Highslide Tutorial

CONTENTS

THUMBNAIL MARKUP

The basic way to mark up your thumbnail is to put an `a`-tag around an `img`-tag. The `a`-tag has to be supplied by the correct `class` and `onclick` attributes like below. In this case the browser points directly to the image if the script fails.

```
<a href="large-image.jpg" class="highslide" onclick="return hs.expand(this)">
    <img src="thumbnail.jpg" alt="Highslide JS"
        title="Click to enlarge" height="100" width="100" />
</a>
```

You may want to create a html page to show if the script fails in old browsers or if the user has disabled JavaScript. This is a good idea for database powered sites, where you can display the caption text in the fallback page. To achieve this you supply the function call of the `onclick`-attribute with a second parameter where you specify a hs.src. Read more about these parameters under Three levels of customizing below.

```
<a href="showimage.php?src=images/large-image.jpg" class="highslide"
        onclick="return hs.expand(this, { src: 'images/large-image.jpg' } )">
    <img src="thumbnail.jpg" alt="Highslide JS" title="Click to enlarge"
        height="100" width="100" />
</a>
```

If you prefer separating content from behaviour in your code, you can download a Highslide configuration with "Unobtrusive" in the Configurator, and mark up the thumbnail in the following clean way using `rel="highslide"`. In this case you can add extra options using the hs.onSetClickEvent event and the hs.isUnobtrusiveAnchor callback function.

```
<a href="large-image.jpg" class="highslide" rel="highslide">
    <img src="thumbnail.jpg" alt="Highslide JS" title="Click to enlarge"
        height="100" width="100" />
</a>
```

CSS RULES

Highslide JS relies on a number of CSS rules for the layout and functionality to work. You can modify the presentational properties, like `border`, `background`, `padding` etc. to suit your design. Other properties, like `position` and `display`, should remain untouched. These CSS-rules are defined:

`.highslide-container`
: The top-level container for all dynamic content generated by Highslide. This class has a font family and type declaration that by default applies to all popups.

`.highslide`
: Used on the `a`-tag surrounding the thumbnail to apply the zoom cursor.

`.highslide-active-anchor`
: Since version 3.3, the additional class name given to the anchor when it is opened. To hide the thumbnail when a full image is opened, set `visibility: hidden` on `.highslide-active-anchor img`.

`.highslide-caption`
: Styles for the caption. Obviously necessary only if you use captions.

`.highslide-controls`
: The wrapper element for the controls overlaid on the image in slideshow mode. Each single button's class name is `.highslide-previous`, `.highslide-play`, `.highslide-pause`, `.highslide-next`, `.highslide-full-expand` and `.highslide-close`.

`.highslide-credits`
: The credits label, "Powered by...".

`.highslide-dimming`
: When using a dimmed page background, this is where you specify the color of the background.

`.highslide-full-expand`
: Since version 3.3, the CSS rule where you define background image and other properties for the full-expand label.

`.highslide-heading`
: Styles for the heading.

`.highslide-image`
: This is where you specify the borders for the expanded image.

`.highslide-image-blur`

Occurs on the expanded image when another image is expanded on top of it. The styles of `.highslide-image` are inherited for the blurred image, but you can specify additional properties. For instance a different `border-color` for the blurred image.

`.highslide-loading`
: Styles for the *Loading*-element that appears on top of the thumbnail when the full size image is loading. You display a loading graphic by defining a `background-image` for this element. For the *Loading*-text not to show, adjust the hs.loadingText.

`.highslide-move`
: The class name to use if you want a text anchor in the caption to allow the user to move the image. See A move handle for the image.

`.highslide-number`
: The styling for the index number, like "Image 1 of 5".

`.highslide-overlay`
: The initial class name for custom overlays. This class only states that the `display` property should be `none`. You can use other class names for your overlay if you specify the correct `display`.

INDIVIDUAL CSS RULES

You can give each specific expander it's own CSS properties. Supply the function call of the `onclick`-attribute with a second parameter where you specify a hs.wrapperClassName. Read more about these parameters under *Three levels of customizing* below.

```
onclick="return hs.expand(this, {wrapperClassName: 'my-wrapper-class'})"
```

Now you can define new CSS rules that inherit the general rules, but whose properties overwrite those of the general rules:

```
.my-wrapper-class .highslide-image {
    border-color: black;
}
.my-wrapper-class .highslide-caption {
    border-color: black;
    background-color: gray;
    color: white;
}
```

THREE LEVELS OF CUSTOMIZING

1) Settings for your entire site

Although you may adjust the settings in the top of the `highslide.js` itself, this approach may be unfortunate if you want to upgrade to a new version of Highslide JS later, because you have to change the settings in the new file. Also, if you use the packed version of the script, you cannot do this. Best practice for site settings is to create a separate file, for instance `highslide.cfg.js` and include it directly after `highslide.js`. In this file, you define settings like this:

```
hs.graphicsDir = 'my-highslide-dir/graphics/';
hs.outlineType = 'outer-glow';
```

2) Settings for each single HTML-page

You can change settings for the object in the HTML page itself. This has to occur after the script inclusion in the source code. If you don't want the script block in the top of all of your HTML files, put it in a separate file like in the above example. The following example shows how you set a new graphics directory and a non-default graphic outline, as well as translate language strings.

```
<script type="text/javascript" src="my-highslide-dir/highslide.js"></script>
<script type="text/javascript">
    hs.graphicsDir = 'my-highslide-dir/graphics/';
    hs.outlineType = 'outer-glow';
</script>
```

3) Settings for each single image

Some settings can be overridden inline for each image in your web page. This applies to the settings listed under hs.overrides. Supply the settings as an object literal in the second parameter of the hs.expand function:

```
onclick="return hs.expand(this, { wrapperClassName: 'my-wrapper-class', align:
'center' })"
```

If you prefer dropping the onclick attribute and separate your content from behaviour, see hs.onSetClickEvent.

For advanced users: you can create custom properties to send to the hs.Expander object. The properties are supplied in the third parameter of the hs.expand function call. For example, you can provide an url to use within HsExpander:

```
onclick="return hs.expand(this, null, { url: 'http://google.com' })"
```

The custom variable can then be accessed within the hs.Expander object through the `this.custom` property. This example shows how you can override click action in the hs.Expander.prototype.onImageClick event:

```
hs.Expander.prototype.onImageClick = function() {
   window.location.href = this.custom.url;
}
```

CAPTIONS AND HEADINGS

Captions and headings are special types of custom overlays, that by default appear below and above the image respectively. Simple caption and heading texts can be

1. pulled from the `alt` or `title` attribute of the thumbnail using hs.captionEval and hs.headingEval,
2. written in the hs.captionText and hs.headingText options or
3. defined as a hidden `div` in your web page containing full HTML. To use full HTML, you create a `div` directly after your thumbnail `<a>` and give it a class name of `highslide-caption` or `highslide-heading`.
4. If you want better control, see hs.captionId and hs.headingId.

A full HTML caption as outlined in point #3 above:

```
<a id="thumb1" href="large-image.jpg" class="highslide"
        onclick="return hs.expand(this)">
    <img src="thumbnail.jpg" alt="Highslide JS" title="Click to enlarge"
        height="100" width="100" />
```

```
</a>
<div class='highslide-caption'>
    Caption text or additional HTML goes here
</div>
```

You can also set hs.captionId = 'theCaption' in the head section to reuse the same caption for all images.

You can add any HTML you want inside the caption div, for example a Close link (see below).

CLOSING THE IMAGE FROM A TEXT LINK

You may want to put a Close link or graphic inside the caption, in a custom overlay or in HTML content. Call hs.close() like this:

```
<a href="#" onclick="return hs.close(this)">Close</a>
```

If you want to close the image from elsewhere on the page, give the thumbnail a-tag an id-attribute, then call hs.close with the id like this:

```
<a href="#" onclick="return hs.close('thumb1')">Close</a>
```

A MOVE HANDLE FOR THE IMAGE

A move handle can also be placed inside the caption, in a custom overlay or in HTML content. All you have to do is to supply the link with the class name highslide-move.

```
<a href="#" class="highslide-move">
    Move
</a>
```

PREVIOUS AND NEXT LINKS

If you have a series of photos, you can put Previous and Next links within the caption, in a custom overlay or in HTML content:

```
<a href="#" onclick="return hs.previous(this)"
        title="Previous (left arrow key)">Previous</a>

<a href="#" onclick="return hs.next(this)"
        title="Next (right arrow key)">Next</a>
```

POSITIONING THE EXPANDED CONTENT

By default the expanded content appears centered above the opening anchor if the viewport allows it. If the popup bumbps into one of the sides of the viewport, it will move so that all of the content is visible. The position can be controled in detail by using one of the methods below. These require that you use the highslide-full.js configuration, or you get your own configuration from the Configurator with "Advanced positioning" turned on.

- To center the expander in the viewport, use the hs.align setting.
- Use the hs.anchor setting to control where the popup expands relative to the opening anchor.
- Use the hs.marginTop, hs.marginRight, hs.marginBottom or hs.marginLeft settings to make space, for example for a top or left menu.
- Use hs.targetX and hs.targetY to gain pixel-precise control of the position of the expanded content.

CUSTOM OVERLAYS

Highslide JS internally uses two overlays on the image, namely the credits and the full-expand link that appears when the image size has been reduced to fit the client. If you use captions or headings, those are also added internally as overlays. Additionally, you can define your own overlays. First, create a `div` somewhere in the page, and give it the class name `highslide-overlay`. The `div` also needs an unique `id`. Please observe that you **cannot use this id to apply styles**, because the entire div is cloned internally in the script, and the clone looses the id. Instead, you can apply additional class names to control CSS styles.

```
<div id="controlbar" class="highslide-overlay controlbar">
    <a href="#" onclick="return hs.previous(this)"
            title="Previous (left arrow key)">
        Previous
    </a>
    <a href="#" onclick="return hs.next(this)"
            title="Next (right arrow key)">
        Next
    </a>
    <a href="#" class="highslide-move" title="Click and drag to move"
            style="margin-left: 10px">
        Move
    </a>
    <a href="#" onclick="hs.close(this)">
        Close
    </a>
</div>
```

Next, you have to tie the overlay to one or more thumbnails. In the head section of the HTML page, after the `highslide.js` script is included, call the hs.registerOverlay function. The function takes an array object as the first parameter. See hs.registerOverlay for what options you have for customising the overlay.

When you want to add individual overlays for each expander, hs.Expander.prototype.createOverlay is generally a better option. Or you can use the hs.Expander.prototype.onCreateOverlay event to change the properties of the overlay before it is added to each single expander.

IMAGE MAPS

Since version 4.0 Highslide is optimised for image maps. Like shown below the `area` object is marked up with an onclick attribute. See a live example at example-image-map.html.

```
<img src="scandinavia.jpg" alt="" usemap="#imgmap1" />
<map id="imgmap1" name="imgmap1">
    <area shape="rect" alt="Iceland" title="Iceland" coords="1,43,110,135"
        href="iceland.jpg" onclick="return hs.expand(this)" />
    <area shape="circle" alt="Denmark" title="Denmark" coords="239,352,36"
        href="denmark.jpg" onclick="return hs.expand(this)" />
</map>
```

With Gallery

Since version 4.0, Highslide supports image galleries with slideshows, internal navigation and shared controls. See hs.addSlideshow for a code example and live demonstration.

With HTML

The Highslide HTML extension allows full HTML content to be shown in a Highslide popup. To use this functionality, you have to use a version of Highslide with the appropriate options checked in the Configurator. The basic way to mark up the HTML expander is as follows:

```
<a href="fallback.htm" class="highslide"
onclick="return hs.htmlExpand(this,
    { contentId: 'my-content' } )">
Click here
</a>

<div class="highslide-html-content" id="my-content"
    style="width: 200px">
    <a href="#" onclick="hs.close(this)">
        Close
    </a>
    <div class="highslide-body">
        Remember to include a means of closing
        the expander.
    </div>
</div>
```

As you see, the content lives anonymously somewhere in your page just like the image captions, until it is picked up and displayed by Highslide. By using AJAX content or iframe content you can move the content out of the page.

ADDITIONAL CSS RULES

`.highslide-html`
: Like `highslide-image`, this is where you specify borders for HTML content.

`.highslide-html-blur`
: Equivalent to `highslide-image-blur`.

`.highslide-html-content`
: General class that you apply to the element you want to expand.

`.highslide-resize`
: The class name to use if you want an element within the content to behave like a resize handle.

`.highslide-body`
: If the HTML content element contains an element with this class name, this element will display scrollbars if the size of the expander is exceeded. This way you can create headers and footers. If not, scrollbars will be shown in the outer element. Also, when you display AJAX, Iframe or Flash content, it will be placed in this container.

`.highslide-header`, `.highslide-footer`, `.highslide-previous`, `.highslide-next`, `.highslide-move`, `.highslide-close`
: These are the CSS rules associated with self rendering.

CONTROLING THE SIZE OF THE EXPANDER

You control the size of the expander by setting CSS width and height on the content div. By setting none at all the expander is behaving like a normal div, and takes the entire viewport width if the content wants it. By setting width and not height, the expander takes the height it needs within the viewport.

For more detailed control on each single expander, use hs.width and hs.height. These properties override those set using CSS on the content div. When you use self rendering content wrappers, this is generally an easier way than using separate hs.wrapperClassName properties and defining the corresponding CSS rules.

The size is also affected by hs.allowSizeReduction, hs.allowWidthReduction and hs.allowHeightReduction.

CONTROLING HEADERS AND FOOTERS

If the content takes up more screen area than the expanded window, scrollbars will appear. However, you are able to control where to put the scrolling area, so that you can display headers and footers that are not scrolled. If you define a div with class name `highslide-body`, this div will swallow all the exceeding content and display scrollbars. Remember you can adjust the appearance of the scrolling area by using CSS margins.

```
<div class="highslide-html-content" id="example1">
    <!-- header -->
    <a href="#" onclick="hs.close(this)">
        Close
    </a>
    <!-- scrolling content -->
    <div class="highslide-body">
        Some lengthy content here.
    </div>
    <!-- footer -->
    <div>
        Footer.
    </div>
</div>
```

SELF RENDERING CONTENT WRAPPERS

Since version 3.3, if hs.contentId is not set for a HTML expander, a self rendered content wrapper is created internally in Highslide. The main content is then either grabbed from Ajax, Iframe, Flash or the maincontent keyword. This wrapper is meant to be heavily styled using CSS, and you can use styling for example to remove elements you don't want. The language strings can be localized through the hs.lang object. To modify the HTML of the wrapper, override the contentWrapper property of the hs.skin object.

The HTML structure of the self rendered content looks like the following. The included content is injected into the highslide-body.

```
<div class="highslide-header">
    <ul>
        <li class="highslide-previous">
            <a href="#" title="{hs.lang.previousTitle}" onclick="return
hs.previous(this)">
            <span>{hs.lang.previousText}</span></a>
        </li>
        <li class="highslide-next">
            <a href="#" title="{hs.lang.nextTitle}" onclick="return
hs.next(this)">
            <span>{hs.lang.nextText}</span></a>
        </li>
        <li class="highslide-move">
            <a href="#" title="{hs.lang.moveTitle}" onclick="return false">
            <span>{hs.lang.moveText}</span></a>
        </li>
        <li class="highslide-close">
            <a href="#" title="{hs.lang.closeTitle}" onclick="return
hs.close(this)">
            <span>{hs.lang.closeText}</span></a>
        </li>
    </ul>
</div>
<div class="highslide-body"></div>
<div class="highslide-footer"><div>
    <span class="highslide-resize"
title="{hs.lang.resizeTitle}"><span></span></span>
</div></div>
```

This is an example of how to style the self-rendered content. This example defines the looks of a wrapper using the default wrapperClassName, "highslide-wrapper". To create groups or individual popups with different styling, apply a hs.wrapperClassName to it, and redefine the CSS rules for that wrapper.

```
.highslide-wrapper {
    background-color: white;
}
/* Set 400px as the default width for expanders */
.highslide-wrapper .highslide-html-content {
    width: 400px;
    padding: 5px;
}
/* The list of controls */
.highslide-wrapper .highslide-header ul {
    margin: 0;
    padding: 0;
    text-align: right;
}
.highslide-wrapper .highslide-header ul li {
    display: inline;
    padding-left: 1em;
}
/* Hide the previous and next links */
.highslide-wrapper .highslide-header ul li.highslide-previous, .highslide-
wrapper .highslide-header ul li.highslide-next {
    display: none;
}
.highslide-wrapper .highslide-header a {
    font-weight: bold;
    color: gray;
    text-transform: uppercase;
    text-decoration: none;
}
.highslide-wrapper .highslide-header a:hover {
    color: black;
}
.highslide-wrapper .highslide-header .highslide-move a {
    cursor: move;
}
.highslide-wrapper .highslide-footer {
    height: 11px;
}
.highslide-wrapper .highslide-footer .highslide-resize {
    cursor: nw-resize;
    float: right;
    height: 11px;
    width: 11px;
    background: url(highslide/graphics/resize.gif);
    position: relative;
    top: 3px;
    left: 3px;
}
```

A RESIZE HANDLE FOR THE POPUP

A resize handle can placed in the content. All you have to do is to add an element with a class name `highslide-resize`. Best practice is to place the element at bottom right. For self rendering content a resize handle is inserted automatically, but relies on CSS to display.

```
<span class="highslide-resize">
    <img src="resize.png" alt="Resize"/>
</span>
```

USING INLINE CONTENT WITH A SELF RENDERING CONTENT WRAPPER

Since version 4.0, you can define inline content to be injected into a common wrapper, unlike hs.contentId that forces you to define the header and footer every time. Equivalently to captions and headings, the maincontent can be

1. pulled from the `alt` or `title` attribute of the thumbnail using hs.maincontentEval,
2. written in the hs.maincontentEval option or
3. defined as a hidden `div` in your web page containing full HTML. To use full HTML, you create a `div` directly after the opening `<a>` and give it a class name of `highslide-maincontent`.
4. If you want better control, see hs.maincontentId.

Option #3 looks like this:

```
<a href="fallback.htm" onclick="return hs.htmlExpand(this)" >
    Open popup
</a>
<div class="highslide-maincontent">
    This is the main content
```

```
</div>
```

DISPLAYING AJAX CONTENT

The `src` of the AJAX content is set using the `href` attribute of the `a`, or the hs.src inline parameter. The other setting we need is `objectType: 'ajax'`. The width and other styling of the AJAX popup is done through the CSS rules associated with self rendering. What separates AJAX content from Iframe is that you can grab a specific `id` from the source file if the file is HTML. There are three depths of grabbing AJAX content:

1. If the source file doesn't contain a `body` tag (non-html), the content is inserted as is.
2. If the source file contains a `body` tag, only the content of the body is inserted, to prevent doctype and head tags to be inserted in the body of the parent page. This means all non-inline CSS disappears, as valid documents only allow CSS rules in the head tag. So you have to define the CSS rules in the parent page rather than in the AJAX source file.
3. If the href points to a specific `id` on the page, for example `href="ajax.htm#intro"`, Highslide tries to grab the inner content of that `id`. This is a great way to keep all the contents of multiple expanders in one file. It also provides a good fallback for JavaScript disabled user agents.

```
<a href="ajax.htm#my-div"
        onclick="return hs.htmlExpand(this, { objectType: 'ajax'} )" >
    AJAX content
</a>
```

If you prefer separating your content from behaviour, download highslide.js with Unobtrusive checked in the Configurator. Then mark up your anchor like this:

```
<a href="ajax.htm#my-div" rel="highslide-ajax">
    AJAX content
</a>
```

Since version 3.3 Highslide by default includes the AJAX content in a self rendering content wrapper. To gain full control of the content wrapper, extending what you can do with CSS, you make an invisible `div` somewhere in your page. This is done the same way as with other HTML expanders, by defining the `contentId` setting. Then within the content div somewhere you put another div with class name "highslide-body". This is where the AJAX content is put dynamically.

```
<a href="ajax.htm#my-div"
        onclick="return hs.htmlExpand(this, { contentId: 'my-content',
        objectType: 'ajax'} )">
    AJAX content
</a>
<div class="highslide-html-content"
        id="my-content" style="width: 700px">
    <div>
     <a href="#" onclick="return hs.close(this)">
         Close
     </a>
    </div>
    <div class="highslide-body"></div>
</div>
```

DISPLAYING CONTENT IN AN IFRAME

The `src` of the iframe is set using the `href` attribute of the `a`, or the hs.src inline parameter. The other required setting is `hs.objectType`. Since version 3.3 Highslide is able to dynamically detect the size of the iframe's content, but this only applies to iframes of the same domain as the parent page. For external pages you need to set `hs.objectWidth` and `hs.objectHeight`. In addition to these, `hs.objectLoadTime` affects the appearance of the iframe. The width and other styling of the iframe popup is done through the CSS rules associated with self rendering.

```
<a href="iframe.htm"
        onclick="return hs.htmlExpand(this, {
            objectType: 'iframe'} )" >
    Content in iframe
</a>
```

If you prefer separating your content from behaviour, download highslide.js with Unobtrusive checked in the Configurator. Then mark up your anchor like this:

```
<a href="iframe.htm" rel="highslide-iframe">
    AJAX content
</a>
```

As with AJAX content, hs.contentId can be used to gain full control of the content wrapper.

AJAX VS. IFRAME - PROS AND CONS

- AJAX runs smoother in the browser as it is (optionally) preloaded and copied into the local DOM tree.
- AJAX can load the contents of a specific element in the source file
- Iframes are better if you need to link to other pages within the popup, for example for multi-page forms.

DISPLAYING FLASH CONTENT

The Highslide JS Flash capabilities make use of SWFObject. Since Highslide version 4.0 SWFObject 2.x is used, and this is not backwards compatible with SWFObject 1.5.

The first requirement is that you include `swfobject.js` in your header together with `highslide.js`. Like with iframes and Ajax, the URL of the included file goes in the href attribute of the opening anchor, or in the hs.src option. Then you use hs.objectType = 'swf' together with hs.objectWidth and hs.objectHeight to define the primary options for the included movie. For extended functionality like sending variables to the Flash movie, see hs.swfOptions.

Like with AJAX and iframes, the Flash content is injected into a self rendering wrapper or a wrapper generated using hs.contentId. Another thing with Flash content is that it looks bad when the size of the expander is reduced to fit the viewport. So you want to set hs.allowSizeReduction to false. Also note that a bug in Firefox and Camino for Mac requires that if you use a dimming page background, you set hs.dimmingGeckoFix.

```

<a href="Flash.swf"
```

```
    onclick="return hs.htmlExpand(this, { objectType:'swf',
        objectWidth: 300, objectHeight: 250, allowSizeReduction: false} )"
    class="highslide">
    Display flash
</a>
```