**CANVAS**

**/\***

**\* AshAlom Gauge Meter. Version 2.0.0**

**\*/**





## What is this?

In JavaScript, the this keyword refers to an **object**.

**Which** object depends on how this is being invoked (used or called).

The this keyword refers to different objects depending on how it is used:

|  |
| --- |
| In an object method, this refers to the **object**. |
| Alone, this refers to the **global object**. |
| In a function, this refers to the **global object**. |
| In a function, in strict mode, this is undefined. |
| In an event, this refers to the **element** that received the event. |
| Methods like call(), apply(), and bind() can refer this to **any object**. |

## Explicit Function Binding

The call() and apply() methods are predefined JavaScript methods.

They can both be used to call an object method with another object as argument.

## See Also:

[The Function call() Method](https://www.w3schools.com/js/js_function_call.asp)

[The Function apply() Method](https://www.w3schools.com/js/js_function_apply.asp)

[The Function bind() Method](https://www.w3schools.com/js/js_function_bind.asp)

The example below calls person1.fullName with person2 as an argument, **this** refers to person2, even if fullName is a method of person1:

### Example

const person1 = {  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
}  
  
const person2 = {  
  firstName:"John",  
  lastName: "Doe",  
}  
  
// Return "John Doe":  
person1.fullName.call(person2);

It doesn't need an index, since this provides the context. As noted by the [docs](http://api.jquery.com/jQuery.each/), "The value can also be accessed through the this keyword." This is accomplished by using [call](https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Objects/Function/call). Something like:

userFunction.call(valueOfElement, indexInArray, valueOfElement);

$.fn.imgAreaSelect = function (options)

means the function is being added to the prototype. This allows it to be used with any instance of the jQuery object.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/call>

# Function.prototype.call()

The **call()** method of [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function) instances calls this function with a given this value and arguments provided individually.

*function Product(name, price) {*

*this.name = name;*

*this.price = price;*

*}*

*function Food(name, price) {*

*Product.call(this, name, price);*

*this.category = 'food';*

*}*

*console.log(new Food('cheese', 5).name);*

*// Expected output: "cheese"*

(function( $ ){

$.fn.createPassword = function( passBase, element, object) {

.....

return this;

};

})( jQuery );

Это расширение библиотеки jQuery методом createPassword. По виду типичный [IIFE](https://developer.mozilla.org/en-US/docs/Glossary/IIFE) с параметром jQuery

Соответственно "Password1Base", "Element", { Constructor: function(params) ...} в

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

## This Precedence

To determine which object this refers to; use the following precedence of order.

|  |
| --- |
|  |
| Precedence | Object |
| 1 | bind() |
| 2 | apply() and call() |
| 3 | Object method |
| 4 | Global scope |

// This is a function constructor:  
function myFunction(arg1, arg2) {  
  this.firstName = arg1;  
  this.lastName  = arg2;  
}  
  
// This creates a new object  
const myObj = new myFunction("John", "Doe");  
  
// This will return "John"  
myObj.firstName;

***$.fn.guageMeter = function(t)***

***{***

***}***

*$('#GaugeMeter\_1').gaugeMeter({ percent: 15 });*

t- вход парам-р

**var defaults = $.extend(**

{

id: ‘’, percent: 0, used: null, min: null, total: null, size: 100, prepend: '', append: '', theme: 'Red-Gold-Green',

color: '', back: 'RGBa(0,0,0,.06)', width: 3, style: 'Full', stripe: '0', animationstep: 1, animate\_gauge\_colors: false,

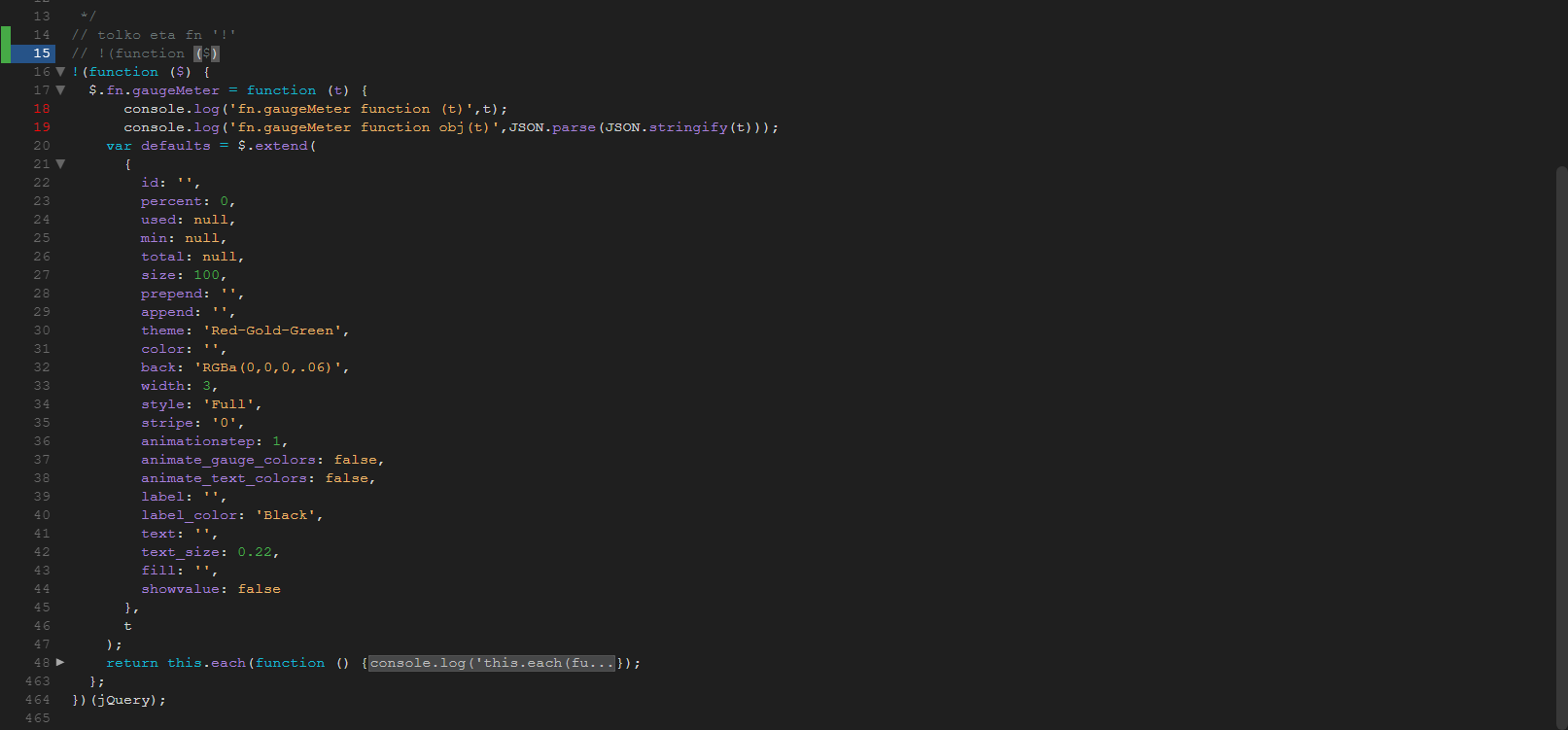
animate\_text\_colors: false, label: '', label\_color: 'Black', text: '', text\_size: 0.22, fill: '', showvalue: false

},

t

An **IIFE** (Immediately Invoked Function Expression)

https://web.archive.org/web/20171201033208/http://benalman.com/news/2010/11/immediately-invoked-function-expression/#iife



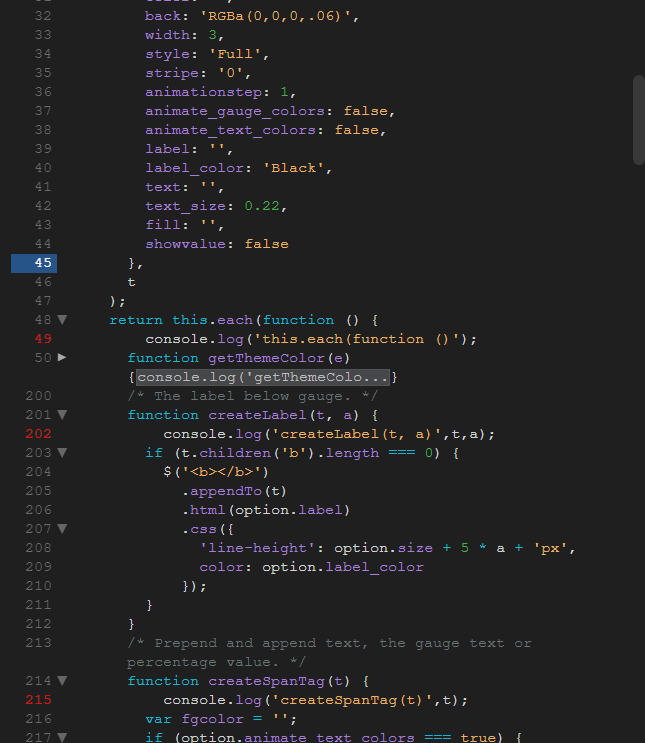
1. The first is the anonymous function with lexical scope enclosed within the [Grouping Operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Grouping) (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.
2. The second part creates the immediately invoked function expression () through which the JavaScript engine will directly interpret the function.

Эта ф-я вызывается каждый раз

**return this.each(function ()**

**getThemeColor**

эта ф-я палитра цветов для каждой цветовой гаммы



**Учавствует**

**return option.color;**

**и конструкции типа**

**'Red-Gold-Green' === option.theme &&**

**(e > 0 && (t = '#d90000'),**

**…**

**'White' === option.theme && (t = '#fff'),**

**'Black' === option.theme && (t = '#000'),**

**t**

**);**

И выбирает аргумент t

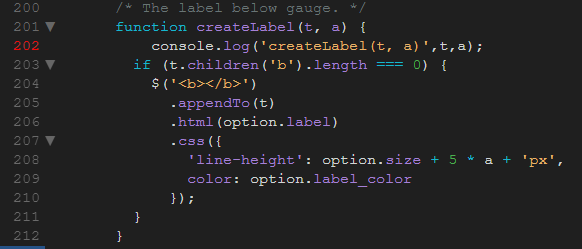
Что значит === в JS?

Оператор строгого равенства === **проверяет равенство без приведения типов**. Другими словами, если a и b имеют разные типы, то проверка a === b немедленно возвращает false без попытки их преобразования

**Созд лэбел под гагой**

t – text

a - size



до

1. "<span style=\"line-height: 100px; font-size: 22px;\">1.5<u> %</u></span><b style=\"line-height: 138.462px; color: black;\"></b><canvas width=\"100\" height=\"100\"></canvas>"

После

1. "<div class=\"GaugeMeter gaugeMeter\" id=\"GauAvPress\" data-append=\" %\" data-size=\"100\" data-width=\"15\" data-style=\"Arch\" data-animationstep=\"0\" data-id=\"GauAvPress\" style=\"width: 100px;\"><span style=\"line-height: 100px; font-size: 22px;\">1.5<u> %</u></span><b style=\"line-height: 138.462px; color: black;\"></b><canvas width=\"100\" height=\"100\"></canvas></div>"
2. outerText: "1.5 %"

/\* Prepend and append text, the gauge text or percentage value. \*/

function createSpanTag(t) {

var fgcolor = '';

if (option.animate\_text\_colors === true) {

fgcolor = option.fgcolor;

}

var child = t.children('span');

if (child.length !== 0) {

child.html(r).css({ color: fgcolor });

return;

}

if (option.text\_size <= 0.0 || Number.isNaN(option.text\_size)) {

option.text\_size = 0.22;

}

if (option.text\_size > 0.5) {

option.text\_size = 0.5;

}

$('<span></span>')

.appendTo(t)

.html(r)

.css({

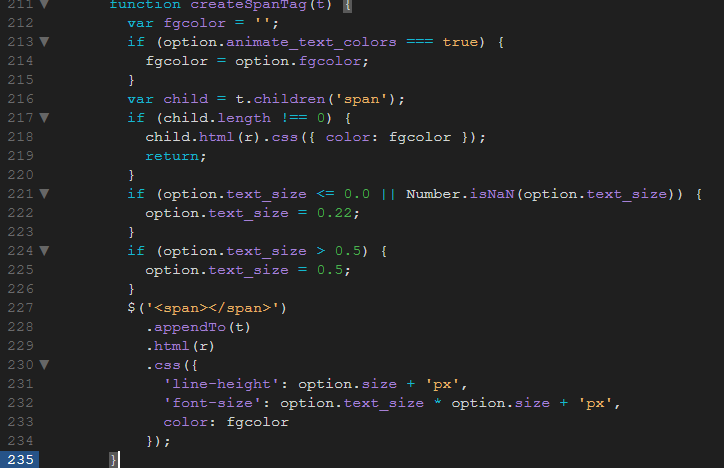
'line-height': option.size + 'px',

'font-size': option.text\_size \* option.size + 'px',

color: fgcolor

});

}



/\* Get data attributes as options from div tag. Fall back to defaults when not exists. \*/

function getDataAttr(t) {

$.each(dataAttr, function (index, element) {

if (t.data(element) !== undefined && t.data(element) !== null) {

option[element] = t.data(element);

} else {

option[element] = $(defaults).attr(element);

}

if (element === 'fill') {

s = option[element];

}

if (

(element === 'size' ||

element === 'width' ||

element === 'animationstep' ||

element === 'stripe') &&

!Number.isInteger(option[element])

) {

option[element] = parseInt(option[element]);

}

if (element === 'text\_size') {

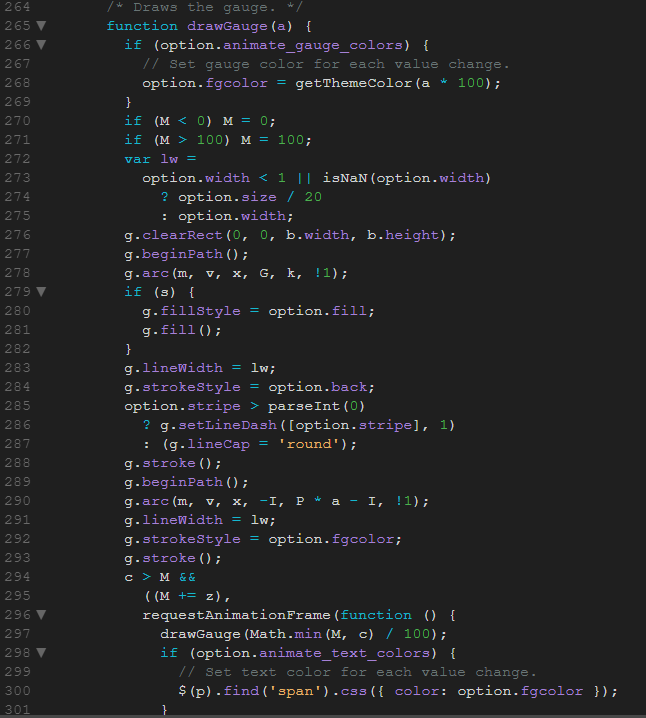
option[element] = parseFloat(option[element]);

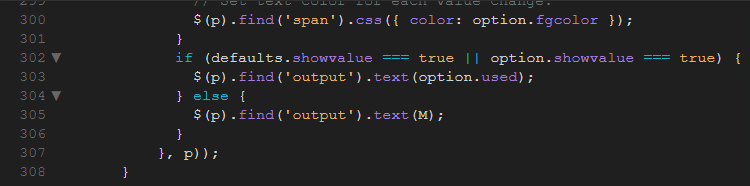
}

});

}







/\* Draws the gauge. \*/

function drawGauge(a) {

if (option.animate\_gauge\_colors) {

// Set gauge color for each value change.

option.fgcolor = getThemeColor(a \* 100);

}

if (M < 0) M = 0;

if (M > 100) M = 100;

var lw =

option.width < 1 || isNaN(option.width)

? option.size / 20

: option.width;

g.clearRect(0, 0, b.width, b.height);

g.beginPath();

g.arc(m, v, x, G, k, !1);

if (s) {

g.fillStyle = option.fill;

g.fill();

}

g.lineWidth = lw;

g.strokeStyle = option.back;

option.stripe > parseInt(0)

? g.setLineDash([option.stripe], 1)

: (g.lineCap = 'round');

g.stroke();

g.beginPath();

g.arc(m, v, x, -I, P \* a - I, !1);

g.lineWidth = lw;

g.strokeStyle = option.fgcolor;

g.stroke();

c > M &&

((M += z),

requestAnimationFrame(function () {

drawGauge(Math.min(M, c) / 100);

if (option.animate\_text\_colors) {

// Set text color for each value change.

$(p).find('span').css({ color: option.fgcolor });

}

if (defaults.showvalue === true || option.showvalue === true) {

$(p).find('output').text(option.used);

} else {

$(p).find('output').text(M);

}

}, p));

}

$(this).attr('data-id', $(this).attr('id'));

var r,

dataAttr = [

'percent',

'used',

'min',

'total',

'size',

'prepend',

'append',

'theme',

'color',

'back',

'width',

'style',

'stripe',

'animationstep',

'animate\_gauge\_colors',

'animate\_text\_colors',

'label',

'label\_color',

'text',

'text\_size',

'fill',

'showvalue',

'data'

],

option = {},

c = 0,

p = $(this),

s = false;

p.addClass('gaugeMeter');

getDataAttr(p);

if (Number.isInteger(option.used) && Number.isInteger(option.total)) {

var u = option.used;

var t = option.total;

if (Number.isInteger(option.min)) {

if (option.min < 0) {

t -= option.min;

u -= option.min;

}

}

c = u / (t / 100);

} else {

if (Number.isInteger(option.percent)) {

c = option.percent;

} else {

c = parseInt(defaults.percent);

}

}

if (c < 0) c = 0;

if (c > 100) c = 100;

if (

option.text !== '' &&

option.text !== null &&

option.text !== undefined

) {

if (

option.append !== '' &&

option.append !== null &&

option.append !== undefined

) {

r = option.text + '<u>' + option.append + '</u>';

} else {

r = option.text;

}

if (

option.prepend !== '' &&

option.prepend !== null &&

option.prepend !== undefined

) {

r = '<s>' + option.prepend + '</s>' + r;

}

} else {

if (defaults.showvalue === true || option.showvalue === true) {

r = '<output>' + option.used + '</output>';

} else {

r = '<output>' + c.toString() + '</output>';

}

if (

option.prepend !== '' &&

option.prepend !== null &&

option.prepend !== undefined

) {

r = '<s>' + option.prepend + '</s>' + r;

}

if (

option.append !== '' &&

option.append !== null &&

option.append !== undefined

) {

r = r + '<u>' + option.append + '</u>';

}

}

option.fgcolor = getThemeColor(c);

createSpanTag(p);

if (

option.style !== '' &&

option.style !== null &&

option.style !== undefined

) {

createLabel(p, option.size / 13);

}

$(this).width(option.size + 'px');

var b = $('<canvas></canvas>')

.attr({ width: option.size, height: option.size })

.get(0),

g = b.getContext('2d'),

m = b.width / 2,

v = b.height / 2,

x = (360 \* option.percent \* (Math.PI / 180), b.width / 2.5),

k = 2.3 \* Math.PI,

G = 0,

M = 0 === option.animationstep ? c : 0,

z = Math.max(option.animationstep, 0),

P = 2 \* Math.PI,

I = Math.PI / 2;

var child = $(this).children('canvas');

if (child.length !== 0) {

/\* Replace existing canvas when new percentage was written. \*/

child.replaceWith(b);

} else {

/\* Initially create canvas. \*/

$(b).appendTo($(this));

}

if ('Semi' === option.style) {

k = 2 \* Math.PI;

G = 3.13;

P = 1 \* Math.PI;

I = Math.PI / 0.996;

} else if ('Arch' === option.style) {

k = 2.195 \* Math.PI;

G = 655.99999;

P = 1.4 \* Math.PI;

I = Math.PI / 0.8335;

}

drawGauge(M / 100);

});

"fn.gaugeMeter function (t)" // [object Object]

{

"style": "Full",

"percent": 50,

"text": "1.5"

}

"fn.gaugeMeter function obj(t)" // [object Object]

{

"style": "Full",

"percent": 50,

"text": "1.5"

}

"this.each(function ()"

"getDataAttr(t)" // [object Array] (23)

["percent","used","min","total","size","prepend","append","theme","color","back","width","style","stripe","animationstep","animate\_gauge\_colors","animate\_text\_colors","label","label\_color","text","text\_size","fill","showvalue","data"]

"getThemeColor(e)" 50

"createSpanTag(t)" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

"<span></span> t r" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} "1.5<u> %</u>"

"createLabel(t, a)" // [object Object]

{

"0": {

"jQuery3310165555865579259052": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

7.6923076923076925

"drawGauge(a)" 0.5

"fn.gaugeMeter function (t)" // [object Object]

{

"style": "Semi",

"percent": 50,

"text": "1.5"

}

"fn.gaugeMeter function obj(t)" // [object Object]

{

"style": "Semi",

"percent": 50,

"text": "1.5"

}

"this.each(function ()"

"getDataAttr(t)" // [object Array] (23)

["percent","used","min","total","size","prepend","append","theme","color","back","width","style","stripe","animationstep","animate\_gauge\_colors","animate\_text\_colors","label","label\_color","text","text\_size","fill","showvalue","data"]

"getThemeColor(e)" 50

"createSpanTag(t)" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

}

"<span></span> t r" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} "1.5<u> %</u>"

"createLabel(t, a)" // [object Object]

{

"0": {

"jQuery331013650325198673242": {

"size": 100,

"append": " %",

"width": 15,

"style": "Arch",

"animationstep": 0

}

},

"length": 1

} 7.6923076923076925

**jQuery Syntax**

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

* A $ sign to define/access jQuery
* A (*selector*) to "query (or find)" HTML elements
* A jQuery *action*() to be performed on the element(s)

**Что такое $.extend**

Соединяет 2 и более объектов

When two or more object arguments are supplied to $.extend(), properties from all of the objects are added to the target object. Arguments that are null or undefined are ignored.

If only one argument is supplied to $.extend(), this means the target argument was omitted. In this case, the jQuery object itself is assumed to be the target. By doing this, you can add new functions to the jQuery namespace. This can be useful for plugin authors wishing to add new methods to JQuery.

Keep in mind that the target object (first argument) will be modified, and will also be returned from $.extend(). If, however, you want to preserve both of the original objects, you can do so by passing an empty object as the target:

|  |  |
| --- | --- |
| 1 | **var** object = $.extend({}, object1, object2); |

(function($){$(function(){

$('select').styleThis();

})})($)

$(function(){}) — это мы запускаем действие в момент загрузки страницы. А оборачиваем в анонимную функцию (function($){ })($);, чтобы избежать конфликта имен с $.  
  
Автор плагина предусмотрел случай, когда доллар занят под другую библиотеку/функцию/переменную. Чтобы избежать конфликта имен, достаточно будет заменить доллар на jQuery, вот так (хотя сразу так и нужно было писать):

(function($){$(function(){

$('select').styleThis();

})})(jQuery);

Вызов функции $() с аргумментом в виде анонимной функции, внутри замыкания (closure) в которое передаётся $. Весь этот изврат нужен для того что бы не было проблем со сборкой мусора... на практике в JQuery это просто выполнение функции по событию onDocumentReady.

Что означает запись $.function в javascript? Пример:

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

Давайте разбирать [$ - это псевдоним](https://stackoverflow.com/questions/8667736/what-does-the-sign-mean-in-jquery-or-javascript/8667760#:%7E:text=In%20jQuery%2C%20the%20%24%20sign%20is,an%20alias%20to%20a%20function.&text=The%20jQuery%20syntax%20is%20tailor,on%20the%20element(s).&text=The%20%24%20symbol%20simply%20invokes%20the%20jQuery%20library%27s%20selector%20functionality.) для jQuery()

Допустим что есть такой код

(function( $ ){

$.fn.createPassword = function( passBase, element, object) {

.....

return this;

};

})( jQuery );

Это расширение библиотеки jQuery методом createPassword. По виду типичный [IIFE](https://developer.mozilla.org/en-US/docs/Glossary/IIFE) с параметром jQuery

Соответственно "Password1Base", "Element", { Constructor: function(params) ...} в

$.createPassword("Password1Base", "Element", {

Constructor: function(params) {

//тело конструктора

}

}

это параметры вызова $.createPassword расширения jQuery

It doesn't need an index, since this provides the context. As noted by the [docs](http://api.jquery.com/jQuery.each/), "The value can also be accessed through the this keyword." This is accomplished by using [call](https://developer.mozilla.org/en/Core_JavaScript_1.5_Reference/Objects/Function/call). Something like:

userFunction.call(valueOfElement, indexInArray, valueOfElement);

$.fn.imgAreaSelect = function (options)

means the function is being added to the prototype. This allows it to be used with any instance of the jQuery object.

В JavaScript, очень распространена практика использования восклицательного знака перед функцией. Это может вызвать некоторое недоумение, особенно у тех, кто только начинает изучать язык. Чтобы разобраться в этом, рассмотрим следующий пример:

|  |  |
| --- | --- |
| 1  2  3 | !function() {      // Какой-то код...  }(); |

Вот что происходит. Здесь восклицательный знак (!) используется перед функцией для того, чтобы JavaScript интерпретировал выражение как функциональное выражение, а не как объявление функции.

В JavaScript, функциональные выражения и объявления функций обрабатываются по-разному. Объявления функций «всплывают» в верхушку своего контекста выполнения, что позволяет вызывать функцию до того, как она была объявлена. С другой стороны, функциональные выражения не «всплывают» и могут быть вызваны только после того, как они были определены.

Когда перед функцией стоит восклицательный знак, JavaScript принимает это как функциональное выражение, а не как объявление функции. Это позволяет немедленно вызвать функцию.

В этом контексте, восклицательный знак действует как логическое отрицание. Он превращает результат функции в булево значение и затем инвертирует его. Но поскольку цель здесь в основном состоит в том, чтобы вызвать функцию, а не получить булево значение, результат этого отрицания обычно игнорируется.

Важно заметить, что вместо восклицательного знака можно использовать и другие операторы, такие как +, -, ~ или void. Они все смогут превратить объявление функции в функциональное выражение, позволяя его немедленно вызвать.

В итоге, восклицательный знак перед функцией в JavaScript — это просто способ сделать функцию немедленно вызываемой.

The each function can take a function accepting an index as a parameter, but it's optional.For simplicity's sake,

.each

 was implemented to have

this

 refer to the current element.However,

.each

can accept an index as a parameter to it's callback.There's an example of that usage in the jQuery API

$('li').each(function(index) {

alert(index + ': ' + $(this).text());

});

The .each() method is designed to make DOM looping constructs concise and less error-prone. When called it iterates over the DOM elements that are part of the jQuery object. Each time the callback runs, it is passed the current loop iteration, beginning from 0. More importantly, the callback is fired in the context of the current DOM element, so the keyword this refers to the element.

$.each(fn) calls fn for each element contained in the current context. Each time it calls fn, it passes the "current" element as this.

So in the following example:

$("div").each(function() {

alert(this.className);

});

Will pop up one alert for each <div> in the DOM, and display the class name of each.

# [**What's the equivalent of 'getElementsByTagName' in jQuery?**](https://stackoverflow.com/questions/4398553/whats-the-equivalent-of-getelementsbytagname-in-jquery)

48

$("tagnamehere")

So:

$("div").each(function() {

// do something exciting with each div

$(this).css("border", "1px solid red");

// do something by directly manipulating the wrapped DOM element

this.style.border = "1px solid red";

// do something only if this particular div has a class of 'pretty'

if($(this).hasClass("pretty")) {

$(this).text("I am the pretty one");

}

});

or just:

// apply some css to all div elements

$("div").css("border", "1px solid red");

Keep in mind that when you use jQuery to select a number of elements, e.g. $("span"), any method you invoke on the object will happen on all matched elements. Think of it as 'implicit iteration' - e.g. $("span").hide(); will hide all span elements on the page.

See:

* <http://www.w3.org/TR/css3-selectors/>
* <http://api.jquery.com/category/selectors/>

Just use the element selector

$('elementname')

E.g.

$('div')

And to do the iteration:

$('div').each(function(){

var $this = $(this);

//insert code here

});

You may not have to iterate, however, as a method called upon the collection will be called for each item in the collection, so

$('div').hide();

...will hide all divs.

[Share](https://stackoverflow.com/a/4398573)

Given a string tagName, this statement:

document.getElementsByTagName(tagName)

...could be written using jQuery:

$(tagName)

Given a native DOM element element and a string tagName, this statement:

element.getElementsByTagName(tagName)

...could be written using jQuery:

$(element).find(tagName)

**Note :** Using getElementsByTagName you get native DOM elements, meanwhile using jQuery statements you get jQuery objects. If you want native DOM elements with jQuery you could use get() - [How do I pull a native DOM element from a jQuery object?](https://learn.jquery.com/using-jquery-core/faq/how-do-i-pull-a-native-dom-element-from-a-jquery-object/)

## Answer: Use the jQuery attr() Method

You can simply use the jQuery attr() method to get or set the ID attribute value of an element.

The following example will display the ID of the DIV element in an alert box on button click.

https://www.tutorialrepublic.com/faq/how-to-get-the-id-of-an-element-using-jquery.php

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=faq&file=jquery-get-id-of-an-element)

<!DOCTYPE html>

<html lang="en">

<head>

<title>jQuery Get ID of an Element</title>

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<style>

div{

padding: 20px;

background: #abb1b8;

}

</style>

<script>

$(document).ready(function(){

$("#myBtn").click(function(){

var elmId = $("#test").attr("id");

alert(elmId);

});

});

</script>

</head>

<body>

<div id="test">#text</div>

<br>

<button type="button" id="myBtn">Show Div ID</button>

</body>

</html>

You can also get the ID of multiple elements having same class through loop, like this:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=faq&file=jquery-get-id-of-multiple-elements)

<!DOCTYPE html>

<html lang="en">

<head>

<title>jQuery Get ID of Multiple Elements</title>

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<style>

div{

padding: 20px;

margin-bottom: 10px;

background: #abb1b8;

}

</style>

<script>

$(document).ready(function(){

$("#myBtn").click(function(){

var idArr = [];

$(".box").each(function(){

idArr.push($(this).attr("id"));

});

// Join array elements and display in alert

alert(idArr.join(", "));

});

});

</script>

</head>

<body>

<div class="box" id="boxOne">#boxOne</div>

<div class="box" id="boxTwo">#boxTwo</div>

<div class="box" id="boxThree">#boxThree</div>

<button type="button" id="myBtn">Show ID List</button>

</body>

</html>

You can also get the ID of individual element when using class selector based on index starting from 0, for example, to get the ID of first element in a set of matched elements you can use $(".box").get(0).id or $(".box")[0].id.

Similarly, to get the ID of last element you can use something like this, $(".box").get($(".box").length - 1).id or $(".box")[$(".box").length - 1].id, because jQuery selector returns a collection of matched elements not a single element.

## Related FAQ

Here are some more FAQ related to this topic:

* [How to find an element based on a data-attribute value in jQuery](https://www.tutorialrepublic.com/faq/how-to-find-an-element-based-on-a-data-attribute-value-in-jquery.php)
* [How to get the children of the this selector in jQuery](https://www.tutorialrepublic.com/faq/how-to-get-the-children-of-the-this-selector-in-jquery.php)
* [How to add attribute to an HTML element in jQuery](https://www.tutorialrepublic.com/faq/how-to-add-attribute-to-an-html-element-in-jquery.php)

Предположим, у вас есть простой неупорядоченный список на странице:

|  |  |
| --- | --- |
| 1  2  3  4 | <ul>  <li>foo</li>  <li>bar</li>  </ul> |

Вы можете выбрать элементы списка и сделать итерацию по ним:

|  |  |
| --- | --- |
| 1  2  3 | $( "li" ).each(**function**( index ) {  console.log( index + ": " + $( **this** ).text() );  }); |

Сообщение будет выведено для каждого элемента следующим образом:

0: foo  
1: bar

https://jquery-docs.ru/examples/

**Работа с jquery Canvas Guage**

https://canvas-gauges.com/documentation/user-guide/

**var** gauge **=** **new** LinearGauge({

renderTo: 'gauge-id',

colorNumbers: 'red',

width: 100,

height: 300

})

<canvas data-type="linear-gauge"

data-color-numbers="red"

data-width="100"

data-height="300"

></canvas>

Canvas gauges supports dynamic re-configuration at runtime calling a special update() method or by dynamically changing HTML element attributes:

gauge.update({ colorNumbers: 'blue' });

is similar to:

$('canvas[data-type="linear-gauge"]').attr('data-color-numbers', 'blue');

<script src="gauge.min.js"></script>

<canvas data-type="linear-gauge"

data-width="200"

data-height="600"

></canvas>

<script>

**var** gaugeElement **=** document.getElementsByTagName('canvas')[0];

gaugeElement.setAttribute('data-border-radius', 20);

gaugeElement.setAttribute('data-color-numbers', 'red');

gaugeElement.setAttribute('data-type', 'radial-gauge');

gaugeElement.setAttribute('data-type', 'linear-gauge');

</script>

<https://canvas-gauges.com/documentation/user-guide/using-as-component>

https://www.tutorialrepublic.com/jquery-tutorial/jquery-events.php

CANVAS GAUGES USER GUIDE

# **Gauges As HTML Components**

How to use canvas gauges as HTML components

#### **Table of Contents**

* [**Declaring Gauges**](https://canvas-gauges.com/documentation/user-guide/using-as-component#declaring-gauges)
* [**Mutating Gauges**](https://canvas-gauges.com/documentation/user-guide/using-as-component#mutating-gauges)
* [**Configuration Attributes Mapping**](https://canvas-gauges.com/documentation/user-guide/using-as-component#configuration-attributes-mapping)

#### **User Guide Chapters**

* [**Introduction**](https://canvas-gauges.com/documentation/user-guide/)
* [**Configuration**](https://canvas-gauges.com/documentation/user-guide/configuration)
* [**Creating Custom Builds**](https://canvas-gauges.com/documentation/user-guide/custom-builds)
* [**Scripting API**](https://canvas-gauges.com/documentation/user-guide/scripting-api)
* [**Advanced Usage**](https://canvas-gauges.com/documentation/user-guide/advanced-usage)
* [**Migration Guide**](https://canvas-gauges.com/migration/)

Canvas gauges provides a declarative way to injection gauges into HTML-page. Each gauge is a simply HTML canvas element. When the custom element obtains attribute “data-type” with one of “linear-gauge” or “radial-gauge” values, it automatically becomes treated as gauge component and is rendered correspondingly.

Gauges support injection/modification of config options via adding/changing/removing configuration attributes at runtime. All configuration attributes should start with “data-“ prefix to produce valid HTML output. All attribute names correspondingly mapped into gauge configuration options.

## Declaring Gauges

Zero-configuration gauge:

<script src="gauge.min.js"></script>

<canvas data-type="linear-gauge"></canvas>

With some configuration options:

<script src="gauge.min.js"></script>

<canvas data-type="linear-gauge"

data-width="160"

data-height="600"

data-border-radius="0"

data-borders="0"

data-bar-begin-circle="25"

data-minor-ticks="10"

data-value="36.6"

data-min-value="35"

data-max-value="42"

data-title="°C"

data-major-ticks="35,36,37,38,39,40,41,42"

data-ticks-width="18"

data-ticks-width-minor="7.5"

data-bar-width="5"

data-highlights="false"

data-color-value-box-shadow="false"

data-value-box-stroke="0"

data-color-value-box-background="false"

data-value-int="2"

data-value-dec="1"

></canvas>

## Mutating Gauges

Canvas gauges support mutations at runtime, so you can easily re-configure gauge by simply changing attributes values. Even gauge type can be changed.

There are some configuration options which are defined for a certain type of gauge only. By the way even if you will declare attributes which gauge does not support it won’t break anything - them will just won’t be parsed and took into account during rendering. So switching gauge type in-runtime always safe.

For example:

<script src="gauge.min.js"></script>

<canvas data-type="linear-gauge"

data-width="200"

data-height="600"

></canvas>

<script>

**var** gaugeElement **=** document.getElementsByTagName('canvas')[0];

gaugeElement.setAttribute('data-border-radius', 20);

gaugeElement.setAttribute('data-color-numbers', 'red');

gaugeElement.setAttribute('data-type', 'radial-gauge');

gaugeElement.setAttribute('data-type', 'linear-gauge');

</script>

## Configuration Attributes Mapping

When using HTML declarative configuration via attributes, mandatory “renderTo” configuration option is not required as far as gauge will be rendered to a proper canvas element directly.

| **Options** | **Attribute** | **Gauge Type** | **Value Type** |
| --- | --- | --- | --- |
| renderTo | - | any | string or HTMLCanvasElement |
| width | data-width | any | number |
| height | data-height | any | number |
| minValue | data-min-value | any | number |
| maxValue | data-max-value | any | number |
| value | data-value | any | number |
| title | data-title | any | string |
| units | data-units | any | string |
| majorTicks | data-major-ticks | any | array of string or number |
| minorTicks | data-minor-ticks | any | integer |
| strokeTicks | data-stroke-ticks | any | boolean |
| animatedValue | data-animated-value | any | boolean |
| borders | data-borders | any | boolean |
| valueInt | data-value-int | any | integer |
| valueDec | data-value-dec | any | integer |
| majorTicksInt | data-major-ticks-int | any | integer |
| majorTicksDec | data-major-ticks-dec | any | integer |
| animation | data-animation | any | boolean |
| animationDuration | data-animation-duration | any | number |
| animationRule | data-animation-rule | any | string or function |
| colorPlate | data-color-plate | any | string |
| colorMajorTicks | data-color-major-ticks | any | string |
| colorMinorTicks | data-color-minor-ticks | any | string |
| colorTitle | data-color-title | any | string |
| colorUnits | data-color-units | any | string |
| colorNumbers | data-color-numbers | any | string |
| colorNeedle | data-color-needle | any | string |
| colorNeedleEnd | data-color-needle-end | any | string |
| colorValueText | data-color-value-text | any | string |
| colorValueTextShadow | data-color-value-text-shadow | any | string |
| colorBorderShadow | data-color-border-shadow | any | string |
| colorBorderOuter | data-color-border-outer | any | string |
| colorBorderOuterEnd | data-color-border-outer-end | any | string |
| colorBorderMiddle | data-color-border-middle | any | string |
| colorBorderMiddleEnd | data-color-border-middle-end | any | string |
| colorBorderInner | data-color-border-inner | any | string |
| colorBorderInnerEnd | data-color-border-inner-end | any | string |
| colorValueBoxRect | data-color-value-box-rect | any | string |
| colorValueBoxRectEnd | data-color-value-box-rect-end | any | string |
| colorValueBoxBackground | data-color-value-box-background | any | string |
| colorValueBoxShadow | data-color-value-box-shadow | any | string |
| colorNeedleShadowUp | data-color-needle-shadow-up | any | string |
| colorNeedleShadowDown | data-color-needle-shadow-down | any | string |
| fontNumbers | data-font-numbers | any | string |
| fontTitle | data-font-title | any | string |
| fontUnits | data-font-units | any | string |
| fontValue | data-font-value | any | string |
| needle | data-needle | any | boolean |
| needleShadow | data-needle-shadow | any | boolean |
| needleType | data-needle-type | any | string |
| needleStart | data-needle-start | any | number |
| needleEnd | data-needle-end | any | number |
| needleWidth | data-needle-width | any | number |
| borderOuterWidth | data-border-outer-width | any | number |
| borderMiddleWidth | data-border-middle-width | any | number |
| borderInnerWidth | data-border-inner-width | any | number |
| borderShadowWidth | data-border-shadow-width | any | number |
| valueBox | data-value-box | any | boolean |
| valueBoxStroke | data-value-box-stroke | any | number |
| valueText | data-value-text | any | string |
| valueTextShadow | data-value-text-shadow | any | boolean |
| valueBoxBorderRadius | data-value-box-border-radius | any | number |
| highlights | data-highlights | any | array of { from: number, to: number, color: string } |
| borderRadius | data-border-radius | linear | number |
| barBeginCircle | data-bar-begin-circle | linear | number |
| barWidth | data-bar-width | linear | number |
| barStrokeWidth | data-bar-stroke-width | linear | number |
| barProgress | data-bar-progress | linear | boolean |
| colorBarStroke | data-color-bar-stroke | linear |  |
| colorBar | data-color-bar | linear | string |
| colorBarEnd | data-color-bar-end | linear | string |
| colorBarProgress | data-color-bar-progress | linear | string |
| colorBarProgressEnd | data-color-bar-progress-end | linear | string |
| tickSide | data-tick-side | linear | string |
| needleSide | data-needle-side | linear | string |
| numberSide | data-number-side | linear | string |
| ticksWidth | data-ticks-width | linear | number |
| ticksWidthMinor | data-ticks-width-minor | linear | number |
| ticksPadding | data-ticks-padding | linear | number |
| barLength | data-bar-length | linear | number |
| fontNumbersSize | data-font-numbers-size | linear | number |
| fontTitleSize | data-font-title-size | linear | number |
| fontUnitsSize | data-font-units-size | linear | number |
| ticksAngle | data-ticks-angle | radial | number |
| startAngle | data-start-angle | radial | number |
| colorNeedleCircleOuter | data-color-needle-circle-outer | radial | string |
| colorNeedleCircleOuterEnd | data-color-needle-circle-outer-end | radial | string |
| colorNeedleCircleInner | data-color-needle-circle-inner | radial | string |
| colorNeedleCircleInnerEnd | data-color-needle-circle-inner-end | radial | string |
| needleCircleSize | data-needle-circle-size | radial | number |
| needleCircleInner | data-needle-circle-inner | radial | boolean |
| needleCircleOuter | data-needle-circle-outer | radial | boolean |
| animationTarget | data-animation-target | radial | string |

Read more: [**all configuration options**](https://canvas-gauges.com/documentation/user-guide/configuration)

# **All Configuration Options**

Complete list of available configuration options for gauges

#### **Table of Contents**

* [**Common Configuration options**](https://canvas-gauges.com/documentation/user-guide/configuration#common-configuration-options)
  + [**Mandatory Options**](https://canvas-gauges.com/documentation/user-guide/configuration#mandatory-options)
  + [**Basic Options**](https://canvas-gauges.com/documentation/user-guide/configuration#basic-options)
  + [**Ticks Bar Options**](https://canvas-gauges.com/documentation/user-guide/configuration#ticks-bar-options)
  + [**Progress Bar Options**](https://canvas-gauges.com/documentation/user-guide/configuration#progress-bar-options)
  + [**Animation Options**](https://canvas-gauges.com/documentation/user-guide/configuration#animation-options)
  + [**Coloring Options**](https://canvas-gauges.com/documentation/user-guide/configuration#coloring-options)
  + [**Needle Configuration Options**](https://canvas-gauges.com/documentation/user-guide/configuration#needle-configuration-options)
  + [**Borders Options**](https://canvas-gauges.com/documentation/user-guide/configuration#borders-options)
  + [**Value Box Options**](https://canvas-gauges.com/documentation/user-guide/configuration#value-box-options)
  + [**Fonts Customization Options**](https://canvas-gauges.com/documentation/user-guide/configuration#fonts-customization-options)
* [**Gauge-Specific Configuration Options**](https://canvas-gauges.com/documentation/user-guide/configuration#gauge-specific-configuration-options)
  + [**Linear Gauge Options**](https://canvas-gauges.com/documentation/user-guide/configuration#linear-gauge-options)
    - [**Borders Options**](https://canvas-gauges.com/documentation/user-guide/configuration#borders-options-1)
    - [**Progress Bar Options**](https://canvas-gauges.com/documentation/user-guide/configuration#progress-bar-options-1)
    - [**Coloring Options**](https://canvas-gauges.com/documentation/user-guide/configuration#coloring-options-1)
    - [**Element Positioning Options**](https://canvas-gauges.com/documentation/user-guide/configuration#element-positioning-options)
    - [**Ticks Bar Options**](https://canvas-gauges.com/documentation/user-guide/configuration#ticks-bar-options-1)
  + [**Radial Gauge Options**](https://canvas-gauges.com/documentation/user-guide/configuration#radial-gauge-options)
    - [**Ticks Bar Options**](https://canvas-gauges.com/documentation/user-guide/configuration#ticks-bar-options-2)
    - [**Coloring Options**](https://canvas-gauges.com/documentation/user-guide/configuration#coloring-options-2)
    - [**Needle Options**](https://canvas-gauges.com/documentation/user-guide/configuration#needle-options)
    - [**Animation Options**](https://canvas-gauges.com/documentation/user-guide/configuration#animation-options-1)

#### **User Guide Chapters**

* [**Introduction**](https://canvas-gauges.com/documentation/user-guide/)
* [**Creating Custom Builds**](https://canvas-gauges.com/documentation/user-guide/custom-builds)
* [**Using As Components**](https://canvas-gauges.com/documentation/user-guide/using-as-component)
* [**Scripting API**](https://canvas-gauges.com/documentation/user-guide/scripting-api)
* [**Advanced Usage**](https://canvas-gauges.com/documentation/user-guide/advanced-usage)
* [**Migration Guide**](https://canvas-gauges.com/migration/)

Canvas gauges can be configured in two ways:

* using JavaScript API
* using HTML-component API

JavaScript API provides two ways of bypassing configuration options: on the object instantiation via constructor; in runtime using update() method.

HTML-component API simply provides an ability to add/change/remove configuration attributes on HTML gauge element.

Semantically JavaScript and HTML use different naming conventions which are standard in the industry for today: HTML attribute name is prefixed with “data-“ and the option name is dash-split; JavaScript options names are following camelCase rule.

JavaScript example:

**var** options **=** {

minValue: **-**100,

maxValue: 100,

animationRule: 'elastic',

animationDuration: 500

};

Equivalent example in HTML-definition would be:

<canvas data-type="linear-gauge"

data-min-value="-100"

data-max-value="100"

data-animation-rule="elastic"

data-animation-duration="500"

></canvas>

Sometimes, the values can be a complex-data structures, like arrays or objects. In this case in JavaScript them simply defined as standard notations, but in HTML-attributes definitions the following rules are applied:

* for arrays of primitives it is allowed to define a comma-separated string of values;
* it is allowed to define the value as valid JSON notation;

Examples:

Array of primitives:

<canvas data-type="radial-gauge"

data-major-ticks="0,20,40,60,80,100"

></canvas>

<canvas data-type="radial-gauge"

data-major-ticks="N,NE,E,SE,S,SW,W,NW"

></canvas>

JSON notations:

<canvas data-type="radial-gauge"

data-major-ticks='["N","NE","E","SE","S","SW","W","NW"]'

></canvas>

<canvas data-type="radial-gauge"

data-highlights='[

{ "from": 0, "to": 50, "color": "rgba(0,255,0,.15)" },

{ "from": 50, "to": 100, "color": "rgba(255,255,0,.15)" },

{ "from": 100, "to": 150, "color": "rgba(255,30,0,.25)" },

{ "from": 150, "to": 200, "color": "rgba(255,0,225,.25)" },

{ "from": 200, "to": 220, "color": "rgba(0,0,255,.25)" }

]'

></canvas>

All available options for configuring gauges listed below.

## Common Configuration options

Common configuration options spread across all type of the gauges and means they are applicable to any gauge type. For been more informative and easy-to-find we split those options into groups below.

### **Mandatory Options**

* **renderTo**: render target in DOM tree. It is expected to be a canvas element, or its identifier in a DOM tree. This option is not required when the gauge injected as a web-component on the page.

### **Basic Options**

* **width**: number in pixels of the canvas element on which the gauge will be drawn.
* **height**: number in pixels of the canvas element on which the gauge will be drawn.
* **minValue**: numeric minimal value which will be shown on a gauge bar.
* **maxValue**: numeric maximal value which will be shown on a gauge bar.
* **value**: current gauge value which will be displayed.
* **units**: should be a string explaining the units for the gauge value, or something falsy to hide this element on a gauge.
* **title**: should be a string to display gauge title or falsy value to hide this element.
* **listeners**: an object defining all handlers which are going to be used on gauge initialization. Each event can have one or more handlers (a function or an array of functions). Example:

**var** gauge **=** **new** RadialGauge({

renderTo: 'some-id',

listeners: {

value: **function**(newValue, oldValue) {

*// do something*

},

animationEnd: [

**function**() { */\* handler one \*/* },

**function**() { */\* handler two \*/* }

]

}

});

### **Ticks Bar Options**

Tick bars on a gauge representing the measuring system which visualize the gauge measuring intervals, and the currently upset value. It should be upset in mind that ticks configuration must be relied properly on a given minValue and maxValue or you could get confusing display result otherwise.

* **exactTicks**: a boolean flag, which switch ticks drawing modes. By default, is turned off (false), meaning the tick bar will be draw by the following rules: a) majorTicks array will define a number of **equal by length** sections; b) minorTicks will define a number of ticks to draw between 2 nearest majorTicks. If the value set to true the rules are following: a) majorTicks array will define exact values where major ticks should be drawn (it should be an array of exact values, not arbitrary labels); b) minorTicks defines a value step for drawing minor tick on a bar. See this [**example**](https://rawgit.com/Mikhus/canvas-gauges/master/examples/exact-ticks-bar.html) [[**Source**](https://github.com/Mikhus/canvas-gauges/blob/master/examples/exact-ticks-bar.html)] for exact ticks bar configuration and behavior.
* **majorTicks**: expected to be an array of numeric or string values which will be displayed on a gauge bar as major ticks. This array defines labels for the ticks. Array length defines a number of sections on a ticks bar.
* **minorTicks**: is an integer number which defines how many minor ticks have to be drawn between two neighbour major ticks.
* **strokeTicks**: boolean value defining if ticks bar of the gauge should be stroked or not. This only relies on a visual effect.
* **majorTicksInt**: integer which defines how many numeric positions should be used to display integer part of the tick number.
* **majorTicksDec**: integer which defines how many positions should be used to display decimal part of the tick number.
* **highlights**: an array of highlights objects, which configures color-highlighted areas on a ticks bar. Each highlight object defines an area to colorize starting **from** value **to** value and using a given **color**, like this: { from: number, to: number, color: string }
* **highlightsWidth**: sets the width of highlights area in relative units.
* **numbersMargin**: defines a margin for tick labels (numbers) in relative units. By default, is 1.

### **Progress Bar Options**

* **barWidth**: bar width in percents in relation to overall width of the gauge. It is limited to 50% anyway.
* **barStrokeWidth**: defines a width of a bar stroke. If set to zero - stroke won’t be drawn.
* **barProgress**: flag, defines if a progress bar should be drawn within this gauge.
* **barShadow**: number, length of the inner bar shadow if required. By default, is 0.

### **Animation Options**

Animations on the gauge can be turned on or off. Whenever the animation is turned on it will automatically run each time gauge changing its value. During the animation gauge will animate its needle or progress bar from the old value to a new value it has been upset. If animatedValue option is turned on it will also constantly update the value displayed in a value box on each animation step.

* **animation**: boolean flag signaling whenever the animation is possible on the gauge or not.
* **animationDuration**: time in milliseconds of the animation duration.
* **animationRule**: defines a type of animation behavior for the gauge. Canvas gauges already knows the most used types of animation rules, or you can define your own animation rule providing the animation rule function within this option. Known rules could be bypassed as string names, which are: “linear”, “quad”, “quint”, “cycle”, “bounce”, “elastic” and their opposites: “dequad”, “dequint”, “decycle”, “debounce”, “delastic”.
* **animatedValue**: a boolean flag, specifies if a value displayed in a value box of the gauge should be constantly updated during animation run. By default, it is falsy, so the upset gauge value will be shown immediately and animation will run visually only on the gauge needle or progress bar.
* **animateOnInit**: a boolean flag, which specifies if gauge should be animated on the first draw, by default is false.

### **Coloring Options**

Canvas gauge provides highly customizable coloring options for the majority of gauge elements. Each color configuration is usually a string value representing the color in one of HEX (#000000-#FFFFFF), RGB (rgb(0, 0, 0)-rgb(255,255,255)) or RGBA (rgba(0,0,0,0)-rgba(255,255,255,1)) formats. Some elements supports gradients. In this case the color of an element could be configured as color start and color end parts.

* **colorPlate**: defines background color of the gauge plate.
* **colorPlateEnd**: if specified wil use gradient fill for the plate.
* **colorMajorTicks**: color of the major ticks lines (also applied to stroke if strokeTicks option is true). It can be an array of colors, for each major tick it is possible to specify specific color. In this case if strokeTicks enabled, the first color from this array will be used for stroking.
* **colorMinorTicks**: color of the minor ticks lines.
* **colorStrokeTicks**: defines a static color for all ticks lines. By default, is not specified. If set to some color value will override major ticks stroke color for lines, but will not influence numbers colors. For minor ticks will be used if colorMinorTicks is not specified.
* **colorTitle**: color of the title text.
* **colorUnits**: color of the units text.
* **colorNumbers**: color of the text for the tick numbers. It can be an array of colors, containing specific color for each number.
* **colorNeedle**: defines color of the gauge needle.
* **colorNeedleEnd**: if defined it enables use of gradient for the gauge needle. If this is falsy, needle will be draw using solid color.
* **colorValueText**: defines a color of the text in a value box.
* **colorValueTextShadow**: defines a color of a text in a value box. If this value is falsy shadow won’t be drawn.
* **colorBorderShadow**: defines a shadow color of the gauge plate. If is falsy the shadow won’t be drawn.
* **colorBorderOuter**: defines a color of the outer border for the gauge plate.
* **colorBorderOuterEnd**: if defined it enables use of gradient on the outer border.
* **colorBorderMiddle**: defines a color of the middle border for the gauge plate.
* **colorBorderMiddleEnd**: if defined it enables use of gradient on the middle border.
* **colorBorderInner**: defines a color of the inner border for the gauge plate.
* **colorBorderInnerEnd**: if defined it enables use of gradient on the inner border.
* **colorValueBoxRect**: defines a color of the value box rectangle stroke.
* **colorValueBoxRectEnd**: if defined it enables use of gradient on value box rectangle stroke.
* **colorValueBoxBackground**: defines background color for value box.
* **colorValueBoxShadow**: defines a color of value box shadow. If falsy shadow won’t be drawn.
* **colorNeedleShadowUp**: defines upper half of the needle shadow color.
* **colorNeedleShadowDown**: defines drop shadow needle color.
* **colorBarStroke**: color of a bar stroke.
* **colorBar**: defines a bar background color.
* **colorBarProgress**: defines a progress bar color.
* **highlightsLineCap**: ‘round’, ‘square’ or ‘butt’. Default is ‘butt’. It sets the context.lineCap within the drawRadialHighLights function of the RadialGauge object. Then format option to set the “units” attribute. For example “{value} % {title}” which replaces the attributes inside {} to the same member in the option object. So if title set to “Hour” and value to “50” the units will be “50% Hour”.

### **Needle Configuration Options**

Gauge needle is an element which visualize the current position of the gauge value on a measuring bar. Currently, canvas gauge supports drawing of two different types of the needles for each gauge - “line” needle and “arrow” needle. By the way, whenever it may be required, needle may be not drawn at all.

* **needle**: boolean, specifies if gauge should draw the needle or not.
* **needleShadow**: boolean, specifies if needle should drop shadow or not.
* **needleType**: string, one of “arrow” or “line” supported.
* **needleStart**: tail part of the needle length, in relative units.
* **needleEnd**: main needle length in relative units.
* **needleWidth**: max width of the needle in the widest needle place.

### **Borders Options**

Canvas gauge plate provides a way to define the borders. There are 3 borders available to draw on the edge of the gauge plate. It is possible to combine the borders display options, their widths and colors to achieve exclusive visual look & feel of your gauges.

* **borders**: boolean, defines if borders should be drawn or not.
* **borderOuterWidth**: specifies a width in pixels of the outer border. If set to zero - border won’t be drawn at all.
* **borderMiddleWidth**: specifies a width in pixels of the middle border. If set to zero - border won’t be drawn at all.
* **borderInnerWidth**: specifies a width in pixels of the inner border. If set to zero - border won’t be drawn at all.
* **borderShadowWidth**: specifies the width of the outer border drop shadow. If zero - shadow won’t be drawn.

### **Value Box Options**

Value box element on the gauge intended to display the digital representation of the current value. it is the most accurate visualisation of the exact value shawn by the gauge on the measuring bar. Whenever it is not required it may be turned off and not drawn.

* **valueBox**: boolean, defines if the value box should be drawn or not on the gauge.
* **valueBoxStroke**: number in relative units which defines the width of stroke of the value box element.
* **valueBoxWidth**: if set and is greater than value text real width - will be set as configured. This value is expected to be a percent in relation to gauge width.
* **valueText**: text to display instead of showing the current value. It may be useful when it is required to display something different in value box.
* **valueTextShadow**: specifies if value text shadow should be drawn or not.
* **valueBoxBorderRadius**: number of radius to draw rounded corners of the value box.
* **valueInt**: integer which defines how many numeric positions should be used to display integer part of the value number.
* **valueDec**: integer which defines how many positions should be used to display decimal part of the value number.

### **Fonts Customization Options**

Canvas gauges enables use of custom fonts when drawing text elements. As far as gauges are build on principals of minimalist code base there is no hardcoded fonts integrated with the gauges. Canvas gauges only provides a way to upset a custom font-family to its different text elements, but the font loading and initialization on the page is a part of the work user has to do himself.

* **fontNumbers**: specifies font family for the tick numbers.
* **fontTitle**: specifies font family for title text.
* **fontUnits**: specifies font family for units text.
* **fontValue**: specifies font-family for value box text.
* **fontNumbersSize**: Size of the font for tick numbers in relative units.
* **fontTitleSize**: Size of the font for title element text in relative units.
* **fontUnitsSize**: Size of the font for units element text in relative units.
* **fontValueSize**: Size of the font using for drawing value in a value box.

Since version 2.0.6 there is added more advanced font styling features. Now it is possible to set font style and font weight on each text element of the gauge. Where font style can be one of ‘normal’ (default) ‘italic’ or ‘oblique’ values. Font weight can be one of ‘normal’ (default), ‘bold’, ‘bolder’, ‘lighter’, ‘100’, ‘200’, ‘300’, ‘400’, ‘500’, ‘600’, ‘700’, ‘800’ or ‘900’ values.

* **fontNumbersStyle**: Font style of tick numbers.
* **fontTitleStyle**: Font style of a gauge title.
* **fontUnitsStyle**: Font style of the gauge units text.
* **fontValueStyle**: Font style for the value text in a value box.
* **fontNumbersWeight**: Font weight of tick numbers.
* **fontTitleWeight**: Font weight of a gauge title.
* **fontUnitsWeight**: Font weight of the gauge units text.
* **fontValueWeight**: Font weight for the value text in a value box.

## Gauge-Specific Configuration Options

Each type of the gauge in this library also has its own specific configuration options available for customization.

### **Linear Gauge Options**

Linear gauge has some specific options for customization, and some drawing rules which is good to know.

First of all it can be drawn vertically or horizontally, depending on the upset width and height options for the canvas element. If width greater than height the gauge will be treated as horizontal, otherwise - as vertical.

Horizontal gauge drawing has a limitation of drawing value box. In this orientation value box rendering disabled as far as it is not possible to find a good place for it within the current design. So it has to be kept im mind when the horizontal orientation is selected to draw the gauge.

#### **Borders Options**

* **borderRadius**: radius for rounded corners of the gauge plate and its borders.

#### **Progress Bar Options**

* **barBeginCircle**: defines if a gauge bar should start with a circle element imitating flask view of the bar. If set to zero it won’t be drawn at all.
* **barLength**: defines bar length in percents in relation to overall gauge length.

#### **Coloring Options**

* **colorBarEnd**: if given, bar background will be drawn as gradient. If falsy bar color will be solid.
* **colorBarProgressEnd**: if given, progress bar color will be drawn as gradient. If falsy bar color will be solid.

#### **Element Positioning Options**

* **tickSide**: defines a side on which ticks bar should be drawn. Available values are: “left”, “right”, “both”. Default value is “both” - on the both sides of a gauge bar. For horizontally aligned gauges “left” value means top position, “right” value means bottom position.
* **needleSide**: defines a side on which needle at the bar should be drawn. Available values are: “left”, “right”, “both”. Default value is “both” - on the both sides of a gauge bar. For horizontally aligned gauges “left” value means top position, “right” value means bottom position.
* **numberSide**: defines a side on which tick numbers should be drawn. Available values are: “left”, “right”, “both”. Default value is “both” - on the both sides of a gauge bar. For horizontally aligned gauges “left” value means top position, “right” value means bottom position.

#### **Ticks Bar Options**

* **ticksWidth**: defines a length of major ticks width (and width of ticks bar overall) in relative units.
* **ticksWidthMinor**: defines a length of minor tick lines in relative units.
* **ticksPadding**: defines a padding used for drawing ticks out of a bar, in relative units.

### **Radial Gauge Options**

Radial gauge controls has their specific customization options, which are enables to drastically customize its view to make it look, for example, like manometer or compass or any other radial-kind of the tool possible too imagine.

#### **Ticks Bar Options**

* **ticksAngle**: defines a max angle for ticks bar. By default, is 270 degrees. If 360 degrees specified ticks bar fills the whole circle.
* **startAngle**: defines a start angle using which ticks bar starts. By default, is 45 degrees.

|  |  |
| --- | --- |
| **barStartPosition**: ‘left’ | ‘right’. By default, is ‘left’. Enable anti-clockwise progress bars and middle start point progress bars. |

#### **Coloring Options**

* **colorNeedleCircleOuter**: defines a color which should be used to draw outer decorative circle element at the middle of the gauge.
* **colorNeedleCircleOuterEnd**: if defined, outer decorative circle gauge element will be drawn as gradient. If falsy - outer circle will be drawn using solid color.
* **colorNeedleCircleInner**: defines a color which should be used to draw inner decorative circle element at the middle of the gauge.
* **colorNeedleCircleInnerEnd**: if defined, inner decorative circle gauge element will be drawn as gradient. If falsy - inner circle will be drawn using solid color.

#### **Needle Options**

* **needleCircleSize**: defines the size in relative units of the decorative circles element of the gauge.
* **needleCircleInner**: a boolean flag, turns on/off inner decorative circle element drawing.
* **needleCircleOuter**: a boolean flag, turns on/off outer decorative circle element drawing.

#### **Animation Options**

* **animationTarget**: defines which part of the gauge should be animated when changing the value. Could be one of ‘needle’ (default) or ‘plate’ values. When ‘plate’ is selected then gauge will animate ticks bar instead of animating the needle.
* **useMinPath** - boolean. Applicable only to radial gauges which have full 360-degree ticks plate. If set to true for this kind of gauges will rotate needle/plate by a minimal rotation path.

# **Gauges Advanced Usage**

Tips and tricks for advanced usage of canvas gauges

#### **Table of Contents**

* [**Async Loading**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#async-loading)
* [**Advanced Animations**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#advanced-animations)
  + [**Improving Performance On Old Browsers**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#improving-performance-on-old-browsers)
  + [**Using custom Animation Rules**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#using-custom-animation-rules)
* [**Integration With Custom Fonts**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#integration-with-custom-fonts)
* [**DOM Mutations Support In Old Browsers**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#dom-mutations-support-in-old-browsers)
* [**Manual DOM Mutations Control**](https://canvas-gauges.com/documentation/user-guide/advanced-usage#manual-dom-mutations-control)

#### **User Guide Chapters**

* [**Introduction**](https://canvas-gauges.com/documentation/user-guide/)
* [**Configuration**](https://canvas-gauges.com/documentation/user-guide/configuration)
* [**Creating Custom Builds**](https://canvas-gauges.com/documentation/user-guide/custom-builds)
* [**Using As Components**](https://canvas-gauges.com/documentation/user-guide/using-as-component)
* [**Scripting API**](https://canvas-gauges.com/documentation/user-guide/scripting-api)
* [**Migration Guide**](https://canvas-gauges.com/migration/)

## Async Loading

Canvas gauges support async loading of their code. If you are using gauges as HTML-components on your page async load of the script is usually safe. So it is enough just to load gauges gode as following:

<script async src="gauge.min.js"></script>

From other hand if you are using scripting on a page for gauges initialization async loading may break your code execution. In this case it is better to wrap gauges initialization with the function which then should be used as handler for gauge script loading, for example:

**<!doctype html>**

<html>

<head>

<title>Canvas Gauges Async Scripting</title>

</head>

<body>

*<!-- ... some code before ... -->*

<canvas id="scripted-gauge"></canvas>

<script>

**function** initGauge() {

**var** gauge **=** **new** RadialGauge({

renderTo: 'scripted-gauge',

width: 300,

height: 300

}).draw();

}

</script>

<script async src="gauge.min.js" onload="initGauge()"></script>

*<!-- ... some code after ... -->*

</body>

</html>

## Advanced Animations

### **Improving Performance On Old Browsers**

Currently canvas gauge supports animations using requuestAnimationFrame calls. For older browsers without support it fallback to use timers, which are not that efficient, so on old platforms you may see performance degrade.

One of a tricks here could be disabling animations if browser does not support requestAnimationFrame if you face a performance problem. This could be done like:

**var** gauge **=** **new** RadialGauge({

*// ... config options ..*

animation: **!!**window.requestAnimationFrame

});

Of course, feature detection may be done more clever way, using vendor prefixed feature detection, as far as gauges detects them also.

### **Using custom Animation Rules**

Currently canvas gauges provides a various pre-defined animation rules, like:

* linear
* quad
* quint
* cycle
* bounce
* elastic

and their opposites:

* dequad
* dequint
* decycle
* debounce
* delastic

Therefore, if it is not enough it is provide a way to create your own rules, which will be used during animations.

Defining a rule must follow the interface:

**public** AnimationRule: **function**(percent: number): number

So it is simply a function which takes a percent of animation completion as an argument and transforms it by some mathematical rule.

For example, implementation of linear rule looks like this:

**var** linearRule **=** **function**(percent) {

**return** percent;

};

**var** gauge **=** **new** RadialGauge({

*// ... some options ...*

animationRule: linearRule

});

Or a bit complicated elastic rule:

**var** gauge **=** **new** LinearGauge({

*// ... some options ...*

animationRule: **function**(percent) {

**return** 1 **-** (**function**(p) {

**var** x **=** 1.5;

**return** Math.pow(2, 10 **\*** (p **-** 1)) **\***

Math.cos(20 **\*** Math.PI **\*** x **/** 3 **\*** p);

})(1 **-** percent);

}

});

So there is no limits except the fantasy of developer to make any type of animation.

## Integration With Custom Fonts

Canvas gauge provides a basic interface to customize fonts of the text element used during the gauge rendering process.

It is done with generic configuration options:

* fontValue: string font-family
* fontNumbers: string font-family
* fontUnits: string font-family

By the way there could be some issues to solve if you are going to use custom loaded fonts on a web page.

As far as gauges are rendered as-fast-as-possible it means that the font can be loaded on the page **after** the gauge has been rendered. And it requires to re-draw the gauge after the font loading. As far as canvas gauge library follow the strategy of providing a minimalist code it won’t try to detect font loading, as far as majority of the users may even not need this feature. Those who require this feature have to take care about font loading themselves.

But font loading detection could be not that simple task, especially for old browsers. Modern browsers provide experimental document.fonts interface which is referring to [**CSS Font Loading API**](https://developer.mozilla.org/en-US/docs/Web/API/CSS_Font_Loading_API)

If you are targeting to a newest browser only it’s not that hard to do. All you need is to wait until font is loaded and redraw the gauge. To hide font-change effect on the gauge it will be enough to make canvas element hidden by default with CSS, like:

<link href="https://fonts.googleapis.com/css?family=Orbitron"

rel="stylesheet">

<script src="gauge.min.js"></script>

<canvas

data-type="radial-gauge"

data-font-value="Orbitron"

data-width="300"

data-height="300"

style="visibility:hidden"

></canvas>

<script>

document.fonts.forEach(font **=>** {

font.loaded.then(() **=>** {

*// using match, because in FF it contains quote marks*

**if** (font.family.match(/Orbitron/)) {

**let** gauge **=** document.gauges[0];

gauge.update();

gauge.options.renderTo.style.visibility **=** 'visible';

}

});

});

</script>

If there is a need to support older browsers it may require to write your own solution or to use some 3d-party solution like [**WebFontLoader**](https://developers.google.com/fonts/docs/webfont_loader) from Google.

## DOM Mutations Support In Old Browsers

In old browsers canvas gauge may not work properly as a web-component. Due to a strategy of minimalist code we were not include any polyfill for [**MutationObserver**](https://developer.mozilla.org/docs/Web/API/MutationObserver). So if you need to support this feature for some old platforms you have to load some polyfill for MutationObserver **before** loading canvas gauge library code.

For example you can use [**this one**](https://github.com/webcomponents/webcomponentsjs) or it’s [**ancestor**](https://github.com/Polymer/MutationObservers) (because of minimalism, despite the fact it’s deprecated).

## Manual DOM Mutations Control

Sometimes it may be required to disable automatic DOM document parse for some reason (for example, in case of performance optimizations, etc.). This could be easily achieved by defining a global constant GAUGES\_NO\_AUTO\_INIT and set it to some truthy value before loading main gauges JavaScript code. For example, such gauge code base loading:

<script>window.GAUGES\_NO\_AUTO\_INIT **=** **true**;</script>

<script src="../gauge.min.js"></script>

will prevent automatic DOM parsing of the page to initialize gauges. This may improve page load time. Then, when it is required each gauge could be re-initialized by using the following BaseGauge interface.

BaseGauge.fromElement(canvasGaugeElement);

Рамка

https://jquery-docs.ru/has-selector/examples/