Want more?

Buzz!

A Javascript HTML5 Audio library

Home
Documentation
Buzz
Sound
Group
Events
Demo

Documentation: Sound

Buzz provides tons of methods to take advantage of the new **HTML5 audio element**. This section lists the methods and provides helpful information about using them.

Supported codecs

Find below the codecs supported by the latest version of browsers on july 2011. The best combination is OGG + MP3 formats.

OGG	MP3	WAV	AAC	Latest version
•		•		FireFox
•	•	•		Chrome
•		•		Opera
	•	•		Internet Explorer
	•	•	•	Safari
	•	•	•	Safari mobile *
•	•			Android browser *

*** NOTE:** In order to save bandwidth, mobile devices don't load/play sounds without user intervention (no autoplay). iPhone and iPad can only play one sound at a time and don't allow to change the volume dynamically. iPad has some issues with multiple sounds on the same page.

Keeping this information up-to-date is virtually impossible, they can be out-dated. So you can perform your own tests with the tool below.

OGG	MP3	WAV	AAC
no	maybe	probably	maybe

Your browser is:
Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_8; fr-fr) AppleWebKit/533.21.1 (KHTML, like Gecko) Version/5.0.5 Safari/533.21.1

Constructor

new buzz.sound(sources, [settings])

Create a new sound instance.

A single file can be set with one URL.

```
var mySound = new buzz.sound("/sounds/mysound.ogg");
```

Many files can be set by an array of URLs.

```
var mySound = new buzz.sound([
    "/sounds/mysound1.ogg",
    "/sounds/mysound2.mp3",
    "/sounds/mysound3.aac",
    "/sounds/mysound4.wav"
]);
```

In order to stay DRY, many files can be set this way as well.

```
var mySound = new buzz.sound("/sounds/mysound", {
    formats: [ "ogg", "mp3", "aac", "wav" ]
});
```

Options

preload metadata / true / false

A boolean specifying whether the sound should be loaded when the page loads. `metadata` specifies that the browser should load only metadata (duration etc...).

autoplay true / false

A boolean specifying whether the sound should play as soon as it can.

loop true / false

A boolean specifying whether the sound should be repeatedly played. Most of the browsers are not good at that yet, don't expect a perfect loop.

```
var mySound = new buzz.sound("/sounds/mysound", {
    formats: [ "ogg", "mp3", "aac", "wav" ],
    preload: true,
    autoload: true,
    loop: false
});
```

Methods

Note that almost all methods are chainable.

```
mySound.loop().play().fadeIn();
```

sound.Load()

Load the sound.

```
mySound.load();
```

Playback

sound.play()

Load the sound and begin playback.

```
mySound.play();
```

sound.pause()

Pause the sound. If playback is started again, it will restart from where it was paused.

```
mySound.pause();
```

sound.togglePlay()

Automatically pause or play the sound.

```
mySound.togglePlay();
```

sound.isPaused()

Return `true` if the sound is paused or is ended. Return `false` otherwise.

```
if ( mySound.isPaused() ) {
    alert("The sound is paused!");
}
```

sound.stop()

Stop the sound. If the playback is started again, it will restart from the start.

```
mySound.stop();
```

sound.isEnded()

Return `true` if the sound is ended. Return `false` otherwise.

```
if ( mySound.isEnded() ) {
    alert("The sound has ended!");
}
```

sound.loop()

Keep re-playing the sound once it has finished.

```
mySound.loop();
```

sound.unloop()

Stop re-playing the sound once it has finished.

```
mySound.unloop();
```

Volume

sound.mute()

Mute the sound.

```
mySound.mute();
```

sound.unmute()

Unmute the sound.

```
mySound.unmute();
```

sound.toggleMute()

Automatically mute or unmute the sound.

```
mySound.toggleMute();
```

sound.isMuted()

Return true if the sound is muted. Return false otherwise.

```
if ( mySound.isMuted() ) {
    alert("The sound is muted!");
}
```

sound.setVolume(volume)

Set the volume of the sound. The range is `0-100`.

```
mySound.setVolume(90);
```

sound.getVolume()

Return the volume of the sound. The range is `0-100`.

```
var volume = mySound.getVolume();
```

sound.increaseVolume([volume])

Increase the volume of the sound by 1 or *`{volume}`*.

```
mySound.increaseVolume(); // Current volume +1
```

sound.decreaseVolume([volume])

Decrease the volume of the sound by 1 or *`{volume}`*.

```
mySound.decreaseVolume(10); // Current volume -10
```

sound.fadeIn([duration], [callback])

Fade the volume of the sound in (from `0` to `100`) in *`{duration}`* milliseconds. A *`{callback}`* function can be set and called when the fade-in is complete.

```
mySound.fadeIn(2000);
```

sound.fadeOut([duration], [callback])

Fade the volume of the sound out (from current volume to `0`) in *`{duration}`* milliseconds. A *`{callback}`* function can be set and called when the fade-in is complete.

```
mySound.fadeOut(2000);
```

sound.fadeTo(volume, [duration], [callback])

Fade the volume of the sound from the current volume to *`{volume}`* in *`{duration}`* milliseconds. A *`{callback}`* function can be set and called when the fade-in is complete.

```
// Fade volume from 20 to 90 in 2 seconds and then back to 20.
mySound.setVolume(20).fadeTo(90, 2000, function() {
    this.fadeTo(20, 2000);
});
```

sound.fadeWith(sound, [duration])

Fade-out the volume of the current sound while fade-in a new sound.

```
var myOtherSound = new buzz.sound("/sounds/myothersound.ogg");
mySound.fadeWith(myOtherSound, 2000);
```

Events

sound.bind(event, callback)

Add one or many event listeners to a sound. See **Event section** for more details.

```
mySound.bind("ended pause", function(e) {
    var percent = buzz.toPercent( this.getTime(), this.getDuration() ),
        message = "Stopped or paused at " + percent + "%";
    document.getElementById("percent").innerHTML = message;
});
```

Events can be namespaced. Namespacing allows to unbind or trigger some events of a type without affecting others.

```
mySound.bind("ended.one pause.one", function(e) {
    alert("Event type one");
}).bind("ended.two pause.two", function(e) {
    alert("Event type two");
});
```

sound.bindOnce(event, callback)

Add one or many event listeners to be executed once. See **Event section** for more details.

```
mySound.bindOnce("pause", function(e) {
    alert("To be executed only at the first pause.");
});
```

sound.unbind(event)

Remove the event listeners bound to a sound. Events to unbind can be namespaced.

```
mySound.unbind("ended.one .two");
```

sound.trigger(event)

Execute the handlers attached to an event. Note that only the functions are trigged, not the native event.

```
mySound.trigger("ended.one");
```

Getters and setters

sound.setTime(seconds)

Set the playback position in seconds.

```
mySound.setTime(90);
```

The **fromTimer** helper can be useful here.

```
mySound.setTime( buzz.fromTimer("01:30") );
```

sound.getTime()

Get the current playback position in seconds.

```
var seconds = mySound.getTime(); // 90
```

The **toTimer** helper can be useful here.

```
var timer = buzz.toTimer( mySound.getTime() ); // 01:30
```

sound.setPercent(percent)

Set the playback position in *{percent}*.

```
mySound.setPercent(50); // 50%
```

sound.getPercent()

Get the playback position in percent.

```
var percent = mySound.getPercent() + "%";
```

sound.getDuration()

Get the total duration of the sound in seconds.

```
var duration = mySound.getDuration() + " seconds";
```

The **toTimer** helper can be useful here.

```
var timer = buzz.toTimer( mySound.getDuration() ); // 05:46
```

sound.SetSpeed(speed)

Set the playback speed where 1 is normal speed, 2 is double speed, etc.

```
mySound.SetSpeed(2); // x2
```

sound.GetSpeed()

Get the playback speed.

```
var speed = mySound.GetSpeed();
```

sound.set(property, value)

Directly set the native properties of an HTML5 audio element.

```
mySound.set("volume", 0.5);
mySound.set("currentTime", 90);
```

sound.get()

Directly get the native properties of an HTML5 audio element.

```
var loop = mySound.get("loop");
```

Time Ranges

sound.getPlayed()

Get an array that represents the ranges of the sound that the browser has played.

```
var played = mySound.getPlayed();
console.info("Played:");
for(var i in played) {
    console.log(played[i]);
}
```

sound.getBuffered()

Get an array with ranges of the sound that the browser has buffered.

```
var buffered = mySound.getBuffered();
console.info("Buffered:");
for(var i in buffered) {
    console.log(buffered[i]);
}
```

sound.getSeekable()

Get an array with ranges of the sound to which it is possible for the browser to seek.

```
var seekable = mySound.getSeekable();
console.info("Seekable:");
for(var i in seekable) {
    console.log(seekable[i]);
}
```

Errors and States

sound.getErrorCode(), sound.getErrorMessage()

Return the code/message of the current error.

- 1 **MEDIA_ERR_ABORTED**: The fetching process for sound was aborted by the user agent at the user's request.
- 2 **MEDIA_ERR_NETWORK**: A network error of some description caused the browser to stop fetching the sound, after the sound was established to be usable.
- 3 **MEDIA_ERR_DECODE**: An error of some description occurred while decoding the sound, after the resource was established to be usable.
- 4 **MEDIA_ERR_SRC_NOT_SUPPORTED**: The `src` attribute was not suitable.

```
mySound.bind("error", function(e) {
    alert("Error: " + this.getErrorMessage());
});
```

sound.getStateCode(), sound.getStateMessage()

Return the code/message of the sound current state.

- 0 **HAVE_NOTHING**: No information is available.
- 1 **HAVE_METADATA**: Enough of the sound has been obtained to get some info.
- 2 **HAVE_CURRENT_DATA**: Data for the immediate current playback position is available.
- 3 **HAVE_FUTURE_DATA**: Data is available for the immediate current playback position and to advance the current playback position.
- 4 **HAVE_ENOUGH_DATA**: All information are available.

sound.getNetworkStateCode(), sound.getNetworkStateMessage()

Return the code/message of the network current state.

- 1 **NETWORK_EMPTY**: The sound has not yet been initialized.
- 2 **NETWORK_IDLE**: The sound has been selected but it is not actually using the network.
- 3 **NETWORK_LOADING**: The browser is actively trying to download data.
- 4 **NETWORK_NO_SOURCE**: The sound is not available.