Ceeblue AWM Player API Guide

Danny Burns

Version 1.0.0, 2 Dec 2019

Table of Contents

The AWM Player is a bit of Javascript, that chooses how to show the stream based on the device it's being accessed from. The goal is to always show a working stream with a similar interface.
If you'd like to write the Javascript that initiates the player yourself, follow these steps:

1. Get the AWM's player code.
2. Call the awmPlay method using awmPlay(streamName,options), where streamName is a string with the name of the stream that you want to show, and options is an object containing the desired settings.

Example:

```
awmPlay('live', {
  target: document.getElementById('live'),
  autoplay: false
});
```

Chapter 1. Basic options

These are basic options that can be used to configure the player.
They are set as a **key**:**value** pair in the options object. Simply leave out the key if you'd like to use the default value.

1.1. `target`

Value: DOM element

Example:

```
document.getElementById('player')
```

Required: Yes
Should point to a DOM element into which the player will be inserted

1.2. `host`

Value: string or the boolean false
For example: *"http://example.com:8080"*
Default: false
Should contain an url to server's HTTP output, where the player files and stream info should be requested from.

1.3. `autoplay`

Value: boolean
Default: true
Whether playback should start automatically. If the browser refuses autoplay, the player will attempt to autoplay with its sound muted.

1.4. `controls`

Value: boolean or the string *"stock"*
Default: true
Whether to show controls in the player. If the value *"stock"* is used, it will not use the AWM players skinned controls, but use the underlying player's default controls. Note that this means the awm-player's appearance will vary with the player that has been selected.

1.5. `loop`

Value: boolean
Default: false
Whether to loop the video. This option is not applicable to live streams.

1.6. `muted`

Value: boolean
Default: false
Whether to start the video with its sound muted.

1.7. `poster`

Value: string or the boolean false
For example: *"/myimage.png"*
Default: false
Url to an image to display while loading the video. If false, no image is shown.

1.8. `fillSpace`

Value: boolean
Default: false
Whether the player should grow to fill the container when the stream resolution is smaller than the target element.

1.9. `skin`

Value: string (name of the skin) or object (skin object definition)
For example: *"dev"*
Default: *"default"*
It's also possible to use a custom skin. This is explained in detail in chapter Skinning.

Chapter 2. Advanced options

2.1. `reloadDelay`

Value: number
Default: 10
When an error window is shown, this value will be used as the default delay in seconds after which the default action is executed.
Note that there may be certain errors which have a different delay time, and that these delays are disabled on the developers' skin.

2.2. `urlappend`

Value: string
For example: *"?userid=1337&hash=abc123"*
Default: none
Appends the specified string to any connections the player opens. This can, for example, be used to pass a user id and passphrase for an access control system.

2.3. `setTrack`

Value: object or the boolean **false**

For example:

```
{
    video: 1,
    audio: -1
}
```

Default: false
If not false, a specific track combination is selected. Use the track type as the key, and the desired track id as its value. A value of **-1** can be used to disable the track type entirely. Leave out the track type to select it automatically.
Note that some players may not support track switching.

2.4. `forceType`

Value: string or the boolean false
For example: *"html5/video/mp4"*
Default: false
If not false, forces the awm-player to select a source with this mimetype.
For your convenience, these are some of the mimetypes:

- **WebRTC**: *"webrtc"*
- **WebM**: *"html5/video/webm"*

- **MP4**: *"html5/video/mp4"*
- **HLS**: *"html5/application/vnd.apple.mpegurl"*
- **Dash**: *"dash/video/mp4"*
- **TS**: *"html5/video/mpeg"*
- **WAV**: *"html5/audio/wav"*
- **Progressive**: *"flash/7"*
- **RTMP**: *"flash/10"*
- **HDS**: *"flash/11"*
- **RTSP**: *"rtsp"*
- **Silverlight**: *"silverlight"*

2.5. `forcePlayer`

Value: string or the boolean false
For example: *"dashjs"*
Default: none
If not false, forces the awm-player to select the player specified.
These players are available:

- **HTML5** player: *"html5"*
- **VideoJS** player: *"videojs"*
- **Dash.js** player: *"dashjs"*
- **WebRTC** player: *"webrtc"*
- **Strobe Flash** media playback: *"flash_strobe"*

2.6. `forcePriority`

Value: object or the boolean false
Default: false
This option can be used to override the order in which sources and/or players are selected by the awm-player.
Use the key source to override the sorting of the sources, the key player to override the sorting of the players.
By default, the awm-player loops through the sources first, and then through the players. To override this, include the key first with a value of *"player"*.

Sorting rules
The value that can be given to source and player should be an array containing sorting rules. If sorting ties on the first rule, the second rule will be used, and so on. The default rule is always appended to the list, so it does not need to be included.
Sorting rules can take several forms:

- A string, which is the key that should be sorted by.

- An array with two values: the first the key to sort by, and the second..
  - *-1* to indicate a reverse sort of this value
  - *an array of values*. The array indicates which values should come first, and their order. Any values not in the array will be treated as equal to eachother.
- A function that will be called for every item to be sorted. It will receive the item as its only argument, and items will be sorted using JavaScript's sort() function on the return values.

Example

```
forcePriority: {
    source: [
      [ "type",  [ "html5/application/vnd.apple.mpegurl", "webrtc" ] ]
    ]
}
```

Passing this value will reorder the sources according to these rules: first try HLS sources, then WebRTC ones.

Example

```
forcePriority: {
  source: [
    [ "type", [ "html5/video/webm", "webrtc" ]],
    [ "simul_tracks": -1 ],
    function(a) { return a.priority * -1; },
    "url"
  ]
}
```

Passing this value will reorder the sources according to these rules: first try WebM sources, then WebRTC ones, then reverse sort by the sources' value of ***simul_tracks***, then reverse sort by the sources' value of ***priority***, then sort alphabetically by the sources' value of ***url***.

2.6.1. `monitor`

Value: object or the boolean false
Default: false
The monitor is part of the awm-player that monitors a stream as it is playing in the browser. It has functions to determine a score, that indicates how well the stream is playing. Should this score fall below a defined threshold, it will take a defined action.
The way the monitor functions can be overridden, in part or in full, by using this option. The default monitor object will be extended with the object passed through this option.
Listed below are the keys of the monitoring object, and their function. A monitoring function should contain, at the very least, these functions:

init()
The function that starts the monitor and defines the basic shape of the procedure it follows. This is

called when the stream should begin playback.

destroy()
Stops the monitor. This is called when the stream has ended or has been paused by the viewer.

reset()
Clears the monitor's history. This is called when the history becomes invalid because of a seek or change in the playback rate.

To tweak the behaviour of the monitor, rather than override it in full, other keys can be used. For example, to automatically switch to the next source / player combination when playback is subpar, pass the below as an option.

```
monitor: {
  action: function() {
    this.AwmVideo.log("Switching to nextCombo because of poor playback in" + this
.AwmVideo.source.type + " (" + Math.round(this.vars.score*1000)/10+"%)");
    this.AwmVideo.nextCombo();
  }
}
```

The default monitor is as follows:

```
monitor = {
  AwmVideo: AwmVideo, // Added here so that the other functions can use it. Do not
override, it
  delay: 1, // The amount of seconds between measurements
  averagingSteps: 20, // The amount of measurements that are saved
  threshold: function() { // Returns the score threshold below which the "action"
should be, taken
    if (this.AwmVideo.source.type === "webrtc") {
      return 0.97;
    }
    return 0.75;
  },
  init: function() { // Starts the monitor and defines the basic shape of the
procedure it, follows. This is called when the stream should begin playback.
    if ((this.vars) && (this.vars.active)) { return; } // It's already running, don't
bother
    this.AwmVideo.log("Enabling monitor");
    this.vars = {
      values: [],
      score: false,
      active: true
    };
    var monitor = this;

    // The procedure to follow
    function repeat() {
```

```
      if ((monitor.vars) && (monitor.vars.active)) {
        monitor.vars.timer = this.AwmVideo.timers.start(function() {
          var score = monitor.calcScore();
          if (score !== false) {
            if (monitor.check(score)) {
              monitor.action();
            }
          }

          repeat();
        }, monitor.delay * 1e3);
      }
    }

    repeat();
  },
  destroy: function() { // Stops the monitor. This is called when the stream has ended
or has, been paused by the viewer.
    if ((!this.vars) || (!this.vars.active)) { return; } // It's not running, don't
bother
    this.AwmVideo.log("Disabling monitor");
    this.AwmVideo.timers.stop(this.vars.timer);
    delete this.vars;
  },
  reset: function() { // Clears the monitor's history. This is called when the
history, becomes invalid because of a seek or change in the playback rate.
    if ((!this.vars) || (!this.vars.active)) {
      // It's not running, start it up
      this.init();
      return;
    }
    this.AwmVideo.log("Resetting monitor");
    this.vars.values = [];
  },
  calcScore: function() { // Calculate and save the current score
    var list = this.vars.values;
    list.push(this.getValue()); // Add the current value to the history

    if (list.length <= 1) { return false; } // No history yet, can't calculate a score

    var score = this.valueToScore(list[0], list[list.length-1]); // Should be 1,
decreases if bad

    // Kick the oldest value from the array
    if (list.length > this.averagingSteps) { list.shift(); }

    // The final score is the maximum of the averaged and the current value
    score = Math.max(score, list[list.length-1].score);
    this.vars.score = score;

    return score;
```

```
  },
  valueToScore: function(a, b) { // Calculate the moving average
    // If this returns > 1, the video played faster than the clock
    // If this returns < 0, the video time went backwards
    var rate = 1;

    if (("player" in this.AwmVideo) &&
        ("api" in this.AwmVideo.player) &&
        ("playbackRate" in this.AwmVideo.player.api)) {
      rate = this.AwmVideo.player.api.playbackRate;
    }

    return (b.video - a.video) / (b.clock - a.clock) / rate;
  },
  getValue: function() { // Save the current testing value and time
    // If the video plays, this should keep a constant value. If the video is stalled,
it will go up, with 1sec/sec. If the video is playing faster, it will go down.
    // Current clock time - current playback time
    var result = {
      clock: (new Date()).getTime() * 1e-3,
      video: this.AwmVideo.player.api.currentTime,
    };

    if (this.vars.values.length) {
      result.score = this.valueToScore(this.vars.values[this.vars.values.length-1],
result);
    }

    return result;
  },
  check: function(score) { // Determine if the current score is good enough. It must
return true, if the score fails.
    if (this.vars.values.length < this.averagingSteps * 0.5) { return false; } //
Gather enough values, first

    if (score < this.threshold()) {
      return true;
    }
  },
  action: function() { // What to do when the check is failed
    var score = this.vars.score;
    // Passive: only if nothing is already showing
    this.AwmVideo.showError("Poor playback: " + Math.max(0, Math.round(score * 100)) +
"%", {
      passive: true,
      reload: true,
      nextCombo: true,
      ignore: true,
      type: "poor_playback"
    });
  }
```

```
}
```

2.6.2. `callback`

Value: function or the boolean false
Default: false
When the awm-player has initialized, and whenever it has thrown an error, the function provided will be called. It will receive the AwmVideo object as its only argument.
This allows other scripts to control the awm-player.

2.6.3. `AwmVideoObject`

Value: object or the boolean false
Default: false
Pass an object with this option to save a reference to the ***AwmVideo*** object, which can then be used by other scripts to control the awm-player.
It can be important to always have an up to date reference to the ***AwmVideo*** object. To achieve this, the ***AwmVideo*** object is saved in the object passed in this option under the key reference, creating the JavaScript equivalent of a pointer.

Example:

```
var mv = {};

awmPlay("stream", {
    target: document.getElementById("stream"),
    AwmVideoObject: mv
});

function killAwmVideo() {
  if ("reference" in mv) {
    mv.reference.unload();
  }
}
```

The variable ***mv.reference*** will always point to the ***AwmVideo*** object that is currently active, so that calling ***killAwmVideo()*** will unload the awm-player, regardless of where it is in its lifetime.

Chapter 3. API methods

3.1. `Events`

The awm-player dispatches events to indicate certain things have happened. Most of these events are standard media events dispatched by the underlying video element. Information about these can be found here: https://www.w3.org/TR/html5/semantics-embedded-content.html#media-elements-event-summary
Some however, are custom, and indicate that the awm-player instance has progressed to a new stage in its start up. You'd expect to receive the **haveStreamInfo**, **playerChosen** and initialized events in this order.

3.1.1. `haveStreamInfo`

This event is dispatched by the target element when the awm-player has retrieved a stream's metadata. From now on, it can be read from `AwmVideo.info`.

3.1.2. `comboChosen`

This event is dispatched by the target element when the awm-player has chosen a player and source combination. This happens before the selected player is asked to build. The player name is now available at `AwmVideo.playerName`, and the source at `AwmVideo.source`.

3.1.3. `initialized`

This event is dispatched by the target element when the interface has been built and the selected player has completed its build method.
If you need to be certain the video is loaded, you will want to listen for the `loaddedmetadata` event, dispatched by the video element.

3.1.4. `initializeFailed`

This event is dispatched by the target element when the awm-player was unable to complete its initialization sequence. The awm-player will always generate either an `initialized` or an `initializeFailed` event, unless it is unloaded before it has reached either point.

3.1.5. `log`

This is event is dispatched by the target element for each new log message. The message itself is available as the event's message property.

3.2. `The AwmVideo object`

These methods and properties can be found directly on the `AwmVideo` object and may be used to control the awm-player.

3.2.1. `AwmVideo.stream`

This property contains the stream name as a string. It should be considered read only.

3.2.2. `AwmVideo.options`

This property contains the options passed to the awm-player. It should be considered read only.

3.2.3. `AwmVideo.info`

This property contains the stream information as an object once the awm-player has loaded the stream metadata from StreamingServer. It should be considered read only.

3.2.4. `AwmVideo.playerName`

This property contains the name of the selected player as a string once the awm-player has selected one. It should be considered read only.

3.2.5. `AwmVideo.source`

This property contains the selected source as an object once the awm-player has selected one. It should be considered read only.

3.2.6. `AwmVideo.video`

This property contains the video element once the player has constructed it. It can be used to add event listeners to, but the element's methods and properties should not be used directly. These should be accessed through `AwmVideo.player.api`.

3.2.7. `AwmVideo.logs`

This property contains any log messages that awm-player instance has sent, as an array. Each log entry is an object, with a time key containing a Date object of when it was dispatched, a message key containing the message itself, and a data key, which contains an object with at least a type key, which is either ”log” or ”error”. It should be considered read only. Use the `AwmVideo.log()` method to add new messages.

3.2.8. `AwmVideo.timers`

In this property, timers associated with this awm-player instance are stored as an object. When the instance is unloaded all the timers are automatically canceled. To start a timer, use `AwmVideo.timers.start(callback, delay)`. A timer id is returned, just like JavaScript's own setTimeout function does. To cancel the timer, use `AwmVideo.timers.stop(timer_id)`.

3.2.9. `AwmVideo.monitor`

This property contains the stream playback monitor, comprised of the default monitor extended with custom code that was passed through the monitor option. It should be considered read only.

3.2.10. `AwmVideo.nextCombo()`

When this method is called, the awm-player instance will be reloaded, using the next best source/player combination. If all combinations have been tried, it will loop from the beginning.

3.2.11. `AwmVideo.unload()`

When this method is called, the video is stopped and the awm-player is removed from the web page. Any running processes may fail to complete.

3.2.12. `AwmVideo.log(message, type)`

When this method is called, a message is added to the log. The type parameter is optional and defaults to "log".

3.2.13. `AwmVideo.checkCombo(options, quiet)`

This method can be used to determine if a source and player combination is available, and if so, which the awm-player would choose.
Options should be an object containing options as they would be passed to awmPlay, and defaults to the options used to start the current instance. If quiet evaluates to true, the usual log messages when choosing a source/player combo won't be sent.
The return value is an object, with these keys:

- `player` The name of the selected player
- `source` The selected source object
- `source index` The index of the selected source

3.2.14. `AwmVideo.showError(message, options)`

Shows a window with some kind of error message.
`message` is the message, in plain text, that should be shown.
`options` is an object controlling the behavior of the error window options may contain these keys:

- `softReload`
  If this property is present and evaluates to true, a button is shown that executes `AwmVideo.player.api.load()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `reload`
  If this property is present and evaluates to true, a button is shown that executes `AwmVideo.reload()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `nextCombo`
  If this property is present and evaluates to true, a button is shown that executes `AwmVideo.nextCombo()`. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.
- `ignore`

If this property is present and evaluates to true, a button is shown that ignores subsequent errors of this type (see below) of error for the lifetime of this awm-player instance. If the value is a number, a countdown is added to the button. When the countdown finishes, the button is pressed.

- `type`
  Is used in combination with the ignore button action to determine whether this error should be displayed or not. If this property is unset or its value is false, it defaults to the contents of the message parameter.
- `polling`
  If this property is present and evaluates to true, a small loading icon is shown, indicating that something is being checked in the background.
- `passive`
  If this property is present and evaluates to true, it will not replace an error window that is already being shown, unless that error is also passive. In that case, the message text will be updated, but the buttons (and their current countdowns values) will not.

3.2.15. `clearError()`

Hides the current error window if there is one.

3.3. `The player API`

Players will also have their own methods and properties, which can be found in `AwmVideo.player`.

3.3.1. `AwmVideo.player.resizeAll()`

Calling this method will ask the awm-player to recalculate its size and resize the video accordingly.

3.3.2. `AwmVideo.player.api`

If the player can respond to most of the methods of a standard HTML5 video element, this property will be set. In it the usual methods and properties of a video element can be found. Please refer to the general HTML5 video element documentation (https://www.w3.org/TR/2011/WD-html5-20110113/video.html) for their workings.
AwmVideo.player.api should be used to control and access the video element rather than AwmVideo.video, as the players may modify the behaviour of the methods stored in the player api to achieve a more consistent experience for the end user. For example, the values of currentTime and duration of the player api will be different to those of the video element for live playback of MP4 in the HTML5 player.

Chapter 4. Skinning

With a skin, the look and effects of the buttons can be changed. To use a skin, set the skin property of the options object to a skin object, or define `AwmSkins.skin` name elsewhere and set it to the name of the skin. AwmPlayer has two predefined skins: default (the default skin intended for production) and dev (with additional information and controls aimed at developers).

4.1. `Skin definition`

A skin can be defined directly in the options object passed to AwmPlay, or elsewhere by setting var `AwmSkins.skin_name` after the awm-player's code, `player.js`, has been loaded.
A skin is made up of several components, which are defined in the skin definition's properties:

- `colors`
  Names of color variables that can be used to quickly modify the awm-player's color scheme.
- `css`
  CSS rules that determine how the various elements are displayed, referencing the color scheme.
- `icons`
  These are SVG icons that can be used by the various blueprints. They are marked with classes so that their appearance is determined by the CSS rules and colors.
- `blueprints`
  A blueprint is a part of the user interface, such as a play button or progress bar.
- `structure`
  The structure defines the layout of the various blueprints creating the awm-player's user interface.

4.1.1. `inherit`

If this property is present and is a skin name, the properties of the given skin will be extended with the current skin definition. If omitted, the default skin will be extended.

4.1.2. `colors`

If this property is present, the inherited skin's color object will be extended with this one. The colors defined here are used in the CSS files pointed to in the css property. The default skin uses these listed below.

- `accent`
  An accent color for things meant to catch the user's eye, like the current progress bar.
- `fill`
  The fill color for icons that use the fill class.
- `semiFill`
  The fill color for icons that use the semiFill class.
- `stroke`
  The text color and stroke color for icons.

- `strokeWidth`
  The stroke width for icons.
- `background`
  The background color of the control bar.
- `progressBackground`
  The background color of the progress bar.

4.1.3. `css`

If this property is present, the inherited skin's css object will be extended with this one. It should be an object that contains urls to css files that control the display and layout of the awm-player's DOM elements.
The object's keys are only used to allow selective overwriting. The values should be urls to special css files. The default skin uses the key skin for its css.
The css file is allowed to contain variables marked with a **$**-sign, pointing to colors named in the color object.

For example:

```
.awmvideo-controls { background-color: $accent; }
```

Will set the controls' background to the accent color. The css rules will be prepended with the awm-player instance's unique id, so that they won't affect other awm-player instances which may use other skins.

4.1.4. `icons`

If this property is present, the inherited skin's icon object will be extended with this one. It should be an object that contains definitions for an svg.
Each key should be an icon name, and each value should be an object, with these properties:

- `size`
  This should be object, with width and height properties set to a number, indicating the height and width of the icon viewbox. If the width and height are the same, just the number value may be used as a shorthand. This isn't necessarily the size the icon will have once on the web page.
- `svg`
  This should be a string containing the contents of the svg, or a function returning that string. If you would like the css and/or color rules to affect the appearance of the icon, these properties should not be defined inline, but through the use of classes. These are the classes currently being used by the default skin:
  - `fill`
    This svg element should be filled with the defined fill color.
  - `semiFill`
    This svg element should be filled with the defined semiFill color.
  - `stroke`
    This svg element should have a stroke with the defined stroke color and width.

- `toggle`
  If the icon has the class off, elements and children of elements with this class that have the class fill or semiFill will have their fill set to none.
- `spin`
  This svg element should have a spinning animation.

Constructing the icon element
Calling the `AwmVideo.skin.icons.build(type, size, options)` method will return the icon as a DOM element.

- `type` should be a string, with the name of the icon.
- `size` should be an object, containing the properties width and height set to a number: the desired size of the icon in pixels. A number may be used as shorthand if its width and height are equal. If only one of width and height is specified, the other will be calculated automatically using the icon's aspect ratio. If omitted, a value of 22 is used.

If the svg property of the icon object is a function, options is passed to it as its parameter

4.1.5. `structure`

If this property is present, the inherited skin's structure object will be extended with this one. It should be an object that defines the layout of the various awm-player's elements, defined in the blueprints property. For convenience, the structure option is split into properties that can be overwritten separately. Each of these properties, if defined, should contain a structure object.

- `main`
  The main structure that contains the video element and controls.
- `videocontainer`
  This structure contains the video element. If for example you'd like to add a logo on top of the video, this would be the best place to do so.
- `controls`
  This structure contains the controls of the awm-player.
- `submenu`
  This structure contains a popup menu that is used in the controls structure.
- `placeholder`
  This structure contains elements returned by the placeholder, loading and error blueprints. It is shown after the skin is loaded but before the player is initialized.
- `secondaryVideo`
  This structure contains a video element and controls, intended to be used for picture-in-picture mode.

Structure object syntax
A structure object should either be an object, or a function that returns a structure object. The function can reference the AwmVideo object as this.
The structure object may contain these keys:

- `type`
  If this property is set and the name of a blueprint, that blueprint is constructed with the current structure as its parameter.
- `classes`
  If this property is set, it should be an array of classes that will be added to the DOM element.
- `title`
  If this property is set, the DOM element's title attribute is set to it.
- `children`
  If this property is set, it should be an array of structure objects that will be appended as children of the DOM element.
- `style`
  If this property is set, it should be an object containing style properties and their desired values that will be applied directly on the DOM element.
- `if`
  If this property is set, it should be a function that will be passed the current structure as its parameter. When this function returns true, the structure object specified by the then key is used, otherwise, it will use the structure object specified by the else key.

Constructing a structure
Calling the `AwmVideo.UI.buildStructure(structure)` method will return the structure as a DOM element.
`structure` should be a structure object.

4.1.6. `blueprints`

If this property is present, the inherited skin's blueprints object will be extended with this one. It should be an object that defines how the various elements should be constructed and how they should function.
The blueprint function is given the current structure as its parameter. It can reference the **AwmVideo** object as **this**. It should return either a DOM element or **false** if nothing should be added to the DOM tree.
All elements returned by blueprint functions will be given a class of their name, prefixed with **awmvideo-**, for example: **.awmvideo-container**.
Listed below are the blueprints that are defined in the default skin.

container
Returns an empty div.

hoverWindow
Returns a DOM element containing a button and a window that is shown when the button or window is hovered over.
To use it, define these properties on the structure object:

- `button`
  Should be a structure object of the element the user should hover over to show the window.
- `window`

Should be a structure object of the window that should be shown.

- `mode`
  Should be the string *"pos"* to enable absolute positioning of the window. More modes may be supported at a later time.
- `transition`
  Should be an object with the properties listed below. Each should contain a string of CSS rules that define the position of the item.
  - `show`
    Will be used as CSS rules for the window when the button is hovered over.
  - `hide`
    Will be used as CSS rules for the window when the button is not hovered over.
  - `viewbox`
    Will be used as CSS rules for the window viewbox. Parts of the window that fall outside of the viewbox will be hidden.

Example:

```
var structure = {
  type: "hoverWindow",
  button: {type: "settings"},
  window: {type: "submenu"},
  mode: "pos",
  transition: {
    hide: "right: -1000px; bottom: 44px;",
    show: "right: 5px;",
    viewport: "right: 0; left: 0; bottom: 0; top: -1000px"
  }
};
```

This structure object will create a settings button and a window containing the submenu structure. When the button is not hovered over, the window will be hidden from view on the right side of the player container. When the button is activated the window is shown with a margin of 5 pixels from the right side of the container. If the window is taller than the container, this part will be visible. Overflow in other directions is hidden from view.

video
Returns the video DOM element. It also improves autoplay behaviour (if the browser blocks autoplay, mutes the video and retries), hides the cursor when it is not moved, and disables right clicking on the element.

videocontainer
Returns the elements defined by the videocontainer structure.

secondaryVideo
Returns the elements defined by the secondaryVideo structure. Starts up a secondary awm-player instance playing the same stream. Define the options property on the structure object to pass those options to the secondary awm-player. It also corrects for desync with the main video. The

secondary video will be muted.

Example:

```
structure.videocontainer = {
  type: "container",
  children: [
    {type: "video"},
    {
      type: "secondaryVideo",
      options: {
        setTracks: {
          video: 3
        }
      }
    }
  ]
};
```

If the video container is replaced with this structure, a secondary video will be added to the awm-player instance. The secondary video will play the video track with an identifyer of 3.

switchVideo
Returns a button that switches the main and secundary video it is attached to.

controls
Returns the elements defined by the controls structure.

submenu
Returns the elements defined by the controls structure.

progress
Returns a progress bar, indicating the current playback position, buffered sections, and allowing seeking. When the stream being played is live, the progress bar will indicate the seek window.

play
Returns a button with a pause icon when the stream is playing and a play icon when the stream is paused. It also asks the video player to play when the video element is double clicked.

speaker
Returns a button with a speaker icon. When the button is clicked, the muted state is toggled.

volume
Returns a button with a volume icon. When the button is clicked or dragged on, the video player's volume is set to the level indicated.
The volume level is scaled quadratically: clicking at 50% will set the volume to 0.25;

currentTime
Returns a div that display the current video time.

totalTime

Retuns a div that displays the duration of the video. If the stream is live, it will receive the class live and it will display the text ”live”.

playername
Returns a div that displays the name of the player.

mimetype
Returns a link to the stream source and displays the source’s mime type.

logo
Returns an element. Set the `element` property to a DOM element to use that, or set the src property to an url to an image.

settings
Returns the settings icon.

loop
Returns a button that toggles the loop state or false if the stream is live.

fullscreen
Returns a button that toggles fullscreen mode.

tracks
Returns an interface that displays the tracks and allows track switching if multiple tracks are available.

text
Returns a span with the text specified with the text property.

placeholder
Returns a div with the size of the stream. If a poster has been configured, it is used as a background image.

timeout
Returns a countdown icon.
The following properties can be used:

- `delay` Should be omitted or set to a number. Indicates the amount of seconds after which the function is executed. If omitted, the delay is set to 5 seconds.
- `function` Should be a function that is executed after the delay has passed.

loading
Returns a div containing a rotating loading icon. It is shown when something is loading in the background and the video is not playing.

error
Returns a div that displays an error when applicable. It adds the **AwmVideo.showError()** and **AwmVideo.clearError()**

tooltip
Returns a div that displays a tooltip.

The returned element has these methods:

- **setText(text)** sets the displayed text to the string provided.
- **setPos(position)** sets the tooltip position. `position` should be an object containing the CSS position properties (`top`, `bottom`, `left` and `right`) set to their desired value. Any position properties that should be set to `auto` can be omitted.
- **triangle.setMode(primary, secondary)** sets the position of the tooltip tip.
  `primary` and `secondary` should be strings containing one of the CSS position property names (`"top"`, `"bottom"`, `"left"` and `"right"`). The primary direction dictates the side on which the tip appears. The secondary direction dictates its alignment.

Example:

```
var tooltip = AwmVideo.UI.buildStructure({type:"tooltip"});
tooltip.setText("Hello world");
tooltip.triangle.setMode("bottom","left");

// apply awm player css to the document body
var uid = false;
AwmVideo.container.classList.forEach(function(a){
  if (a.indexOf("uid") == 0) { uid = a; }
});
if (uid) { document.body.classList.add(uid); }

document.body.appendChild(tooltip);
document.body.addEventListener("mousemove", function(e){
  var pos = {bottom: this.clientHeight - e.clientY + 10};
  var leftPercentage = e.clientX / this.clientWidth * 100;
  if (leftPercentage > 50) {
    pos.right = (100 - leftPercentage) + "%";
    tooltip.triangle.setMode("bottom", "right");
  } else {
    pos.left = leftPercentage + "%";
    tooltip.triangle.setMode("bottom","left");
  }
  tooltip.setPos(pos);
});
```

This code creates a tooltip and appends it to the body. The tooltip will follow the cursor around. If the cursor is on the left side of the screen the tooltip appears to the right of the cursor. If the cursor is on the right side of the screen the tooltip appears to the left of the cursor.

`button`
Returns a button element.
The following structure object properties can be used:

- **label** The text that should be displayed on the button.
- **onclick** The function that should be executed when the button is clicked.

- **delay** If specified, this should be a number indicating the number of seconds after which the onclick function should be executed. A timeout icon is prepended on the button.

4.2. Examples

4.2.1. Other colors

The following skin definition changes the default skin's color scheme.

```
AwmSkins.white = {
  colors: {
    fill: "rgba(34,136,187,1)",
    semiFill: "rgba(34,136,187,0.5)",
    stroke: "#000",
    background: "#fff",
    progressBackground: "rgba(34,136,187)",
    accent: "cyan"
  }
};
```

4.2.2. Disabling a blueprint

The following skin definition disables the progress bar.

```
AwmSkins.noprogress = {
  blueprints: {
    progress: function() { return false; }
  }
};
```

4.2.3. Skin inheritance

The following code adds a random accent color to the developers' skin.

```
function getRandomFromArray(array) {
  return array[Math.floor(Math.random() * array.length)];
}

// create an element to hold the awm-player
var c = document.createElement("div");
document.body.appendChild(c);

// build the awm-player
awmPlay("stream", {
  target: c,
  skin: {
    inherit: "dev",
    colors: {
      accent: getRandomFromArray(["red", "yellow", "cyan"])
    }
  }
});
```

4.2.4. `Using logos`

The following code constructs a awm player instance with a static logo and a dynamic banner

```
// Pre-create an image element that will be passed on to the awm-player
var banner_bottom = document.createElement("img");

// Some pre-defined sources for the banner
var banner_srcs = [
  "my_banner1.png",
  "my_banner2.png",
  "my_banner3.png",
  "my_banner4.png"
];

// Loop over the banners
var i = 0;
function next_banner() {
  banner_bottom.src = banner_srcs[i];
  i++;
  if (i >= banner_srcs.length) { i = 0; }
}

setInterval(function(){
  next_banner();
}, 30e3);
next_banner();

// Create an element to hold the awm-player
var c = document.createElement("div");
```

```
document.body.appendChild(c);

// Build the awm-player
awmPlay(streamname, {
  target: c,
  skin: {
    structure: {
      videocontainer: {
        type: "container",
        children: [
          { type: "video"},
          {
            type: "logo",
            src: "my_logo.svg", //the logo blueprint can be given an image url
            style: {
              position: "absolute",
              width: "10%",
              top: "1em",
              right: "1em",
              filter: "drop-shadow(1px 1px 2px black)",
            }
          },
          {
            type: "logo",
            element: banner_bottom, //..or a DOM element
            style: {
              position: "absolute",
              width: "80%",
              maxHeight: "15%",
              objectFit: "contain",
              bottom: "37px",
              left: 0,
              right: 0,
              filter: "drop-shadow(1px 1px 2px black)",
              margin: "0 auto"
            }
          }
        ]
      }
    }
  }
});
```