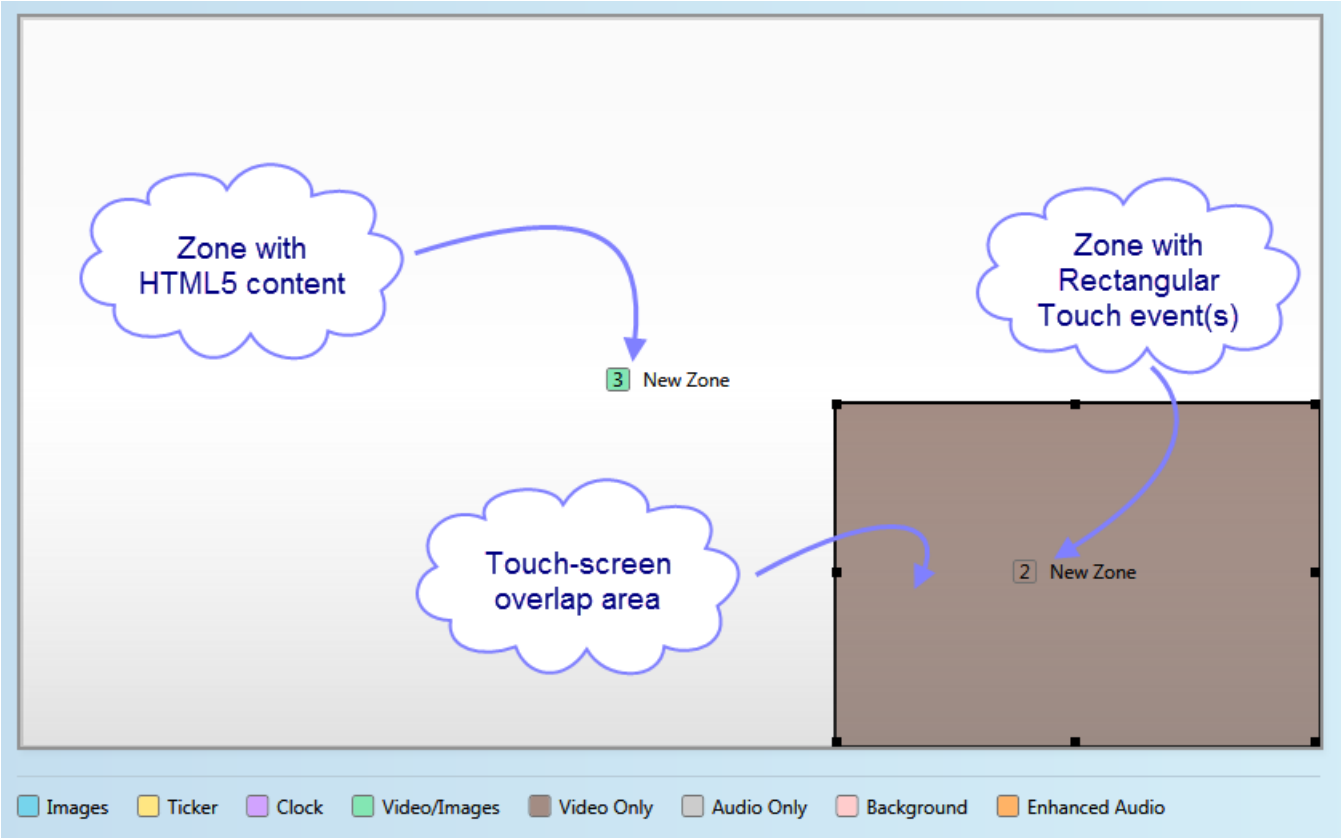




Integrating Touchscreen Content

You can enable touch-screen events for an HTML5 page by checking the **Enable mouse and touch events** box when creating an HTML5 state (see Using HTML5 Pages in BrightAuthor for more details).

Note that touch events are received by both HTML5 pages and BrightAuthor Rectangular Touch events. Therefore, if you have a zone with an HTML5 page overlapping a zone containing a Rectangular Touch event, touching the area of overlap will send an event to both zones at the same time. This is the case even if one zone completely covers the other visually.



Creating an overlap as shown above may have unintended consequences.

Depending on the type of action triggered in each zone, touch-event overlap may cause crashing or other issues with presentation stability. Unless you are certain of the consequences, make sure that zones with touch-enabled HTML5 content and zones with Rectangular Touch events do not overlap.

Known Issues

BrightSign's HTML5 rendering engine is a constant work-in-progress. The following are known issues that we are working to resolve in future versions of player firmware:

- Displaying an HTML page and rotated video while using a 4K output mode will cause the video to glitch. This is the case even if the video is not part of the HTML page (i.e. it's displayed and rotated using BrightScript).
- The JavaScript `toLocaleTimeString()` call does not retrieve localized time formats (i.e. 24-hour vs. 12-hour clock): Instead, the hour/minute clock defaults to 24-hour time on the BrightSign player. The below code provides a workaround in JavaScript if you would like to display time using a 12-hour clock:
    1. Create the following function:

```
function format12Hour(date)
{
    var zero = '0';

    hh = date.getHours();
    mm = date.getMinutes();
    ss = date.getSeconds()

    if((hh % 12) == 0)
        hh = 12;
    else
        hh %= 12;

    // Pad zero values to 00
    hh = (zero+hh).slice(-2);
    mm = (zero+mm).slice(-2);
    ss = (zero+ss).slice(-2);
```

```
    return hh + ':' + mm + ':' + ss + ' ' + ((date.getHours() < 12) ? 'AM' :
'PM');
}
```

2. Optionally, if you would prefer not to display seconds information, you can replace the above “return” line with the following:

```
return hh + ':' + mm + ' ' + ((date.getHours() < 12) ? 'AM' : 'PM');
```

3. Implement the function in the HTML script as follows:

```
var dateString = (startJSDate.getMonth() + 1) + "/" +
startJSDate.getDate();
    if (!startDateTime.isDateOnly()) {
      dateString += " -- " + format12Hour(startJSDate);
    }
```

HTML VIDEO

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, or 4K1142 (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 the pause/resume commands currently work for HLS streams only.

Media Stream Extensions (MSE)

Firmware versions 6.1.x and later support Media Stream Extensions (e.g. YouTube live streams).

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”.

```
<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.

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

View Mode

In firmware versions 6.0.x and later, HTML `<video>` elements maintain the aspect ratio of the source video. If the aspect ratio of the source video doesn't match that of the video window, the video will be letterboxed appropriately. This default behavior can be modified using the "viewmode" attribute, which can be assigned one of the following values:

- `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. This option is identical to the default behavior described above.
- `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 take an “hwz” attribute. When “hwz” is disabled, the video is rendered as a graphics element. When “hwz” is enabled, the video is rendered as a video element, ensuring the highest possible frame rate and video quality.

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

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.

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

The "fade" parameter allows you to control the fading behavior between videos in a `<video>` element. The "fade" parameter accepts the following values:

- `always`: When a video ends, the video window will go black. The new video will then fade in.
- `auto`: Videos transition without fade effects. This is the default behavior.

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 Tag Extensions

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
- `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 packer identifier (PID) of the audio stream you wish to play
- `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 packer 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, and all integer values must have no leading zeroes.
- Numerical values must match the stream description exactly. 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 correctly 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;;"
```

Audio Routing <video> Elements

BrightSign players have unique audio attributes for `<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:N"` (`N` specifies the analog audio enumeration; use `0` to specify the standard 3.5mm audio output)

Note: *If you don't assign any audio attributes to a* `<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>
```

JavaScript Video Playlists

If you create a JavaScript video playlist that uses the `load()` and `play()` methods (and no height/width is specified in the video tag), you may notice a single-frame glitch at the beginning of each video, when a small window appears before the video plays. This occurs because `load()` displays the video window before retrieving resolution information from the source.

You can avoid this glitch by applying the `visibility:"hidden"` attribute to the video player before `load()` is called; then, wait until the load is completed (when the video player returns the `"canplay"` event) to change the `visibility:` to `"inherit"`.

Example:

```
<script>
	var videos = ['video1.mp4', 'video2.mp4'];
	var videoIndex = videos.length;
	var playerVideo = null;

	function PlayNext()
	{
		videoIndex++;
		if(videoIndex >= videos.length) {
			videoIndex = 0;
		}

		playerVideo.src = videos[videoIndex];
		playerVideo.style.visibility = "hidden";
		playerVideo.load();
	}

	function PlayNow()
	{
		playerVideo.style.visibility = "inherit";
		playerVideo.play();
	}

	function LoadPlayer()
	{
		playerVideo = document.getElementById('playerVideo');
```

```
            playerVideo.addEventListener("ended", PlayNext);
            playerVideo.addEventListener("canplay", PlayNow);
            PlayNext();
        }
</script>
```

HTML5 RESOURCES

There are a large number of online resources—including tutorials, samples, templates, and widgets—available to help you get started creating content with HTML5. The HTML5 standard offers huge advantages to web developers, including digital signage authors. The following websites are great places to learn how to create pages using HTML5:

- http://www.html5report.com
- http://www.w3schools.com

Wordpress

Wordpress is an excellent HTML5 resource that provides an intuitive approach to creating digital signage. Here are some of the benefits of using the Wordpress architecture:

- Wordpress offers advanced HTML5 support, with premade widgets ranging from weather to e-commerce. The system also supports advanced HTML5 options using CSS3 features. The http://www.jqwidgets.com site also provides HTML5 widgets.
- You can either run Wordpress from the website or install a Wordpress instance on your own servers.
- Wordpress has an ecosystem of template creators that offer sophisticated templates for a wide range of industries: Template creators include the following:
    a. http://www.rockettheme.com/wordpress
    b. http://www.templatemonster.com
    c. http://graphpaperpress.com

HTML5 Authoring

These are some of the common HTML5 authoring applications:

- Adobe CS Tools: Dreamweaver, Illustrator, InDesign, Photoshop
- Aptana Studio
- CoffeeCup Software

ADVANCED TECHNIQUES

This section is intended for those who are familiar with scripting languages. See the BrightScript Reference Manual and BrightSign Object Reference Manual on the BrightSign Documentation page for more details.

Simple Webpage Script

The simple script outlined below opens a webpage stored on a remote server. Save this script as an *autorun.brs* file, place it on an SD card, and publish it to the player. Alternatively, you can play an HTML page from the local storage without an autorun script by placing the *index.html* file into the root folder of the player storage.

Notice that the script has a `sleep(10000)` line. This line delays loading the web URL, which is necessary to account for connection delays that may occur on your network. If you don't use this script, the BrightSign player may not connect in time to load the page, resulting in a "cannot resolve host", which indicates that the player does not have an Internet connection. In certain network configurations, especially when the player utilizes DHCP, you may need to increase the `sleep` amount (in milliseconds) to give the player more time to establish a connection.

```
Sub Main()

msgPort = CreateObject("roMessagePort")
r = CreateObject("roRectangle", 0, 0, 1920, 1080)
h = CreateObject("roHtmlWidget", r)
h.SetPort(msgPort)

h.SetURL("http://www.brightsign.biz")
h.SetURL("file:///testpage.html")

sleep(10000)
```

```
h.Show()

while true
	msg = wait(0, msgPort)
	print "type(msg)=";type(msg)
	if type(msg) = "roHtmlWidgetEvent" then
		eventData = msg.GetData()
		if type(eventData) = "roAssociativeArray" and type(eventData.reason) = "roString"
then
			print "reason = ";eventData.reason
			if eventData.reason = "load-error" then
				print "message = ";eventData.message
			endif
		endif
	endif
end while

End Sub
```

Portrait Orientation

On your *roHtmlWidget* instance, call `SetTransform("rot90")` for clockwise portrait orientation or `SetTransform("rot270")` for counter-clockwise portrait orientation.