The power of web-based widgets mixed in a HTML cocktail

Goincompany.com

DRINKS Toolkit

No matter what platform or framework you use, DRINKS Toolkit has changed the way to build control panels or dashboards, moving this technology on the web.

DRINKS Toolkit is a library to create widgets on a webpage and interface its with real world.
This book teachs all you need to know about DRINKS, from “Installation” to “How to realize a full working dashboard”.
Pratical examples show you how to use DRINKS advanced features in your pages.

Good reading and enjoy discovering DRINKS Toolkit.

DRINKS Toolkit

Goincompany.com

To Labview.

Table of Contents

Preface

DRINKS Toolkit is a collection of web-based widget powered with HTML5 Canvas API and Javascript, designed to make easily and fast development on webpages.
In this first release we have developed a toolkit based to realize web instruments, but in the future DRINKS, will be more than a toolkit. We want to create many others instruments, realize hardware to control everything, provide support, and useful features to make your work faster and easier and create web widgets that can improve your web development.
Using just one tag you can manipulate a more complex javascript interface taking advantage of the power of HTML, so you can create your instrument using Drinks API, bind it to a tag and include it to your web pages easily.
Drinks is distributed under new BSD license and you can download the Open Pack for free or donating something that we'll use to improve Drinks. In other way, you can help us only downloading the package and giving us your positive feedback, this will be a personal satisfaction and it will pay so much work made to realize this project.
If you need support or other services provided by Goincompany.com, you can buy Business Pack, visit our site for knowing more.

- **AJAX Powered**: all widgets uses AJAX technology to allow communications from Client to Server, retrieve all data in real-time.
- **HTML5 Powered**: HTML5 is cornerstone of the W3C's open web platform; a framework designed to support innovation and foster the full potential the web has to offer.
- **Absolutely cross-platform:** Every device which has a browser that supporting canvas can use Drinks.
- Enjoy Barman Ide, the integrated development environment that allow to create your Drinks applications in a click.
- Is an open source project, therefore you can redesign, create or compose your instruments like you want.
- No Flash and no plugins are needed, only a web browser Canvas and Javascript compatible.
- Is fully compatible with Chrome, Firefox, Opera and Safari.
- Is W3C compatible.
- Add every widgets on the page using an HTML TAG.

Installation

Installing Drinks is simply. Just download the toolkit from www.goincompany.com/drinks.php , extract and upload the folder in the server where you want to place your application.

In your html page you need to link the javascript files just inserting the script tag:

*<script type=\"text/Javascript\" src=\"**Path to Drinks.js**\"></script>*.

Your application is ready to work.

First Design

Create your first dashboard!

Creating a dashboard is very simple with Drinks. You just know a little bit of HTML and your application is ready to work.

For example now, I want to create a simple Knob that controls the value of a Gauge.

First, we have to declare our Knob:

<knob id="k1"></knob>

Note that, **the presence of the *id* is mandatory: every widget must have its own id**.

Also the tags must be closed.

Second, We have to declare our Gauge:

<display id=”gau”></display>

Then we have to connecting the instruments together so, the value of the Knob, will propagate to the Gauge.

We add the onchange event to the Knob.

<knob id=”k1” onchange=”gau.value = this.value;”></knob>

Example 1

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
    </head>
    <body>
        <knob id="k1" onchange="gau.value=this.value"></knob>
        <display id="gau"></display>
    </body>
</html>
```

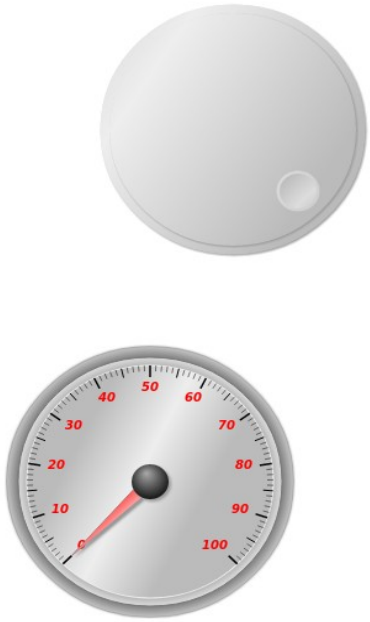

Figure 1: The result of Example 1.

Improve your dashboard.

Combine HTML standards.

The Dashboard you have created is empty. For example, you could want that the instruments are inside a container and position them where you like more.

We create a div with some style properties...

<div ***style="width:600px; height:300px; background-color:gray; border-radius:15px; box-shadow:1px 1px 3px #333;"****>*

.....Instruments.....

* *

... and then we append the instruments to it.

Example 2

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
    </head>
    <body>
        <div style="width:600px; height:300px; background-color:gray;
border-radius:15px; box-shadow:1px 1px 3px #333;">
            <knob id="k1" onchange="gau.value=this.value"
radius="80" style="float:left;"></knob>
            <display id="gau" radius="80" style="float:left; margin-
top:55px;"></display>
        </div>
    </body>
</html>
```

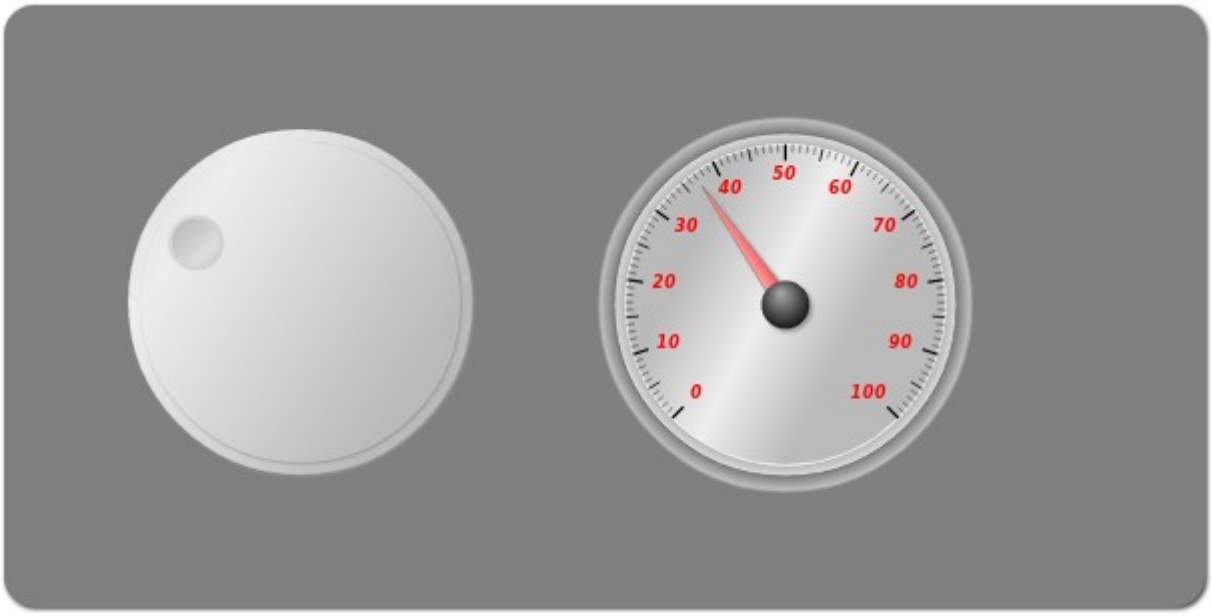

Figure 2: The result of Example 2.

We can resize them using the radius attribute.

Another example:

<knob id="k1" onchange="gau.value=this.value" radius="80" ***style="position: absolute; top:89px; left:57px; "****></knob>*

or instead of inline style CSS:

```
<style>
	#k1{
		position:absolute;
		top: 89px;
		left: 57px;
	}

</style>
<knob id="k1" onchange="gau.value=this.value"></knob>
```

or if you have so much knobs:

```
<style>
	knob.kclass{
		position:absolute;
		top: 89px;
		left: 57px;
	}

</style>
<knob id="k1" onchange="gau.value=this.value" class="kclass"></knob>
```

Add a description.

Every widget has a label attribute used to add a little description at the bottom of the instrument.

<knob id="k1" ***label="Turn this knob"*** *onchange="gau.value=this.value" radius="80" style="float:left;" ></knob>*

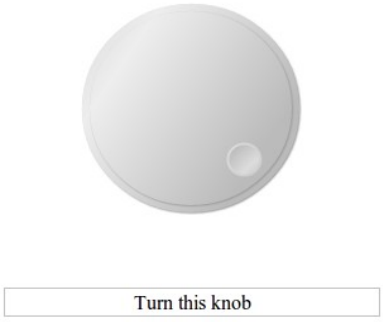

Figure 3: A Knob with the label attribute setted to "Turn this knob".

Set the size of the instruments.

Using the width and height attributes you can resize your instruments. The size is expressed in pixel. The circular widgets have, instead of these fields, the radius attribute and, the width and height, are calculated from that.

<switch type="side" id="sw" ***width="100" height="50"****></switch>*

<display id="gau" ***radius="80"*** *></display>*

Set a default value.

To explain this, is necessary to understand that Instruments are divided into different categories. Some of these are Analog, Digital and Binary.

Binary widgets can take only two values, 0 or 1. If the value is 1 the instrument is on, else is off.

For example:

<switch id="sw" ***value=”1”****></switch>*

This button is on.

<led id="led1" ***value=”1”****></led>*

<led id="led2" ***value=”0”****></led>*

The first LED is on while the second is off.

Analog instruments can take a value limited to *max_range* and *min_range* and can be float.

<display id="gau" ***value=”37.5”****></display>*

Digital instruments can take limited number of values.

<slider id=”sl” type=”digital” ***value=”bad”****>*

<option value=”good”>good</option>

<option value=”bad”>bad</option>

<option value=”ugly”>ugly</option>

</slider>

Example 3

A button turns on a LED.

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
    </head>
    <body>
        <switch id="sw" onchange="if(this.value==1)
{led.on();}else{led.off();}"></switch>
        <led id="led"></led>
    </body>
</html>
```

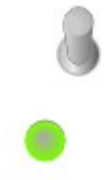

Figure 4: The result of Example 4.

The *on* and *off* functions assign the value of the LED at 1 or 0. There is another way to do this, using the *toggle* function that changes the status of the binary instrument.

```
<switch id="sw" onchange="led.toggle();"></knob>
<led id="led"></led>
```

The type attribute.

This attribute identifies a class of instruments that belongs to the specific tag.

Knob	loop, analog, digital
Display	gauge, digital, level, thermo, analog, graph
Led	round, rect, triangle
Switch	toggle, arc, side, circle, rect, rocker
Slider	analog, digital

The type attributes of every instruments in the 1.0 Drinks release.

The first types of each tag are the default types and is possible to create them without specifying it.

Changing the range of the instruments.

The range of values that the instrument can take is assigned by *min_range* and *max_range* attributes that respectively represent the min and the max value that it can take.

Example 4

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
    </head>
    <body>
```

```
                <knob id="k1" onchange="gau.value=this.value" min_range="-
100" max_range="100"></knob>
                <display id="gau" min_range="-100"
max_range="100"></display>
            </body>
        </html>
```

Figure 5: The result of Example 4. Knob and Gauge have the min_range and max_range attribute setted.

Drinks and Javascript.

How to access the instrument object.

Drinks provides two way to access the objects. Every instrument is stored in a variable named as its id.

For example you can access the value of the Gauge by:

*<knob id="k1" onchange="**gau.value=this.value;**" ></knob>*

With the *this* keyword you access the instrument where the onchange event occurred.
Or you can use:

*<knob id="k1" onchange="**Drinks.getElementById('gau').value=this.value;**"*
></knob>

Example 5

```
<html>
        <head>
            <script type="text/javascript" src="Drinks.js"></script>
        </head>
        <body>
            <knob id="k1" onchange="gau.value=this.value; "
min_range="-100" max_range="100"></knob>
            <display id="gau" min_range="-100"
max_range="100"></display>
            <input type="button" onclick="gau.max_range=200;
gau.min_range=-200; gau.radius=80; k1.radius=80;" value="Click">
    </body>
    </html>
```

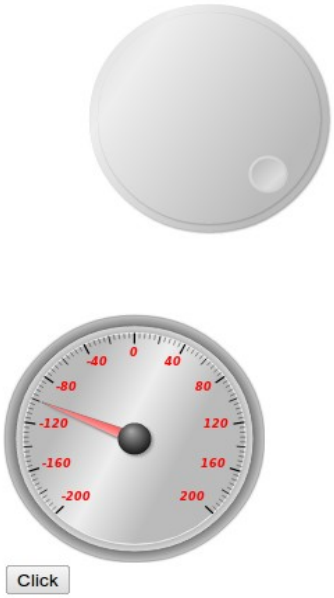

Figure 6: The result of Example 5. When the button is pressed the Gauge changes its range and its radius.

If you want to access all the instruments with a certain name or class you have to use the *getElementsByName* and *getElementsByClassName* function respectively.

Example 6

```
<html>
        <head>
            <script type="text/javascript" src="Drinks.js"></script>
        </head>
        <body>
            <knob id="k1" name="hello" onchange="gau.value=this.value;
" min_range="-100" max_range="100"></knob>
            <display id="gau" name="hello" min_range="-100"
max_range="100"></display>
```

```
                <display id="gau2" class="bye" min_range="-100"
max_range="100"></display>
                <input type="button"
onclick="alert(Drinks.getElementsByName('hello').length)" value="Click">
                <input type="button"
onclick="alert(Drinks.getElementsByClassName('bye').length)" value="Click">
            </body>
</html>
```

The Drinks class stores the instruments grouped by tag, in an array for each one, so you can access all the Leds or all the Displays by these arrays.

Example 7

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
        <script>
            function turnsOn(){
                for(var i in Drinks.leds){
                    var element = Drinks.leds[i];
                    if(element.type=="round")
                        element.on();
                }
            }
        </script>
    </head>
    <body>
        <led id="led1" style="float:left;"></led>
        <led id="led2" style="float:left;"></led>
        <led id="led3" type="rect" style="float:left;"></led>
```

```
        <led id="led4" style="float:left;"></led>
        <led id="led5" style="float:left;"></led>
        <input type="button" onclick="turnsOn();" value="click"/>
    </body>
</html>
```

Figure 7: The result of Example 7. Only the round leds are turned on.

Create your instrument using JS

The Drinks API provides a way to create your instruments using JS.

You can do this by the *createElement* function:

```
Drinks.createElement('display');
```

The attributes can be setted in two different ways:

1. Passing a second attribute to *createElement*: an object in JSON form that has the attributes and the respective value.

```
var gauge = Drinks.createElement('display', {"id":"gau1", "min_range":"-50", "max_range":"50","style":"float:left; margin-top:50px;"});
```

2. Using the *setAttribute* method.

```
var gauge = Drinks.createElement('display');
gauge.setAttribute("id":"gau1");
```

gauge.setAttribute("min_range":"-50");

gauge.setAttribute("max_range":"50");

gauge.setAttribute("style":"float:left; margin-top:50px;");

Append your instrument in a container

After you have created the HTML element, you have to append it to an HTML container. In order to do this you have to use the *appendChild* function, provided by the Drinks class.

Drinks.appendChild('body', gauge);

'*body*' is the id of the HTML container. If the parent is an instrument, this function doesn't work. We have to use the *appendChild* method of the instrument, but we'll see this later.

Example 8

```
<html>
        <head>
            <script type="text/javascript" src="Drinks.js"></script>
            <script>
                function create(){
                    var gauge = Drinks.createElement("display",
{"id":"gau1", "min_range":"-50", "max_range":"50","style":"float:left;
margin-top:50px;"});
                    Drinks.appendChild('body', gauge);
                }
            </script>
        </head>
        <body id="body">
```

```
                <knob id="k1" style="float:left;"
onchange="if(Drinks.getElementById('gau1')){gau1.value=this.value;}"></knob>

                <input type="button" onclick="create();" value="click"/>
            </body>
        </html>
```

HTML Element vs Instrument

There is a difference between the HTML Element and the JS Drinks instrument.

For example, the knob tag generates an HTML element and the respective JS object.

In order to edit instrument's properties, you have to manipulate the JS object. On page load, from the HTML elements of the tags, Drinks creates the instruments and stores them in an array. The respective HTML element is stored into the JS object and to access the HTML properties you have to reference to it.

So you can't use the *document.getElementById* function to retrieve the instruments, because that returns the HTML element. The same thing for the other functions of the document. The Drinks widgets aren't belong to DOM.

Example 9

```
<html>
            <head>
                <script type="text/javascript" src="Drinks.js"></script>
            </head>
            <body id="body">
                <knob id="k1" style="float:left;"></knob>
                <input type="button" onclick="k1.html.style.display='none';"
value="click"/>
            </body>
        </html>
```

For example, is not possible to do something like this:

```
var sw = Drinks.createElement('switch');
sw.type = …..
sw.min_range=....
```

Because sw is an HTML element and it hasn't the instrument's fields.
Instead, this is possible:

```
var sw = Drinks.createElement('switch');
sw.setAttribute(“type”, “ToggleSwitch”);
sw.setAttribute(“id”, “sw1”);
sw.setAttribute(“width”, “100”);
Drinks.appendChild('body', sw);
```

after the appendChild you can use the id to access your instrument.

```
sw1.value=1;
sw1.onchange='...';
```

Some important attributes

How to propagate an input to many outputs

Sometimes may be useful to link the value of an input instrument to many output instruments. This is possible by the *display* attribute of the Input class.

The argument of this attribute is a name that all the displays must have setted to receive the value, in another attribute called *display_name*.

So we have a knob:

```
<knob id="k1" display="out"></knob>
```

And three display:

```
<display id="dis1" display_name="out">
<display id="dis2" type="level" display_name="out">
<display id="dis3" type="analog" display_name="out">
```

The value of the knob will propagate to the displays.

Example 10

```
<html>
        <head>
            <script type="text/javascript" src="Drinks.js"></script>
        </head>
        <body >
            <knob id="k1" display="out" style="float:left;" type="analog"
radius="70"></knob>
            <display id="dis1" display_name="out"
style="float:left;"></display>
            <display id="dis2" type="level" display_name="out"
style="float:left;"></display>
            <display id="dis3" type="analog" display_name="out"
style="float:left;"></display>
        </body>
</html>
```

Using the autoscale property

The autoscale property is common to analog and digital instruments and allows to resize the range of values automatically if exceed the *min_range* or *max_range* value.

Example 11

```
<html>
        <head>
            <script type="text/javascript" src="Drinks.js"></script>
        </head>
        <body >
            <knob id="k1" display="out" style="float:left;"
type="analog"></knob>
            <display id="dis1" display_name="out" style="float:left;"
max_range="50" autoscale="true"></display>
            <display id="dis2" type="level" display_name="out"
style="float:left;"></display>
            <display id="dis3" type="analog" display_name="out"
style="float:left;"></display>
        </body>
    </html>
```

OnAlert and OnLeaveAlert Events

If you want to control if your instrument reachs a certain range of values, *OnAlert* event is what you are looking for. By the usage of range_from and range_to attributes you can define start and end of range. If the value comes in this range, the onalert event triggers. Once the onalert event is triggered, if the value goes out the range, onleavealert is activated.

Example 12

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
        <script>
        function onAlert(){
            document.getElementById('info').innerHTML='ARGH, very
HOT!';
            document.getElementById('info').style.backgroundColor='red';
            dis1.color='red';
            l1.on();
        }
        function onLeaveAlert(){
            document.getElementById('info').innerHTML='Ok, it\'s
normal';
        document.getElementById('info').style.backgroundColor='green';
        dis1.color='blue';
            l1.off();
        }
        </script>
        </head>
        <body >
            <knob id="k1" radius="50" style="float:left;" type="analog"
onchange="dis1.value=this.value;" max_range="50"></knob>
            <display id="dis1" type="thermo" style="float:left;"
max_range="50" range_from="40" range_to="50" onalert="onAlert();"
onleavealert="onLeaveAlert();"></display>
            <led id="l1" color="red" style="float:left;"></led>
```

```
                <div style="border:1px solid black; width:150px; height:20px;
float:left; background-color:green;" id="info">Ok, it's normal</div>
            </body>
        </html>
```

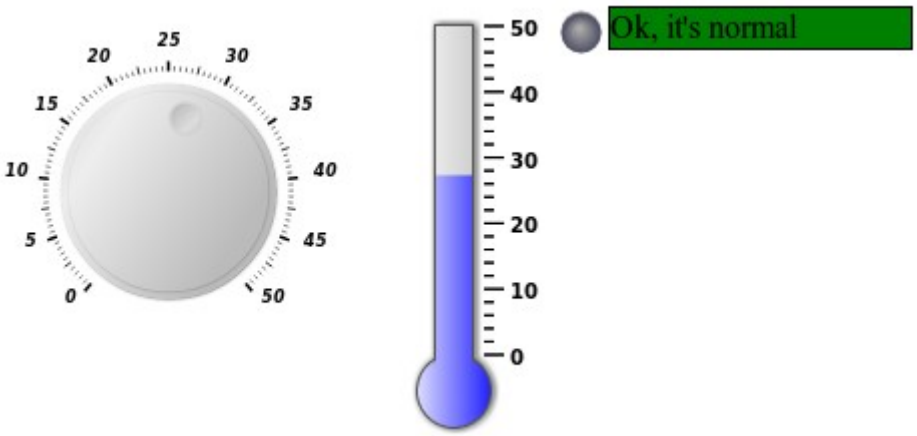

Figure 8.a: The result of Example 12 when the value of the ThermoMeter is out of alert range.

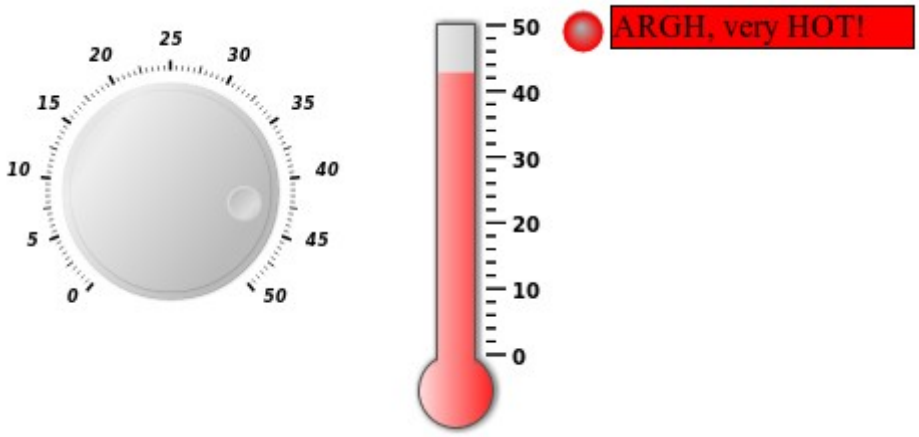

Figure 8.b: The result of Example 12 when the value of the ThermoMeter is in alert range.

The option tag

There are two digital input in this first release: DigitalSlider and DigitalKnob.

To append values to them we use the option tag.

<option value="1">Good</option>

For each option we have a value and a description that will appear in the instrument.

<option value="20">20</option>

Digital Slider and Digital Knob have an object called *options* where every item has two attribute, value and inner, that represent the value and the innerHTML of the option tag.

With this array we can append or remove options by JS functions *push* and *splice*.

Example 13

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
    </head>
    <body>
        <knob type="digital" id="k1" radius="80"
onchange="document.getElementById('test').innerHTML=this.value;">
            <option value="1">good</option>
            <option value="2">bad</option>
            <option value="3">ugly</option>
        </knob>
        <input type="button" onclick="k1.options.push({'value':4,
'inner':'yes'});" value="click">
        <input type="button" onclick="k1.options.splice(0, 1);" value="click">
        <div id="test"></div>
    </body>
```

</html>

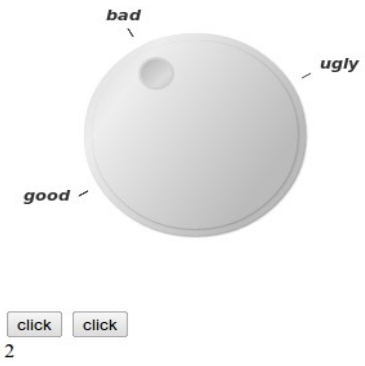

Figure 9.a: The result of Example 13 on page load.

Figure 9.b: The result of Example 13 when the first button was pressed and the "yes" option was appended.

Figure 9.c: The result of Example 13 when the second button was pressed and the "good" option was removed.

Setting the value of a digital instrument

There are two ways for setting value:

set the value attribute or add the selected attribute in the option tag.

1.

```
<knob type="digital" id="k1" radius="80"  value="2">
                    <option value="1">good</option>
                    <option value="2">bad</option>
                    <option value="3">ugly</option>
</knob>
```

2.

```
<knob type="digital" id="k1" radius="80">
                <option value="1">good</option>
                <option value="2"  selected>bad</option>
                <option value="3">ugly</option>
</knob>
```

Using JS we can set the value:

```
k1.value='2';
```

or

```
k1.selectedIndex='1';
```

Example 14

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
```

```
    </head>
    <body>
        <knob type="digital" id="k1" radius="80"
onchange="document.getElementById('test').innerHTML=this.value;">
            <option value="1">good</option>
            <option value="2" selected>bad</option>
            <option value="3">ugly</option>
        </knob>
        <input type="button" onclick="k1.value='3';" value="click">
        <input type="button" onclick="k1.selectedIndex=0;" value="click">
        <div id="test"></div>
    </body>
</html>
```

The align attribute

LevelMeter, ThermoMeter and Slider have this attribute. Its value is a string that can be *vertical* or *horizontal* and represents the alignment of the instrument.

For example we have:

```
<display type="level" id="l1" align="vertical"></display>
<display type="level" id="l2" align="horizontal"></display>
```

The default value is *vertical*.

And the result is:

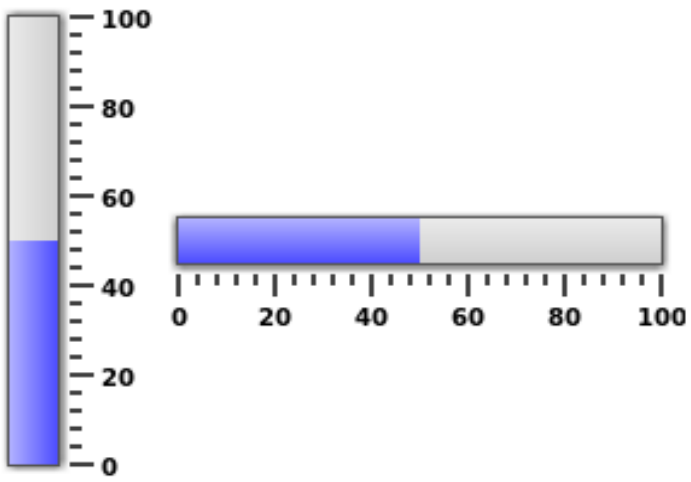

Figure 10: Two LevelMeter with different align values.

Example 15

```
<html>
<head>
    <script type="text/javascript"src="Drinks.js"></script>

</head>
<body>
    <display type="level" id="l1" align="vertical" value="50"
style="float:left;"></display>
    <display type="level" id="l2" align="horizontal" value="50"
style="float:left; margin-top:60px;"></display>
    <input type="button" onclick="l1.align='horizontal';" value="click">
    <input type="button" onclick="l2.align='vertical';" value="click">

</body>
```

</html>

We can access the align property by JS.

The topdown attribute

LevelMeter, ThermoMeter and Slider have this attribute. Its value is a boolean that can be *true* or *false* and represents the direction of the value: from bottom to top (false) or from top to bottom (true).

```
<display type="level" id="l1" topdown="false"></display>
<display type="level" id="l2" topdown="true"></display>
```

The default value is *false*.

Ant the result is:

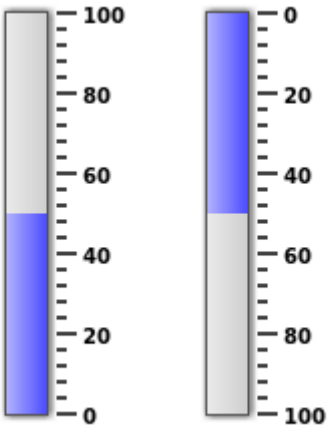

Figure 11: Two LevelMeter with different topdown values.

Example 16

```
<html>
<head>
    <script type="text/javascript"src="Drinks.js"></script>

</head>
<body>
    <display type="level" id="l1" topdown="false" style="float:left;"
value="50"></display>
    <display type="level" id="l2" topdown="true" style="float:left;"
value="50"></display>
    <input type="button" onclick="l1.topdown=true;" value="click">
    <input type="button" onclick="l2.topdown=false;" value="click">

</body>
</html>
```

We can access the topdown property by JS.

Divisions and precision

Analog instruments has two important attributes: divisions and precision.
For example we want to change the divisions of a Gauge:

```
<display id="gauge" divisions="20"></display>
```

or of an AnalogKnob

<knob type="analog" id="knob" ***divisions="40"****></knob>*

Obviously, sometimes the size and the divisions are conflicting because, if the instrument is too small you can't use much divisions.

Precision is the number of cipher which the instrument use to aproximate the value.

<display id="gauge" min_range="0.001" max_range="1" ***precision="3"****></display>*

or of an AnalogKnob

<knob id="knob" type="analog" min_range="0.001" max_range="1" ***precision="3"****></display>*

The OnLoad Event

OnLoad trigger when the render method of the instrument is executed.
For example a LED that toggle every second:

<led id="l1" color="white" value="1" x="350" y="170" ***onload="setInterval(function(){this.toggle();}.bind(this), 1000);"****></led>*

and a display that change its value every second:

<display id="d1" type="digital" ***onload="setInterval(function() {this.value=Math.random()*100;}.bind(this), 1000);"*** *significative="1" cipher="3" width="400" height="200" bgcolor="black" color="red">*

Interface Drinks with real world

Before now we haven't talked about server. If you want to interface Drinks with the real world, you have to send requests to server; the sensors will be linked to it and they will pass their values to a deamon which is in execution.

This process will pass data to a database or a file that the server can read.

Once the data are read the work is done because, you can communicate the values to the client using html standard methods or AJAX.

In this section we'll use the PHP as server side script but you can use what you want.

HTML standard form

The first method to communicate with server is using forms.

For example this is our form:

```
<form action="server.php" method="post">
    ...Instruments....
</form>
```

And now we append to it the instruments which the data must be passed. It's necessary to specify a name to each tag.

```
<form action="server.php" method="post">
    <display id="dis1" type="analog" value="0.8" name="dis1"
style="float:left;" max_range="2" text="A"></display>
    <display id="dis2" type="analog" value="30" name="dis2"
style="float:left;" max_range="50" text="V"></display>
    <input type="submit">
</form>
```

And this is the server.php file:

```
<?
	echo $_POST["dis1"]." : ".$_POST["dis2"]
?>
```

Example 17

The knobs change the value of the two displays.

```
<html>
<head>
	<script type="text/javascript"src="Drinks.js"></script>
</head>
<body >
	<form action="server.php" method="post">
		<display id="dis1" type="analog" name="dis1"
style="float:left;" max_range="2" text="A"></display>
		<display id="dis2" type="analog" name="dis2"
style="float:left;" max_range="50" text="V"></display>
		<input type="submit">
	</form>
	<div style="width:100%; float:left;">
		<knob type="analog" radius="80" style="float:left;" id="ka"
onchange="dis1.value = this.value;" max_range="2"></knob>
		<knob type="analog" radius="80" style="float:left;" id="kv"
onchange="dis2.value = this.value;" max_range="50"></knob>
	</div>
</body>
</html>
```

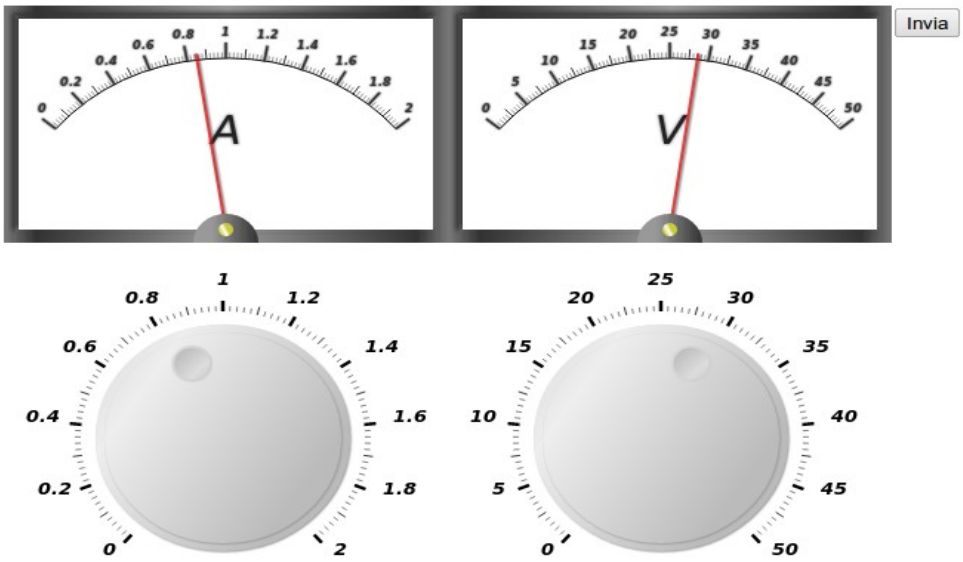

Figure 12: The result of Example 17.

Build dinamic pages

Now imagine that we have a database. After a query, we want to create as much "displays" as the records in a table and for each, set the values stored in the respective rows.

```
<?
        $result = query(...);    // Execute query
        $i=0;    //initialize counter
        while($row = get_row()){  //while there are some rows

                echo " <display id=\"gau$i\" value=\"".$row["value"]."\"> ";
                //create a display
                $i++;
        }
?>
```

The AJAX cocktail

Drinks provides an AJAX API. Using the *href* attribute you can refer to a server side script and pass or receive data from it. An other attribute that you can use to support this functionality is *refresh*. It specifies the seconds that pass between a request and another. First we have to distinguish between input and output instruments: input pass data to server while output receive data from it.

For example, an output instrument:

```
<display id=" dis1" href=" server.php"  refresh="5" ></display>
```

And in the php file:

```
<?
    $value = get_from_db();
    echo $value;
?>
```

The gauge will receives the value every 5 seconds.

Another example with an input instrument:

```
<knob id="k1" href="server.php" refresh="5"></knob>
```

And in the php file:

```
<?
    $value = $_GET["k1"];
    set_to_db($value);
?>
```

The knob will sends the value every 5 seconds.

By default the refresh attribute is set to 5 seconds. If you set 0, Drinks will send a request every 100 ms.

Two or more instruments can't refer the same href file. In order to this we have to use the Manager class.

Example 18

client.htm

```
<html>
<head>
    <script type="text/javascript"src="Drinks.js"></script>
</head>
<body >
    <display id="dis1" href="server.php" type="analog" refresh="5"
style="float:left;" max_range="2" text="A"></display>
</body>
</html>
```

server.php

```
<?
    echo rand(0, 200)/100;
?>
```

The Manager Class

When you have to send or receive more than one value, you can use the Manager class. Since there is a difference between input and output instruments, you can't use the same Manager for all kind of them. You can create all Manager that you want, but

at least one for input and one for output.
For example we have 3 inputs:

```
<knob type="analog" id="an1"></knob>
<knob type="analog" id="an2"></knob>
<knob type="analog" id="an3"></knob>
```

and 1 output:

```
<display type="analog" id="an4"></display>
```

The first manager is for input:

```
var man = Drinks.createManager();
man.href='server.php';
man.input=new Array("an1", "an2", "an3");
man.start();
```

and second is for output:

```
var man2 = Drinks.createManager();
man2.href='server.php';
man2.start();
```

But let me explain. The Manager is created by the *createManager* method of Drinks class, then you have to set the *href* attribute and *refresh* if is different from default value. After settings, the *start* method can be called and the requests will start.

The *input* attribute is an array that contains the ids of the input instruments that we want to control. If *input* is empty the Manager controls output instruments else input.

Example 19

client.htm

```
<html>
	<head>
		<script type="text/javascript"src="Drinks.js"></script>
		<script>
			var man = Drinks.createManager();
			man.href='server.php';
			man.input=new Array("an1", "an2", "an3");
			man.start();
			var man2 = Drinks.createManager();
			man2.href='server.php';
			man2.start();
		</script>
	</head>
	<body id="body">
		<knob type="analog" id="an1"></knob>
		<knob type="analog" id="an2"></knob>
		<knob type="analog" id="an3"></knob>
		<display type="analog" id="dis1"></display>
		<display type="analog" id="dis2"></display>
		<display type="analog" id="dis3"></display>
	</body>
</html>
```

server.htm

```
<?
	if(count($_GET)>0){   //input section
		send_to_db($_GET["an1"]);
```

```
        send_to_db($_GET["an2"]);
        send_to_db($_GET["an3"]);
    }
    else{  //output section
        $values["dis1"] = get_from_db("dis1");
        $values["dis2"] = get_from_db("dis2");
        $values["dis3"] = get_from_db("dis3");
        echo json_encode($values);
    }
?>
```

get_from_db and *set_to_db* are imaginary functions to explain the possible actions.

In the output section, in order to send the values to the manager, we have to build an hashmap where every key is the id of the instrument, and send it to the client by json encode of the array.

The AJAX Class

If you have experiences with AJAX, you can use the AJAX class to send post and get requests. We have two important functions *load* and *send.* The first executes get requests and has two argument: the server side script URL and the id of the HTML element you want to set its *innerHTML* attribute.

First, we have to declare our class:

```
var ajax = new Ajax();
```

then call the *load* method.

```
ajax.load("server.php", "test_div");
```

and so:

```
<div id="test_div"></div>
```

The response of server.php will be put into *test_div*. If id is *null*, the response will not be put in the div.

The second function is send. It executes post requests taking the values from a form. For example:

```
<form>
    <switch id="sw" name="sw"></switch>
    <display id="lev" type="level" name="lev"></display>
    <knob id="kn" name="kn"></switch>
    <input type="button" onclick="ajax.send(this.form, 'server.php', null);"
</form>
```

The first parameter is the form object, second is server script url and third is the id of the HTML element.

A third important method is *addCall.* This function append an additional piece of javascript code to execute after the response is occurred and to evaluate the AJAX response text.

Example 20

client.htm

```
<html>
<head>
    <script type="text/javascript"src="Drinks.js"></script>
    <script>
        var ajax = new Ajax();
```

```
                function onload(){
                    ajax.send(document.forms[0], 'server.php', "test_div");
                }
            </script>
        </head>
        <body onload="setInterval(onload, 2000);">
            <form style="width:100%; height:300px;">
                <switch id="sw" name="sw" style="float:left;"></switch>
                <slider id="sl" name="sl" style="float:left;"
width="300"></slider>
                <knob id="kn" name="kn" style="float:left;"
radius="50"></knob>
            </form>
            <led id="led1" style="float:left;"></led>
            <switch type="side" id="sw2" style="float:left;"
onchange="if(this.value==1){ajax.addCall(' if(ajaxobj.responseText==\'1\')
{led1.on();}'); ajax.load('server.php?get', null);}"></switch>
            <div id="test_div"></div>
        </body>
</html>
```

server.php

```
<?
    session_start();
    if(isset($_GET["get"])){
        if($_SESSION["sw"]=="1"){
            echo "1";
        }
    }
    else{
```

```
        $_SESSION["sw"] = $_POST["sw"];
        $_SESSION["sl"] = $_POST["sl"];
        $_SESSION["kn"] = $_POST["kn"];
        echo $_SESSION["sw"].":".$_SESSION["sl"].":".$_SESSION["kn"];
    }
?>
```

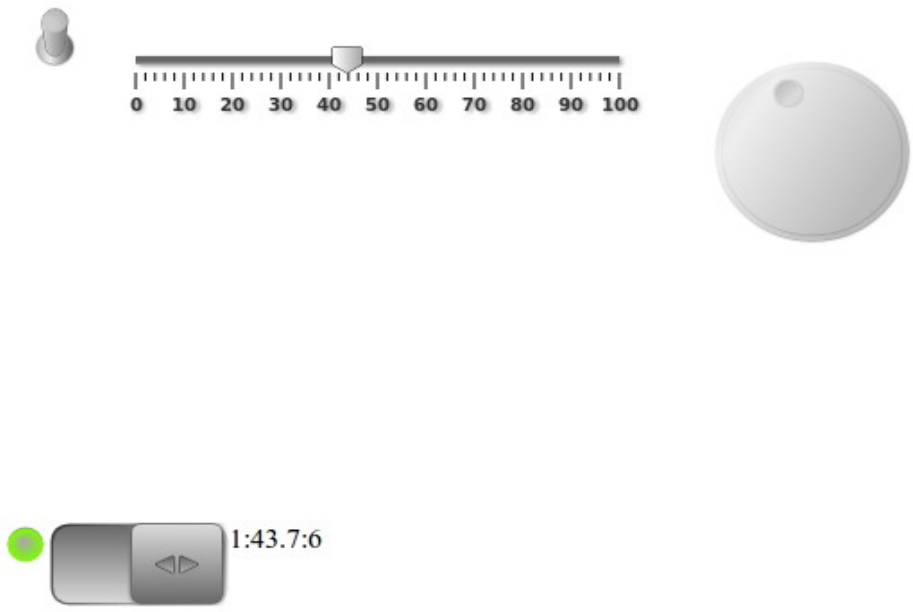

Figure 13: The result of Example 20. If the values of ToggleSwitch, Knob and AnalogSlider change, the innerHTML of a Div changes by an AJAX call. When the SideSwitch turns on, if the response of server.php is "1", the LED is turned on.

With the addCall function we append an additional code to execute at the end of the load request and control the response from the server, if is "1" then we turn on the LED. a*jaxobj* is the AJAX response object.

Combine Instruments.

Append a child

Drinks allows to combine instruments just putting a child element inside the tag.

Every instrument can have children, and its children can have other children that are stored in an array called *inner*.

For example we put a LED into a gauge:

```
<display id="gau">
    <led id="led1"></led>
</display>
```

The LED will be placed at (0,0) coordinates. To put it in the desired position, we have two attributes *x* and y.

```
<display id="gau">
    <led id="led1" x="105" y=180></led>
</display>
```

Now we put another display inside the gauge.

```
<display id="gau" value="90">
    <display id="dis1" type="digital" width="40" height="20" x="100"
y="180" link></display>
</display>
```

With the *link* attribute we link the value of the digital display to the value of the gauge. We link the value of a child to the value of the parent.

Example 21

A Gauge with a Display and a LED inside. If the value comes into the range the LED turns on.

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
    </head>
    <body>
        <display id="gau" value="90" range_from="60" range_to="100"
onalert="led1.on();" onleavealert="led1.off();">
            <display id="dis1" type="digital" width="40" height="20"
x="100" y="160" link></display>
            <led id="led1" radius="5" x="112" y="170"></led>
        </display>
    </body>
</html>
```

And now we repeat the same example with Javascript.

Example 22

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
        <script>
            function putInside(){
                var disp = Drinks.createElement("display", {"id":"dis1",
"type":"digital", "width":"40", "height":"20", "x":"100", "y":"160", "link":"true"});
                gau.appendChild(disp);
                var ledd = Drinks.createElement("led", {"id":"led1",
```

```
"radius":"5", "x":"112", "y":"170"});
                        gau.appendChild(ledd);
                }
        </script>
    </head>
    <body>
        <display id="gau" value="90" range_from="60" range_to="100">
        </display>
        <input type="button" onclick="putInside()" value="press">
    </body>
</html>
```

Create a new tag

After we have created our combined instrument, we can create a new tag that represents it. For example the Gauge with Display and LED could be called <gaugeadv> and could be used when you want by this tag.

1. Add the tag to *all_type* array

In the Drinks.js file there is the *all_type* array, you have to append the name of your tag followed by the name of the Javascript class.

```
var all_type={"knob":{"loop":"NoEndKnob", "analog":"EndedKnob",
"digital":"DigitalKnob"}, "display":{"gauge":"Gauge", "digital": "dMeter", "level":
"LevelMeter", "thermo":"ThermoMeter", "analog":"VuMeter"}, "led":
{"round":"RoundLed", "rect":"RectLed", "triangle":"TriangleLed"}, "switch":
{"rocker":"RockerSwitch", "arc":"ArcSwitch", "side":"SideSwitch", "circle":
"CircleSwitch", "rect":"RectSwitch", "lswitch":"lSwitch"}, "oscope":"Oscilloscope",
"slider":{"analog":"AnalogSlider", "digital":"DigitalSlider"},
"gaugeadv":"GaugeAdv"};
```

Then if you want to put this class in a new file you have to include this

includeJS(['Display.js', 'Knob.js', 'Led.js', 'Oscilloscope.js', 'Switch.js', 'Slider.js', ***'GaugeAdv.js'****], path, initialize);*

else put the class in an existing file already included.

2. Create the class.

```
function GaugeAdv(element){
        GaugeAdv.inherits(Instrument);   //GaugeAdv inherit from instrument.
        Instrument.call(this, element);   //Call the instrument constructor

        //CREATION CODE

        this.render = function(){      //Render function
                this.instrumentCommonOperations();     //Necessary to implement the
                                                       common operations for every
                                                            instrument.
        }
}
```

element is the HTML element of your tag.

3. Create your instrument.

First create the gauge:

```
        var gau = Drinks.createElement("display", {"id": "gau", "range_from":"60",
"range_to":"100"});
```

and append it to gaugeadv element.

```
        this.appendChild(gau);
```

Then create the digital display:

var disp = Drinks.createElement("display", {"id":"dis1", "type":"digital", "width":"40", "height":"20", "x":"100", "y":"160", "link":"true"});

and append it to first child (gauge).

this.inner[0].appendChild(disp);

Then create the LED:

var ledd = Drinks.createElement("led", {"id":"led1", "radius":"5", "x":"112", "y":"170"});

and append it to first child (gauge).

this.inner[0].appendChild(ledd);

Then add what you want:

this.inner[0].onalert="led1.on();";
this.inner[0].onleavealert="led1.off();";

In HTML code:

<gaugeadv></gaugeadv>

and your instrument will appear.

IMPORTANT NOTE:

If you have created your tag, maybe you want to use it often or in the future.
But you know that the id of the instruments is unique and you can't duplicate it.
Therefore we must do a trick:

var parent_id = element.getAttribute("id");
var disp = Drinks.createElement("display",
***{"id":parent_id+"gau"**, "type":"digital", "width":"40", "height":"20", "x":"100", "y":"160", "link":"true"});*

So if you add this tag:

<gaugeadv id="adv"></gaugeadv>

the id of the gauge will be *"advgau"* and the same thing for the other child instruments.

So, to access the gauge for example:

advgau.value=90;

You can add other attributes, for example:

<gaugeadv id="adv" ***style="float:left;" led_color="red" max_range="200"*** ></gaugeadv>

And in your class:

```
var style = element.getAttribute("style");
var led_color = element.getAttribute("led_color");
var max_range= element.getAttribute("max_range");

…....

gauge.setAttribute("style", style);
gauge.setAttribute("max_range", max_range);

…..

ledd.setAttribute("color", led_color);
```

Example 23

```
function GaugeAdv(element){
    GaugeAdv.inherits(Instrument);   //GaugeAdv inherit from instrument.
    Instrument.call(this, element);   //Call the instrument constructor
    var style = element.getAttribute("style") || "";
    var led_color = element.getAttribute("led_color") || "";
    var max_range = element.getAttribute("max_range") || "";
    var range_from = element.getAttribute("range_from") || "";
```

```
        var range_to = element.getAttribute("range_to") || "";
        var parent_id = element.getAttribute("id") || "";
        var cipher = element.getAttribute("cipher") || "";
        var significative = element.getAttribute("significative") || "";
        var value = element.getAttribute("value") || "";

        var gau = Drinks.createElement("display", {"id": parent_id+"gau",
"max_range":max_range, "range_from":range_from, "range_to":range_to,
"style":style, "value":value});
        this.appendChild(gau);

        var disp = Drinks.createElement("display", {"id": parent_id+"dis",
"type":"digital", "width":"40", "height":"20", "x":"100", "y":"160", "link":"true",
"cipher": cipher, "significative":significative, "max_range":max_range,});
        this.inner[0].appendChild(disp);

        var ledd = Drinks.createElement("led", {"id":parent_id+"led", "radius":"5",
"x":"112", "y":"170", "color":led_color});
        this.inner[0].appendChild(ledd);

        this.inner[0].onalert="this.inner[1].on();";
        this.inner[0].onleavealert="this.inner[1].off();";

        this.render = function(){      //Render function
                this.instrumentCommonOperations();     //Necessary to implement the
                                                       common operations for every
                                                            instrument.
        }
}
```

newtag.htm

```
        <html>
        <head>
```

```
        <script type="text/javascript"src="Drinks.js"></script>

    </head>
    <body>
        <gaugeadv id="adv" style="float:left;" led_color="red"
max_range="200" value="110" range_from="100" range_to ="130" cipher="3"
significative="1" ></gaugeadv>
    </body>
    </html>
```

Figure 14: The result of Example 23. A Gauge with a DigitalDisplay and a LED.

The gaugeadv is already provided by Drinks toolkit.

HTML elements as childs

Example 24:

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>
    </head>
```

```
    <body>
        <display style="float:left; position:relative;" id="gau1"
max_range="6000" text="rpm"
onchange="document.getElementById('t1').value=this.value;">
            <input type="text" id="t1" style="position:absolute; z-index:5;
top:170px; width:60px; left:87px;">
            <img src="pink.png" width="32" height="32"
style="position:absolute; z-index:5; top:140px; left:100px;">
        </display>
        <display id="gau2" style="float:left; position:relative;" radius=100
max_range="1" divisions="10" value="0.8" color="blue">
            <img src="fuelIcon.png" width="32" height="32"
style="position:absolute; z-index:5; top:160px; left:105px;">
        </display>
    </body>
</html>
```

and the result is:

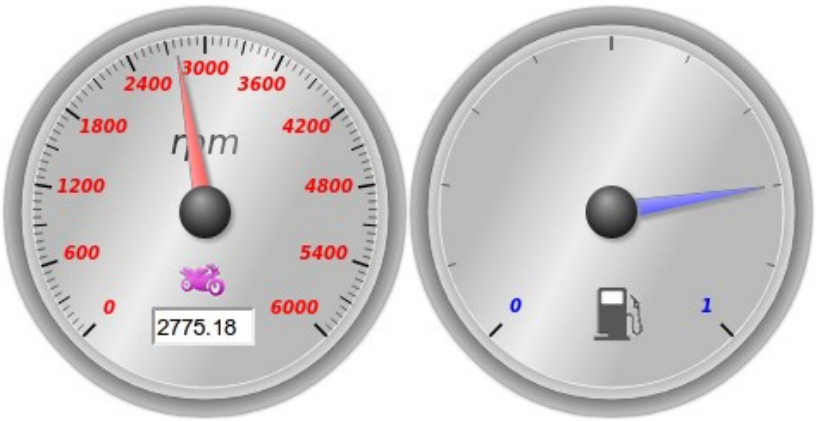

Figure 15: The result of Example 24. Two Gauges with two images inside and the first with a input text that shows the Gauge value.

The rotate attribute

This attribute rotates your instrument of a desired angle specified as parameter.

Example 25:

```
<html>
	<head>
		<script type="text/javascript"src="Drinks.js"></script>

	</head>
	<body>
		<switch type="arc" id="sw" rotate="270">
			<led type="triangle" id="tr" width="20" rotate="90"
height="20" x="45" y="25" link></led>
		</switch>
	</body>
</html>
```

And this is the result when is on and off:

Figure 16.a: The ArcLed switch when is off.

Figure 16.b: The ArcLed switch when is on.

This instrument is already provided by Drinks toolkit and its tag is <arcled>.

Example 26

A led meter.

```
<html>
    <head>
        <script type="text/javascript"src="Drinks.js"></script>

    </head>
    <body>
        <div style="border: 2px outset #ddd; float:left; width:60px;
height:200px; background-color:#333;">
            <led type="rect" width="60" height="20" id="led1"
style="float:left;" color="red"></led>
            <led type="rect" width="60" height="20" id="led2"
style="float:left;" color="red"></led>
            <led type="rect" width="60" height="20" id="led3"
style="float:left;" color="red"></led>
            <led type="rect" width="60" height="20" id="led4"
style="float:left;" color="red"></led>
            <led type="rect" width="60" height="20" id="led5"
style="float:left;" color="yellow"></led>
            <led type="rect" width="60" height="20" id="led6"
style="float:left;" color="yellow"></led>
            <led type="rect" width="60" height="20" id="led7"
style="float:left;" color="yellow"></led>
            <led type="rect" width="60" height="20" id="led8"
```

```
style="float:left;" color="green"></led>
                    <led type="rect" width="60" height="20" id="led9"
style="float:left;" color="green"></led>
                    <led type="rect" width="60" height="20" id="led10"
style="float:left;" color="green"></led>
            </div>

            <knob id="k1" radius="60" onchange="for(var i in Drinks.leds)
{if(i<this.value){Drinks.leds[9-i].on();}else{Drinks.leds[9-i].off();}}">
        </body>
</html>
```

And the result is:

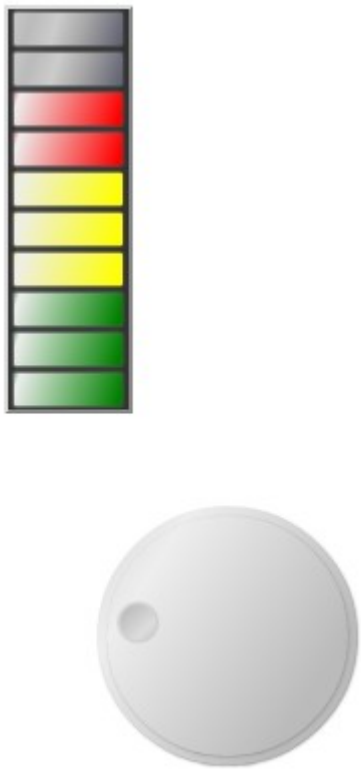

Figure 17: The result of Example 26.

The Display Graph

First steps

This instrument is used to represent a collection of points. In the X-axis we have the time and in the Y-axis the value of the point. This points are stored in an array by the *setPoints* function. First we'll see how to create this instrument.

```
<display type="graph" id="g1">
</display>
```

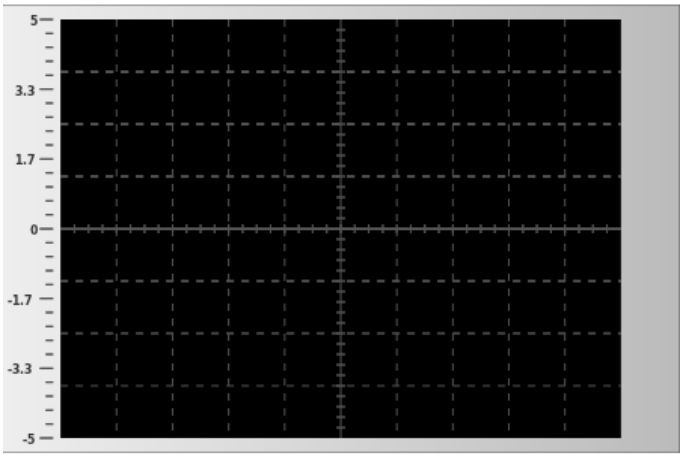

Figure 18: The Display Graph.

How we can see, we have a scale on the left and divisions inside the screen.

Using the scale attribute, we can set how the wave will expand in the y-axis. There are two mode: The first is using the *max_range* and *min_range* attribute (Range Scale) and second using the divisions (Divisions Scale).

Range Scale

In this mode the points of the wave will be drawed according the value of the scale. The values outside the range will not be drawed but is possible to use the *autoscale*

property in order to resize the range if your wave goes out.

```
<display type="graph" id="g1" scale="range" autoscale="true">
</display>
```

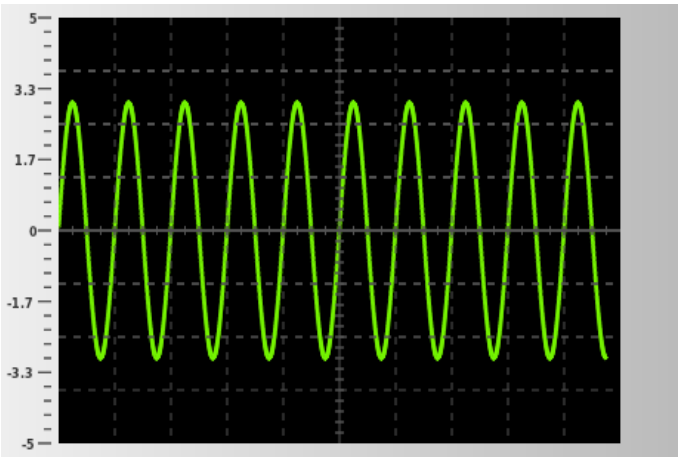

Figure 19: A wave with a amplitude value of 3 drawed using Range Scale.

Divisions Scale

In this mode you have to set the amplitude value of each division.
For example if you set a value of 3 per divisions:

```
<display type="graph" id="g1" scale="divisions">
</display>
```

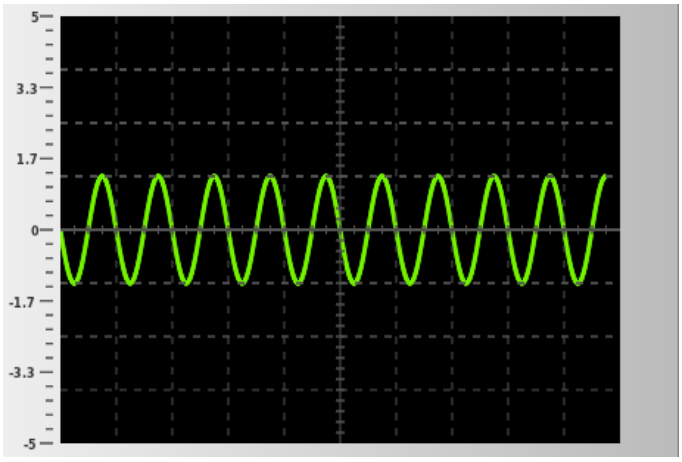

Figure 20: A wave with a amplitude value of 3 drawed using Divisions Scale with a value of amplitude per divisions of 3.

Some Customizations

In order to use the graph for your purposes, you can customize the instruments for example hiding the divisions setting the *hide_divisions* attribute.

```
<display type="graph" id="g1" hide_divisions="true">
</display>
```

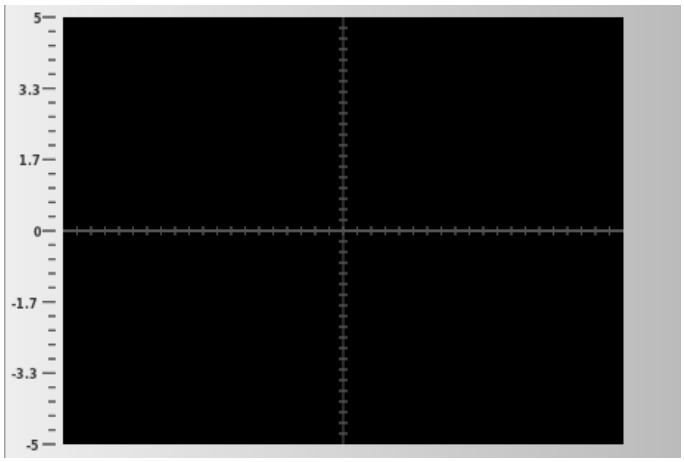

Figure 21: The Display Graph with the hide_divisions attribute setted.

Or hiding the axis setting the *hide_axis* attribute.

*<display type="graph" id="g1" **hide_axis="true"** hide_divisions="true">*
</display>

Figure 22: The Display Graph with the hide_axis and hide_divisions attributes setted.

Or changing the background color setting the *bgcolor* attribute.

<display type="graph" id="g1" hide_axis="true" color="white" hide_divisions="true">
</display>

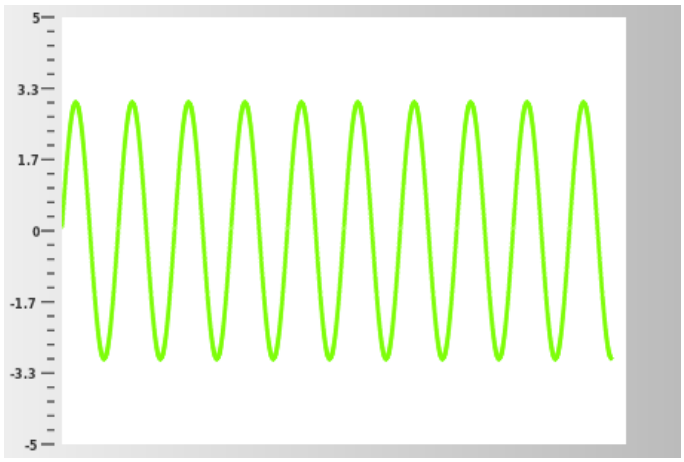

Figure 23: The Display Graph with the bgcolor attribute setted to white.

Or if you are using the Divisions Mode, you can hide the scale on the left setting the *hide_grid* attribute.

```
<display type="graph" id="g1" hide_grid="true">
</display>
```

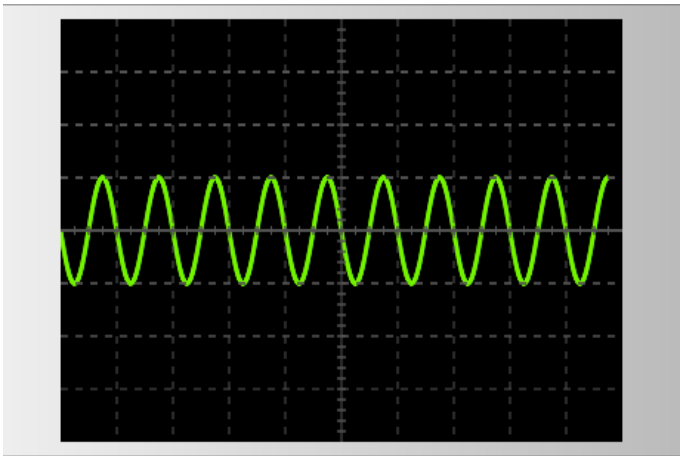

Figure 23: The Display Graph with the hide_grid attribute setted.

How to load the points.

The channel tag

In the innerHTML of the instrument we can have at most two channels (for this version) and set some properties for each of them. For example:

```
<display type="graph" id="g1">
        <channel amplitude="1"></channel>
        <channel sweep="1"></channel>
</display>
```

But how to set the points to draw?

1. Using JS:

```
<script>
    function load(){
        var arr = new Array();
        for(var i=0; i<360; i++){
            arr.push(Math.sin(i*Math.PI/180)*2);
        }
        g1.channel[0].setPoints(arr);
    }
</script>
<display type="graph" id="g1" onload="load();">
    <channel></channel>
</display>
```

2. Inside the channel tag using a server script:

```
<?php
    for($i=0; $i<360; $i++){
        $num[$i] = round(sin($i* M_PI / 180 + M_PI/4)*3*100)/100;
    }
    echo "
        <display type=\"graph\" id=\"g1\">
            <channel>".json_encode($num)."</channel>
        </display>";
?>
```

Note: Hide the channel to avoid to see the text inside the element.

```
<channel style="display:none;">".json_encode($num)."</channel>
```

3. Using the onLoad event and a server script:

```
<?php
    for($i=0; $i<360; $i++){
        $num[$i] = round(sin($i* M_PI / 180 + M_PI/4)*3*100)/100;
    }
    echo "
        <display type=\"graph\" id=\"g1\"
    onload=\"this.channel[0].setPoints(".json_encode($num).");\">
            <channel></channel>
        </display>";
    ?>
```

4. Using AJAX and *href* attribute.

```
    <display type="graph" id="g1">
            <channel href="osc1.php" refresh="4"></channel>
    </display>;
```

in osc1.php:

```
    <?
        for($i=0; $i<360; $i++){
            $num[$i] = round(sin($i* M_PI / 180)*5*100)/100;
        }
        echo json_encode($num);
    ?>
```

and using the refresh attribute, the osc1.php script will be called every 4 seconds and the points will be updated.

Set amplitude to a channel

Amplitude is an attribute that allow to increase or decrease the amplitude of the wave in that channel. It is a number that is multiplied for each point of the wave.

For example:

<*channel* ***amplitude="5"***></*channel*>

So if you have a max_range of 5, you'll see that the wave fills the screen and the real peak value will be 1.

If you are using the Divisions Scale, you can set *amp4div* attribute. This attribute set the amplitude value for a division.

So if you have a wave which has a peak value of 9, and you set *amp4div* to 3, you'll see the wave that fills three divisions.

For example:

<*channel* ***amp4div="3"***></*channel*>

Set sweep and frequency to a channel

Sweep is an attribute that allow to set a time duration for each division. So a wave with a frequency of 1 Hz (period 1sec) and a sweep of 1 sec per divisions, takes one division to complete its period.

For example:

<*channel* ***sweep="0.001" frequency="1000"***></*channel*>

We have that our wave has a frequency of 1 Khz and one division is 1 ms.

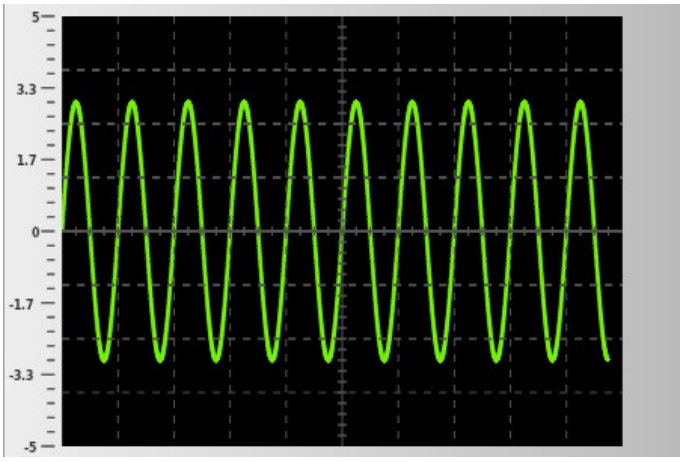

Figure 24: A wave with a frequency of 1KHz and the Graph has a sweep of 1 ms.

Or if one division is 100 us:

<channel ***sweep="0.0001" frequency="1000"****></channel>*

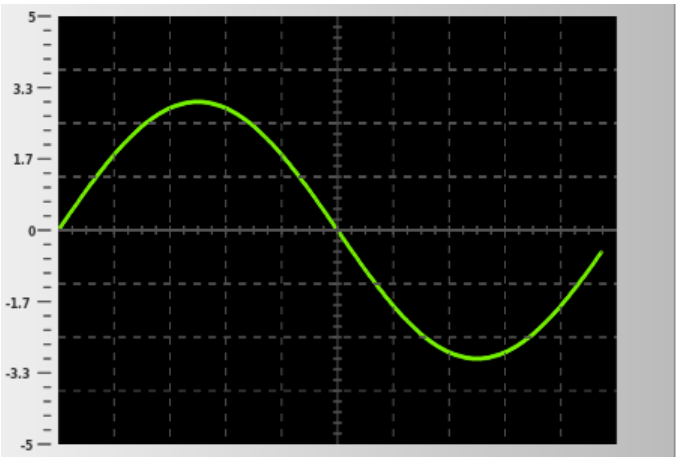

Figure 25: A wave with a frequency of 1KHz and the Graph has a sweep of 100 us

Set X and Y position

X and Y position can be setted for each channel and represent the x and y coordinates that the instrument uses to start drawing. It is expressed in pixels.

For example:

```
<channel  x_position=\"100\" y_position=\"-100\"></channel>
```

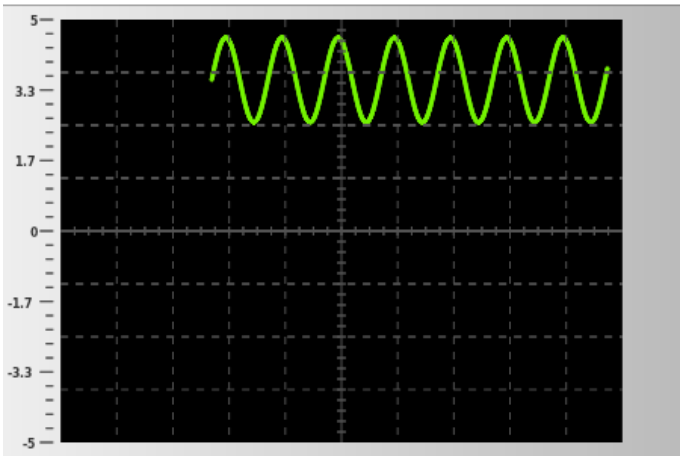

Figure 26: The Display Graph with x_position and y_position attributes setted.

A first complete example

Example 27

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
        <script>
        </script>
    </head>
```

```
<body id="body">
<?
        //Create first wave
        for($i=0; $i<360; $i++){
                $num[$i] = round(sin($i* M_PI / 180)*3*100)/100;
        }
        //Create second wave
        $k=-1;
        $num2 = array();
        for($i=0; $i<800; $i++){
                if($i >= $k*100 && $i<($k+1)*100 && $k%2==0)
                        $num2[$i] = 1;
                else
                        $num2[$i] = -1;

                if($i%100==0)
                        $k++;
        }
    echo "
        <display type=\"graph\" id=\"g1\" power_onload=\"true\">
                <channel style=\"display:none;\" frequency=\"1000\"
sweep=\"0.001\">".json_encode($num)."</channel>
                <channel style=\"display:none;\" amplitude=\"2\" frequency=\"1\"
sweep=\"0.1\" color=\"red\">".json_encode($num2)."</channel>
        </display>";
    ?>
    </body>
    </html>
```

This first example has two channels with the points inside them and some attributes setted. We have seen what are frequency, sweep, amplitude but what about *power_onload* and *color*?

Color is the color of the wave.

The Display Graph instrument is normally off. To turn on, you have to call the *on* method for example by a switch or you can automatically activate it with the power_onload attribute setted. There is the *off* method too that allows to turn off the graph.

```
<switch id="sw1" onchange="
    if(this.value=='1')
        {g1.on();}
    else{g1.off();}"></switch>
```

The mode attribute

If you try the last example you'll see only the wave in the first channel. And the second? *Mode* allows to select different working mode of the graph.

1. CH1 mode

```
<display type="graph" id="g1" mode="ch1">
</display>
```

The graph will draw only the first channel.

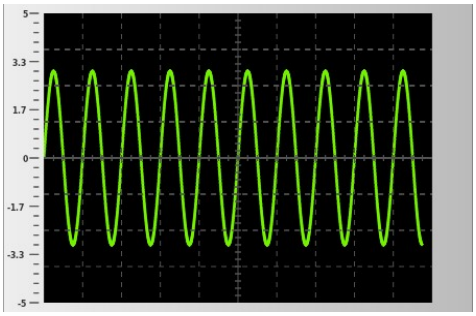

Figure 27: CH1 mode.

2. CH2 mode

```
<display type="graph" id="g1" mode="ch2">
</display>
```

The graph will draw only the second channel.

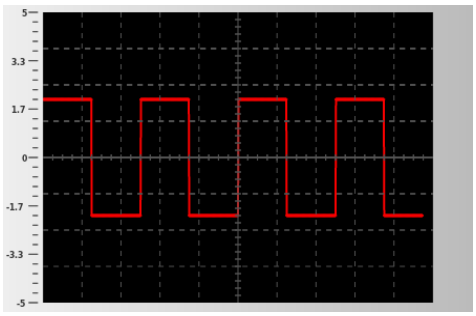

Figure 28: CH2 mode.

3. DUAL mode

```
<display type="graph" id="g1" mode="dual">
</display>
```

The graph will draw both channel.

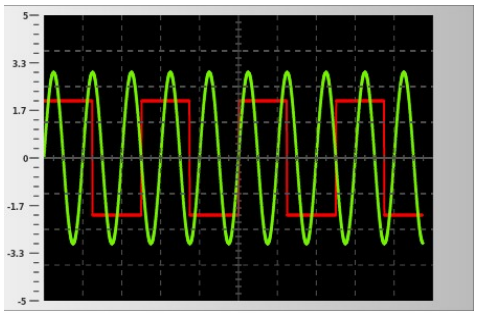

Figure 29: DUAL mode.

4. X/Y

```
<display type="graph" id="g1" mode="x/y">
</display>
```

The graph will draw the first channel in the x-axis and second channel in y-axis.

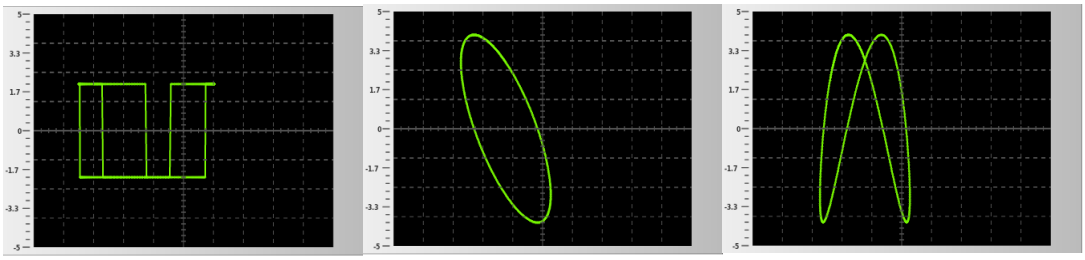

Figure 30: Some pictures about x/y mode.

Channel informations

It is possible to retrieve some informations from each channel like peak value, peaktopeak value and average of the wave.

The functions to retrieve this values are respectively *getPeak*, *getPeakToPeak* and *getAverage.*

The OnCompleteScreen Event

This event triggers when the Graph complete the rendering of a screen and for example you can do something like this:

This JS functions retrieve the info for each channel and set the values of some DigitalDisplay.

```
function getInfo(){
        d1.value=oscope1.channel[0].getPeak();
```

```
            d2.value=oscope1.channel[0].getPeakToPeak();
            d3.value=oscope1.channel[0].getAverage();
        }
        function getInfo2(){
            d4.value=oscope1.channel[1].getPeak();
            d5.value=oscope1.channel[1].getPeakToPeak();
            d6.value=oscope1.channel[1].getAverage();
        }

<display type="graph" id="g1">
        <channel style="display:none;" oncompletescreen="getInfo();"
href="channel1.php" refresh="2"></channel>
        <channel style="display:none;"  oncompletescreen="getInfo2();"
href="channel2.php" refresh="2"></channel>
    </display>";
```

Example 28

```
<html>
    <head>
        <script type="text/javascript" src="Drinks.js"></script>
    <script>
        function getInfo(){
            d1.value=g1.channel[0].getPeak();
            d2.value=g1.channel[0].getPeakToPeak();
            d3.value=g1.channel[0].getAverage();
        }
        function getInfo2(){
            d4.value=g1.channel[1].getPeak();
            d5.value=g1.channel[1].getPeakToPeak();
            d6.value=g1.channel[1].getAverage();
```

```
        }
    </script>
    </head>
    <body id="body">
    <?
        for($i=0; $i<360; $i++){
            $num[$i] = round(sin($i* M_PI / 180)*2*100)/100;
        }
        $k=-1;
        $num2 = array();
        for($i=0; $i<800; $i++){
            if($i >= $k*100 && $i<($k+1)*100 && $k%2==0)
                $num2[$i] = 1;
            else
                $num2[$i] = 0;

            if($i%100==0)
                $k++;
        }
        $num3 = array();
        $m=0;
        for($i=0; $i<360; $i++){
            $m+=2;
            $num3[$i] = round(sin($m* M_PI / 180+M_PI/4)*2*100)/100;

        }
    echo "
        <display type=\"graph\" id=\"g1\" power_onload=\"true\"
mode=\"dual\">
            <channel style=\"display:none;\" frequency=\"1000\"
sweep=\"0.001\"
oncompletescreen=\"getInfo();\">".json_encode($num)."</channel>
```

```
                <channel style=\"display:none;\" amplitude=\"2\"
frequency=\"1\" sweep=\"0.1\" color=\"red\"
oncompletescreen=\"getInfo2();\">".json_encode($num2)."</channel>
            </display>";
        ?>
            <div style="width:100%; float:left;">
                <display type="digital" id="d1" label="peak CH1"
style="float:left;" width="100" height="50" cipher="3" significative="1"
></display>
                <display type="digital" id="d2" label="peak2peak CH1"
style="float:left;" width="100" height="50" cipher="3" significative="1"
></display>
                <display type="digital" id="d3" label="average CH1"
style="float:left;" width="100" height="50" cipher="3"
significative="1"></display>
            </div>
            <div style="width:100%; float:left;">
                <display type="digital" id="d4" label="peak CH2"
style="float:left;" width="100" height="50" cipher="3" significative="1"
></display>
                <display type="digital" id="d5" label="peak2peak CH2"
style="float:left;" width="100" height="50" cipher="3" significative="1"
></display>
                <display type="digital" id="d6" label="average CH2"
style="float:left;" width="100" height="50" cipher="3"
significative="1"></display>
            </div>

        </body>
    </html>
```

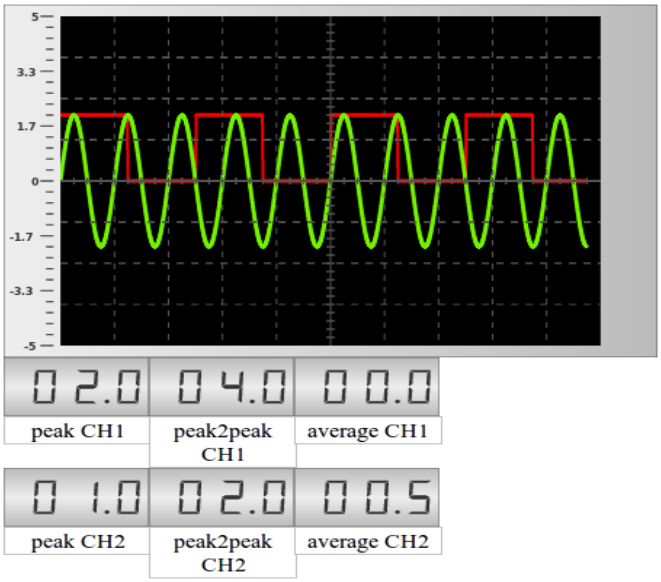

Figure 31: The result of example 28.

Set attribute from JS

And now we are going to create a panel to manipulate easily the various configurations using JS.

```
<html>
<head>
	<script type="text/javascript" src="Drinks.js"></script>
	<script>
	</script>
</head>
<body style="background-color:#ddd;" id="body" >
	<div style="width:450px; height:320px; float:left;">
	<?
		$m=0;
```

```
        for($i=0; $i<360; $i++){
            $m+=2;
            $num[$i] = round(sin($m* M_PI / 180 + M_PI/4)*3*100)/100;

        }
        for($i=0; $i<360; $i++){
            $m++;
            $num2[$i] = round(sin($m* M_PI / 180)*3*100)/100;
        }
    echo "<display type=\"graph\" id=\"g1\" style=\"float:left;\"
scale=\"divisions\">
        <channel style=\"display:none;\" >".json_encode($num)."</channel>
        <channel style=\"display:none;\"
color=\"red\">".json_encode($num2)."</channel>
    </display>";
    ?>
```

And now we add a slider that changes mode.

```
    </div>
    <div style="width:650px; height:320px; float:left;">

    <slider id="m" type="digital" style="float:left;" align="vertical"
onchange="g1.setMode(this.value);" height="180" width="150">
        <option value="ch1">CH1</option>
        <option value="ch2" >CH2</option>
        <option value="dual">Dual</option>
        <option value="x/y">X/Y</option>
    </slider>
    </div>
```

And a button that turns on the Graph.

```
<div style="width:450px; height:300px; float:left;">
    <div style="width:100%; float:left;">
        <switch  type="toggle" id="toggle1" style="float:left;"
onchange="if(this.value=='1'){g1.on();}else{g1.off()} power_led.toggle();"
width="100" ></switch>
        <led  id="power_led" style="float:left; margin-top:30px;"
radius="10" color="red"></led>
    </div>
</div>
```

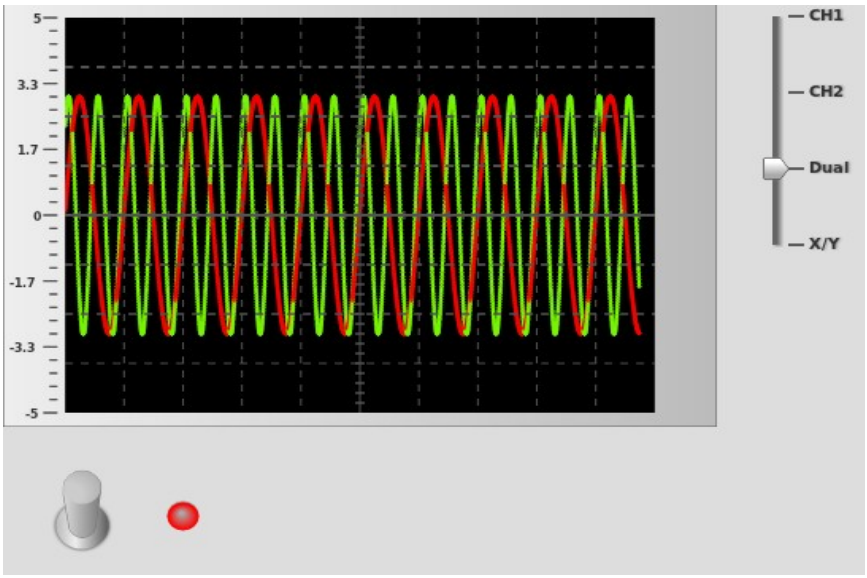

Figure 32: First part of final example.

Now we add two knobs after the slider to set Sweep for ch1 and ch2.

```
<knob id="sweep1"  type="digital" label="Sweep Time/Div CH1"
radius="60" style="float:left" onchange="g1.channel[0].sweep=this.value;">
    <option value="0.5">.5</option>
    <option value="0.2">.2</option>
    <option value="0.1">.1</option>
    <option value="0.05">50</option>
    <option value="0.02">20</option>
```

```
            <option value="0.01">10</option>
            <option value="0.005">5</option>
            <option value="0.002">2</option>
            <option value="0.001">1</option>
            <option value="0.0005">.5</option>
            <option value="0.0002">.2</option>
            <option value="0.0001">.1</option>
            <option value="0.00005">50</option>
            <option value="0.00002">20</option>
            <option value="0.00001">10</option>
            <option value="0.000005">5</option>
            <option value="0.000002">2</option>
            <option value="0.000001">1</option>
            <option value="0.0000005">.5</option>
            <option value="0.0000002">.2</option>
            <option value="0.0000001">.1</option>
        </knob>
        <knob id="sweep2"  type="digital" label="Sweep Time/Div CH2"
radius="60" style="float:left" onchange="g1.channel[1].sweep=this.value;">
            <option value="0.5">.5</option>
            <option value="0.2">.2</option>
            <option value="0.1">.1</option>
            <option value="0.05">50</option>
            <option value="0.02">20</option>
            <option value="0.01">10</option>
            <option value="0.005">5</option>
            <option value="0.002">2</option>
            <option value="0.001">1</option>
            <option value="0.0005">.5</option>
            <option value="0.0002">.2</option>
            <option value="0.0001">.1</option>
```

```
        <option value="0.00005">50</option>
        <option value="0.00002">20</option>
        <option value="0.00001">10</option>
        <option value="0.000005">5</option>
        <option value="0.000002">2</option>
        <option value="0.000001">1</option>
        <option value="0.0000005">.5</option>
        <option value="0.0000002">.2</option>
        <option value="0.0000001">.1</option>
    </knob>
```

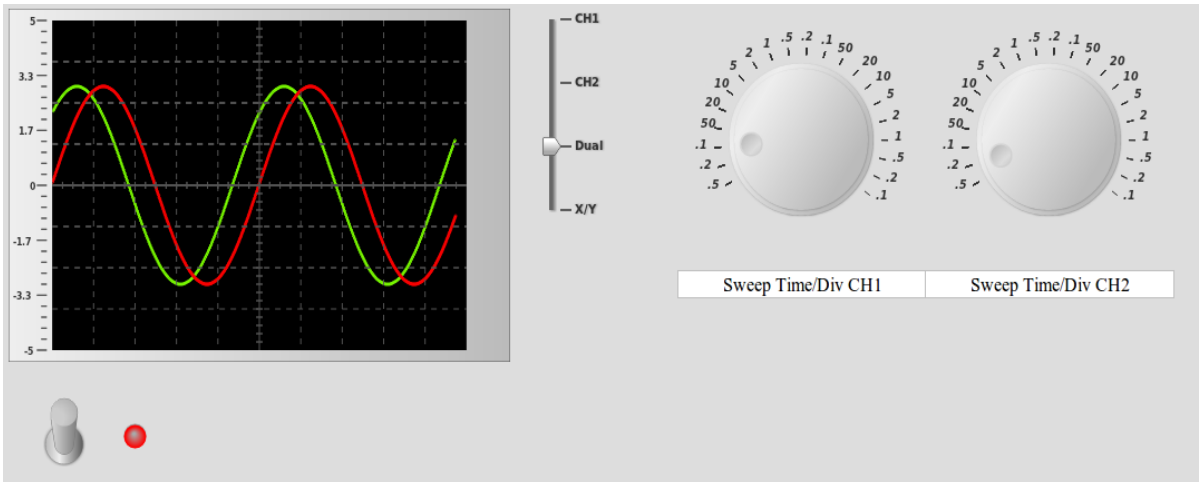

Figure 33: Second part of final example.

And we finish with 6 Knobs to manipulate Amplitude, X position and Y position.

```
    <div style="width:765px; height:300px; float:left;">
        <div style="float:left; width:160px;">
            <knob id="posy1" type="loop" label="Y Position CH1"
radius="40" value="0" min_range="-140" max_range="140" step="5"
style="float:left" onchange="g1.channel[0].y_position=this.value;"></knob>
            <knob id="posx1" type="loop" label="X Position CH1"
radius="40" min_range="0" max_range="400" step="5" style="float:left"
onchange="g1.channel[0].x_position=this.value;"></knob>
```

```
</div>
<knob id="amp1"  type="digital" label="Amplitude Volts/Div CH1" radius="60" style="float:left;" onchange="g1.channel[0].amp4div=this.value;">
    <option value="5">5</option>
    <option value="2">2</option>
    <option value="1">1</option>
    <option value="0.5">.5</option>
    <option value="0.2">.2</option>
    <option value="0.1">.1</option>
    <option value="0.05">50</option>
    <option value="0.02">20</option>
    <option value="0.01">10</option>
    <option value="0.005">5</option>
    <option value="0.002">2</option>
    <option value="0.001">1</option>
</knob>
<knob id="amp2"  type="digital" label="Amplitude Volts/Div CH2" radius="60" style="float:left" onchange="g1.channel[1].amp4div=this.value;">
    <option value="5">5</option>
    <option value="2">2</option>
    <option value="1">1</option>
    <option value="0.5">.5</option>
    <option value="0.2">.2</option>
    <option value="0.1">.1</option>
    <option value="0.05">50</option>
    <option value="0.02">20</option>
    <option value="0.01">10</option>
    <option value="0.005">5</option>
    <option value="0.002">2</option>
    <option value="0.001">1</option>
</knob>
```

```
        <div style="float:right; width:160px;">
            <knob id="posy2" type="loop" label="Y Position CH2"
radius="40" value=0 min_range="-140" max_range="140" step="5"
style="float:left" onchange="g1.channel[1].y_position=this.value;"></knob>

            <knob id="posx2" type="loop" label="X Position CH2"
radius="40" min_range="0" max_range="400" step="5" style="float:left"
onchange="g1.channel[1].x_position=this.value;"></knob>
        </div>
    </div>
```

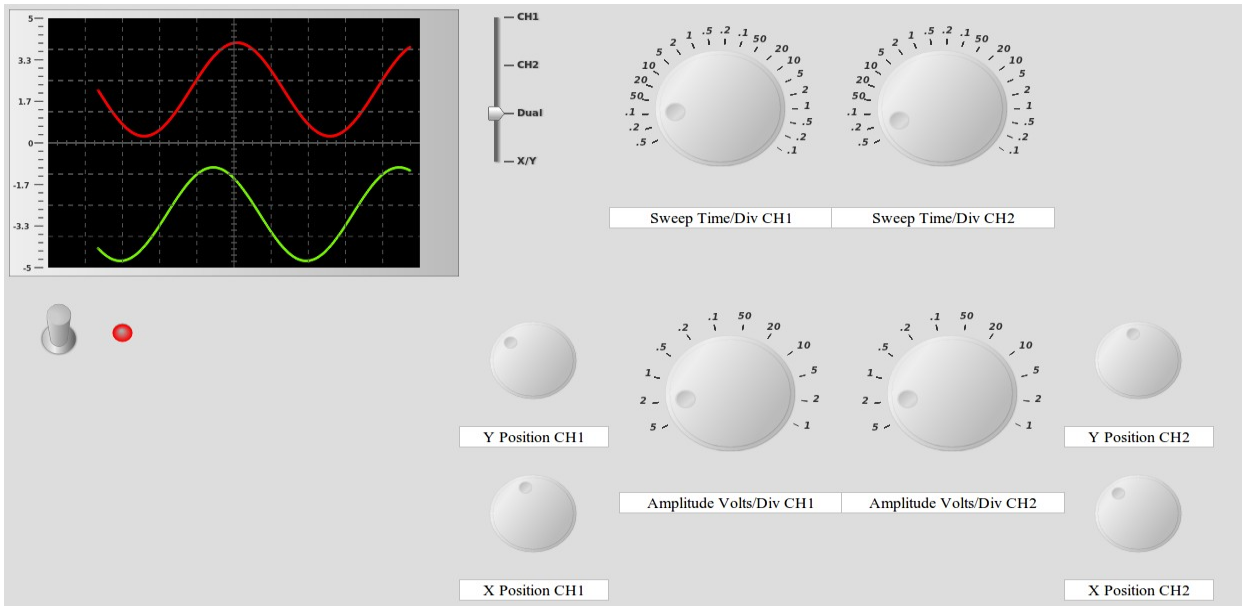

Figure 33: Third part of final example.

The Complete Reference

The Instrument class

Instrument is the root class. Every instrument inherit attributes from this class.

HTML and Javascript Attributes

value [0]

It represents the value of the instrument.

min_range [0]

The minimum value that the instrument can take.

max_range [100]

The maximum value that the instrument can take.

id

Instrument identificator.

href

It links the value of your instrument to a server side script specified in this attribute using AJAX.

refresh [5]

It specifies the seconds interval that divide every Ajax requests to the href file.

onchange

The event that triggered when the instrument changes its value.

onload

The event that triggered when the instrument starts.

label

A instrument descriptions that appear at the bottom of its.

rotate

The rotation angle of the container.

type

Type of instrument that belongs to the less specific class represented by tag name.

width

Instrument width.

height

Instrument height.

radius

Is used for circular instrument but can be used to all.

x [0]

X coordinate of the instrument child inside.

y [0]

Y coordinate of the instrument child inside.

Javascript Attributes

canvas

The HTML Canvas element.

label_container

The div that cointains the label.

pen

The 2d canvas rendering context.

ajax

Ajax object used to href requests.

inner

It is an array that contains all the instrument childs.

prec_value [value]

The precedent value of the instrument.

parent

References to the parent instrument.

container

The div that contains canvas and label.

html

Is the html element.

Methods

appendChild

Appends a child instrument.

Parameters

element

The html element.

setWidth

Sets the canvas width, the instrument width attribute and the container width.

Parameters

width

The width you want to set.

setHeight

Sets the canvas height, the instrument height attribute and the container height.

Parameters

height

The height you want to set.

The Drinks class

It provides a series of useful methods and collections to approach to your widgets.

ready

Is a function that is called after the instrument's initialization.

Methods

appendChild

Appends a child to the parent instrument.

Parameters

id

The id of the parent instrument.

element

The HTML element.

buildElement

build the javascript Drinks object.

Parameters

element

The HTML element.

getElementById

Returns the Drinks object by id.

Parameters

id

The id of the object.

Return parameters

obj

The Drinks object.

getElementsByName

Returns the Drinks objects by name.

Parameters

name

The name of the objects.

Return parameters

objs

An array containing the Drinks objects.

getElementsByClassName

Returns the Drinks object by class name.

Parameters

class

The class of the objects.

Return parameters

objs

An array containing the Drinks objects.

createElement

create the html element from Javascript.

Parameters

tag

The tag of the element you want to create.

obj

A JSON object like {“attribute”:”value”, …} to initialize your html element.

Return parameters

element

The html element.

createManager

create a manager.

Return parameters

man

The manager.

The AJAX class

Provides an useful interface to manipulate AJAX requests.

Methods

setLoader

Sets an HTML content to the div which will retrieve the response of the AJAX request until it is completed.

Parameters

src

The HTML code.

addCall

Add an additional instructions set in an array and to execute all after the response.

Parameters

func

A string with the instructions.

load

Execute an AJAX get request.

Parameters

file

The url of the server side script.

loc

The id of the HTML element which will receive the response.

send

Execute an AJAX post request.

Parameters

form

The form object of the post request.

file

The url of the server side script.

loc

The id of the HTML element which will receive the response.

The Manager class

This class is used to do AJAX requests and update the values of a certain number of instruments.

Using the href attribute, every instrument needs of a server page that sends the own value.

In the server side script, the values must be appended in an array and sended with JSON. The manager splits the array and updates the values of the instruments.

Javascript Attributes

href

The url of the server side script which sends the value.

refresh [0]

The number of seconds that passes from a request to another.

values

An array containing the values of the instruments to update.

Methods

start

Starts the refresh operation.

The Analog class

All the analog instrument inherit attributes from this class.

HTML and Javascript Attributes

hide_grid [false]

Used to hide the grids in the analog instrument.

precision [1]

Is a pow of ten that represents the precision of the instrument.

divisions [100]

The number of divisions of the instrument.

The Digital class

Actually a simbolic class.

The Binary class

All the binary instruments, like buttons or led, inherit attributes from this class.

Methods

on

Turn the value to 1.

off

Turn the value to 0.

toggle

toggle the value.

The Input class

All the input instruments inherit attributes from this class.

Inherits from Instrument.

HTML and Javascript Attributes

displays

Is an array of output objects that the input instrument manage. Used to propagate the value of the input to various output.

The HTML attribute is *display*.

The Output class

All the output instruments inherit attributes from this class.

Inherits from Instrument.

The An2DigOutput class

Group common attributes and methods for analog and digital instruments.

Inherits from Output.

HTML and Javascript Attributes

autoscale [false]

Is a boolean value used to set the autoscale mode. In this way the min_range and max_range attributes will change if you exceed these thresholds.

range_from

It specifies a start value to a range of values that you want to monitor.

range_to

It specifies a end value to a range of values that you want to monitor.

onalert

The event that triggered when the value enter in the range.

onleavealert

The event that triggered when the value leave the range.

The Analog Output class

Inherits from An2DigOutput and Analog.

HTML and Javascript Attributes

text

A text inside the instrument.

Javascript Attributes

position [0]

The position of the pointer.

prec_position [0]

The precedent position of the pointer.

actual_value [value]

The current value in analog instruments. To reach the value, this instruments take a certain time and in each rendering cycle the actual value is stored in this variable.

The Binary Output class

Actually a simbolic class.

Inherits from Output and Binary.

The Digital Output class

HTML and Javascript Attributes

cipher [2]

The number of digit of the instrument.

significative

The number of significative digit of the instrument.

Methods

splitComma

It adapts the float value to a correct digital visualization.

Return parameters

obj

An object with the three attributes.

s*tring*

A string representation of the value, without the comma.

c*omma*

The index of the comma.

negative

A boolean value which indicates that the number is negative.

The Move Input class

All the objects, which are sensibles on the mouse move event, inherit from this class.

Javascript Attributes

x_root [0]

The x coordinate of the movement.

y_root [0]

The y coordinate of the movement.

up_func

An additional function to do when the mouse up.

move_func

An additional function to do when the mouse move.

click_func

An additional function to do when the mouse click.

decx [0]

A x coordinate value as to decrement from the x_root.

decy [0]

A y coordinate value as to decrement from the y_root.

The Knob class

This is a generic class that all the knobs inherit.

Inherits from MoveInput.

Redefined Attributes

radius [100]

The radius of the knob.

decx [width/2]

A x coordinate value as to decrement from the x_root.

decy [height/2]

A y coordinate value as to decrement from the y_root.

x_root [1]

The x coordinate of the movement.

y_root [1]

The y coordinate of the movement.

Methods

setCoordsByAngle

Sets the x and y coords by the angle of the knob value indicator.

getCoords

Find the angle of the indicator by x_root and y_root.

The Digital Knob class

This is a digital knob which may take a limited number of values. You can add the values adding an option tag inside the knob tag.

Inherits from Knob.

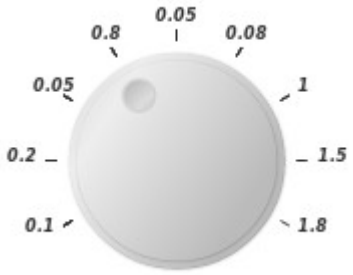

Figure 34: DigitalKnob.

Example:

```
<knob id="k1"  type="digital"  radius="50" >
        <option value="0.1">0.1</option>
        <option value="0.2">0.2</option>
        <option value="10" >0.05</option>
        <option value="0.8" selected>0.8</option>
        <option value="0.05">0.05</option>
        <option value="0.08">0.08</option>
        <option value="1">1</option>
        <option value="1.5">1.5</option>
        <option value="1.8">1.8</option>
</knob>
```

Javascript Attributes

options

Is an array which contains all the options.

selectedIndex [0]

The index of the selected option.

The Analog Knob class

Is a simbolic class made to group common attributes for all the analog knobs.

Inherits from AnalogInput and Knob classes.

The Loop Knob class

This is a knob which hasn't a end. The value is incremented if it is turned clockwise or decremented if anticlockwise, of a value given by step.

Inherits from Knob.

Default type for Knob.

Figure 35: LoopKnob.

HTML and Javascript Attributes

step [1]

The increment or decrement of the value that the knob do when it turns.

Example:

```
<knob id="k1"  type="loop" radius="50"  step="1" >
</knob>
```

The Ended Knob class

This is a knob which has a end. If you cross the angle of the *max_range* value, it returns to *min_range* value.

Inherits from AnalogKnob

Figure 36: EndedKnob.

Redefined Attributes

divisions [100]

The number of divisions of the instrument.

Example:

```
<knob id="k1"  type="analog" radius="50">
</knob>
```

The Slider class

This is a generic class that all the sliders inherit.

Inherits from MoveInput.

cursor [pointer]

A string to select the type of cursors.

pointer

The default value.

square

A square cursor.

round

A round cursor.

align [horizontal]

A string to select the type of slider alignment.

vertical

Vertical alignment.

h*orizontal*

Horizontal alignment.

topdown [false]

A boolean that represents the usage of the instrument. From top to bottom or not.

The Analog Slider class

A slider which can take values from a min and a max value.

Inherits from Slider and AnalogInput.
Default type for Slider.

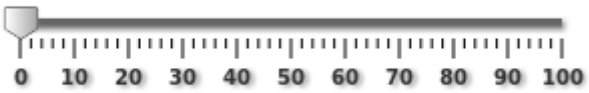

Figure 37: AnalogSlider.

divisions [50]

The number of divisions of the instrument.

decx [5*width/200]

A x coordinate value as to decrement from the x_root.

decy [5*height/200]

A y coordinate value as to decrement from the y_root.

Example:

```
<slider id="s1"  type="analog" width="300">
</slider>
```

The Digital Slider class

This is a digital slider which may take a limited number of values. You can add the values adding an option tag inside the slider tag.

Inherits from Slider and DigitalInput.

Figure 38: DigitalSlider.

options

Is an array which contains all the options.

s*electedIndex* [0]

The index of the selected option.

Example:

<slider id="s1" type="digital" >
</slider>

The Gauge class

A basic indicator.

Inherits from AnalogOutput.
Default type for Display.

Figure 39: Gauge.

HTML and Javascript attributes

color [red]

The color of the numbers and pointer.

Redefined Attributes

radius [100]

The radius of the gauge.

divisions [100]

The number of divisions of the instrument.

Example:

```
<display id="gau1"  type="gauge">
</display>
```

The AnalogMeter class

An analog indicator.

Inherits from AnalogOutput.

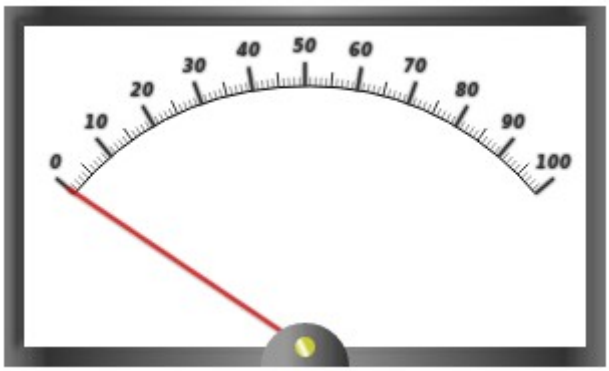

Figure 40: AnalogMeter.

Redefined Attributes

divisions [100]

The number of divisions of the instrument.

Example:

<display id="an1" type="analog">
</display>

The LevelMeter class

A meter which is used often to represent levels.

Inherits from AnalogOutput.

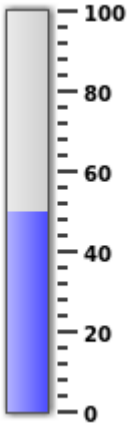

Figure 41: LevelMeter.

HTML and Javascript attributes

color [blue]

The color of the level.

align [vertical]

A string which indicates the alignment of the instrument.

vertical

Vertical alignment.

horizontal

Horizontal alignment.

topdown [false]

A boolean that represents the usage of the instrument. From top to bottom or not.

rounded [false]

A boolean used to set if the level is rounded or not.

Redefined Attributes

divisions [25]

The number of divisions of the instrument.

Example:

<display id="lev1" type="level" value="50">

</display>

The ThermoMeter class

A meter which is used to represent temperature.

Inherits from AnalogOutput.

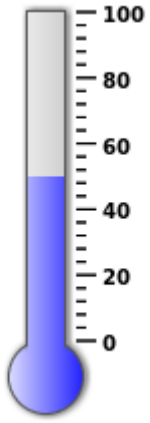

Figure 42: ThermoMeter.

HTML and Javascript attributes

color [blue]

The color of the Thermometer.

align [vertical]

A string which indicates the alignment of the instrument.

vertical

Vertical alignment.

horizontal

Horizontal alignment.

topdown [false]

A boolean that represents the usage of the instrument. From top to bottom or not.

Redefined Attributes

divisions [25]

The number of divisions of the instrument.

The Digital Meter class

A seven-segment display.

Inherits from DigitalOutput.

Figure 43: DigitalDisplay.

HTML and Javascript attributes

color [auto]

The color of the segments.

bgcolor [auto]

The background color of the DigitalDisplay.

Example:

```
<display id="dis1"  type="digital">
</display>
```

The Led class

This is a generic class that all the leds inherit.

Inherits from BinaryOutput.

HTML and Javascript attributes

color [lawngreen]

The color of the LED when is turned on.

The Round Led class

A round shaped LED.

Inherits from LED.
Default type for LED.

Figure 44: RoundLed.

Redefined Attributes

radius [10]

The radius of the LED.

Example:

```
<led id="led1"  type="round" radius="20" value="1">
</led>
```

The Rect Led class

A rect shaped LED.

Inherits from LED.

Figure 45: RectLed.

Example:

```
<led id="led1"  type="rect" >
</led>
```

The Triangle Led class

A triangle shaped LED.

Inherits from LED**.**

Figure 46: TriangleLed.

Example:

```
<led id="led1"  type="triangle" >
</led>
```

The Toggle Button class

This is a generic class that all the toggle buttons inherit.

Inherits from BinaryInput.

The Rocker Switch class

Inherits from ToggleButton.

Figure 47: RockerSwitch.

Example:

```
<switch id="sw1"  type="rocker" >
</switch>
```

The Arc Switch class

Inherits from ToggleButton.

Figure 48: ArcSwitch.

Example:

```
<switch id="sw1"  type="arc" >
</switch>
```

The Side Switch class

Inherits from ToggleButton.

Figure 49: SideSwitch.

Example:

```
<switch id="sw1"  type="side" >
</switch>
```

The Circle Switch class

Inherits from ToggleButton.

Figure 50: CircleSwitch.

Example:

```
<switch id="sw1"  type="circle" >
</switch>
```

The Rect Switch class

Inherits from ToggleButton.

Figure 51: RectSwitch.

HTML and Javascript attributes

text

A text inside the button.

Example:

<switch id="sw1" type="rect" >

</switch>

The Toggle Switch class

Inherits from ToggleButton**.**

Default type for Switch.

Figure 52: ToggleSwitch.

Example:

<switch id="sw1" type="toggle" >

</switch>

The Display Graph class

Inherits from AnalogOutput**.**

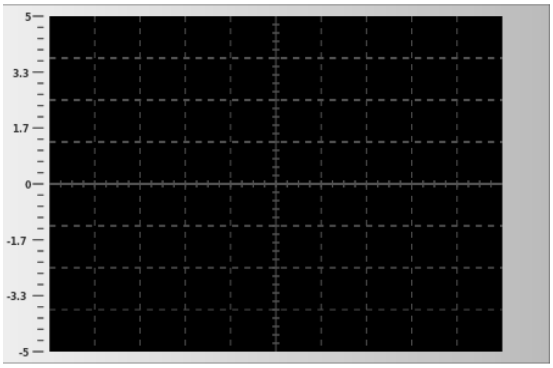

Figure 53: Graph.

HTML and Javascript attributes

channel

An array containing the channels.

divisions [30]

The divisions of the range scale.

hide_divisions [false]

A boolean used to hide the divisions inside the Graph.

hide_axis [false]

A boolean used to hide the axisinside the Graph.

scale [range]

A string to switch scale.

r*ange*

Range Scale

divisions

Divisions Scale

bgcolor [black]

The background color of the Graph.

power_onload [false]

A boolean used to turn on the Graph on page load.

Methods

on

Turns on the Graph.

off

Turns off the Graph.

setMode

Set the working mode of the Graph

Parameters

mode

A string that can be:

ch1

CH1 mode.

ch2

CH2 mode.

dual

DUAL mode.

x/y

X/Y mode.

The Channel class

frequency [1]

The frequency of the wave.

amplitude [1]

The amplitude of the wave.

amp4div [1]

The amplitude value for each division.

y_position [0]

The y_position of the wave.

x_position [0]

The x_position of the wave.

color [lawngreen]

The color of the wave.

href

It links the points of the channel to a server side script specified in this attribute using AJAX.

onCompleteScreen

The event that triggers when the Graph completes to draw a screen.

Methods

setPoints

Load the points in the channel.

Parameters

points

an array of points.

getPeakToPeak

Retrieve the peaktopeak value.

getPeak

Retrieve the peak value.

getAverage

Retrieve the average.

Composite Instruments provided by Drinks

The GaugeAdv Class

Inherits from Instrument**.**

Instruments: Gauge, DigitalDisplay, RoundLED.

Figure 36: GaugeAdv.

HTML and Javascript attributes

radius

The radius of the Gauge.

style

The style of the Gauge.

led_color

The color of the LED.

max_range

max range for Gauge and DigitalDisplay.

min_range

min range for Gauge and DigitalDisplay.

range_from

The start value for range of alert of Gauge and DigitalDisplay.

range_to

The end value for range of alert of Gauge and DigitalDisplay.

id

The id of the GaugeAdv HTML element.

This is the parent_id.

Gauge id : parent_id+'gau';

DigitalDisplay id: parent_id+'dis';

LED id: parent_id+'led';

cipher

The cipher of the DigitalDisplay.

significative

The significative cipher of the DigitalDisplay.

value

The value for Gauge and DigitalDisplay.

DigitalDisplay has the *link* attribute set to true.

OnAlert

The onalert event for the Gauge.

By default it turns on the LED.

OnLeaveAlert

The onleavealert event for the Gauge.

By default it turns off the LED.

The ArcLed Class

Inherits from Instrument**.**

Instruments: ArcSwitch**,** TriangleLed.

width

The width of the ArcSwitch.

height

The height of the ArcSwitch.

id

The id of the ArcLed HTML element.

This is the parent_id.

ArcSwitch id : parent_id+'sw';
TriangleLed id: parent_id+'led';

style

The style of the ArcSwitch.

onchange

The onchange event of the ArcSwitch.

rotate

The rotate angle of the ArcSwitch.

color

The color of the TriangleLed.