
Developing jQuery Plugins

creating an edit-on-demand plugin

Jakob Westhoff <jakob@php.net>

PHP Unconference Hamburg 2010
September 25, 2010

About Me

Jakob Westhoff

jQuery about itself

From the jQuery Website:

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.

jQuery about itself

From the jQuery Website:

jQuery is a fast and concise JavaScript **Library** that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.

jQuery about itself

From the jQuery Website:

jQuery is a **fast** and **concise** JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.

jQuery about itself

From the jQuery Website:

jQuery is a fast and concise JavaScript Library that simplifies HTML **document traversing**, **event handling**, **animating**, and **Ajax interactions** for rapid web development. jQuery is designed to change the way that you write JavaScript.

jQuery about itself

From the jQuery Website:

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. **jQuery is designed to change the way that you write JavaScript.**

Introduction to jQuery

- Compact
 - only 71kb minified
 - 24kb minified and gzipped

Introduction to jQuery

- Compact
 - only 71kb minified
 - 24kb minified and gzipped
- Cross-browser compatible
 - Internet Explorer 6.0+
 - Firefox 2+
 - Safari 3.0+
 - Opera 9.0+
 - Chrome

Introduction to jQuery

- Compact
 - only 71kb minified
 - 24kb minified and gzipped
- Cross-browser compatible
 - Internet Explorer 6.0+
 - Firefox 2+
 - Safari 3.0+
 - Opera 9.0+
 - Chrome
- Easily extendable

jQuery Example

```
$("#p#example").addClass("highlight").fadeIn("slow");
```

jQuery Example

```
$("#p#example").addClass("highlight").fadeIn("slow");
```

- Find all paragraphs with the id example

jQuery Example

```
$("p#example").addClass("highlight").fadeIn("slow");
```

- Find all paragraphs with the id `example`
- Add the css class `highlight` to them

jQuery Example

```
$("p#example").addClass("highlight").fadeIn("slow");
```

- Find all paragraphs with the id `example`
- Add the css class `highlight` to them
- Fade in the paragraph slowly

Working with jQuery

```
$("#p#example").addClass("highlight").fadeIn("slow");
```

- Accessed using \$ or jQuery

Working with jQuery

```
$("p#example").addClass("highlight").fadeIn("slow");
```

- Accessed using \$ or jQuery
- Document centric
 - \$(css selector).operation

Working with jQuery

```
$("#p#example").addClass("highlight").fadeIn("slow");
```

- Accessed using \$ or jQuery
- Document centric
 - \$(css selector).operation
- Fluent interface paradigm
 - operation().operation().operation()

Beginning Plugin Development

- **Before** you start
 - Check <http://plugins.jquery.com/>

Beginning Plugin Development

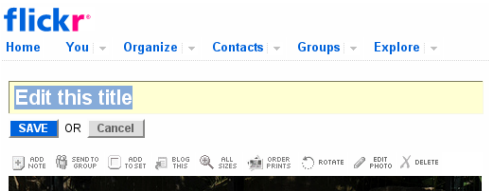
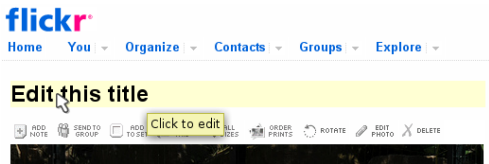
- Before you start
 - Check <http://plugins.jquery.com/>
- Read **Plugins/Authoring** documentation
 - <http://docs.jquery.com/Plugins/Authoring>

Beginning Plugin Development

- **Before** you start
 - Check <http://plugins.jquery.com/>
- Read **Plugins/Authoring** documentation
 - <http://docs.jquery.com/Plugins/Authoring>
- **Buy** "Pluginentwicklung mit jQuery" (german)
 - <http://entwicklerpress.de/jquery>

Introducing the Example

- On-Demand-Editing Plugin
- Used by Flickr and others



Plugin Requirements

- Applicable to every **block level** element

Plugin Requirements

- Applicable to every **block level** element
- Capture on **mouseover** / **mouseout** events for color change on hovering

Plugin Requirements

- Applicable to every **block level** element
- Capture on **mouseover** / **mouseout** events for color change on hovering
- Transform content to **edit box** and **save/cancel** buttons **onclick**

Plugin Requirements

- Applicable to every **block level** element
- Capture on **mouseover** / **mouseout** events for color change on hovering
- Transform content to **edit box** and **save/cancel** buttons **onclick**
- Execute **user-definable callback function** once new data is provided

Let's begin

- Choose a name for our plugin: `editable`

Stylesheet for our Plugin

- Choose a name for our plugin: `editable`
- Create `jquery.editable.css`
 - `Hover` effect
 - `Textbox` look and feel
 - `Button` positioning / look and feel

Stylesheet for our Plugin

```
1  .editable-hover ,
2  .editable-active
3  {
4      background-color: #efeeee;
5  }
6
7  .editable-container span
8  {
9      font-size: 12px;
10     color: #000000;
11     margin: 0 8px 0 8px;
12 }
13
14 .editable-container input ,
15 .editable-container textarea
16 {
17     display: block;
18 }
```

The problem with the \$ shortcut

- \$ should **not** be used in plugins
 - jQuery allows redefining of this alias for compatibility reasons
 - May break your plugin

The problem with the \$ shortcut

- \$ should **not** be used in plugins
 - jQuery allows redefining of this alias for compatibility reasons
 - May break your plugin
- **Always** use jQuery

The problem with the \$ shortcut

- \$ should **not** be used in plugins
 - jQuery allows redefining of this alias for compatibility reasons
 - May break your plugin
- **Always** use jQuery
- Or use **nifty workaround**:

```
1 (function($, jQuery) {  
2     ...  
3     // Your plugin code goes here  
4     ...  
5 })(jQuery, jQuery);
```

Registering the Plugin Method

- Choose a name for our plugin method: **editable**

Registering the Plugin Method

- Choose a name for our plugin method: `editable`
- Register the new method in the `jQuery.fn` namespace:

```
1 jQuery.fn.editable = function(options) {  
2     ...  
3     // Plugin method  
4     ...  
5 }
```

Handling Options

- Most plugins need configuration options

Handling Options

- Most plugins need configuration options
- Use a **default** if a certain option is not provided

Handling Options

- Most plugins need configuration options
- Use a **default** if a certain option is not provided
- Options are supplied as **associative array** / **object**

Default Options

- Allow to replace defaults from outside the plugin:

```
1 jQuery.fn.editable.defaults = {  
2   "multiline": false ,  
3   "prefix": "editable_" ,  
4   "savefn": function () {}  
5 };
```

Default Options

- Allow to replace defaults from outside the plugin:

```
1 jQuery.fn.editable.defaults = {  
2   "multiline": false ,  
3   "prefix": "editable_" ,  
4   "savefn": function () {}  
5 };
```

- Utilize the `jQuery.extend` function:

```
1 options = jQuery.extend(  
2   {},  
3   jQuery.fn.editable.defaults ,  
4   options  
5 );
```

this Context inside the Plugin Method

- The **this** context inside a plugin method ...
 - points to the **called** jQuery set
 - may contain **zero**, one or **multiple** element(s)

Using each to Handle Sets

- The `this` context inside a plugin method ...
 - points to the `called` jQuery set
 - may contain `zero`, one or `multiple` element(s)
- Use `each` to handle sets correctly:

```
1 return $(this).each(function( index, value ) {  
2     ...  
3     // "this" maps to the currently handled element  
4     ...  
5 });
```

What we have got so far

```
1 (function($, jQuery) {
2   jQuery.fn.editable = function(options) {
3     // Option handling
4     options = jQuery.extend(
5       {},
6       jQuery.fn.editable.defaults,
7       options
8     );
9     // Handle sets correctly.
10    // Ensure the fluent interface paradigm.
11    return $(this).each(function() {
12      // "real" plugin code goes in here
13      ...
14    });
15  }
16  jQuery.fn.editable.defaults = {
17    // Option defaults
18  };
19 })(jQuery, jQuery);
```

- Add a css class to the container element:

```
$(this).addClass('editable-container');
```

Register events with namespaces

- Register events for onhover effect:

```
1 $(this).bind(  
2     "mouseenter.editable mouseleave.editable",  
3     function(e) {  
4         $(this).toggleClass(  
5             'editable-hover'  
6         });  
7     }  
8 );
```

Register events with namespaces

- Register events for onhover effect:

```
1 $(this).bind(  
2   "mouseenter.editable mouseleave.editable",  
3   function(e) {  
4     $(this).toggleClass(  
5       'editable-hover'  
6     });  
7   }  
8 );
```

- All events can be registered for a specified **namespace**
- Namespaces are separated from the event name using a **dot** (**.**)

Register events with namespaces

- Register events for onhover effect:

```
1 $(this).bind(  
2     "mouseenter.editable mouseleave.editable",  
3     function(e) {  
4         $(this).toggleClass(  
5             'editable-hover'  
6         });  
7     }  
8 );
```

- All events can be registered for a specified **namespace**
- Namespaces are separated from the event name using a **dot** (**.**)
- Allow easy de-registration/management

Enter Edit Mode on Click

- On click enter edit mode
 - Registered using **one** as once only event
 - Calling **plugin private** function `enterEditMode`

```
1 $(this).one( 'click.editable',  
2   function() {  
3     enterEditMode( $(this) );  
4   }  
5 );
```

Private Plugin Functions

- Function in the plugin scope

Private Plugin Functions

- Function in the plugin scope
- Access to all variables **globally** declared

Private Plugin Functions

- Function in the plugin scope
- Access to all variables **globally** declared
- Access to all plugin method **parameters** (options)

Private Plugin Functions

- Function in the plugin scope
- Access to all variables **globally** declared
- Access to all plugin method **parameters** (options)

```
1 jQuery.fn.editable = function(options) {
2     options = ...
3
4     return $(this).each(function() {
5         ...
6     });
7
8     function enterEditMode(container) {
9         // Access to: container, options,
10        // every other declared var
11    }
12 }
```

Needed Private Functions

- `enterEditMode(container)`
 - Called whenever data input should be possible

Needed Private Functions

- `enterEditMode(container)`
 - Called whenever data input should be possible
- `leaveEditMode(container, updatedText)`
 - Called whenever the edit process is complete and the normal state should be restored

enterEditMode - Cleanup a little bit

- Store the **original** content

```
var originalTextContent = container.html();
```

enterEditMode - Cleanup a little bit

- Store the original content

```
var originalTextContent = container.html();
```

- Disable onhover effects

```
container.unbind('mouseenter.editable');  
container.unbind('mouseleave.editable');  
container.removeClass('editable-hover');
```

- Thanks to event namespaces only the events registered by the plugin are removed.

enterEditMode - Cleanup a little bit

- Store the original content

```
var originalTextContent = container.html();
```

- Disable onhover effects

```
container.unbind('mouseenter.editable');  
container.unbind('mouseleave.editable');  
container.removeClass('editable-hover');
```

- Thanks to event namespaces only the events registered by the plugin are removed.
- Set correct CSS classes

```
container.addClass('editable-active');
```

enterEditMode - Create input field and buttons

- Create the needed input field

```
var edit = $('<input type="text"></input>');
```

enterEditMode - Create input field and buttons

- Create the needed input field

```
var edit = $('<input_type="text"></input>');
```

- Create the needed buttons

```
var save = $('<button_/>', {  
  css: {  
    "class": "save"  
  },  
  "text": "Save"  
});
```

```
var cancel =  
  $('<button_class="cancel">Cancel</button>');
```

enterEditMode - Register button onClick

- Register onClick for **Cancel**

```
1 cancel.bind('click.editable', function(e) {  
2     e.stopPropagation();  
3     leaveEditMode(container, originalTextContent);  
4 });
```

enterEditMode - Register button onClick

■ Register onClick for Cancel

```
1 cancel.bind('click.editable', function(e) {
2     e.stopPropagation();
3     leaveEditMode(container, originalTextContent);
4 });
```

■ Register onClick for Save

```
1 save.bind('click.editable', function(e) {
2     e.stopPropagation();
3     leaveEditMode(
4         container,
5         options.savefn(
6             edit.val()
7         )
8     );
9 });
```

- Replace the current content with input field and buttons

```
1 container.empty()  
2   .append(edit)  
3   .append(save)  
4   .append(cancel);
```

leaveEditMode - Reactivate onHover

- Remove state from css classes

```
container.removeClass('editable-active');
```

leaveEditMode - Reactivate onHover

- Remove state from css classes

```
container.removeClass( 'editable-active' );
```

- Bind events for onHover effect

```
container.bind(  
    "mouseenter.editable_mouseleave.editable",  
    function(e) {  
        container.toggleClass( 'editable-hover' );  
    }  
);
```

leaveEditMode - Reactivate onClick

- Re-register the onClick handler

```
1 $(this).one( 'click.editable',  
2   function() {  
3     enterEditMode( $(this) );  
4   }  
5 );
```

leaveEditMode - Update text content

- Update the container with the given text content

```
container.html(updatedText);
```

Calling the plugin

- Simply call the `editable` method on the selected node.

```
$("#h1").editable({  
  savefn: function(content) {  
    // Do some AJAX magic here...  
    return content;  
  }  
});
```

Let's take a look

Live Demo!

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name
- 3 **Methods** have always to **return the object** they are working on

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name
- 3 **Methods** have always to **return the object** they are working on
- 4 Use **each** to ensure your method is applied to all elements in a set

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name
- 3 **Methods** have always to **return the object** they are working on
- 4 Use **each** to ensure your method is applied to all elements in a set
- 5 Always use `jQuery` instead of the `$` shortcut in your plugins
 - or use the nifty trick shown

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name
- 3 **Methods** have always to **return the object** they are working on
- 4 Use **each** to ensure your method is applied to all elements in a set
- 5 Always use jQuery instead of the \$ shortcut in your plugins
 - or use the nifty trick shown
- 6 Always use jQuerys event namespaces for registration

7 Golden Rules of Plugin Development

- 1 Name your file `jquery.[insert name of plugin].js`
 - eg. `jquery.myplugin.js`
- 2 Prefix your CSS classes with the plugin name
- 3 **Methods** have always to **return the object** they are working on
- 4 Use **each** to ensure your method is applied to all elements in a set
- 5 Always use jQuery instead of the \$ shortcut in your plugins
 - or use the nifty trick shown
- 6 Always use jQuerys event namespaces for registration
- 7 Make default options accessible from the outside

- Set methods (`jQuery.fn`)

Plugintypes

- Set methods (`jQuery.fn`)
- Utility functions (`jQuery`)

Plugintypes

- Set methods (`jQuery.fn`)
- Utility functions (`jQuery`)
- Special event handler

Plugin types

- Set methods (`jQuery.fn`)
- Utility functions (`jQuery`)
- Special event handler
- `jQuery.UI` widgets

Plugintypes

- Set methods (`jQuery.fn`)
- Utility functions (`jQuery`)
- Special event handler
- `jQuery.UI` widgets
- CSS-Selektoren

Plugintypes

- Set methods (`jQuery.fn`)
- Utility functions (`jQuery`)
- Special event handler
- `jQuery.UI` widgets
- CSS-Selektoren
- Easing functions (Animation)

Questions, comments or annotations?

Slides: <http://westhoffswelt.de/portfolio.htm>

Contact: Jakob Westhoff <jakob@php.net>

Unit testing with QUnit

- Unit testing? **YES** you want to!

Unit testing with QUnit

- Unit testing? YES you want to!
- QUnit

Unit testing with QUnit

- Unit testing? YES you want to!
- QUnit
 - jQuerys unit testing framework
 - <http://docs.jquery.com/QUnit>

Unit testing with QUnit

- Unit testing? **YES** you want to!
- QUnit
 - jQuerys unit testing framework
 - <http://docs.jquery.com/QUnit>
 - No stable release, yet

Unit testing with QUnit

- Unit testing? **YES** you want to!
- QUnit
 - jQuerys unit testing framework
 - <http://docs.jquery.com/QUnit>
 - No stable release, yet
 - However core features are extremely stable

Simple QUnit example

- Include `js` and `css`

```
<script src="jquery-latest.js"></script>
<script type="text/javascript" src="qunit/testrunner.js"></script>
<link rel="stylesheet" href="qunit/testsuite.css"
      type="text/css" media="screen" />
```

Simple QUnit example

- Include js and css

```
<script src="jquery-latest.js"></script>
<script type="text/javascript" src="qunit/testrunner.js"></script>
<link rel="stylesheet" href="qunit/testsuite.css"
      type="text/css" media="screen" />
```

- Create a simple html page to be filled with information

```
<h1>QUnit example</h1>
```

```
<h2 id="banner"></h2>
```

```
<h2 id="userAgent"></h2>
```

```
<ol id="tests"></ol>
```

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

Simple QUnit example

- Define a **module** (optional)

```
module("My_simple_example_module");
```

Simple QUnit example

- Define a **module** (optional)

```
module("My_simple_example_module");
```

- Add a **test**

```
test("My_first_test", function() {  
    ok(true, "Everything_is_fine.");  
});
```

Possible QUnit assertions

- `ok(state, [message])`
 - A boolean assertion, equivalent to JUnit's `assertTrue`

Possible QUnit assertions

- `ok(state, [message])`
 - A boolean assertion, equivalent to JUnit's `assertTrue`
- `equal(actual, expected, [message])`
 - Compare content of simple types (String, Number, ...)
 - Compare references of Objects (Array, Function, ...)

Possible QUnit assertions

- `ok(state, [message])`
 - A boolean assertion, equivalent to JUnit's `assertTrue`
- `equal(actual, expected, [message])`
 - Compare content of simple types (String, Number, ...)
 - Compare references of Objects (Array, Function, ...)
- `deepEqual(actual, expected, [message])`
 - A deep recursive comparison of content on all types

Asynchronous assertions

- Use `start()` and `stop()` for asynchronous testing

```
test("My_async_test", function () {
  stop(2000);
  setTimeout(function () {
    ok(true, "Everything_is_fine");
    start();
  }, 1000);
});
```