

---

# Piece by piece

An introduction to jQuery-UI widget development

Jakob Westhoff <jakob@php.net>

WebTech Conference  
13. October, 2010

Jakob Westhoff

@jakobwesthoff

<http://westhoffswelt.de>

## About You?

Questions?

# Questions!

# What comes next?

---

## jQuery-UI

# jQuery-UI about itself

---

From the jQuery-UI Website:

jQuery UI provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

# jQuery-UI about itself

---

From the jQuery-UI Website:

jQuery UI provides abstractions for low-level interaction and animation, advanced effects and **high-level, themeable widgets**, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

# jQuery-UI about itself

---

From the jQuery-UI Website:

jQuery UI provides abstractions for **low-level interaction and animation**, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

# jQuery-UI about itself

---

From the jQuery-UI Website:

jQuery UI provides abstractions for low-level interaction and animation, **advanced effects** and high-level, themeable widgets, built on top of the jQuery JavaScript Library, that you can use to build highly interactive web applications.

# jQuery-UI about itself

---

From the jQuery-UI Website:

jQuery UI provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, **built on top of the jQuery JavaScript Library**, that you can use to build highly interactive web applications.

# jQuery-UI - A short overview

- A set of high level widgets
- Low level framework to create own Widgets, Behaviors and Effects
- Compatible with all major Browsers (thanks to jQuery)
- Modular code base between 5-200kb
- Fully themeable using a graphical tool: ThemeRoller
- Highly extensible
- Download: <http://jqueryui.com>

# What comes next?

---

## Widgets

# Widgets included in jQuery UI

---

- Accordion
- Autocomplete
- Button
- Dialog
- Slider
- Tabs
- Datepicker
- Progressbar

## Section 1

Mauris mauris ante, blandit et, ultrices a, suscipit eget, quam. Integer ut neque. Vivamus nisi metus, molestie vel, gravida in, condimentum sit amet, nunc. Nam a nibh. Donec suscipit eros. Nam mi. Proin viverra leo ut odio. Curabitur malesuada. Vestibulum a velit eu ante scelerisque vulputate.

## Section 2

## Section 3

## Section 4

## Section 1

## Section 2

Sed non urna. Donec et ante. Phasellus eu ligula. Vestibulum sit amet purus. Vivamus hendrerit, dolor at aliquet laoreet, mauris turpis porttitor velit, faucibus interdum tellus libero ac justo. Vivamus non quam. In suscipit faucibus urna.

## Section 3

## Section 4

## Section 1

## Section 2

## Section 3

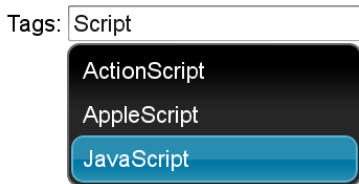
Nam enim risus, molestie et, porta ac, aliquam ac, risus. Quisque lobortis. Phasellus pellentesque purus in massa. Aenean in pede. Phasellus ac libero ac tellus pellentesque semper. Sed ac felis. Sed commodo, magna quis lacinia ornare, quam ante aliquam nisi, eu iaculis leo purus venenatis dui.

- List item one
- List item two
- List item three

## Section 4

# Autocomplete

---



# Buttons

---

A button element

A submit button

An anchor

# Dialog

Sed vel diam id libero **rutrum convallis**. Donec aliquet leo vel magna. Phasellus rhoncu ante. Etiam bibendum, enim faucibus aliquet rhoncus, arcu felis ultricies neque, sit an elit eros a lectus.

text in

checkb  
 radio

select

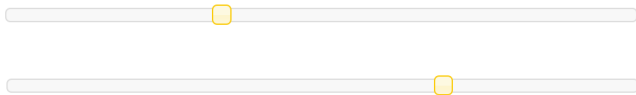
textare

**Basic modal dialog** ✕

Adding the modal overlay screen makes the dialog look more prominent because it dims out the page content.

# Slider

---



# Tabs

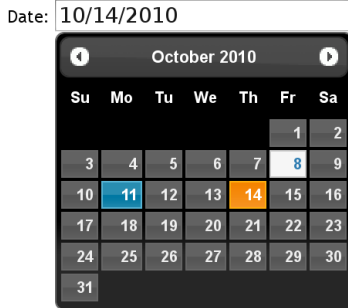
Nunc tincidunt

Proin dolor

Aenean lacinia

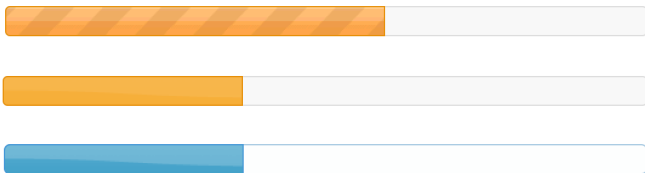
Proin elit arcu, rutrum commodo, vehicula tempus, commodo a, risus. Curabitur nec arcu. Donec sollicitudin mi sit amet mauris. Nam elementum quam ullamcorper ante. Etiam aliquet massa et lorem. Mauris dapibus lacus auctor risus. Aenean tempor ullamcorper leo. Vivamus sed magna quis ligula eleifend adipiscing. Duis orci. Aliquam sodales tortor vitae ipsum. Aliquam nulla. Duis aliquam molestie erat. Ut et mauris vel pede varius sollicitudin. Sed ut dolor nec orci tincidunt interdum. Phasellus ipsum. Nunc tristique tempus lectus.

# Datepicker



# Progressbar

---



# What comes next?

---

## Behaviors

# Behaviors included in jQuery UI

---

- Draggable
- Droppable
- Resizable
- Selectable
- Sortable

# What comes next?

---

## Effects

# Effects included in jQuery UI

---

- Blind
- Bounce
- Clip
- Drop
- Explode
- Fold
- Highlight
- Pulsate
- ...

What comes next?

---

# Animation Enhancements

# Animation Enhancements over default jQuery

---

- Color animations
- New easing functions
  - `easeInBounce`
  - `easeInQuad`
  - ...
- CSS class based animation

# What comes next?

---

## Plugintypes

# jQuery-UI plugintypes

---

- Widgets
- Behaviors
- Effects
- Normal jQuery plugintypes
  - Set methods (`jQuery.fn`)
  - Utility functions (`jQuery`)
  - Special event handler
  - CSS-Selectors
  - Easing functions (`Animation`)

# jQuery-UI plugintypes

---

- Widgets
- Behaviors
- Effects
- Normal jQuery plugintypes
  - Set methods (`jQuery.fn`)
  - Utility functions (`jQuery`)
  - Special event handler
  - CSS-Selectors
  - Easing functions (Animation)

# What comes next?

---

## Creating a Widget

# Creating a Widget without jQuery-UI

- Widgets usually have a state
  - Excessive use of `.data` method
  - Nested closures and custom objects
- Different initialization phases
- Handling of default and user options
- Multiple public methods
- Destruct and remove Widget on request

# What comes next?

---

## Widget Factory

# The Widget factory of jQuery-UI

---

- Instance based president states
- "Magic" methods for different initialization phases
- Automatic merging of default and user supplied options
- Custom methods without namespace pollution
- Distinguish between public and private methods

# What comes next?

---

## Example Widget

# The myprogressbar Widget

- A custom made progressbar
- Less flexible due to heavy usage of images
- More sophisticated graphical design possible



# Filename conventions

---

- Name of this widget: `myprogressbar`
- Namespace of this widget: `ui`
- All widgets should follow a certain filename convention:
  - `jquery.ui.myprogressbar.js`
  - `jquery.ui.myprogressbar.css`
  - `jquery.ui.myprogressbar/image.png`
  - ...

# The widget factory in action

---

- Call `$.widget` to create a new Widget
- First argument: namespace and identifier
- Namespaces **do not** protect against naming conflicts
- All default widgets use the `ui` namespace

# The widget factory in action

---

```
1 $.widget(  
2   'ui.myprogressbar',  
3   { /* Widget implementation */ }  
4 );
```

# Calling conventions

---

- Factory creates new method in `jQuery.fn` namespace
- Given identifier used as method name
- All interaction with the Widget through this method

# Calling conventions

---

- Widget creation

```
$("#id").myprogressbar();  
$("#id").myprogressbar({...});
```

# Calling conventions

---

- Widget creation

```
$("#id").myprogressbar();  
$("#id").myprogressbar({...});
```

- Invoking a method without arguments

```
$("#id").myprogressbar("method");
```

# Calling conventions

- Widget creation

```
$("#id").myprogressbar();  
$("#id").myprogressbar({...});
```

- Invoking a method without arguments

```
$("#id").myprogressbar("method");
```

- Invoking a method with arguments

```
$("#id").myprogressbar("method", arg1, arg2, ...);
```

# What comes next?

---

## Options

# Handling of Options

---

- Store default options in `options` property
- Override `_setOptions` method to handle option changes

# Handling of Options - options property

```
1 $.widget( "ui.myprogressbar", {
2   options: {
3     borderImage: 'images/progressbar_border.png',
4     borderThickness: 2,
5     borderImageWidth: 400,
6     borderImageHeight: 24,
7     progressImage: 'images/progressbar_inside.png',
8     progressImageSize: 600,
9     movementTimeout: 300,
10    initialValue: 0
11  },
12  ...
13 });
```

# Handling of Options - `_setOptions` method

```
1 $.widget( "ui.myprogressbar", {
2   ...,
3   _setOption: function( option, value ) {
4     // Call _setOption base implementation
5     $.Widget.prototype._setOption.apply( this, arguments );
6
7     switch ( option ) {
8       case "borderImage":
9         // Set the new border image inside the DOM
10        break;
11       case "progressImage":
12        // Set the new progressbar background image inside the DOM
13        break;
14       case "borderThickness":
15       case "borderImageWidth":
16       case "borderImageHeight":
17       case "progressImageSize":
18        // Recalculate progressbar position
19        break;
20     }
21   },
22   ...
23 });
```

# Handling of Options - `_setOptions` method

```
1 $.widget( "ui.myprogressbar", {
2   ...,
3   _setOption: function( option, value ) {
4     // Call _setOption base implementation
5     $.Widget.prototype._setOption.apply( this, arguments );
6
7     switch ( option ) {
8       case "borderImage":
9         // Set the new border image inside the DOM
10        break;
11       case "progressImage":
12        // Set the new progressbar background image inside the DOM
13        break;
14       case "borderThickness":
15       case "borderImageWidth":
16       case "borderImageHeight":
17       case "progressImageSize":
18        // Recalculate progressbar position
19        break;
20    }
21  },
22  ...
23 });
```

# Handling of Options - \_setOptions method

```
1 $.widget( "ui.myprogressbar", {
2   ...,
3   _setOption: function( option, value ) {
4     // Call _setOption base implementation
5     $.Widget.prototype._setOption.apply( this, arguments );
6
7     switch ( option ) {
8       case "borderImage":
9         // Set the new border image inside the DOM
10        break;
11       case "progressImage":
12        // Set the new progressbar background image inside the DOM
13        break;
14       case "borderThickness":
15       case "borderImageWidth":
16       case "borderImageHeight":
17       case "progressImageSize":
18        // Recalculate progressbar position
19        break;
20     }
21   },
22   ...
23 });
```

# Handling of Options - \_setOptions method

```
1 $.widget( "ui.myprogressbar", {
2   ...,
3   _setOption: function( option, value ) {
4     // Call _setOption base implementation
5     $.Widget.prototype._setOption.apply( this, arguments );
6
7     switch ( option ) {
8       case "borderImage":
9         // Set the new border image inside the DOM
10        break;
11       case "progressImage":
12        // Set the new progressbar background image inside the DOM
13        break;
14       case "borderThickness":
15       case "borderImageWidth":
16       case "borderImageHeight":
17       case "progressImageSize":
18        // Recalculate progressbar position
19        break;
20     }
21   },
22   ...
23 });
```

# Handling of Options - `_setOptions` method

```
1 $.widget( "ui.myprogressbar", {
2   ...,
3   _setOption: function( option, value ) {
4     // Call _setOption base implementation
5     $.Widget.prototype._setOption.apply( this, arguments );
6
7     switch ( option ) {
8       case "borderImage":
9         // Set the new border image inside the DOM
10        break;
11       case "progressImage":
12        // Set the new progressbar background image inside the DOM
13        break;
14       case "borderThickness":
15       case "borderImageWidth":
16       case "borderImageHeight":
17       case "progressImageSize":
18        // Recalculate progressbar position
19        break;
20    }
21  },
22  ...
23 });
```

# Handling of Options - Access merged options

---

- Default options are automatically merged with user options
- Result is written back to the `options` property

# Handling of Options - Access merged options

```
1 $.widget( 'ui.progressbar', {
2   options: {
3     borderImage: 'images/progressbar_border.png',
4     ...
5   },
6   getBorderImage: function() {
7     alert( this.options.borderImage );
8   },
9   ...
10  });
```

# Handling of Options - Access merged options

```
1 $.widget( 'ui.progressbar', {  
2   options: {  
3     borderImage: 'images/progressbar_border.png',  
4     ...  
5   },  
6   getBorderImage: function () {  
7     alert( this.options.borderImage );  
8   },  
9   ...  
10  });
```

# Handling of Options - Access merged options

```
1 $.widget( 'ui.progressbar', {
2   options: {
3     borderImage: 'images/progressbar_border.png',
4     ...
5   },
6   getBorderImage: function() {
7     alert( this.options.borderImage );
8   },
9   ...
10  });
```

# What comes next?

---

## Initialization

# "Magic" `_create` method

---

- `_create` is automatically invoked on widget creation
- Only called once for each widget
- Little brother `_init` invoked every time

# Access to the targeted element

---

- Targeted element stored in `element` property
- Saved as jQuery set
- Other magic properties exist: `options`, `widgetName`, `widgetEventPrefix`

# "Magic" \_create method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     _create: function() {
4         this.element.empty();
5
6         $( '<img_>', {
7             src: this.options.borderImage,
8             css: {
9                 border: 'none',
10                width: this.options.borderImageWidth,
11                height: this.options.borderImageHeight,
12                background: ...
13            }
14        }).appendTo( this.element );
15
16        this.value( this.options.initialValue );
17    }
18 });
```

# "Magic" \_create method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     _create: function() {
4         this.element.empty();
5
6         $( '<img_>', {
7             src: this.options.borderImage,
8             css: {
9                 border: 'none',
10                width: this.options.borderImageWidth,
11                height: this.options.borderImageHeight,
12                background: ...
13            }
14        }).appendTo( this.element );
15
16        this.value( this.options.initialValue );
17    }
18 });
```

# "Magic" \_create method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     _create: function() {
4         this.element.empty();
5
6         $( '<img_>', {
7             src: this.options.borderImage,
8             css: {
9                 border: 'none',
10                width: this.options.borderImageWidth,
11                height: this.options.borderImageHeight,
12                background: ...
13            }
14        }).appendTo( this.element );
15
16        this.value( this.options.initialValue );
17    }
18 });
```

# "Magic" \_create method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     _create: function() {
4         this.element.empty();
5
6         $( '<img_>', {
7             src: this.options.borderImage,
8             css: {
9                 border: 'none',
10                width: this.options.borderImageWidth,
11                height: this.options.borderImageHeight,
12                background: ...
13            }
14        }).appendTo( this.element );
15
16        this.value( this.options.initialValue );
17    }
18 });
```

# "Magic" \_create method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     _create: function() {
4         this.element.empty();
5
6         $( '<img_>', {
7             src: this.options.borderImage,
8             css: {
9                 border: 'none',
10                width: this.options.borderImageWidth,
11                height: this.options.borderImageHeight,
12                background: ...
13            }
14        }).appendTo( this.element );
15
16        this.value( this.options.initialValue );
17    }
18 });
```

# What comes next?

---

## Destruction

# Destruction of Widgets

---

- Every widgets inherits certain methods
- In most cases they need to be extended
- `destroy()` is one of them
- Others are: `option`, `widget`, `enable` and `disable`

# The destroy method

---

- Remove all events namespaced with the widget identifier
- Remove the item stored in the `element` property
- Remove the item returned by the `widget` method

# The destroy method

```
1 $.widget( "ui.myprogressbar", {
2     ...,
3     destroy: function() {
4         // Call the base implementation of destroy
5         $.Widget.prototype.destroy.apply( this );
6
7         // Do more cleanup here
8     },
9     ...
10 });
```

# What comes next?

---

## Event namespaces

# Event namespaces - a mostly unknown feature

- Group registered events by a certain identifier
- As usual use `bind` for listening to events
- Namespaces and event types separated using a **dot** (`.`)
  - eg. `click.namespace`
- Namespaces allow easy de-registration/management
  - `.unbind("click.namespace")`
  - `.unbind(".namespace")`

# Event namespaces in widget development

---

- **Always** use event namespaces inside your widgets
- Use the widget identifier as namespace
- No conflict between your event handlers and others
- Easy de-registration possible without remembering the callback

# Register events with namespaces

---

```
1 $(this).bind(  
2     'mouseover.myprogressbar',  
3     function(event) {  
4         ...  
5     }  
6 );
```

# What comes next?

---

## Methods / Properties

# Widget methods and properties

---

- Use custom properties and methods to structure your code
- Both are defined in the Widget object

# Widget methods and properties

- Use custom properties and methods to structure your code
- Both are defined in the Widget object

```
1 $.widget(  
2   'ui.myprogressbar',  
3   {  
4     'property': 42  
5     'method': function () {...}  
6   }  
7 );
```

# One instance per Widget

---

- A new instance is created for each Widget invocation
- Properties are persistent

# One instance per Widget

- A new instance is created for each Widget invocation
- Properties are persistent

```
1 $.widget( "ui.myprogressbar", {
2     setValue: function( val ) {
3         this.currentValue = val;
4
5         // Calculate new background position
6         ...
7     }
8 });
```

# Fluent interface and returned values

---

- Widget factory takes care of fluent interface handling
- Returning a value from a method circumvents this behavior

# Fluent interface and returned values

- Widget factory takes care of fluent interface handling
- Returning a value from a method circumvents this behavior

```
1 $.widget( "ui.myprogressbar", {  
2     getValue: function(val) {  
3         return this.currentValue;  
4     }  
5 });
```

# Private methods

---

- Methods prefixed with underscore (`_`) are private
- Properties are always private
- Properties are always private

# Private methods

- Methods prefixed with underscore (`_`) are private
- Properties are always private
- Properties are always private

```
1 $.widget( "ui.myprogressbar", {  
2   _calculateBackgroundPosition: function() {  
3     ...  
4   }  
5 });
```

# What comes next?

---

## Formatting

# CSS formatting in widgets

---

- Use CSS rules for visual formatting if possible
- Decoupling of functionality and representation
- Store CSS in appropriate position
  - `jquery.ui.myprogressbar.css`
- Always use classes not ids (multiple invocation)
- Use widget name and namespace as prefix
  - `ui-myprogressbar-container`
- Follow the hierarchy of your elements
  - `ui-myprogressbar-container-bar`

# ThemeRoller

---

- Custom Widgets can profit from ThemeRoller
- Use jQuery-UI CSS framework for your elements
- Just a set of predefined CSS classes

- Structural helper
  - `ui-helper-hidden`, `ui-helper-clearfix`, ...
- Widget look and feel
  - `ui-widget-header`, `ui-widget-content`, ...
- Button and input element marker
  - `ui-priority-primary`, `ui-state-default`, ...
- Visual states
  - `ui-state-highlight`, `ui-state-error`, ...
- Icons
  - `ui-icon`, `ui-icon-folder-collapsed`, ...

# What comes next?

---

## Conclusion

# What you have learned today - In general

---

- 1 Obey the filename rules: `jquery.namespace.widget.js`

# What you have learned today - In general

---

- 1 Obey the filename rules: `jquery.namespace.widget.js`
- 2 Use the Widget factory (`$.widget`)

# What you have learned today - In general

---

- 1 Obey the filename rules: `jquery.namespace.widget.js`
- 2 Use the Widget factory (`$.widget`)
- 3 Don't be afraid of using many options

# What you have learned today - In general

---

- 1 Obey the filename rules: `jquery.namespace.widget.js`
- 2 Use the Widget factory (`$.widget`)
- 3 Don't be afraid of using many options
- 4 Always use Event namespaces to register events

# What you have learned today - In general

- 1 Obey the filename rules: `jquery.namespace.widget.js`
- 2 Use the Widget factory (`$.widget`)
- 3 Don't be afraid of using many options
- 4 Always use Event namespaces to register events
- 5 Prefix your CSS classes with the widget name, follow the hierarchy

# What you have learned today - About the factory

---

- 1 State of your Widget is preserved

# What you have learned today - About the factory

---

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)

# What you have learned today - About the factory

---

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)
- 3 User options and defaults are automatically merged

# What you have learned today - About the factory

---

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)
- 3 User options and defaults are automatically merged
- 4 Implement `_setOptions` to allow option changes

# What you have learned today - About the factory

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)
- 3 User options and defaults are automatically merged
- 4 Implement `_setOptions` to allow option changes
- 5 `_create` is the right place to initialize your widget

# What you have learned today - About the factory

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)
- 3 User options and defaults are automatically merged
- 4 Implement `_setOptions` to allow option changes
- 5 `_create` is the right place to initialize your widget
- 6 Provide a `destroy` method to clean up

# What you have learned today - About the factory

- 1 State of your Widget is preserved
- 2 Properties are always private, methods can be (underscore)
- 3 User options and defaults are automatically merged
- 4 Implement `_setOptions` to allow option changes
- 5 `_create` is the right place to initialize your widget
- 6 Provide a `destroy` method to clean up
- 7 The fluent interface is taken care of if no value is returned

- Widgets

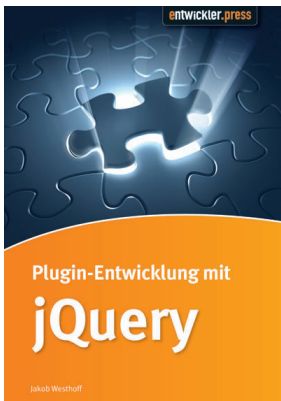
# jQuery-UI plugintypes

---

- Widgets
- Behaviors
- Effects
- Normal jQuery plugintypes
  - Set methods (`jQuery.fn`)
  - Utility functions (`jQuery`)
  - Special event handler
  - CSS-Selectors
  - Easing functions (`Animation`)

# Want to learn more?

- Buy "Pluginentwicklung mit jQuery" (german)
- <http://entwicklerpress.de/jquery>



Questions, comments or annotations?

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

Contact: Jakob Westhoff <[jakob@westhoffswelt.de](mailto:jakob@westhoffswelt.de)>

Follow Me: @jakobwesthoff