

jSliderPro

v1.2

Reference Manual

Developed by: Mauro Pesci (mauropesci@infodreams.it)

Contents

Overview.....	3
Initialization.....	4
Dependencies.....	4
Creating new jSliderPro instance.....	4
Adding contents to the slider.....	5
Images adaptation.....	6
Main options.....	8
Buttons.....	10
Bullets.....	13
Working with callbacks.....	15
Transitions.....	18
Animations.....	21
Selectors.....	22
Effects.....	25
Examples.....	27
Captions.....	32

Overview

jSliderPro is a complete and highly customizable banner rotation and is equipped with a complex set of features that make it extremely flexible and adaptable. With little effort you can create unique, fully responsive - and compatible with all major browsers -, HTML5 slideshows for your websites.

First of all, the central strength of jSliderPro, lies in its ability to offer thousands of totally different and stunning animations. In fact, unlike other similar components that implement a limited set of atomic transitions, jSliderPro separates some basic concepts. In this way the user can mix them together in countless ways, limited only to him/her imagination.

Such concepts, which will be analyzed in detail below, are:

- **TRANSITION** - Represents a full step from one slide to the next
- **ANIMATION** - Is the unit of action that form a transition. You can have multiple animations in a single transition.
- **SELECTOR** - Indicates to an animation which tiles of a slide should be animated, at any given time.
- **EFFECT** - Constitutes the real action that is assigned to each tile of the slide.
- **SKIP** - Indicates a selector such tiles must skip within an animation. This is useful for creating complex effects.

The transitions are specified through simple JSON objects and this makes possible the on-the-fly manipulation of them.

In addition, jSliderPro, provides controls to allow users to navigate through the various slides. These controls consist of:

- **Navigation buttons** - That allows to move forward and backward.
- **Bullets** - That allow the selection of a specific slide.

Even the controls can be customized in many ways and you can also choose the appearance - by setting an image - or change their location in relatively to the main slider.

Many features will be implemented in future releases, such as support for text, its animation, other selectors, effects, and much more.

Initialization

The first action to be taken for the proper functioning of jSliderPro is to prepare the page by loading all dependencies and style sheets in addition to creating a new instance of its main object.

As a demonstration I inserted into the package containing the library code a sample HTML page that you can use as a quick reference.

Dependencies

First of all you must meet all of the dependencies on third-party JavaScript libraries. jSliderPro is based entirely on the services offered by jQuery and its plug-in for the functions of easing. So you will need to enter in the `<head>` section of the HTML page the following statements:

```
<script type="text/javascript" src="[jquery-lib-path]/jquery.min.js"></script>
<script type="text/javascript" src="[jquery-lib-path]/jquery.easing.[version].min.js"></script>
```

Where the tag `[jquery-lib-path]` is the path where the jQuery library is contained. In the attached package I included the latest version 1.10.2, but you can use a more recent one.

Then you must include in the page a reference to the jSliderPro's code in this way:

```
<script type="text/javascript" src="[jsliderpro-path]/js/jsliderpro.min.js"></script>
```

Now as above, the tag `[jsliderpro-path]` refers to the path where you placed the content of the package. jSliderPro also needs a minimum of css' statements, so you have to add a reference to the main stylesheet:

```
<link href="[jsliderpro-path]/css/jsliderpro.css" rel="stylesheet" type="text/css">
```

This is all that is necessary to prepare the page. Now we can proceed to create a new instance of the slider.

Creating new jSliderPro instance

To maximize the friendliness with search engines, the declaration of a slider jSliderPro is split into two parts, a part of HTML - through which you can specify the content and options related to the behavior of each one - and a part of JavaScript, which is responsible for managing the entire application logic and the options regarding the behavior of the component.

So let's add where needed, within the HTML page, the code that specifies the elements within the DOM:

```
<div class="jsliderpro-container">
  <div class="jsliderpro">
    ...
  </div>
</div>
```

Simple enough! Note you can have multiple instances within a single page, so the reason of a class for those tags

instead of an identifier.

Now that the HTML page contains the item that will represent the slider within the DOM, we must create a new javascript object. Normally you will put the code inside a jQuery start() event handler, at the bottom of the body of the HTML page, such the following:

```
<script type="text/javascript">
    $( document ).ready( function() {
        var jslp = new jsliderpro({ ... });
    });
</script>
```

The component would already be able to operate in this way, since it is equipped with a rich set of default settings. In order to meet different needs while creating you can pass a JSON object that contains the preferences to be set. This topic will be discussed later in this document.

Adding contents to the slider

Though as I said jSliderPro would already be able to function we have not yet specified any content to show! Now if we run the HTML page in a browser we would see just a completely empty rectangle.

Doing so is a very simple task, the content list is nothing more than a list enclosed in HTML tags `<ul>` `</ul>`. The component recognizes each element of the list identified with the class attribute 'jsliderpro-content'. In this way we can already place some code within the HTML statements for the component declaration:

```
<div class="jsliderpro-container">
    <div class="jsliderpro">
        <ul>
            <li class="jsliderpro-content">

            </li>
            <li class="jsliderpro-content">

            </li>
            <li class="jsliderpro-content">

            </li>
            ...
        </ul>
    </div>
</div>
```

Within each `<li>` `</li>` You can specify the items to be displayed in each of the slides. An image can be declared using the standard HTML tag `<img>`, so I'll show a little example:

```
<div class="jsliderpro-container">
    <div class="jsliderpro">
        <ul>
```

```

        <li class="jsliderpro-content">
            
        </li>
        <li class="jsliderpro-content">
            
        </li>
        <li class="jsliderpro-content">
            
        </li>
        ...
    </ul>
</div>
</div>

```

Fairly straightforward, but what is the meaning of the strange parameters 'data-xxxx'? Well, web professionals know that it is always necessary to process the images to fit the size you need. To achieve this objective it is necessary to use special external software for graphics and photo editing and as well as knowing how to perform resizing and cropping, you must create a new file and takes up disk space and/or download time.

So jSliderPro provides a small tool to perform these operations on the fly, without the need of having to create other files.

Note 1: At the time jSliderPro v1.0 supports only the images as animated element of a slide, but future versions will also add other types.

Note 2: jSliderPro processes tags at the time of its initialization preventing them from being loaded by browsers. Doing so it can minimize load times and the occupation of memory.

Note 3: jSliderPro assumes that a content contains only a single tag, other images inside a single will be ignored.

Images adaptation

When you choose an image to be displayed by the slider is unlikely that its dimensions coincide perfectly with those of the latter.

Usually the operation which is accomplished in these cases is the image resize, but also in this case it's possible that the proportions between width and height don't coincide, so it becomes necessary to perform some clipping operations.

jSliderPro carries everything for you automatically, the only task that the developer has to perform is to pass the information necessary to obtain the desired result through 'data' parameters, directly into the tag. For this

purpose, these parameters are available:

'fit' parameter

Can assume one from these values: **'width'** or **'height'**. It tells the component to resize 'one' of the two dimensions of the image so that it coincides with 'one' of the slider. The other dimension will be scaled by the same ratio used for the one that was chosen.

Thus, for example, if the slider has dimensions of 960x600 pixels and the image size is 1920x1080 pixels, specifying **'width'** as the parameter value, the final image will have a size of $(960/1920 = 0.5 \times 1080 = 540)$ 960x540 pixels. Conversely, specifying **'height'** as the parameter value, the image will become $(600/1080 \times 1920 = 0.56 = 1067)$ 1067x600 pixels.

You'll surely notice that in both cases the image don't coincides perfectly with the size of the slider (960x600 pixels). In fact, in the first case the height is lower (resulting in an area not covered within the slide) while in the second the width is even greater so the final image can not be correctly displayed.

If your goal is to maintain the correct proportions the only option you have is to select a part of the image to display properly or place it inside the slide (if the size is less) in an optimal way. For this purpose you need to use the parameters **'valign'** and **'halign'** explained later.

'vscale' parameter

Instead of automatically scale an image you can control the resizing by specifying an arbitrary value. The parameter **'vscale'** acts on the height and in this way passing the value of 1 will leave unchanged the image height, a value between 1 and 0 will reduce the size, while a value greater than 1 will result in an enlargement.

Note that this parameter is applied after the **'fit'**, which ensures the ability to resize further in conjunction with its result.

'hscale' parameter

The same rules explained for the parameter **'vscale'** with the difference that in this case the changes are made to the width.

'valign' parameter

Can assume one from these values: **'top'**, **'middle'** or **'bottom'**. As mentioned previously, it may happen that you have the need to align the image within the slide to display only a certain part of it. This parameter allows you to choose a position in relation to the height of an image.

Setting the value to **'top'** the top edge of the image will be positioned to coincide with the top of the slide. Similarly, the value **'bottom'** will position the lower edge of the image with the lower one of the slide.

Instead, the value of **'middle'** will ensure that the center of the height of both slide and image coincide. In this way if the image is larger than the slide it will show only the portion at the image's center, while if a smaller image will be displayed in the center of the slide.

'halign' parameter

Can assume one from these values: **'left'**, **'center'** or **'right'**. The behavior of this parameter is analogous to that explained for the previous with the only difference that is applied to the width.

Main options

It's time to show how you can customize the behavior and appearance of the slider. First of all it is important to know how to change the size of the slider.

Setting the size

In order to not complicate the code reimplementing functionality easily obtainable through the tools provided by web browsers is not possible, at the time, set the size through direct parameters to the main object. We must instead open the style sheet contained in the file "css/jsliderpro.css" and change some properties.

Set properly the properties 'width' and/or 'height' inside the classes '.jsliderpro-container', '.jsliderpro', '.jsliderpro-container-fixed', '.jsliderpro-fixed'. jSliderPro can operate in two modes: **responsive** (that is able to automatically adapt the dimensions according to the window size, through the appropriate media queries) and **fixed**, so the first pair of classes refers to the maximum size of the slider in **responsive** mode, while the other two are for the fixed mode.

Below I have an excerpt of the code to be more easily seen:

```
.jsliderpro-container {  
    background-color: black;  
    width: 960px;  
    position: relative;  
    margin-left: auto;  
    margin-right: auto;  
    overflow: hidden;  
}
```

```
.jsliderpro {  
    width: 960px;  
    height: 500px;  
    position: relative;  
    overflow: hidden;  
    background-color: black;  
}
```

```
.jsliderpro-container-fixed {  
    padding: 0px;  
    background-color: black;  
    width: 960px;  
    position: relative;  
    margin-left: auto;  
    margin-right: auto;  
}
```

```
.jsliderpro-fixed {  
    width: 960px;  
    height: 500px;  
    position: relative;  
    overflow: hidden;
```

```
}
```

In particular, with regard to the responsive mode you can change the media queries at your like, adding or changing the values. The component is able to detect changes in size of the window and automatically adapt all settings related to the measures.

In this way, the settings for the positioning and size of images, buttons, bullets and so on will be appropriately scaled to preserve the original appearance.

Initialization parameters

As seen above, at the time of creation, the object jSliderPro accepts a JSON object as an argument. Such object can contain the following properties:

'target'

It allows you to associate the created object to a particular element of the DOM. This parameter is useful if you intend to have multiple sliders on the same page. It is normally omitted and its default value coincides with the first slider found on the page. Note that the passed object must be a valid jQuery object.

'delay'

Sets the time duration, in milliseconds, of a single slide from a transition to another one. The default value is 1000 milliseconds.

'interactive'

A simple boolean that tells the component whether to display or not the navigation controls: buttons and bullets. By default it is set to 'true'.

'buttons' and 'bullets'

Both are JSON objects and allow you to configure the appearance and position of the buttons and bullets in a number of ways and for this reason each of the two parameters will be discussed in the appropriate sections.

Here's an example of initialization through the parameters described above:

```
<script type="text/javascript">
    $( document ).ready( function() {
        var jslp = new jsliderpro({
            target: $(".jsliderpro"),    // Not necessary if you have only one
                                       // slider in the page
            delay: 2000,
            interactive: false
        });
    });
</script>
```

Buttons

Normally, in a website you want the user to have the ability to navigate within the various slide forward or backward, and not just passively observe a series of images that flows independently. As in a web browser you want to have control buttons relating to the mentioned actions: moving forward and backward.

As mentioned parameters related to the buttons are specified through JSON objects, so let us look closely at their structure.

Buttons images

jSliderPro allows you to choose different images to be displayed as navigation buttons at the time of the creation of a new slider object. When you do this, you must specify an image for each state that can take on each of the two buttons: **'normal'**, **'hover'** and **'pressed'**.

Button images are stored within the parameters of **'left'** and **'right'** of the main parameter **'buttons'** and can have the following properties:

'normal'

Is a valid URL that points to the image that will be displayed when the button is in the initial state.

'hover'

Is a valid URL that points to the image that will be displayed when you place the mouse pointer over the button.

'pressed'

Is the image that appears when you click the mouse button on it.

Setting appropriately the parameters **'left'** and **'right'** you can then set an image for each of the two buttons. For your convenience jSliderPro uses a default for those values, so you can be able to show buttons even without placing any value for the parameters above.

The object-parameter **'buttons'** also supports the following parameters that allow you to work on the size of them:

'width'

To set a specific width for the buttons. It's possible even to express the values through classical css units: em, percent, pixels and so on.

'height'

To set a specific height for the buttons. It's possible even to express the values through classical css units: em, percent, pixels and so on.

'autoresize'

A boolean value. This option is useful with the responsive mode as it allows you to enable / disable the automatic resizing of the buttons according to the changed size of the slider. The default value is **'true'**.

'autohide'

Setting to **'true'** this parameter, you can make sure that the buttons disappear when the mouse pointer is not inside the slider's area.

The default value is **'false'**, which makes the buttons to remain visible.

'hidespeed'

When 'autohide' is on, you can set the show/hide animation duration, in milliseconds.

Follows an example that uses all parameters described:

```
<script type="text/javascript">
    $( document ).ready( function() {
        var jslp = new jsliderpro({

            ... other initialization parameters ...

            buttons: {
                left: {
                    normal: 'images/btn_left_normal.png',
                    hover: 'images/btn_left_hover.png',
                    pressed: 'images/btn_left_pressed.png',
                },
                right: {
                    normal: 'images/btn_right_normal.png',
                    hover: 'images/btn_right_hover.png',
                    pressed: 'images/btn_right_pressed.png',
                },
                width: '50%',
                height: '50%',
                autoresize: true,
                autohide: false
            }

            ... other initialization parameters ...

        });
    });
</script>
```

Buttons location

Surely the appearance is not the only thing you'll want to customize, depending on the style of the site you may have to set a location of the buttons different from the default one. You can place them everywhere and arrange them in ways completely different from each other.

In addition to parameters 'left' and 'right' the object-parameter 'buttons' can have the following:

'orientation'

Accepts these values: '**horizontal**' and '**vertical**'. This parameter indicates the way in which the buttons are positioned relatively to each one. The value '**horizontal**' means that the two buttons must be placed horizontally on the same line. Setting the value '**vertical**' they will be positioned vertically one above the other.

The default value is '**horizontal**'.

'arrangement'

Can assume the values: '**opposed**' or '**joined**'. To simplify things, this parameter allows you to indicate whether the two buttons, as well as to be vertically or horizontally on the same line, they must also be placed on opposite sides of the slider's rectangle. Setting '**joined**' the buttons will be placed adjacent according with the chosen '**orientation**'.

Note that the value of this parameter influences the invalidation of one of the two parameters '**vAlign**' or '**hAlign**'. Infact, the value '**opposed**' ensure that jSliderPro will consider only the value of the parameter '**vAlign**' if the orientation is '**horizontal**', and '**hAlign**' otherwise.

The default value is '**opposed**'.

'vAlign'

Accepts these values: '**top**', '**middle**', '**bottom**', '**above**', '**below**' and places the buttons respectively at the top, middle and bottom of the slider rectangle, while '**above**' and '**below**' allow to place them outside.

The default value is '**middle**'.

'hAlign'

Accepts these values: '**left**', '**center**', '**right**', '**out-left**', '**out-right**' and places the buttons respectively at the left, center and right of the slider rectangle, while '**out-left**' and '**out-right**' allow to place them outside.

The default value is '**center**'.

'separation'

If you have set an arrangement of type '**joined**' this parameter allows you to adjust the spacing between the two buttons. Obviously also the alignment affect the final positioning of them, in fact if you have aligned the buttons at the edges, it will be moved only the one that is the innermost of the two. If you choose '**middle**' or '**center**' alignment, the two buttons will be moved equally one away from the other.

The default value is 0.

'hMargin'

When you set the alignment using the parameter '**hAlign**' in the vertical edges of the image of the buttons coincide with the vertical slider. Through this parameter it is possible to adjust the distance between them.

The default value is 0.

'vMargin'

When you set the alignment using the parameter 'vAlign' the horizontal edges of buttons' image coincide with the one of the slider. Through this parameter it is possible to adjust the distance between them.

The default value is 0.

Bullets

The bullets represent a placeholder for each slide in the presentation and are visible at the same time all together. They have a double function: to show you which slide is currently displayed and to allow you to jump directly to a specific slide.

As mentioned parameters related to the buttons are specified through JSON objects, so let us look closely at their structure.

Bullets images

jSliderPro allows you to choose different images to be displayed as bullets at the time of the creation of a new slider object. When you do this, you must specify an image for each state that can take a bullet: **'normal'**, **'hover'** and **'selected'**.

Bullets images are stored in the object **'bullets'** the following properties:

'normal'

Is a valid URL that points to the image that will be displayed when the bullet is in the initial state.

'hover'

Is a valid URL that points to the image that will be displayed when you place the mouse pointer over the bullet.

'selected'

Is the image that appears when you click the mouse button on it and when the slide assigned to the bullet become visible.

For your convenience jSliderPro uses a default for those values, so you can be able to show bullets even without placing any value for the parameters above.

The object-parameter **'bullets'** also supports the following parameters that allow you to work on the size of them:

'width'

To set a specific width for the bullets. It's possible even to express the values through classical css units: em, percent, pixels and so on.

'height'

To set a specific height for the bullets. It's possible even to express the values through classical css units: em, percent, pixels and so on.

'autoresize'

A boolean value. This option is useful with the responsive mode as it allows you to enable / disable the automatic resizing of the bullets according to the changed size of the slider. The default value is 'true'.

'autohide'

While jSliderPro is performing a transition from one slide to another, all the controls are disabled. Setting to **'true'** this parameter, you can also make sure that the bullets disappear during the time of a transition.

The default value is **'false'**.

Follows an example that uses all parameters described:

```
<script type="text/javascript">
    $( document ).ready( function() {
        var jslp = new jsliderpro({

            ... other initialization parameters ...

            bullets: {
                normal: 'images/blt_normal.png',
                hover: 'images/blt_hover.png',
                selected: 'images/blt_selected.png',
                width: '50%',
                height: '50%',
                autoresize: true,
                autohide: false
            }

            ... other initialization parameters ...

        });
    });
</script>
```

Bullets location

Surely the appearance is not the only thing you'll want to customize, depending on the style of the site you may have to set a location of the bullets different from the default one. You can place them everywhere and arrange them in ways completely different from each other.

In addition the **'bullets'** object can have the following properties:

'orientation'

Accepts these values: **'horizontal'** and **'vertical'**. This parameter indicates the way in which the bullets are positioned relatively to each one. The value **'horizontal'** means that the bullets must be placed horizontally on the same line. Setting the value **'vertical'** they will be positioned vertically one above the other.

The default value is **'horizontal'**.

'vAlign'

Accepts these values: **'top'**, **'middle'**, **'bottom'**, **'above'**, **'below'** and places the buttons respectively at the top, middle and bottom of the slider rectangle, while **'above'** and **'below'** allow to place them outside.

The default value is **'middle'**.

'hAlign'

Accepts these values: **'left'**, **'center'**, **'right'**, **'out-left'**, **'out-right'** and places the buttons respectively at the left, center and right of the slider rectangle, while **'out-left'** and **'out-right'** allow to place them outside.

The default value is **'center'**.

'separation'

If you have set an arrangement of type **'joined'** this parameter allows you to adjust the spacing between the bullets. Obviously also the alignment affect the final positioning of them, in fact if you have aligned the buttons at the edges, it will be moved only the ones that are the innermost of the set. If you choose **'middle'** or **'center'** alignment, the bullets will be moved equally one away from the others.

The default value is 0.

'hMargin'

When you set the alignment using the parameter **'hAlign'** in the vertical edges of the image of the bullets coincide with the vertical slider. Through this parameter it is possible to adjust the distance between them.

The default value is 0.

'vMargin'

When you set the alignment using the parameter **'vAlign'** the horizontal edges of bullets' image coincide with the one of the slider. Through this parameter it is possible to adjust the distance between them.

The default value is 0.

Working with callbacks

In some situations you may have the need to interact programmatically with the slider not only to issue commands, but also to detect the start or completion of certain actions.

For example, you might want to wait for the end of a transition to play a sound, display a text describing a particular slide (since this version does not support the caption yet), or perform more complex operations such as retrieving data from the server.

jSliderPro offers this service through the mechanism of registration of listeners to events. It's all very simple, just call some methods of the main object and pass a custom function to be performed when the particular event is raised.

To register a listener you need to call the method:

```
function addEventListener(event, listener);
```

while you can remove a listener trough:

```
function removeEventListener(event, listener);
```

where the parameter '**listener**' is a function that accepts a single parameter and '**event**' can assume one of these values:

'EVENT_SLIDER_READY'

This event is raised just after the slider object has been created and has successfully completed all the initialization operations. No parameters will be passed.

'EVENT_TRANSITION_START'

This event is raised when a particular transition starts its animations. No parameters will be passed.

'EVENT_TRANSITION_END'

This event is raised when a particular transition has completed its animations. No parameters will be passed.

'EVENT_ANIMATION_START'

This event is raised when a single animation starts. As parameter will pass the JSON object representing a particular animation.

'EVENT_ANIMATION_END'

This event is raised when a single animation ends. As parameter will pass the JSON object representing a particular animation.

'EVENT_EFFECT_START'

This event is raised when an effect for a particular tile inside a slide starts. As parameter will pass a JSON object containing two properties:

'effect': Containing the name of the executed effect.

'elementIndex' : Containing the index of the animated tile.

'EVENT_EFFECT_END'

This event is raised when an effect for a particular tile inside a slide ends. As parameter will pass a JSON object containing two properties:

'effect': Containing the name of the executed effect.

'elementIndex' : Containing the index of the animated tile.

'EVENT_NEXT_CLICKED'

This event is raised when user clicks on the '**next**' navigation button. No parameters will be passed.

'EVENT_PREVIOUS_CLICKED'

This event is raised when user clicks on the '**previous**' navigation button. No parameters will be passed.

'EVENT_INDEX_CHANGED'

This event is raised when user clicks on a bullet. As parameter will pass an integer representing the index of the clicked bullet.

'EVENT_SLIDER_RESIZED'

This event is raised when the slider is resized in response to one of the media queries.

'EVENT_MOUSE_OVER'

This event is raised when the mouse pointer enter inside the slider's area.

'EVENT_MOUSE_OUT'

This event is raised when the mouse pointer exits outside the slider's area.

You may attach as many listeners you want to a single event. Here's an example of use:

```
// . . . jsliderpro initialization . . .
var jslp = new jsliderpro({...});

// Listener function
var alistener = function(parm) {
    if (parm === 0) {
        // Do your stuff
    }
};

// Add the listener
jslp.addEventListener(jslp.EVENT_INDEX_CHANGED, alistener);

// Remove the listener
jslp.removeEventListener(jslp.EVENT_INDEX_CHANGED, alistener);
```

Remember that you should include the name of the main object's instance to access its methods and constants.

Transitions

The transitions are the central component of the sliders, allowing you to change from one image to the next through eye-catching effects.

Some time ago, these results could be obtained only through the use of external technologies such as Adobe Flash, but the advent of HTML5 (through the functions of painting on canvas) in conjunction with CSS has made it possible to create very simple and lightweight Javascript software.

Since then many solutions of this type was created, each of which offers a certain set of pre-built transitions, because despite all you still need to write a lot of code to create one.

jSliderPro tries to overcome this limitation by implementing a very complete system to generate thousands of various transition effects without efforts.

As said in the overview section of this document, this was achieved through the identification and separation of the fundamental concepts that constitute a transition. The current version of jSliderPro supports these concepts:

TRANSITION - Represents a full step from one slide to the next

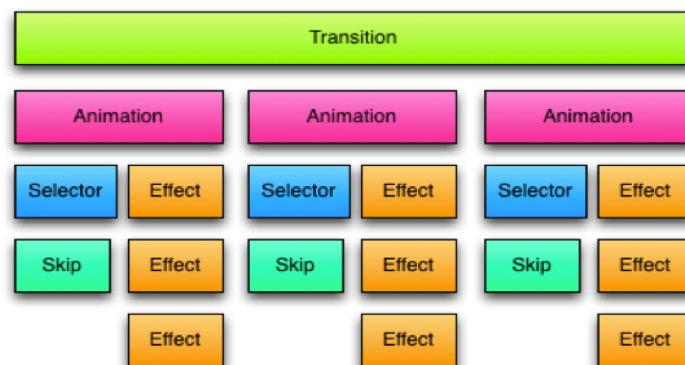
ANIMATION - Is the unit of action that form a transition. You can have multiple animations in a single transition.

SELECTOR - Indicates to an animation which tiles of a slide should be animated, at any given time.

EFFECT - Constitutes the real action that is assigned to each tile of the slide.

SKIP - Indicates a selector such tiles must skip within an animation. This is useful for creating complex effects.

The relationships between them can be summarized with the following figure.



As you can see a single transition can be formed by one or more animations. An animation is basically the application of a graphic effect to an image or a part of it, but which part the animation must involve is decided by the selector. For the moment, please accept this thin definition because I will devote special sections to go into detail.

Since the transitions usually involve at least two images, jSliderPro associates a role to each of the two. At any given time, in fact, we have a currently displayed image and one that is about to enter into the visible area.

Not only that, jSliderPro also distinguishes three modes of transition 'in' or 'out' and 'mixed'. Let's look at each in detail:

'in'

In this mode, the image that is animated the next. The current image remains stationary and has a lower z-index in order to be placed 'behind' the incoming one.

'out'

In this mode, the image that is animated the current. The next image remains stationary and has a lower z-index in order to be placed 'behind' the exiting one.

'mixed'

This mode is the most interesting of the previous in that it allows the creation of very complex transitions. Here both images, the current and the next, are at the same level and can be animated simultaneously. In the next section you can learn how to animate them.

Another aspect to consider is that, until now, we have spoken about images, which are specified by a file name. It follows that by applying an animation, the entire image will be involved. You do not want to always see, for example, a big image exiting from the visible area. This would give the ability to create only very few types of transitions ...

A very common technique to avoid this, is to divide a single image into many small subparts, this technique is called **'tiling'**. Another difference between jSliderPro from other similar software is constituted by its ability to tile an image on the fly. In this way you can decide for each transition the number of tiles in which the original image have to be converted. To do so you can specify the properties: **'rows'** and **'columns'**, and this converts the selected image into an array of rows x columns tiles. By default they both have a value 1 meaning that the entire image is animated.

There are other parameters associated with a whole transition and are the following:

'easing'

As the entire component is based on the jQuery library, we can use its very useful feature to increase the dynamism of our animations. As in all jQuery animations we can alter the trend strictly linear of movements (or value changes) through the easing, that is the series of rebounds, braking, acceleration and so on available through some plug-in.

The change is effective on all the animation of a transition and for the moment it is not yet possible to specify more than one. When specified this parameter, accepts a string containing the name of a particular ease effect.

The default value is **'linear'**.

'duration'

Is an integer numeric value that express milliseconds. With this parameter you can decide the total duration of the transition. In particular, this value is used to calculate the activity time of the selectors in each animation. Please refer to the section on selectors.

The default value is **'1000'**.

jSliderPro wants every transition has been previously registered, in order to be used. You can manage your transitions using these methods:

```
function registerTransition(name, obj);
```

This method accepts a string as a name to associate to the transition. The name is the value used principally by contents to declare the transition to be applied on each of them. The other argument consists in a JSON object containing all parameters described above and others that will be presented soon.

```
function unregisterTransition(name);
```

By this method you can unregister a transition from the slider.

```
function isTransitionRegistered(name);
```

By this method you can check whether a particular name is associated to a transition.

```
function getTransition(name);
```

Rarely you could need to retrieve the transition object registered by the slider. This method returns the same JSON object passed during the registration phase.

So to register a transition for mode 'in' with a duration of 2 seconds you can write this code:

```
// . . . jsliderpro initialization . . .
var jslp = new jsliderpro({...});

jslp.registerTransition('a_transition', {
    mode: 'in',
    duration: 2000,
    rows: 5,
    columns: 10
});
```

After the slider knows your transition 'a_transition' you can associate it to your contents declared into the HTML page:

```
... other jsliderpro declarations ...
<li class="jsliderpro-content" data-transition="a_transition">
    
</li>
```

the data parameter 'data-transition' indicates which transition effect you want to apply when transiting from this content to the next. Someone could argue that to have a lightly different transition you must to duplicate your code and change its settings.

jSliderPro helps you through the mechanism of dynamic extension of objects. In this way you can simply tell to the component what properties to change and that's all! Through data parameters you can alter (or extend) the original transition object without creating a new one:

```
... other jsliderpro declarations ...
```

```

<li class="jsliderpro-content"
    data-transition="a_transition"
    data-transitionparms="{ duration : 3000 }">
    
</li>

```

So we changed the parameter '**duration**' of the transition '**a_transition**'.

Animations

Animations represents the unit that contain the specifications for a particular phase of a transition. You can have multiple animations running at the same time or you can create sequences of them.

To declare an animation inside a transition you must include the property '**animation**' in the transition declaration object. Animations objects can have these properties:

'layer'

Can take the values: '**above**' and '**below**'. The property is considered only when a transition is in '**mixed**' mode, and it is useful to select one of the two images to which you want to apply an animation.

The default value is '**above**'.

'startDelay'

Is an integer number representing the milliseconds of delay from the transition start time. Use it if you don't want that your animation start simultaneously or for creating an animation sequence.

The default value is 0.

'selector'

This is another JSON object representing the tool discussed in the proper section below.

'effect'

This is another JSON object representing the tool discussed in the proper section below. It can also be an array of effect objects if your need is to merge multiple effects in one.

So we can proceed to expand our previous example, adding the lastly discussed properties:

```

// . . . jsliderpro initialization . . .
var jslp = new jsliderpro({...});

jslp.registerTransition('a_transition', {
    mode: 'in',
    duration: 2000,
    rows: 5,

```

```

        columns: 10,
        animation: {
            startDelay: 1000,
            selector: { . . . },
            effect: { . . . }
        }
    });

```

Since, as previously said, you can have multiple animations within a single transition it's even possible to pass an array of them through the parameter:

```

// . . . jsliderpro initialization . . .
var jslp = new jsliderpro({...});

jslp.registerTransition('a_transition', {
    mode: 'in',
    duration: 2000,
    rows: 5,
    columns: 10,
    animation: [{
        startDelay: 1000,
        selector: { . . . },
        effect: { . . . }
    }, {
        startDelay: 2000,
        selector: { . . . },
        effect: { . . . }
    }]
});

```

Selectors

It's time to clarify the topic behind this mysterious name, so many time appointed. When I talked about the mechanism that converts images into a tile-set probably someone become a bit confused trying to imagine how jSliderPro deals with each of them.

When an animation begins it invokes the selector that is an object able to tell which tiles needs to be animated at any given time. With that information jSliderPro can execute the selected effects on each tile according with their location in the timeline. In this manner everything is arranged by sequences and animations gain a different aspect every time.

The JSON object takes a '**name**' parameter through which you can set a selector function:

```

selector: {
    name: 'horizontalScan'
}

```

Every other parameter is specific to each selector and is placed directly inside this object.

At the moment it's not possible to implement new selectors, but you can choose from a prebuilt pool. Following each of them will be discussed.

'horizontalScan' selector

The tiles are selected horizontally in sequence one by one starting from the leftmost one on the first row. When the end of a row is reached the selection continues from the first tile of the next row.

You can flip the selection order of the tiles by setting the **'flip'** parameter that accepts one of these values: **'no'**, **'horizontal'**, **'vertical'**, **'both'**. If you set an horizontal flip the selection will start from the rightmost on the first row and backwards. The vertical option inverts the order of the rows selected, starting from the last row ending with the first. Obviously the option **'both'** inverts the horizontal direction as well as the vertical. So you can make the selector travel from all directions.

The default value is **'no'**.

'verticalScan' selector

The tiles are selected vertically in sequence one by one starting from the topmost one on the first column. When the end of a column is reached the selection continues from the first tile of the next column.

You can flip the selection order of the tiles by setting the **'flip'** parameter that accepts one of these values: **'no'**, **'horizontal'**, **'vertical'**, **'both'**. If you set an horizontal flip the selection will start from the rightmost on the last column and backwards. The vertical option inverts the order of the rows selected, starting from the last row ending with the first. Obviously the option **'both'** inverts the horizontal direction as well as the vertical. So you can make the selector travel from all directions.

The default value is **'no'**.

'diagonalScan' selector

Sum the behavior of the two previous selectors by moving through all diagonals starting from left to right. This selector accepts all values of the parameter flip.

'random' selector

This function chooses the tiles randomly one by one from the first to the last. No tile is selected more than once and has no support for the **'flip'** option.

'rows' selector

Instead of each tile one at a time, in this case an entire row is selected at a time starting from the first at the top. This selector supports the parameter to flip for the only option **'vertical'** because the other values have no meaning.

'columns' selector

In this case an entire column is selected at a time starting from the first at the left. This selector supports the parameter to flip for the only option **'horizontal'** because the other values have no meaning.

'diagonals' selector

Similarly to the above, this selector chooses a whole diagonal at a time. It also supports all values for the **'flip'** option.

'rectangles' selector

It can create unique animations in that it determines the selection of whole rectangles one inside the other. It does not support at all the 'flip' option, but you can reverse the order of the selected rectangles by setting to true the value of 'inverse'. In this way the selection start from the smallest rectangle in the center of the slide up to the largest.

'horizontalWave' selector

Performs a scan of the tiles one by one, similarly to what happens for the 'horizontalScan' selector with the difference that when the last tile of a line is encountered, rather than start over again from the one to the left of the next row, it reverses the direction and come back. The same thing happens when it touches the first tile of row generating a back and forth movement.

This selector accepts all values of the parameter flip.

'verticalWave' selector

Performs a scan of the tiles one by one, similarly to what happens for the 'verticalScan' selector with the difference that when the last tile of a column is encountered, rather than start over again from the one to the top of the next column, it reverses the direction and come back. The same thing happens when it touches the first tile of column generating a back and forth movement.

This selector accepts all values of the parameter flip.

jSliderPro accepts for selectors also their name as the only parameter instead of a JSON object. In this way it selects all default parameters. Here's an example:

```
animation: {  
    startDelay: 1000,  
    selector: 'verticalScan',  
    effect: { . . . }  
}
```

Skips

Each selector is designed to select all the tiles of a slide independently. To be able to create multiple animation effects you need to start multiple selectors. In doing so, however, would be animated several times the same tiles. How to fix it? Simple! jSliderPro supports the functionality of skips. It is an aid to the selectors able to indicate which of the tiles selected must be left out, so not animated.

An example will help to more easily understand the concept. You have chosen the '**columns**' selector to animate the tiles in columns. Do you want to do, however, so that the even columns are animated in one way, while in another the odd ones.

Obviously you need to create two separate animations with the same selector, but in this case you must also specify the skip option for everyone, in this way:

```
animation: [{  
    startDelay: 1000,  
    selector: {  
        name: 'columns',  
        skip: 'odd'  
    }  
},
```

```

    },
    effect: { . . . }
  }, {
    selector: {
      name: 'columns',
      skip: 'even'
    },
    effect: { . . . }
  }
]]

```

In the first case, the parameter 'skip: odd' will skip all the odd columns leaving only the even to the first animation, vice versa with the second animation. In case you have chosen a type selector 'scan' the skip would be applied on the individual not on the tile rows or columns.

JSliderPro currently only supports these two skip 'odd' and 'even', but in the next versions will be added many others.

Effects

Now that we know how to create an animation and set the selectors all we can do is apply some effect to the tiles of a slide. Please note in advance that with the term of effect I intend to mean a very large class of graphical operations: scrolling, zoom, fade and so on.

The effects are in jSliderPro operations that are applied within an animation to all the selected items without distinction. The task of the selector is to determine 'when' the given effect must be applied to a certain tile.

The syntax to activate an effect within an animation is similar to what seen for the selectors. The object that contains all the parameters must have at least one property called 'name' to indicate the particular effect. All other properties are interpreted as specific parameters of each effect. So, for example we can write:

```

{
  startDelay: 1000,
  selector: 'verticalScan',
  effect: {
    name: 'fade'
  }
}

```

to activate a fade effect for the animation.

The supported effects are these:

'change' effect

This is a null-effect, used to obtain the instantaneous disappearance of a tile. It has no further parameters.

'fade' effect

Is perhaps the most widely used effect because it is very charming and adaptable to any situation. It is a smooth transition for a tile from its fully opaque state to the completely transparent.

Note that some effects are influenced by the transition mode **'in'** and **'out'** as in the first case the tile of the next image pass by a state of complete transparency to an opaque (since it is the incoming image the one to be displayed), conversely in **'out'** mode the tiles of current image will switch from the opaque to the transparent, showing details of the next image below. It has no further parameters.

'scroll' effect

This effect simply moves the tile from the initial position to outside of the visible area (in mode **'out'**) and vice versa (in mode **'in'**). You can choose the direction of movement by setting the **'direction'** with one of these values: **'up'**, **'down'**, **'left'**, **'right'**, **'up-left'**, **'up-right'**, **'down-left'**, **'down-right'**.

The default value for **'direction'** is **'right'**.

'clip' effect

This is very simple to understand: in **'out'** mode it carves out a tile progressively until it disappears. Conversely in mode **'in'** brings up progressively a tile. The parameter **'direction'** can take many values, some of which are discussed separately: **'up'**, **'down'**, **'left'**, **'right'**, **'horizontal'**, **'vertical'**, **'upper-left'**, **'lower-left'**, **'upper-right'**, **'lower-right'**, **'left-center'**, **'right-center'**, **'upper-center'**, **'lower-center'**, **'all'**.

The first four options **'up'**, **'down'**, **'left'** and **'right'** make the cut line by line or column by column, as appropriate. All others are a mix of these.

The following four **'upper-xxxx'** and **'lower-xxxx'** merge the vertical cut with the horizontal so that the cut appears to move towards one of the four corners of the tile.

Then we find other four options that end with **'xxxx-center'**. In this case were mixed three cuts, leaving time to time one out of the mix. The resulting effect is that the cut appears to move towards the center of the various edges of the tile.

Finally we have the option **'all'** that mixes all four fundamental cuts creating an effect that shows the tile gradually come down from all sides towards its center.

The default value for **'direction'** is **'up'**.

'zoom' effect

Is very simple to explain this effect, depending on the transition mode, it enlarges or reduces a tile.

Supports the parameter **'direction'** which can take the following values: **'away'** and **'close'**, which default value is **'away'**. According therefore to the combination of transition mode and parameter **'direction'** can be determined by four distinct movements. In the **'out'** mode and the parameter **'direction'** to **'away'** the tile will be reduced from the initial size to the minimum and eventually disappear, with **'close'** will be enlarged to their maximum.

In the **'on'** mode and the parameter **'direction'** to **'close'** the tile will be enlarged from the minimum size to reach their normal size, with **'away'**, starting from the maximum size will be reduced to their normal size.

Why talk about minimum and maximum dimensions? Because by the parameters **'min'** and **'max'**, that accept a numeric integer, we can define these values. By default the **'min'** option is Set to 0, resulting in a complete disappearance of tile that are being reduced. L'opzione **'max'** per default è impostata a 1000.

The effects of zoom usually should be used in combination with another effect, especially the fade, because they can generate unpleasant results. In fact, if you try a zoom with option **'close'** in **'out'** mode you would see the tile grow up to a certain point, occupying the image below, and then disappear suddenly when they reach their maximum value.

Other options

Effects support a further option:

'duration'

As in other cases, it is an integer value, which establishes the duration of the effect, in milliseconds, applied to a single tile. By default the value is set to 1000.

In this case it is important to note that the overall duration of a transition is also influenced by the duration of the effects, as well as the number of animations and their placement in time. When run an animation this does not finish until ends the last animated tile effect.

Examples

At this point it is necessary to have some explanatory example to better illustrate the entire process of transitions creation. These examples were taken from the file '**jsliderpro-demo.js**' contained in the source package. There you will find hundreds of other examples like this and editing them you could see the end result.

Example 1 – Simple fade

Now we will create a transition of the whole image with a fade effect.

```
jslp.registerTransition('a_transition', {  
    animation: {  
        effect: "fade"  
    }  
});
```

Quite simple! Using all the default settings can simply create an animation that contains only an indication of the effect fade. If you want to vary the duration of the fade, we may change the default value of the '**duration**' parameter, but in this case we need an object for the effect:

```
jslp.registerTransition('a_transition', {  
    animation: {  
        effect: {  
            duration: 3000,  
            name: "fade"  
        }  
    }  
});
```

But why is animated the whole picture? Because the parameters '**row**' and '**column**' of the transition have the default value of 1... And why there is not even the parameter '**selector**'? Because even the selector has a default value that points to '**horizontalScan**' (as well as the effect that has default in '**change**'), but in this case having only one tile for the entire image, any selector would have the same effect!

Example 2 – Simple left scroll

In this case we want that the image of the next slide enters the visual field with a scroll effect from right to left.

```
jslp.registerTransition('a_transition', {
    animation: {
        mode: 'in',
        effect: {
            duration: 2000,
            name: "scroll",
            direction: "right"
        }
    }
});
```

First you set the transition mode 'in' to make sure that it is the next slide to be animated. After that it was enough to choose a 'scroll' effect by setting its parameter 'direction' to right.

Example 3 – Zoom out with fade

I decided to present this example because I think the following is the only method to get a nice zoom effect.

```
jslp.registerTransition('a_transition', {
    easing: "easeOutCubic",
    animation: {
        effect: [{
            duration: 2000,
            name: "zoom",
            direction: "close"
        }, {
            duration: 2000,
            name: "fade"
        }]
    }
});
```

Note two important things. The first is that we have created an array of 'effect' objects to associate with the same animation two separate effects. The second is that in order to align the duration of both we had to add a parameter 'duration' with the same value of 2000 milliseconds.

Example 4 – Rows fade from bottom

Here I introduce a simple trick to lighten the workload when dealing with entire rows (or columns). Simply just set the number of rows you want while to 1 that of the columns. In this way we can obtain good performance even with a high number of lines (50, 100 or more ...).

```
jslp.registerTransition('a_transition', {
    rows: 20,
    columns: 1,
    animation: {
        selector: {
```

```

        name: "rows",
        direction: "up"
    },
    effect: "fade"
}
});

```

Example 5 – Random falling tiles

A really funny transition, looks like a wall that crumbles piece by piece.

```

jslp.registerTransition('a_transition', {
    rows: 5,
    columns: 10,
    duration: 3000,
    animation: {
        selector: "random",
        effect: [{
            name: "scroll",
            direction: "down"
        }, "fade"];
    }
});

```

Note also the syntax used to set the effect 'fade'.

Example 6 – Alternate columns scroll up and down

Let's begin to realize more complex transitions through the use of multiple animations.

```

jslp.registerTransition('a_transition', {
    easing: "easeOutBounce",
    rows: 1,
    columns: 10,
    duration: 2000,
    animation: [{
        selector: {
            name: "columns",
            skip: "odd"
        },
        effect: {
            name: "scroll",
            direction: "up"
        }
    }, {
        selector: {
            name: "columns",

```

```

        skip: "even"
    },
    effect: {
        name: "scroll",
        direction: "down"
    }
}
    ]]
});

```

Example 7 – Mixed mode alternate scrolling rows

With this example I conclude roundup, employing all the techniques explained to here. For simplicity I have added inline comments.

```

jslp.registerTransition('a_transition', {
    easing: "easeOutBack",
    rows: 10,
    columns: 1,
    mode: "mixed",
    duration: 2000,
    animation: [
        // This first animation moves to left the even rows of
        // the exiting image.
        {
            layer: "above",
            selector: {
                name: "rows",
                skip: "odd"
            },
            effect: {
                name: "scroll",
                direction: "left",
                duration: 1000
            }
        },
        // Instead this animation moves to right the odd rows of
        // the exiting image.
        {
            layer: "above",
            selector: {
                name: "rows",
                skip: "even"
            },
            effect: {
                name: "scroll",
                direction: "right",
                duration: 1000
            }
        },
    ],
});

```

```

// Now the same for the entering image. This animation
// moves left its even rows
{
    layer: "below",
    selector: {
        name: "rows",
        skip: "odd"
    },
    effect: {
        name: "scroll",
        direction: "left",
        duration: 1000
    }
},
// Last but not least the animation to scroll right the
// odd rows of the entering image.
{
    layer: "below",
    selector: {
        name: "rows",
        skip: "even"
    },
    effect: {
        name: "scroll",
        direction: "right",
        duration: 1000
    }
}
]
});

```

Captions

Another exciting feature of jSliderPro is the ability to apply the same rules explained for the background images, to animate captions. As captions you can consider not only simple texts, but normal HTML code. Yes, it's true, you can use everything: images, videos, lists, tables, divs and so on.

The only aspect you must keep in mind is the manner in which selectors are used with HTML contents. Since makes no sense to think in terms of columns and rows, jSliderPro simplify everything assuming the HTML as is forming a square of 1 single column x many rows, where each row is represented by the corresponding HTML node. For example, if you write the following code:

```
<h1>Exciting set of core features</h1>
<ol style="font-size: 1.5em;" data-split="true">
  <li>100,000+ fully customizable transition effects</li>
  <li>Animated HTML captions with thousands of transition effects</li>
  <li>Customizable buttons and bullets with 400+ position configurations</li>
  <li>Compatible with all major browsers (PC, iOS, Android)</li>
  <li>Fully responsive</li>
  <li>Automatic memory management</li>
  <li>Javascript callbacks</li>
  <li>Extensive public interface</li>
</ol>
```

Selectors will 'see' a square of 1 column x 2 rows, where the first row is represented by the 'h1' statement, and the latter by the 'entire ol' (with each internal 'li').

At this point let's see how to create a caption for one slide. Simple! Add these statements to a content:

```
<li class="jsliderpro-content"
...>

... IMAGE DECLARATION ...

<div class="jsliderpro-caption" data-enter="" data-exit="">
  ...YOUR HTML CODE HERE...
</div>

</li>
```

The attributes 'data-enter' and 'data-exit' are needed to set respectively for the enter and exit animation of a caption's transition. That's it!

As said before jSliderPro animates each 'first level' HTML element as a whole, even if it's formed by a lot of nested elements. If your needs forces you to animate everything you can add a 'split' attribute to the choosen element, as this:

```
<li class="jsliderpro-content"
...>

... IMAGE DECLARATION ...

<div class="jsliderpro-caption" data-enter="" data-exit="">
  <h1>Exciting set of core features</h1>
  <ol style="font-size: 1.5em;" data-split="true">
```

```

        <li>100,000+ fully customizable transition effects</li>
        <li>Animated HTML captions with thousands of transition effects</li>
        <li>Customizable buttons and bullets with 400+ position configurations</li>
        <li>Compatible with all major browsers (PC, iOS, Android)</li>
        <li>Fully responsive</li>
        <li>Automatic memory management</li>
        <li>Javascript callbacks</li>
        <li>Extensive public interface</li>
    </ol>
</div>
</li>

```

Now all child of the 'ol' node will be taken into account. Now the animating elements are not just 2 as before, but 10. Note that also the 'ol' tag will be animated as a separated entity. That's useful, for instance, if you changed the background color via CSS.

If you, instead, don't want to include the parent element (as in div, ol, ul and so on) place a 'skip' attribute into that:

```

<li class="jsliderpro-content"
...>
    ... IMAGE DECLARATION ...
    <div class="jsliderpro-caption" data-enter="" data-exit="">
        <h1>Exciting set of core features</h1>
        <ol style="font-size: 1.5em;" data-split="true" data-skip="true">
            <li>100,000+ fully customizable transition effects</li>
            <li>Animated HTML captions with thousands of transition effects</li>
            <li>Customizable buttons and bullets with 400+ position configurations</li>
            <li>Compatible with all major browsers (PC, iOS, Android)</li>
            <li>Fully responsive</li>
            <li>Automatic memory management</li>
            <li>Javascript callbacks</li>
            <li>Extensive public interface</li>
        </ol>
    </div>
</li>

```

The full power of jSliderPro is expressed when you add multiple caption animations each with different 'enter' and 'exit' transition attributes:

```

<li class="jsliderpro-content"
...>
    ... IMAGE DECLARATION ...
    <div class="jsliderpro-caption" data-enter="" data-exit="">
    </div>
    <div class="jsliderpro-caption" data-enter="" data-exit="">
    </div>
    <div class="jsliderpro-caption" data-enter="" data-exit="">
    </div>

```

