Kapitel 3

jQuery – der Einstieg

Sie werden sehen, wie Ihnen jQuery Arbeit abnimmt. Sie schreiben weniger Code, um zum Ergebnis zu kommen. Und Sie werden sehen, welche Wege es gibt, jQuery in Ihr Projekt zu inkludieren.

In diesem Kapitel erfahren Sie einiges über die Grundlagen von jQuery und über die Hintergründe des jQuery-Objekts und seines Einsatzes. Sie lernen, wie Sie mit jQuery Elemente Ihrer HTML-Seite selektieren und anschließend Aktionen ausführen können. Zum Verständnis benötigen Sie zumindest Grundkenntnisse in HTML und JavaScript sowie – was in Zusammenhang mit jQuery beinahe noch wichtiger ist – ein Grundverständnis von CSS und der Formulierung von CSS-Selektoren. Sollten Sie hier einen Nachschlag benötigen, raten wir Ihnen, zuerst die beiden Kapitel im Anhang zu lesen: Anhang A, »HTML und CSS«, und Anhang B, »JavaScript und DOM«. Lassen Sie uns nun also einfach beginnen!

3.1 Vergleich: JavaScript mit und ohne jQuery

Um zu verdeutlichen, wie jQuery dem User tagtäglich viel Arbeit abnimmt und ihm so das Leben im Web erleichtert, möchten wir hier zunächst mit einem Beispiel anfangen, das – noch ohne die Hilfe von jQuery – die Manipulation eines HTML-Elements vornimmt.

Angenommen, Sie wollen, abhängig von seinem Textinhalt, einem von mehreren `<p>`-Absätzen die Klasse `green` zuweisen und ihm damit eine neue Schriftfarbe geben. Dann betrachten Sie dazu zunächst einmal den Inhalt des `<script>`-Elements:

```
<html>
<head>
<title>Listing 1</title>
   <style type="text/css">
      .green {
         color:#009933;
      }
   </style>
```

```
    <script type="text/javascript">
      window.onload = function() {
        var elements = document.getElementById("box")
            .getElementsByTagName("p");
        for (var i = 0; i < elements.length; i++) {
          if (elements[i]
          .firstChild.data == "Zweiter Absatz") {
            elements[i].className = "green";
          }
        }
      }
    </script>
  </head>
  <body>
    <div id="box">
      <p>Erster Absatz</p>
      <p>Zweiter Absatz</p>
      <p>Dritter Absatz</p>
    </div>
  </body>
  </html>
```

Listing 3.1 Schriftfarbe ändern ohne jQuery

Was in diesem Scriptabschnitt passiert, ist Folgendes: Sobald die komplette HTML-Seite geladen ist (`window.onload`), wird eine Kollektion mit allen Absätzen (`"p"`) erstellt, die sich innerhalb des Containers mit der ID (`"box"`) befinden.

Zum Zeilenumbruch vor dem Punktoperator

Wichtig: Sie dürfen *vor* dem Punktoperator in Objekten praktischerweise einen Zeilenumbruch machen, da der Punkt am Beginn der Folgezeile der Script-Engine deutlich macht, dass die Anweisung weitergeht. Überlange Programmzeilen wie diese können so vermieden werden:

```
var elements = document.getElementById("box")
             .getElementsByTagName("p");
```

Diese Kollektion wird in einer Variablen `elements` gespeichert. Anschließend werden über eine `for`-Schleife und eine `if`-Anweisung alle gefundenen `<p>`-Container dahingehend geprüft, ob sie einen Textknoten mit dem Inhalt »Zweiter Absatz« enthalten.

Wenn dem so ist, wird dem betreffenden Absatz die Klasse `green` hinzugefügt und damit dessen Schriftfarbe auf »Grün« gesetzt. Sie brauchen an dieser Stelle nicht zu sehr in die Tiefe zu gehen, um sich klarzumachen, dass Sie hier genau überlegen müs-

sen, wie Sie in Ihrem Script mit Bedingungen und Schleifen Ihre Suchergebnisse filtern, bis am Ende das gewünschte Ergebnis im Browser erscheint. Ein Framework kann Ihnen dieses Kopfzerbrechen ersparen. Werden Sie also aktiv!

3.2 jQuery einbinden

Es versteht sich, dass das jQuery-Framework, um es nutzbar zu machen, zunächst in eine HTML-Seite eingebunden werden muss: Dem ausführenden Browser müssen die Funktionalitäten zur Verfügung stehen, um die besonderen Eigenschaften und Methoden von jQuery zu interpretieren. Zunächst sollten Sie sich, falls dies nicht bereits geschehen ist, eine aktuelle Version von jQuery besorgen – dies können Sie direkt bei der offiziellen jQuery-Website *http://jquery.com/download* tun. Laden Sie von dort sowohl die minimierte Produktionsversion als auch die Developer-Version von jQuery herunter. Dort finden Sie auch das Plugin *migrate.js*.

Downloaden oder doch nicht downloaden?

Sie finden die erforderlichen jQuery-Dateien auch im Ordner *jquery* im Onlinebereich zu diesem Buch. Alle Beispiele basieren auf Version 3.x. Sollte es in Zukunft eine wesentlich neuere jQuery-Version geben, sind Probleme jedoch nicht auszuschließen. Verwenden Sie dann die Version aus dem Begleitmaterial.

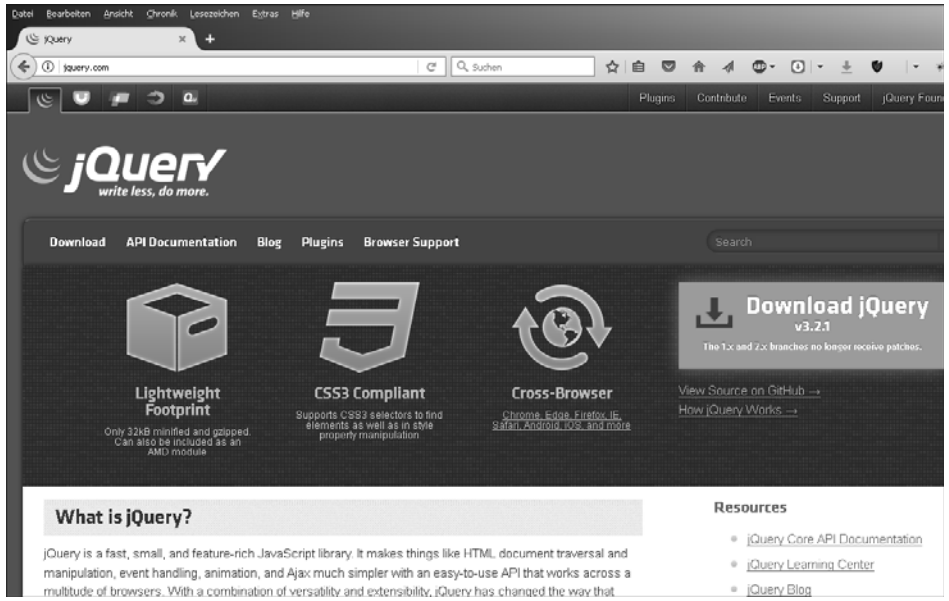

Abbildung 3.1 Download von »jquery.com«

Es gibt nun also, wie bereits gesagt, zwei verschiedene Versionen von jQuery. Einmal die Version *uncompressed*, d. h. die unkomprimierte Datei, und einmal die Version *compressed*, also eine verkleinerte Datei. Sind Sie daran interessiert, einmal einen Blick in das »Getriebe« von jQuery zu werfen? Dann nehmen Sie die unkomprimierte Datei zur Hand. Im Normalfall werden Sie das, selbst als Entwickler, kaum tun wollen, sondern nur die angebotenen Funktionen im Produktiveinsatz nutzen. In diesem Fall verwenden Sie die kompaktere *compressed*-Datei, deren Größe weniger als die Hälfte der unkomprimierten Datei beträgt.

3.2.1 jQuery 1.x , 2.x oder 3.x – was denn nun?

Mit dem Erscheinen von jQuery 1.9 wurden die ersten von ein paar harten Schnitten vorgenommen. Es wurden Methoden und Eigenschaften, die schon lange als *deprecated* markiert waren, aus dem Framework entfernt. Daneben wurden noch diverse Rückgabewerte von Methoden verändert, das Pseudo-Event *Hover* wurde entfernt (nicht zu verwechseln mit `.hover()`, das bleibt erhalten), und einige andere kleine Altlasten wurden über Bord geworfen. Einen vollständigen Artikel des jQuery-Teams zu diesem Thema finden Sie hier: *http://jquery.com/upgrade-guide/1.9*

In Tabelle 3.1 sehen Sie eine Auflistung der wichtigsten Änderungen in Bezug auf die entfernten Methoden.

Methode	Grund
`.toggle(fn,fn,…)`	Verwechslungsgefahr mit gleichlautender Methode `.toggle([duration] [,complete ])`, die ein Element sichtbar und unsichtbar schaltet.
`jQuery.browser()`	Seit der Version 1.3 deprecated, da es dem Prinzip der Feature Detection widerspricht.
`.live()`	`.live()` wurde zu Gunsten des moderneren Konzepts von `.on()` entfernt.
`.die()`	`.die()` wurde auf Grund des moderneren Konzepts von `.off()` entfernt (siehe `.live()`).
`jQuery.sub()`	Die Zahl der Anwendungsfälle, für die diese Methode genutzt werden könnte, ist zu gering, um sie weiter mit herumzuschleppen.

Tabelle 3.1 Entfernte Methoden in jQuery 1.9

In der aktuellen Version 3.0 wurden ebenfalls noch einmal einige, vorher schon als veraltet markierte Methoden entfernt.

In Tabelle 3.2 sehen Sie eine Auflistung der wichtigsten Änderungen in Bezug auf die in Version 3.0 entfernten Methoden.

Methode	Grund
`.error()`	Da die Methode nur eine Kurzform der Variante `.on("error", handler)` darstellt, wurde sie aus Gründen der Stringenz entfernt.
`.load()`	Da die Methode nur eine Kurzform der Variante `.on("load", handler)` darstellt, wurde sie aus Gründen der Stringenz entfernt. Zudem konnte `.load()` mit der gleichnamigen Methode des Ajax-Moduls verwechselt werden.
`.size()`	`.size()` lieferte die Anzahl der Elemente in der aktuellen Selection. Dafür existiert die native JavaScript-Eigenschaft `.length`, die stattdessen verwendet werden soll.
`.unload()`	Da die Methode nur eine Kurzform der Variante `.on("unload", handler)` darstellt, wurde sie aus Gründen der Stringenz entfernt.
`.andSelf()`	Die Methode wurde seit Version 1.8 als veraltet markiert und wird durch die Methode `.addBack()` vollständig ersetzt.

Tabelle 3.2 Entfernte Methoden in jQuery 3.0

Aber was tun Sie, wenn Sie eine der entfernten oder geänderten Methoden weiterhin in einem Projekt benötigen? Wenn Sie beispielsweise ein älteres Plugin aus verschiedensten Gründen weiter nutzen müssen? Sich ärgern? Nein. Dafür haben die Macher von jQuery das Plugin *migrate.js* entwickelt. Binden Sie dieses Plugin unterhalb des eigentlichen Frameworks ein, und Ihnen stehen alle Methoden und Eigenschaften der »alten« API zur Verfügung. Ein Beispiel einer Einbindung finden Sie weiter unten.

Darüber hinaus wurde in der Version 2.0 ein harter Schnitt vorgenommen: Die Unterstützung für ältere Browser wurde gekappt. Die Version 2.0 unterstützt lediglich den IE ab der Version 9.x und höher sowie bei Google Chrome, Firefox und Opera die aktuelle und eine Vorgängerversion. Safari wird ab der Version 5.x unterstützt. Es wurden sämtliche Browserquirks und Hacks konsequent herausgenommen; so wurde die Performance beträchtlich erhöht. Die Version 1.x wurde dennoch weiterentwickelt – derzeit, Stand Oktober 2017, sind wir hier bei den Versionen 1.12 und 2.2 angelangt. Diese beiden Versionen 1.x und 2.x werden in der Zukunft aber nicht mehr weiterentwickelt. Wenn Sie immer noch relativ alte Systeme unterstützen müssen, dann können Sie das mittels des Plugins migrate.js tun, Sie sollten aber möglichst bald auf Version 3.x updaten. Und das Plugin migrate.js erhält Ihnen auch noch ältere Features. Was will man mehr? Das jQuery-Team hat eben auch an die gedacht, die

ältere Browser oder ältere Scripte bedienen müssen, und einen intelligenten Upgradepfad bereitgestellt.

Die Version 3.x unterstützt (Stand Oktober 2017) folgende Browser: Chrome ab Version 60, Microsoft Edge ab Version 39, Firefox ab Version 55, Internet Explorer ab Version 9, Safari ab Version 10 und Opera ab Version 45.

3.2.2 jQuery online und offline nutzen

Vielleicht zunächst eine kleine Klärung im Vorfeld: Benötigen wir zum Test von jQuery eigentlich einen Webserver oder nicht? Man könnte antworten, im Prinzip nein. Jedoch kann in diesem Fall dann auch nur ein Teil der jQuery-Funktionalität genutzt werden: Einige der Tricks, die das Framework zur Verfügung stellt, basieren auf der Nutzung von Ajax, was in der Regel einen Server voraussetzt, der die HTTP-Anfragen interpretiert, die das Framework dann absetzen wird.

Benötigen wir kein Ajax, kann auch direkt aus dem Dateisystem heraus (sozusagen »offline«) gearbeitet werden: Sie können beispielsweise ein HTML-Dokument unmittelbar aus dem Verzeichnis ins Browserfenster ziehen, um ein Script zu testen. Ansonsten müssten Sie einen lokalen Webserver zur Verfügung haben und die Übungsdateien in dessen Dokumentenverzeichnis ablegen (siehe Abschnitt 2.3, »Webserver«).

3.2.3 jQuery lokal einbinden

Sie benötigen von jeder HTML-Seite, in der Sie jQuery nutzen möchten, einen Link zur jQuery-Datei. Sie haben die Wahl zwischen der minimierten Version `jquery-1.12.4.min.js` und der unkomprimierten Version `jquery-1.12.4.js` bzw. `jquery-2.2.4.js` und `jquery-2.2.4.min.js` sowie `jquery-3.2.1.js` und `jquery-3.2.1.min.js`. Für die lokalen Übungen in diesem Buch macht es keinen Unterschied, ob Sie eine unkomprimierte oder eine komprimierte Version verwenden. Legen Sie einfach beide Dateien in einem Unterverzeichnis des *htdocs*-Verzeichnisses des lokalen Servers auf Ihrem Rechner ab. (Wir nennen dieses Verzeichnis *buchbeispiele* – selbstverständlich können Sie auch jeden beliebigen Namen wählen.)

Wir empfehlen für die Arbeit mit den Beispielen dieses Buchs folgende Ordnerstruktur:

```
buchbeispiele/
    lib/
        jquery-x.x.js
        jquery-x.x.min.js
    _beispiel_/
    ...
```


Dies erleichtert die kapitelübergreifend konstante Einbindung des Frameworks, dessen Dateien im Ordner *jquery* abgelegt werden (dies ist der Bezeichner, den wir für dieses Buch gewählt haben). Parallel zu diesem Ordner legen Sie einen oder mehrere Ordner mit Ihren Test- und Übungsdateien an. Ressourcen, die von den HTML-Seiten benötigt werden (CSS- und Grafikdateien), platzieren Sie sinnvollerweise ebenfalls in die Übungsordner.

Dies ermöglicht eine stets gleichförmige Formulierung des Links zur jQuery-Datei im Dokumentkopf der Übungsdateien:

```
<script type="text/javascript"
        src="../lib/jquery-3.2.1.js"></script>
```

Eine ähnliche Anordnung empfiehlt sich auch für Projekte, in denen Sie jQuery einsetzen (erinnern Sie sich aber daran, für Produktionszwecke auf die minimierte Version der jQuery-Datei zurückzugreifen). Reproduzieren Sie für den Onlinezugriff die gleiche Verzeichnisstruktur auf Ihrem Produktionsserver. Sie können dann die relativen Links zum Framework unverändert beibehalten.

Das Plugin migrate.js binden Sie folgendermaßen ein:

```
<script type="text/javascript"
        src="../lib/jquery-3.2.1.js"></script>
<script type="text/javascript"
        src="../lib/jquery-migrate-3.0.0.js"></script>
```

Moderne HTML5-Seiten und auch HTML5-Templates wie HTML5 Boilerplate sind dazu übergegangen, die Scripte vor dem Ende des Dokuments einzubinden:

```
<html>
<head>...</head>
<body>
<navigation>...<navigation>
<article>...</article>
<script type="text/javascript"
        src="../lib/jquery-1.12.4.js"></script>
<script type="text/javascript"
        src="../lib/jquery-migrate-3.0.0.js"></script>
<script type="text/javascript"
        src="../js/eigenes-script.js"></script>
</body>
</html>
```

Die Begründung ist, dass das Laden und das Ausführen der Seite auf diese Weise beschleunigt werden. Entscheiden Sie selbst, beide Methoden sind gültig.

3.2.4 jQuery aus dem Google Online Repository einbinden

Eine Alternative zur lokalen Einbindung von jQuery ist die Nutzung des *Google Online Repository*, das ein frei zur allgemeinen Nutzung zur Verfügung stehendes Framework anbietet. (Gedacht ist dieses Repository für den Live-Betrieb einer Website. Sie können das Prinzip jedoch auch für die Buchbeispiele einsetzen, sofern Ihr Rechner jederzeit über eine Online-Verbindung verfügt.)

Das *Content Delivery Network* (CDN) von Google hält neben jQuery auch andere JavaScript-Frameworks bereit, wie Prototype, MooTools, Dojo und Ext JS. Im Falle von jQuery 3.2.1 geschieht die Einbindung wie folgt:

```
<script  type="text/javascript" src=" https://ajax.googleapis.com/ajax/libs/
jquery/3.2.1/jquery.min.js"></script>
```

Man kann auch einen Schritt weiter gehen und jQuery über den `load`-Befehl der *Google JavaScript API* einbinden. Hierfür muss allerdings zunächst eben diese API verlinkt werden. Anschließend steht die Methode `google.load()` zur Verfügung, mit der jQuery ebenfalls bereitgestellt werden kann.

Deren erstes Argument bestimmt das Framework. Achtung: Der zweite Parameter, der die gewünschte Version nennt, ist obligatorisch und kann daher nicht einfach weggelassen werden! Mit der folgenden Anordnung wird die minimierte Version von jQuery 1.10.2 aus dem Google CDN geladen:

```
<script type="text/javascript"
        src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
        google.load("jquery", "3.2.1");
</script>
```

Wollen Sie die unkomprimierte Variante einbinden, fügen Sie noch ein drittes Argument hinzu (achten Sie auf die geschweiften Klammern):

```
<script type="text/javascript">
        google.load("jquery", "3.2.1",{uncompressed:true});
</script>
```

Pro und Kontra

Diese Lösung hat wie so vieles ihre Licht- und Schattenseiten. Als Vorteil könnte man werten, dass man nicht gezwungen ist, jQuery auf dem eigenen Server abzulegen. Die Anbieter werden zudem meist mehrere Versionen des Frameworks (darunter sicherlich die jeweils aktuelle) zur Verfügung stellen. Bei einer stehenden Onlineverbindung steht auch einer Nutzung von Repositories im Rahmen von ansonsten lokalen Tests nichts im Wege.

Ein Nachteil ist, dass die Funktion der eigenen Website von der konstanten Bereitstellung des Frameworks durch eine andere Partei abhängig ist. Bei Microsoft oder Google kann man immerhin davon ausgehen, dass der Service dauerhaft zur Verfügung stehen wird. Nicht vergessen sollte man jedoch, dass die Einbindung einer extern gehosteten Datei auch ein Protokollieren der Nutzung der eigenen Website durch den Provider des Frameworks ermöglicht (ohne dies unterstellen zu wollen). Ob man sich darauf einlassen möchte, sei dem Einzelnen überlassen. (Wir werden in diesem Buch die konventionelle Einbindung verwenden und jQuery aus dem lokalen Verzeichnis einbinden.)

3.2.5 Das Beste aus beiden Welten

Es gibt auch die Möglichkeit, jQuery über das Google CDN zu laden und einen Fallback für den Fall bereitzustellen, dass kein Internetzugang zur Verfügung steht:

```
<script type="text/javascript" src=" https://ajax.googleapis.com/ajax/libs/
  jquery/3.2.1/jquery.min.js ">
</script>
<script>window.jQuery || document.write('<script src="../lib/
  jquery-3.2.1.min.js">
<\/script>')
</script>
```

Sie laden jQuery über das Google CDN. Anschließend fragen Sie mit `window.jQuery` ab, ob das jQuery-Objekt vorhanden ist. Ist es das nicht, schreiben Sie per JavaScript mit `document.write()` den lokalen Pfad zum Framework. So ist sichergestellt, dass auch beim lokalen Testen oder bei Netzwerkfehlern seitens Google immer eine Version geladen werden kann.

3.3 Das erste richtige Beispiel mit jQuery

Legen Sie die heruntergeladene Framework-Datei wie beschrieben in ein Verzeichnis parallel zu dem, in dem die HTML-Seite gespeichert ist. Bevor Sie nun das Script konstruieren, müssen Sie erst das Framework in die Seite einbinden. Ähnlich einem externen Stylesheet wird die Datei in das Hauptdokument inkludiert.

Im ersten `<script>`-Element geben Sie den Pfad zur Bibliothek an:

```
<script type="text/javascript"
        src="../jquery/jquery-3.2.1.min.js"></script>
```

Wo genau soll die Einbindung im Quelltext erfolgen?

Unser Tipp: Wenn Sie das jQuery-Framework im `<head>`-Verzeichnis inkludieren, dann stets nach der Einbindung der CSS-Dateien. Achten Sie darauf, dass Ihre eigenen JavaScript-Funktionen bzw. -Dateien wiederum stets nach dem Framework eingebunden werden.

Legen Sie nun ein weiteres `<script>`-Element an, und fügen Sie die jQuery-Funktionen ein. Das gesamte Beispiel sieht jetzt folgendermaßen aus (der jQuery-Code ist fett hervorgehoben):

```
<html>
<head>
   <title>Listing 2</title>

   <!-- Zunächst die Style-Angaben: -->
   <style type="text/css">
      .green {
         color:#009933;
      }
   </style>
   <!-- Nun das Framework einbinden: -->
   <script type="text/javascript"
           src="../jquery/jquery-3.2.1.min.js"></script>

   <!-- ... und jetzt unseren eigenen Code: -->
   <script type="text/javascript">
      $(document).ready(function() {
         $("#box p:contains('Zweiter Absatz')")
          .addClass("green");
      });
   </script>
</head>
<body>
   <div id="box">
      <p>Erster Absatz</p>
      <p>Zweiter Absatz</p>
      <p>Dritter Absatz</p>
   </div>
</body>
</html>
```

Listing 3.2 Schriftfarbe ändern mit jQuery

Das jQuery-Objekt als Factory-Funktion

Es springt sofort ins Auge, dass gerade einmal drei Zeilen Code benötigt werden, um das gesteckte Ziel zu erreichen. Das fundamentale Konstrukt in jQuery ist die Funktion `$()`. Es handelt sich dabei um eine so genannte Factory-Funktion – sie wird so bezeichnet, weil sie das erzeugt, was anschließend als Arbeitsgrundlage dient.

Zum Vergleich: Die im vorigen Listing eingesetzten DOM-Funktionen `getElementById()` und `getElementsByTagName()` holen nur existierende Bestandteile des Dokuments, verändern diese aber nicht. Sie müssen mit den »natürlichen« Eigenschaften der gefundenen Dokumentknoten zurechtkommen.

Anders bei der `$()`-Funktion. Diese Funktion erhält zwar auch ein Argument, das bestimmt, womit die Funktion arbeitet. Hier ist es jedoch das gesamte Dokument – Sie werden aber auch noch andere Fälle kennenlernen. Allerdings begnügt sich die `$()`-Funktion nicht damit, sich das bezeichnete Objekt zu beschaffen, sie stattet dieses mit weiteren Fähigkeiten aus. Diese neuen Fähigkeiten umgeben das alte Objekt wie ein Mantel – man spricht daher von *Wrapping*. Das so erzeugte neue Objekt wird (unabhängig davon, welche Art von Objekt als Grundlage fungiert) als *jQuery-Objekt* bezeichnet.

Mit dem Aufruf von `$()` wird also immer ein neues jQuery-Objekt erzeugt und zurückgegeben. Danach steht es mit seinen ganzen Methoden und Eigenschaften für das weitere Scripting zur Verfügung. Eine dieser Methoden nennt sich `.ready()`. Sie wird allerdings speziell dann verwendet, wenn das jQuery-Objekt aus einem `document`-Objekt gebildet wurde (zugegeben, ein Sonderfall).

Jetzt wissen Sie für den Anfang genug – analysieren wir nun den hervorgehobenen Code aus dem vorigen Listing Zeile für Zeile:

1. Sobald das Dokument fertig geladen wurde, ...

   ```
   $(document).ready( ... );
   ```

2. ... führen Sie die in `ready()` befindliche anonyme Funktion aus, ...

   ```
   $(document).ready( function() {
         ...
   });
   ```

3. ... die besagt: Bilden Sie aus allen `<p>`-Elementen mit dem Inhalt `'Zweiter Absatz'` innerhalb des `<div>`-Containers mit der ID `box` ein jQuery-Objekt:

   ```
   $(document).ready( function() {
      $("#box p:contains('Zweiter Absatz')")
         ...
   });
   ```

4. Wenden Sie nun auf dieses die Funktion `addClass()` an, um den `<p>`-Containern die Klasse `green` hinzuzufügen (wir machen wieder einen Zeilenumbruch vor dem Punktoperator – das ist nicht nur für das Listing im Buch praktisch):

```
$(document).ready( function() {
    $("#box p:contains('Zweiter Absatz')")
      .addClass("green");
});
```

Das war's. Sie brauchen sich nicht mit Schleifen und Bedingungen herumzuschlagen. Stattdessen navigieren Sie mittels des Selektors `"#box p:contains('Zweiter Absatz')"` direkt zum gewünschten Element, verkapseln es als jQuery-Objekt und verändern es. (Übrigens, der Filterpseudoselektor `:contains()` ist eine Dreingabe von jQuery, die Ihnen in reinem CSS nicht zur Verfügung steht.)

Die gewünschte Veränderung geschieht durch den Aufruf von `addClass()`. Mit dieser Methode können Sie beliebige CSS-Klassen an beliebige HTML-Elemente binden. (Dass die Klasse mit ihren Eigenschaften vorher im Stylesheet definiert werden muss, versteht sich von selbst.)

Warum muss es ein jQuery-Objekt sein?

Die *Verkapselung* in ein jQuery-Objekt ist in diesem Fall der springende Punkt. Den `<p>`-Container bekämen Sie zwar auch anders zu fassen (wie zuvor bereits beschrieben), aber er wäre eben nur ein `<p>`-Container. Versuchten Sie, auf ihn *direkt* die Methode `addClass("green")` anzuwenden, hätten Sie Pech. Anders sieht es aus, wenn der `<p>`-Container »jQuerifiziert« wurde (in ein jQuery-Objekt verpackt ist). In diesem Fall stehen ihm unmittelbar alle Methoden von jQuery zur Verfügung.

Das jQuery-Objekt als Knotenliste

Wir haben das jQuery-Objekt als Wrapper für ein Einzelobjekt kennengelernt. Mit Hilfe dieser Verkapselung stehen Ihnen für dieses Objekt alle jQuery-Tricks zur Verfügung. Ein Einzelobjekt war es aber schlicht deshalb, weil der an `$()` übergebene Ausdruck lediglich ein Objekt als Ergebnis hatte.

Wenn dies aber nicht so eindeutig ist, was dann? Ganz einfach – jQuery macht aus *allen* übergebenen Objekten (nehmen wir mal an, es seien Elementknoten) *gemeinsam* ein jQuery-Objekt. Dieses nimmt dann, wie man sagt, die Eigenschaften einer »Liste« an. Im herkömmlichen JavaScript spricht man von einer Knotenliste (*Node-List*). In jQuery läuft dies darauf hinaus, dass für *jedes* Element der Liste alle jQuery-Optionen zur Verfügung stehen.

Legen wir einmal mehrere Elemente im jQuery-Objekt ab, indem wir jetzt *alle* Absätze innerhalb des Containers `#box` selektieren. Hierfür genügt es, den Filter

`:contains('Zweiter Absatz')` wegzulassen.[1] Damit ist die Einschränkung auf den zweiten Absatz aufgehoben:

```
$(document).ready(function() {
    $("#box p").addClass("green");
});
```

Das Ergebnis ist wenig spektakulär, vielleicht sogar als »intuitiv vorhersehbar« zu bezeichnen: Alle Textabsätze erhalten die grüne Schrift und Hintergrundfarbe.

Obwohl … das bedeutet doch, dass jQuery *alle* Elemente seiner Liste nimmt und auf *jedes einzelne* die angehängte Methode anwendet? Genau: jQuery arbeitet eine Knotenliste Item für Item ab, und zwar vollautomatisch. Behalten Sie das im Hinterkopf – Stichwort: implizite Schleife.

Welches Argument erwartet die Funktion $()?

An dieser Stelle möchten wir kurz auf die Frage eingehen, *was* für ein Argument die `$()`-Funktion eigentlich erwartet. Wir haben gesagt, dass es sich um CSS-Selektoren handelt, teilweise noch mit jQuery-spezifischen Erweiterungen. Es geht aber auch noch mehr, wie Sie später noch sehen werden. Bleiben wir jedoch zunächst bei CSS.

jQuery unterstützt im Rahmen der `$()`-Funktion nahezu alle CSS-Selektoren, von CSS 1.0 bis CSS 3.0. Dies macht es für Webworker mit CSS-Erfahrung leichter, sich in jQuery einzuarbeiten. Wenn nicht, in Anhang A, »HTML und CSS«, stellen wir Ihnen die wichtigsten Grundbegriffe vor.

3.4 Wir haben fertig

Gehen wir kurz auf die `.ready()`-Methode im folgenden Ausdruck ein:

```
$(document).ready(fn)
```

Bei `.ready()` handelt es sich um eine grundlegende Methode innerhalb des Event-Moduls von jQuery: Eine Funktion `fn`, die an `.ready()` übergeben wird, führt jQuery aus, sobald das Dokument geladen wurde. In unserem Fall ist es eine anonyme Funktion, die die Dokumentabfrage enthält.

Der Grund für dieses Manöver ist folgender: Wenn Sie die Anweisung `$("#box p").addClass("green")` unmittelbar auswerteten, erzielten Sie keine Treffer – in diesem Moment gäbe es schlicht noch kein Dokument, aus dem das Script den `<p>`-Container holen könnte. Das ist nämlich noch gar nicht geschrieben. Also muss abgewartet werden. JavaScript bietet uns mit `window.onload` zu diesem Zweck bereits von Haus aus

1 Wir könnten hier sogar noch einfacher `$('p').addClass('green')` schreiben. Schön, nicht?

einen passenden Event-Handler an. Ist `fn` die aufzurufende Funktion, schreiben Sie mit seiner Hilfe:

```
window.onload = fn;
```

Den gleichen Dienst leistet der `onload`-Event-Handler im `<body>`-Element. Wo ist der Unterschied zum (zudem komplizierter zu schreibenden) Ansatz von jQuery? Der Unterschied ist das Timing: Mit `window.onload` wird die Funktion `fn` erst aufgerufen, wenn alle Abhängigkeiten des Dokuments aufgelöst sind, beispielsweise die Grafiken geladen wurden. Die Grafiken brauchen Sie aber gar nicht, wenn Sie nur an die Elemente wollen!

Der Vorteil von `$(document).ready(fn)` besteht darin, dass das Script bereits dann die Callback-Funktion `fn` ausführt, wenn die für Sie relevante HTML-Struktur bereitsteht. Das DOM ist dann zwar nur »weitestgehend geladen«, aber Sie können auch schon früher darauf zugreifen. Ein Problem ist das nicht – das Laden der Bilder können wir bei der Analyse und Manipulation des HTML-Dokuments in vielen Fällen vernachlässigen.

Sie könnten beliebig oft `$(document).ready(fn)` in einer Seite ausführen lassen. Im Endeffekt mag das aber ein wenig unübersichtlich werden. Hier besteht der Hintergedanke primär darin, Funktionscode aus dem `ready`-Block auszulagern, um diesen möglichst einfach zu halten (sonst könnten die Funktionsblöcke von `tuDies()` und `tuJenes()` auch ebenso im `ready`-Block selbst stehen – der würde dann allerdings möglicherweise ziemlich lang werden):

```
// definieren, was tuDies() machen soll:
function tuDies() {
   $("div p").click(function(){ ... });
}

// erster ready()-Block:
$(document).ready( function() {
   // und tuDies() aufrufen, wenn's so weit ist:
   tuDies();
});

// definieren, was tuJenes() machen soll:
function tuJenes() {
   $("ul li").click(function(){ ... });
}

// zweiter ready()-Block:
$(document).ready(function() {
```



```
// und tuJenes() aufrufen, wenn's so weit ist:
   tuJenes();
});
```

Wie gesagt, das geht, bietet aber keine Vorteile. Eine Funktion aus dem `ready`-Block auszulagern, ist jedoch immer eine gute Sache. Sie sehen auch, dass Sie dort ebenfalls die jQuery-Schreibweise verwenden dürfen. (Dies ist also nicht etwa auf den `ready`-Block beschränkt, um es einmal deutlich zu sagen.)

Um Übersicht zu schaffen, beschränken Sie sich besser auf nur eine einzige `ready`-Anweisung. Innerhalb der anonymen Callback-Funktion dieser Anweisung (von der es nur eine geben darf!) können Sie »huckepack« beliebig viele Funktionen aufrufen:

```
$(document).ready(function() {
   tuDies();
   tuJenes();
   // etc.
});
```

3.5 Das Mausereignis – Bindung eines Click-Events

Nun macht unser allererstes Beispiel scheinbar nicht allzu viel Sinn, da Sie doch Stileigenschaften allein mittels Stylesheet festlegen könnten,[2] ohne JavaScript zu Hilfe zu rufen. Geschenkt – wenn Sie eine solche Manipulation aber mit einem Mausereignis verbinden, wird es interessanter: Sie wollen als Nächstes erreichen, dass ein Absatz seine Schriftfarbe ändert, sobald Sie ihn anklicken. Gehen wir kurz durch, wie dies ohne die Hilfe von jQuery erfolgen könnte.

3.5.1 Zunächst – die »aufdringliche« Variante

Warum wir dies mit »aufdringlich« betiteln, wird gleich klarer. Jedenfalls werden Aktionen wie der Klick auf ein Element gewöhnlich mit Hilfe von Event-Handlern erfasst. Diese werden wie Attribute in das Start-Tag des Elements geschrieben.

Um die `<p>`-Container klickbar zu machen, genügt dann folgender Code:

```
<div id="box">
      <p onclick="farbwechsel(this)">Erster Absatz</p>
      <p onclick="farbwechsel(this)">Zweiter Absatz</p>
      <p onclick="farbwechsel(this)">Dritter Absatz</p>
</div>
```

2 Bedenkt man aber, dass Sie den Style anhand des Elementinhalts binden können, ist es vielleicht doch ganz spannend, oder?

Was an dieser Stelle kurz erläutert werden soll, ist das Schlüsselwort `this`. Der Ausdruck stellt ein Objekt dar, das an die aufgerufene Funktion `farbwechsel()` übergeben wird. Es bezieht sich jeweils auf das `<p>`-Element, das gerade angeklickt wurde. Es handelt sich hier um das so genannte Kontextobjekt (also das Objekt, an dem aktuell etwas passiert). So weit, so gut – auch die Funktion `farbwechsel()` ist nicht sonderlich kompliziert:

```
    // das übergebene this landet in der Variablen meinP:
function farbwechsel(meinP) {
    // der geklickte Absatz bekommt die Klasse zugewiesen:
    meinP.className = "green";
}
```

Erst einmal funktioniert alles. Sie sehen, dass Sie auf diese Weise beliebige HTML-Elemente mit einem *Click-Event* versehen können. Allerdings sollten Sie, um die Benutzerführung eindeutig zu halten, bei entsprechend »aktivierten« Elementen die Cursor-Eigenschaft per CSS auf `pointer` setzen, damit der Benutzer erkennt, dass hier eine Interaktion möglich ist.

Dies geschieht in unserem Beispiel in einer Styleregel für `<p>`-Container. Des Weiteren fügen Sie der Klasse `green` eine Hintergrundfarbe hinzu und setzen die Cursor-Darstellung dort wieder auf »normal« – immerhin braucht der Absatz, sobald er die Klasse hat, ja kein zweites Mal geklickt zu werden:

```
p {
    cursor:pointer;
}
.green {
    color:#009933;
    background-color:#E2FFEC;
    cursor:default;
}
```

3.5.2 Etwas weniger aufdringlich, bitte!

Was ist nun schlecht daran? Unschön ist, dass hier der Click-Event-Handler samt Funktionsaufruf in das HTML-Dokument geschrieben wurde, obwohl er nicht Teil der Informationsstruktur ist. Er ist außerdem überflüssig, wenn kein JavaScript aktiv ist (weil es abgeschaltet oder vom Client nicht unterstützt ist).

Im Sinne des Ansatzes *Unobtrusive JavaScript* (wir haben dazu im vorangegangenen Kapitel ein paar Worte verloren) bevorzugt man daher, den Event-Handler wegzulassen und die Klickfunktionalität per JavaScript nachträglich hinzuzufügen. Das HTML bleibt auf diesem Weg »sauber«.

Um einem Absatz »von außen« das Click-Event zuzuweisen, müssen zunächst alle `<p>`-Container innerhalb des Bereichs `#box` gesammelt werden.

Dies geschieht wie zuvor:

```
var meineP = document.getElementById("box")
             .document.getElementsByTagName("p");
```

Über das Array mit den Textabsatzknoten läuft eine Schleife, die das Click-Event bindet. Die Zahl der Knoten wird in einer Variablen `len` abgelegt, damit das Array nicht bei jedem Schleifendurchlauf erneut ausgelesen werden muss:

```
for(var i =0, len= meineP.length; i<len; i++) {
    // Fein. Jetzt das Click-Event binden. Aber wie?
   }
```

Für die Art und Weise, wie Sie das Click-Event »ordentlich« binden, gibt es verschiedene, untereinander unverträgliche Varianten: die offizielle und die für den Internet Explorer. Eine Fallunterscheidung ist möglich, aber umständlich. Der Rettungsanker findet sich beim guten alten DOM Level 0 und der dort definierten `onclick`-Eigenschaft. Diese Vorgehensweise ist zwar nicht zeitgemäß, hat aber den Vorteil, dass sie dann doch browserübergreifend funktioniert:

```
window.onload = function() {
    var meineP = document.getElementsByTagName("p");
    for(var i=0, len=meineP.length; i<len; i++) {
        meineP[i].onclick= function() {
                this.className = "green";
                }
    }
}
```

Dies ist in seiner Kompaktheit durchaus vertretbar. Sollte es noch schöner gehen? Oder zumindest kürzer?

3.5.3 Ein unaufdringlicher Dreizeiler, dank jQuery

Innerhalb von jQuery verwenden Sie nicht eine DOM-Eigenschaft namens `onclick`, sondern eine jQuery-Methode, die den (durchaus treffenden) Namen `click()` trägt. Mit ihrer Hilfe kann jedem Absatz innerhalb des Elements mit der ID `box` ein Click-Event-Handler hinzugefügt werden. Schreiben Sie also das Script einfach einmal entsprechend um. Ob Ihnen wieder drei Zeilen reichen werden? Na sicher!

```
$(document).ready( function() {
    $('#box p').click( function() {
        $(this).addClass("green");
    });
});
```

Dass `$('#box p')` in seiner Kürze dasselbe bewirkt wie der längere Ausdruck `document.getElementById("box").getElementsByTagName("p")`, haben Sie vielleicht bereits akzeptiert. Erinnern Sie sich, dass Sie hier auf die `for`-Schleife verzichten können: jQuery wendet ja im Fall einer Knotenliste (Sie wissen, dass es sich nicht um eine »gewöhnliche« Knotenliste handelt) eine Methode auf alle Items der Liste an (als »implizite Schleife«). Praktisch!

Sobald ein `<p>`-Element tatsächlich geklickt wird, tritt die anonyme Funktion im Inneren von `click()` in dessen Namen in Aktion. (Wie Sie bereits zuvor erfahren haben, bezeichnet man diese Funktion als *Callback-Funktion*. Als solche wird sie erst dann ausgeführt, wenn das Click-Event eintritt.)

Das `this` in ihrem Inneren bezieht sich auf den geklickten `<p>`-Container. Da dieses `this` (als herkömmlicher `<p>`-Container) mit `addClass()` nichts anfangen könnte, wrappen Sie es über die `$()`-Funktion wieder in ein jQuery-Objekt.

Unter die Lupe damit:

```
// die anonyme Funktion in click():
$('#box p').click( function() {
   // das Kontextobjekt this wird jQueryfiziert:
   $(this).addClass("green");
});
```

Hier folgt zusammenfassend der Quellcode der gesamten Datei:

```
<html>
<head>
   <title>Listing 3</title>

  <!-- Zunächst die Style-Angaben: -->
   <style type="text/css">
      p {
         cursor:pointer;
      }
      .green {
         color:#009933;
         background-color:#E2FFEC;
         cursor:default;
      }
```



```
   </style>

   <!-- Nun das Framework einbinden: -->
   <script type="text/javascript"
           src="../jquery/jquery-3.2.1.min.js"></script>

   <!-- ... und jetzt unseren eigenen Code: -->
   <script type="text/javascript">
      $(document).ready(function() {
        $("#box p").click(function() {
             $(this).addClass("green");
        });
      });
   </script>
</head>
<body>
   <div id="box">
      <p>Erster Absatz</p>
      <p>Zweiter Absatz</p>
      <p>Dritter Absatz</p>
   </div>
</body>
</html>
```

Listing 3.3 Schriftfarbe ändern mit Click-Event (mit jQuery)

Wollen Sie innerhalb eines klickbaren Textabsatzes die Koordinaten des Klick-Ereignisses erfassen, stoßen Sie auf eine weitere Diskrepanz zwischen den JavaScript-Implementierungen der Browser.

Der Standard verlangt die Übergabe der Informationen über ein Ereignis (u. a. Art und Ort des Ereignisses) an die durch das Ereignis getriggerte Funktion, die die Information auswerten kann. Die Informationsstruktur wird als ein JavaScript-Objekt übergeben, das allgemein als Event-Objekt bezeichnet wird. Es wird von der Zielfunktion durch einen Parameter (meist wird für diesen der Bezeichner `e` wie *event* verwendet) in den Argumentklammern aufgefangen:

```
meineP[i].onclick = function(e) { ... }, false);
```

Leider implementiert der Internet Explorer vor der Version 10 dies nicht auf die gleiche Weise. Nicht dass es hier keine Informationen über das Ereignis gäbe, das stattgefunden hat. Allerdings lagern diese in einem Objekt `event`, das dem `window`-Objekt unterstellt ist: `window.event`. Die Funktion bekommt keinen Wert übergeben, sondern muss sich diesen vielmehr holen.

In standardkonformen Browsern können Sie, um die Klickkoordinaten zu erfassen, Folgendes schreiben:

```
meineP[i].onclick = function(e) {
    // Zugriff auf das übergebene Event-Objekt e:
    alert('Klick an ' + e.clientX + ' und ' + e.clientY);
};
```

Im älteren Internet Explorer müsste dies wie folgt geschehen (Sie schreiben kurz `event` statt `window.event` und lassen dafür den Übergabeparameter in den Funktionsklammern weg – zum Glück sind wenigstens die Eigenschaften des Event-Objekts gleich benannt):

```
meineP[i].onclick = function() {
    // direkter Zugriff auf window.event:
    alert('Klick an ' + event.clientX +
          ' und ' + event.clientY);
};
```

Nun müssten wir, um zu entscheiden, nach welchem Modell gearbeitet wird, feststellen, ob der Funktion etwas übergeben wird oder nicht. Hierfür gibt es einen alten Programmierertrick, der den Übergabeparameter prüft, ohne umständlich auf eine `if-else`-Bedingung zurückzugreifen. Sie setzen einfach (sehr viel kürzer) den ternären Operator `? :` ein – hier ein erläuterndes Beispiel:

```
meineP[i].onclick = function(e) {
    // wurde ein e übergeben?
    e ? alert("e definiert!") : alert("e undefiniert!");
    // und nun weiter ...
};
```

Geprüft wird das übergebene `e`. Ist es `undefiniert` (wurde das Script also im IE ausgeführt) und damit `false`, wird der Ausdruck *rechts* vom Doppelpunkt verwendet, ansonsten (`e` wird zu `true` ausgewertet) *links* davon. Die Inkompatibilität überwinden Sie demnach, indem Sie, bei `false`, `e` gleich `window.event` setzen (ansonsten `e` einfach belassen) und anschließend nach »Schema F« fortfahren:

```
meineP[i].onclick = function(e) {
    e? e : e = window.event;
    // Prima, auch in IE heißt window.event jetzt e:
    alert('Klick an ' + e.clientX + ' und ' + e.clientY);
};
```


Abgesehen von dieser kleinen »Verrenkung« ist die Lösung nun recht einfach browserübergreifend zu schreiben. Der gesamte Code sieht jetzt so aus:

```
window.onload = function() {
  var meineP = document.getElementsByTagName("p");
  for(var i =0, len=meineP.length; i<len; i++) {
    meineP[i].onclick = function(e) {
      e? e : e = window.event;
      alert('Klick an ' + e.clientX + " und " + e.clientY);
      };
  }
}
```

Schauen Sie aber nun einmal, wie jQuery dies löst:

```
$(document).ready( function() {
    $('p').click( function(e) {
      alert('Klick an ' + e.clientX + " und " + e.clientY);
    });
});
```

Offensichtlich spart jQuery nicht nur (wie Sie bereits wissen) die `for`-Schleife ein, sondern sorgt obendrein dafür, dass die durch ein Ereignis getriggerte Funktion stets ein Event-Objekt übergeben bekommt. Die `$()`-Funktion bietet aber noch mehr, lassen Sie sich überraschen!

3.6 Give me more! – Verkettung von jQuery-Methoden

Eine angenehme Eigenheit von jQuery ist, dass Sie mehrere jQuery-Methoden hintereinander an ein jQuery-Objekt hängen können. Das Geheimnis besteht darin, dass jede jQuery-Methode wieder ein jQuery-Objekt zurückgibt.

Dieses jQuery-Objekt (es handelt sich um das gleiche wie am Anfang, nur mit den »eingearbeiteten« Veränderungen der eben angewendeten Methode) steht somit quasi »am Ausgang« zur Verfügung, so dass eine weitere Methode darauf angewendet werden kann. Und dies funktioniert immer so weiter. Sie können sich das wie ein Werkstück vorstellen, das eine Montagestraße entlangläuft:

```
// die jQuery-Methoden werden einfach verkettet:
$(selektorausdruck).methode1().methode2().methode3();
```

Das könnten Sie (nur zur Erinnerung) auch wie folgt schreiben (Vorsicht: Bloß kein Semikolon an die Zeilenenden setzen – außer ganz am Schluss!):

```
// die jQuery-Methoden werden einfach verkettet:
$(selektorausdruck)
  .methode1()
  .methode2()
  .methode3();
```

Aber schauen Sie es sich einfach einmal genau an: Sie haben bereits mit `$(this).addClass("green")` gesehen, dass Sie einem Element eine CSS-Klasse hinzufügen können. Lassen Sie uns dieses Beispiel erweitern.

3.6.1 Den Elternknoten eines Elements manipulieren

Sagen wir, wir wollen das Elternelement von `this` (den die `<p>`-Container umgebenden `<div>`-Container) mit den Klassen `boxColor-0` bis `boxColor-2` versehen.

Anklickbar sein sollen hierbei nach wie vor die Textabsätze selbst. Sie müssen es nun so einrichten, dass jQuery dem Div die CSS-Klasse abhängig davon zuweist, welcher `<p>`-Container angeklickt wurde. Das heißt, für den ersten `<p>`-Container erhält der Div die Klasse `boxColor-0`, für den zweiten vergeben Sie `boxColor-1` etc.

Bei jedem Klick muss die vorher an den Div vergebene Klasse allerdings entfernt werden. Ebenso soll nur der eben angeklickte `<p>`-Container die Klasse `green` besitzen, von allen anderen muss sie wieder verschwinden. Eine ganze Menge an Randbedingungen! Allerdings ist das halb so schlimm, wenn Sie sich Stück für Stück an die Lösung heranarbeiten.

Selektieren Sie zunächst alle `<p>`-Container in `Div#box`, und machen Sie sie anklickbar:

```
$("#box p").click( function() {
   ...
});
```

So weit, so gut. Der angeklickte `<p>`-Container soll die Klasse `green` erhalten. Diesen Container holen Sie mit `$(this)`. Auch das kennen Sie bereits:

```
$("#box p").click( function() {
   $(this).addClass("green");
});
```

Jetzt wird es interessanter – Sie haben den angeklickten `<p>`-Container bereits in der Hand, wollen jetzt aber etwas mit dessen Elternknoten, dem Div, machen. Gut, wechseln Sie einfach vom `<p>`-Container dorthin.

jQuery bietet Ihnen hierfür die Methode `parent()`. Weil Sie im Baum von einem Element zum anderen klettern (Bergsteiger bezeichnen dies als *traversieren*), zählt jQuery die `parent()`-Methode zu den *Tree-traversing-Methoden*.

Gehen Sie also zum Div:

```
$("#box p").click( function() {
   $(this).addClass("green").parent();
});
```

Nochmals zum Verständnis – der Ausdruck `$(this).addClass("green")`. Der `parent()` *ist* jetzt der `<div>`-Container. Kürzer hätten Sie `$(this).parent()` schreiben können, aber Sie hatten ja »unterwegs noch etwas zu erledigen« ...

Nun sorgen Sie dafür, dass am `<div>`-Container CSS-mäßig »Tabula rasa« gemacht wird: Mit `removeClass()` entfernen Sie eventuell dort befindliche CSS-Klassen:

```
$("#box p").click( function() {
   $(this).addClass("green").parent().removeClass();
});
```

Jetzt soll dem Div die gewünschte Klasse hinzugefügt werden. Dies machen Sie wiederum mit `addClass()`. Weichen Sie auf die mehrzeilige Schreibweise aus, und kommentieren Sie das Geschehen ein wenig mit:

```
$("#box p").click( function() {
   $(this)              // der angeklickte P-Container
    .addClass("green")  // erhält die Klasse green
    .parent()           // wir traversieren zum Elternknoten
    .removeClass()      // entfernen dort alle CSS-Klassen
    .addClass( ... );   // und fügen eine neue hinzu
});
```

Stopp – wo bekommen Sie nun den Klassennamen her, den Sie `.addClass()` übergeben müssen? Der Bezeichner beginnt immer mit `boxColor-` und geht dann mit einer der Ziffern `0`, `1` oder `2` weiter, die den `<p>`-Container bezeichnen. Das erinnert an die Indexziffern eines Arrays.

Handelt es sich hier denn um ein Array? Sozusagen. Und zwar, wenn Sie den grundlegenden Ausdruck `$("#box p")` nehmen. Von diesen drei `<p>`-Containern ist einer angeklickt worden, der im Rahmen unserer Funktion durch `this` bezeichnet wird. jQuery bietet Ihnen nun eine Methode `.index()`, die Ihnen die Position `pos` eines Objekts in einer Knotenliste nennen kann.

Das Prinzip ist dieses:

```
var pos = $(knotenliste).index(vergleichsobjekt);
```

Auf unseren Fall angewendet, schreiben Sie beispielsweise:

```
var pos = $("#box p").index(this);
```

Speichern Sie aber den Rückgabewert nicht, sondern setzen Sie ihn direkt ein:

```
$("#box p").click( function() {
   $(this)              // der angeklickte P-Container
    .addClass("green")  // erhält die Klasse green
    .parent()           // wir traversieren zum Elternknoten
    .removeClass()      // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" +  $("#box p").index(this));
});
```

Sie haben nun ganz vergessen, dafür zu sorgen, dass noch die Klasse `green` von vorher geklickten `<p>`-Containern entfernt werden muss. Dies erledigen Sie einfach pauschal zu Beginn der Funktion. Es macht nichts, dass einer der Absätze sofort wieder die gleiche Klasse erhält …

```
$("#box p").click( function() {
   // pauschal überall die Klasse green entfernen:
   $("#box p").removeClass("green");
   $(this)              // der angeklickte P-Container
    .addClass("green")  // erhält die Klasse green
    .parent()           // wir traversieren zum Elternknoten
    .removeClass()      // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" +  $("#box p").index(this));
});
```

Das ist jetzt insoweit schön, als es funktioniert. Allerdings ist es ein wenig unordentlich. Außerdem haben Sie nun dreimal ein jQuery-Objekt mit demselben Ausdruck `$("#box p")` gebildet. Das wirkt unökonomisch. Sie könnten dieses Objekt auch speichern, also in eine Variable packen, und »recyceln«:

```
var obj = $("#box p");
```

Sie erhalten dann:

```
var obj = $("#box p");
obj.click( function() {
   // pauschal überall die Klasse green entfernen:
   obj.removeClass("green");
   $(this)              // der angeklickte P-Container
    .addClass("green")  // erhält die Klasse green
    .parent()           // wir traversieren zum Elternknoten
    .removeClass()      // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" + obj.index(this));
});
```

Damit lässt sich schon besser arbeiten. Nun könnten Sie auch das ständige Hin und Her mit Entfernen und Hinzufügen von Klassen sortieren:


```
var obj = $("#box p");
obj.click( function() {
   // erst alle Klassen entfernen
   obj.removeClass("green").parent().removeClass();
   // jetzt Klassen hinzufügen:
   $(this).addClass("green").parent()
    .addClass("boxColor-" + obj.index(this));
});
```

Lagern Sie nun noch die Indexerzeugung aus dem unteren Ausdruck aus:

```
var obj = $("#box p");
obj.click( function() {
   // welcher P-Container wurde angeklickt?
   var pos = obj.index(this);
   // erst alle Klassen entfernen
   obj.removeClass("green").parent().removeClass();
   // jetzt Klassen hinzufügen:
   $(this).addClass("green").parent()
    .addClass("boxColor-" + pos);
});
```

So sieht die Funktion nun im Großen und Ganzen aus – das vorher Geschriebene muss lediglich noch in einen `ready`-Block eingefügt werden:

```
<html>
<head>
   <title>Listing 4</title>

   <!-- Zunächst die Style-Angaben: -->
   <style type="text/css">
      p {
         cursor:pointer;
      }
      .green {
         color:#009933;
         background-color:#E2FFEC;
         cursor:default;
       }
      .boxColor-0 {
         background-color:#CCCCCC;
      }
      .boxColor-1 {
         background-color:#EEEEEE;
      }
```

```
   .boxColor-2 {
      background-color:#999999;
   }
</style>

<!-- Nun das Framework einbinden: -->
<script type="text/javascript"
        src="../jquery/jquery-3.2.1.js"></script>

<!-- ... und jetzt unseren eigenen Code: -->
<script type="text/javascript">
   $(document).ready(function() {
      var obj = $("#box p");
      obj.click(function() {
         var pos = obj.index(this);
         obj.removeClass()
         .parent().removeClass();
         $(this).addClass("green")
         .parent().addClass("boxColor-" + pos);
      });
    });
</script>
   </script>
</head>
<body>
   <div id="box">
      <p>Erster Absatz</p>
      <p>Zweiter Absatz</p>
      <p>Dritter Absatz</p>
   </div>
</body>
</html>
```

Listing 3.4 Verkettung von jQuery-Funktionen

3.7 Zusammenfassung

jQuery stellt Ihnen ein wichtiges Event zur Verfügung, mit dem Sie überprüfen können, ob ein HTML-Baum fertig geladen wurde, und zwar `$(document).ready(fn)`, um erst anschließend, nach der Prüfung, eine Funktion auszuführen. Es stellt Ihnen ein einfaches Modell zur Navigation durch den Dokumentenbaum zur Verfügung, mit Hilfe von Selektoren, die sich an den Spezifikationen der CSS-Selektoren orientieren.

Sie können aus einem Arsenal an Werkzeugen wählen, mit denen Sie eine HTML-Seite manipulieren können, wobei Sie bei Ihren ersten Gehversuchen `.addClass(name)` kennengelernt haben. Sie können nun zudem mehrere jQuery-Methoden miteinander verketten. Alles in allem entsteht damit ein schlanker Code, und eine strikte Strukturierung Ihrer Projekte wird möglich, da Sie die HTML-Seite von allem möglichen Ballast befreien und diesen zu Ihrer besseren Übersicht in externe Dateien auslagern können.

Und wie jetzt weiterarbeiten?

Mit Kapitel 4, »jQuery – die Übersicht«, folgt ein längeres (zugegeben sehr langes) Kapitel, in dem wir Ihnen das gesamte Vokabular von jQuery vorstellen und es ausführlich besprechen. Irgendwo muss ja alles stehen. Sie können es natürlich hier und jetzt durchlesen, um einen Kompletteindruck zu erhalten. Vielleicht möchten Sie aber stattdessen die Ärmel hochkrempeln, die Siebenmeilenstiefel anziehen und direkt zum Praxisteil in Kapitel 5, »jQuery – der Praxiseinsatz«, springen. Dort werden die jQuery-Methoden angewendet, aber nicht erklärt. Erklärt werden natürlich die Beispiele. Sie können die vorgestellten Beispiele nachvollziehen und immer dann in den Referenzteil zurückblättern, wenn Sie Hintergrundinformationen benötigen.

Kapitel 6
jQuery UI

Das Paket jQuery UI ist ein Aufsatz auf das jQuery-Basisframework. Im Prinzip besteht UI aus Interaktionsroutinen und einer Reihe aufeinander abgestimmter, über CSS-Themes optisch konfigurierbarer Plugins, die Oberflächenelemente (so genannte »Widgets«) erzeugen.

Das jQuery UI steht für *jQuery User Interface*. Es ist eine Erweiterung des jQuery-Frameworks, das einerseits erweiterte Interaktion wie Effekte, Drag & Drop, Easing und Farbanimationen bereitstellt, andererseits komplexe Oberflächen-Controls (Widgets) wie Datepicker, Navigationen und Dialogboxen zur Auswahl bietet.

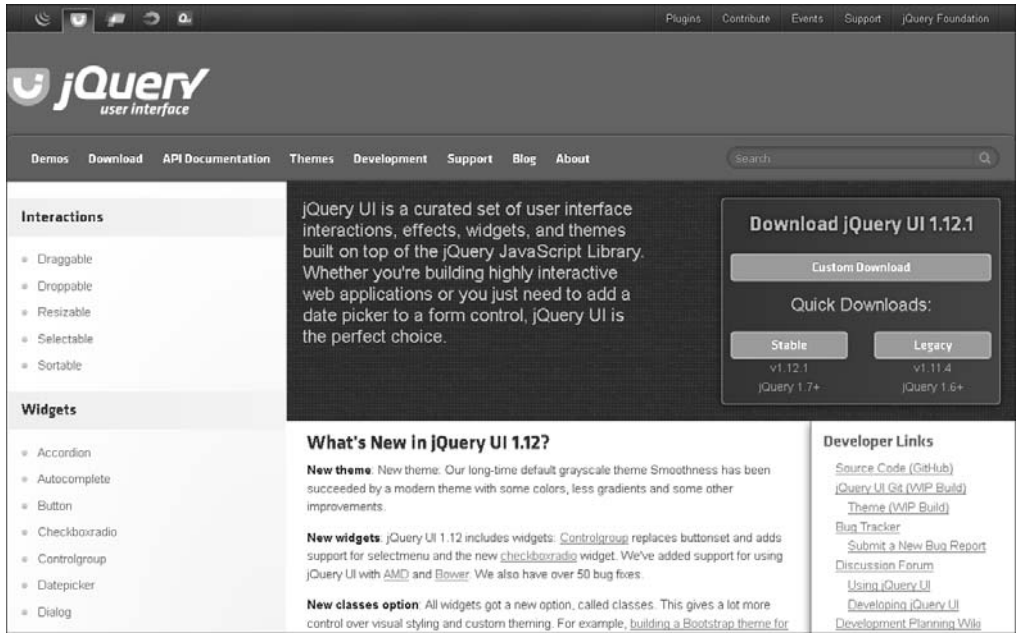

Abbildung 6.1 Die Website http://jqueryui.com

jQuery UI umfasst eine Sammlung von leicht individualisierbaren, einfach zu konfigurierenden Plugins. Im Einzelnen unterscheidet man folgende Bereiche.

Interaktion	**Drag/Drop, Größenänderung, Auswahlen, Sortierungen**
Widgets	Accordion, Autovervollständigung, Button, Datepicker, Dialog, Fortschrittsbalken, Schieberegler, Tabs
Utilitys	Positionierung, Mausinteraktion
Effekte	Die Methoden `effect()`, Farbanimationen mit `animate()`, `switchClass()`, Erweiterungen für Methoden `show()`, `hide()`, `toggle()`, `addClass()`, `removeClass()`, `toggleClass()`.

Tabelle 6.1 jQuery UI – Übersicht

6.1 Download und Konfiguration von jQuery UI

jQuery UI kann als Komplettpaket heruntergeladen und in vollem Umfang in ein Projekt eingebunden werden.

- Die aktuelle Version von jQuery UI ist 1.12.1 (247 KB).
- Die Legacy-Version von jQuery UI ist 1.11.4 (234 KB).

Beide Versionen werden gepflegt; in der Regel wird man die aktuelle Version bevorzugen (die ältere unterstützt noch IE7).

Abbildung 6.2 Download auf der Startseite

Der Nachteil des Komplettpakets besteht darin, dass man Features und Widgets downloadet und in die Projektdaten einbezieht, die möglicherweise nie verwendet werden. Die UI-Datei wird unnötig groß.

Aus diesem Grund ist ein frei konfigurierbarer *Custom Download* möglich, bei dem beliebig Elemente an- und abwählbar sind.

6.1.1 Der Download Builder von jQuery UI

Der Download Builder zum Erstellen eines Custom Downloads gilt sowohl für die aktuelle Version als auch das Legacy-UI.

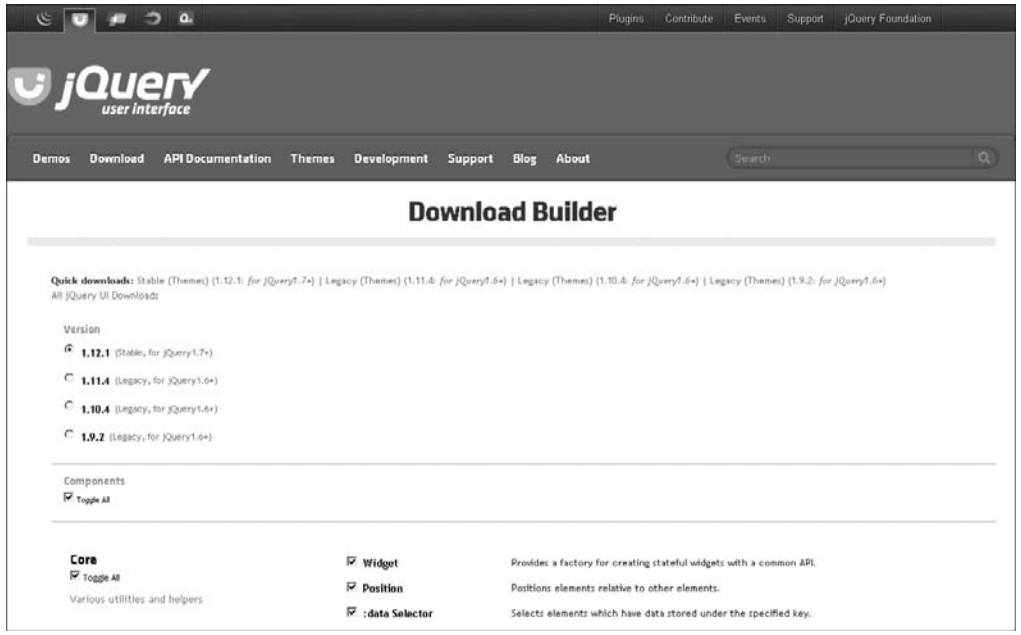

Abbildung 6.3 Der Download Builder von jQuery UI

Der Build ist unterteilt in folgende Bereiche:

- **UI Core** (Dienstfunktionen und Basis-Utilitys)
  Core, Widget, Mouse, Position
- **Interactions** (Verhalten für Elemente und Widgets)
  Draggable, Droppable, Resizable, Selectable, Sortable
- **Widgets** (Interface-Elemente, benötigen Core und Interactions)
  Accordion, Autocomplete, Button, Checkboxradio, Controlgroup, Datepicker, Dialog, Menu, Progressbar, Selectmenu, Slider, Spinner, Tabs, Tooltip
- **Effects** (autarke Effekte mit vielfältiger API)
  Effects Core, Blind Effect, Bounce Effect, Clip Effect, Color Animation, Drop Effect, Easings, Explode Effect, Fade Effect, Fold Effect, Highlight Effect, Puff Effect, Pulsate Effect, Scale Effect, Shake Effect, Slide Effect, Transfer Effect

Zwischen den Abteilungen bestehen *Abhängigkeiten* – so sind weder Interactions noch Widgets ohne (zumindest Teile des) UI Core anwendbar. Der Downloader warnt aber bei Diskrepanzen und wählt entsprechende Module selbstständig zu oder ab.

6.2 Theming von jQuery UI

Die Widgets von jQuery UI sind *themeable*, das bedeutet, es existieren vordefinierte CSS-Klassen innerhalb der HTML-Struktur des Widget-Markups. Auf diese Klassen sind CSS-Stylesheet-Dateien (Themes) abgestimmt, die im Rahmen des UI-Pakets mit heruntergeladen werden.

Sie können bereits beim Download im Builder ein Theme auswählen. Das Default-Theme trägt den Namen »Base«. Es handelt sich um ein extrem zurückhaltendes Farbschema. Alternativ sind auch ausgesprochen bunte Themes erhältlich.

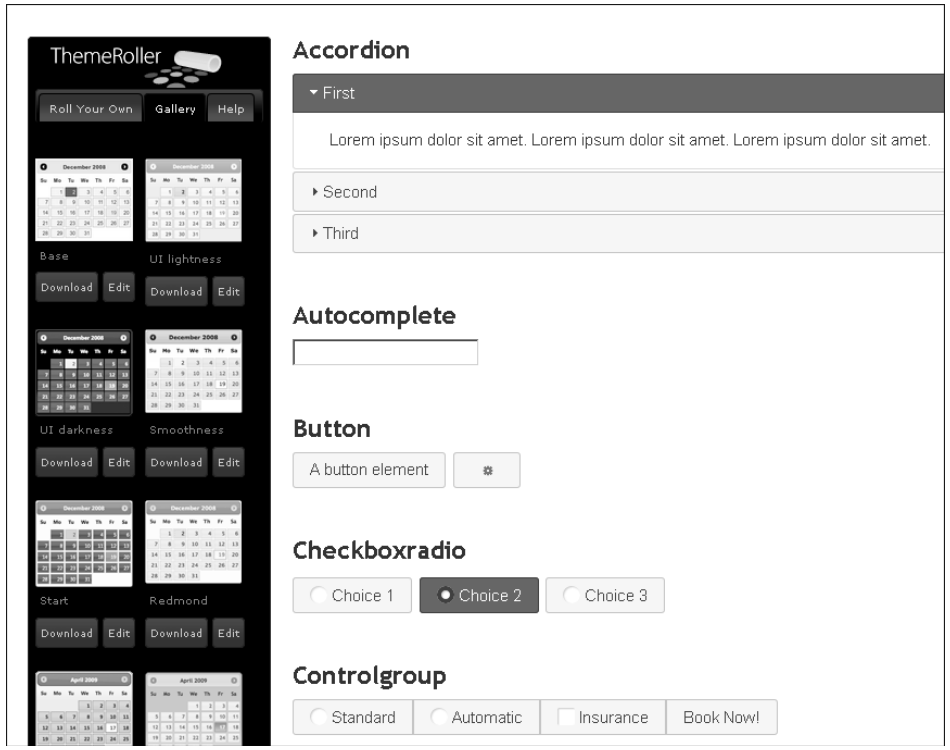

Abbildung 6.4 Auswahl eines Themes im ThemeRoller

Wenn Sie das Aussehen der Widgets mit einem bestimmten Theme begutachten möchten, wechseln Sie in die Abteilung *Themes*.

Hier haben Sie im Rahmen des *ThemeRollers* die Option, entweder ein komplett eigenes Theme zu definieren (planen Sie hierfür etwas Zeit ein) oder anhand einer Gallery die Wirkung der vordefinierten Themes zu begutachten.

Das gewählte Theme kann als Ausgangspunkt für eine Eigenkreation dienen. Sie können (u. a.) Schriften, Eckenradius, Farben von Kopf- und Fußleisten und anderen Bereichen frei wählen.

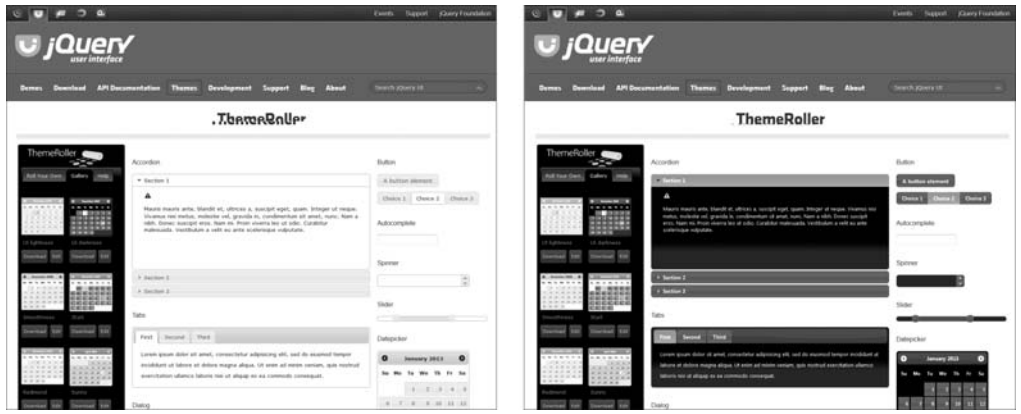

Abbildung 6.5 Die ThemeRoller Gallery – links »Smoothness«, rechts »Darkness«

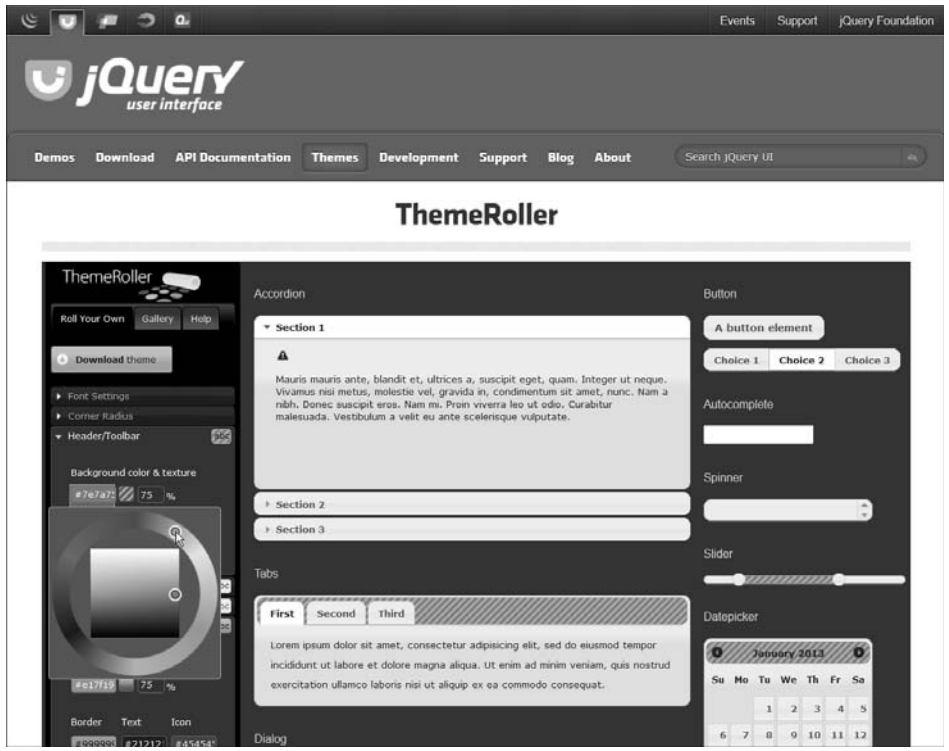

Abbildung 6.6 Farbwahl für den Header-Bereich eines eigenen Themes

Das fertige Theme steht nach Abschluss der Bearbeitung als Download zur Verfügung. (Sie wählen dann noch die UI-Komponenten dazu aus.)

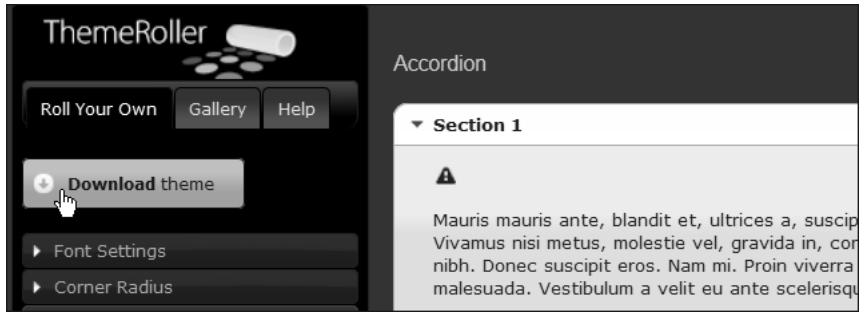

Abbildung 6.7 Download-Button des ThemeRollers

Bookmarken Sie Ihren Download

Bookmarken Sie die ThemeRoller-Seite vor dem Download. Sie haben so die Möglichkeit, den *Bearbeitungszustand* Ihres Themes zu speichern, um beispielsweise später daran weiterzuarbeiten oder es zu modifizieren.

Alle Theme-Parameter werden im *URI-String* festgehalten:

http://jqueryui.com/themeroller/#zThemeParams=5d00000100490600...

Es besteht ansonsten keine Möglichkeit, ein heruntergeladenes Theme zur Bearbeitung wieder upzuloaden (anders als bei jQuery Mobile).

6.2.1 Scoped Themes

Möchten Sie mehrere Themes gleichzeitig in Ihrem Dokument verwenden oder einfach zwischen zwei Themes wechseln können, so besteht die Möglichkeit, beim Download einen Scope (aka eine *Basisklasse*) für das Theme festzulegen.

Geben Sie den gewünschten Klassennamen einfach in das Feld CSS SCOPE ein. Vergessen Sie nicht den Punkt vor dem Bezeichner!

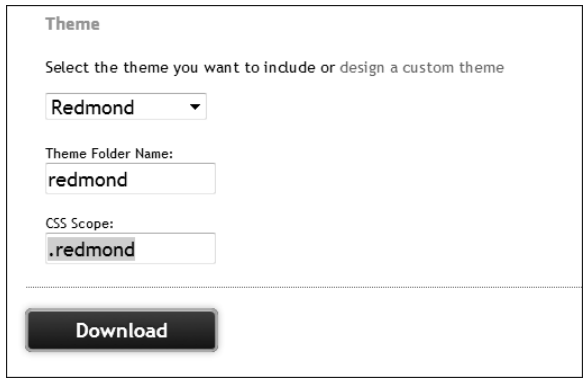

Abbildung 6.8 Angabe einer Basisklasse für einen Theme-Download

Der Effekt ist der, dass die Basisklasse allen Selektoren des Themes vorangestellt wird:

```
.redmond .ui-widget {
   font-family: Lucida Grande, sans-serif;
   font-size: 1.1em;
}
```

Dies gibt Ihnen die Möglichkeit, ein Theme durch Einsatz seiner Basisklasse auf Bereiche zu beschränken oder an- und abzuschalten:


```
<body class="redmond">
... überall im Dokument Theme "Redmond" ...
</body>

<body class="cupertino">
... überall im Dokument Theme "Cupertino" ...
</body>

...
<div class="redmond">... hier Theme "Redmond".</div>
<div class="cupertino">... hier Theme "Cupertino".</div>
...
```

Hinweis: Die von den UI-CSS-Dateien eingesetzten Grafiken sind farblich und dem Dateinamen nach auf das jeweilige Theme angepasst und liegen relativ zur CSS-Datei stets in einem Unterordner *images/*.

Setzen Sie mehrere Themes gleichzeitig oder alternativ ein, so ist eine Ordnerstruktur wie diese hier sinnvoll (CSS entsprechend umbenennen):

css/
smoothness/
smoothness.css
images/
redmond/
redmond.css
images/
cupertino/
cupertino.css
images/
...

Hierbei könnte Smoothness (ohne Basisklasse) als Default fungieren.

6.3 Einsatz von jQuery UI

jQuery UI wird grundsätzlich als gezipptes Paket ausgeliefert – das ist unabhängig davon, ob Sie das Komplettpaket oder einen konfigurierten Download herunterladen.

Auch wenn Sie lediglich ein *Theme* laden, also alle UI-Optionen im Konfigurator *abgewählt* haben, ergibt sich dieselbe Struktur. Die JavaScript-Dateien sind dann aber weitgehend leer.

6.3.1 Dateistruktur von jQuery UI

Betrachten Sie nun bitte die Struktur des entpackten Pakets. Das Downloadpaket enthält im jQuery-UI-Ordner eine Startseite *index.html*, alle CSS-Dateien, die beiden jQuery-UI-Dateien (*jquery-ui.js*, *jquery-ui.min.js*) und zwei Unterordner:

- **[jquery-ui-ordner]/external/jquery**
  Hier befindet sich eine Datei mit dem jQuery-Framework in Version *jquery-1.12.4.js*.
- **[jquery-ui-ordner]/images/**
  Hier finden Sie alle Icon-Grafiken.

6.4 Einbinden von jQuery UI

Um jQuery UI in Ihrem Projekt einzusetzen, kopieren Sie einen Teil des entpackten UI-Zips in Ihr Projektverzeichnis.

Legen Sie die JavaScript-Dateien *jquery-ui.js* und *jquery-ui.min.js* aus *js/* in den gleichen Ordner, in dem bereits Ihr *jquery.js* liegt. Benennen Sie die Dateien um, wenn Sie möchten.

Kopieren Sie den Ordner *smoothness/* aus dem *css*-Verzeichnis des UI-Zips (inklusive des Unterorders *images/*).

Benennen Sie die CSS-Dateien darin um, wenn Sie dies möchten (wir wählen den Namen *smoothness.css* bzw. *smoothness.min.css*).

Ihr Projektordner könnte nun aussehen wie folgt:

projekte/
  js/
    jquery.js
    jquery-ui.js
    jquery-ui.min.js

  css/
    smoothness/
      jquery-ui.css
      jquery-ui.min.css
      jquery-ui.structure.css
      jquery-ui.structure.min.css
      jquery-ui.theme.css
      jquery-ui.theme.min.css
      images/
  uebungen-ui/
  ...

Das Einbinden von jQuery UI ist einfach – zu beachten ist lediglich, dass Sie (zumindest sofern Sie Widgets einsetzen möchten, was aber meist der Fall ist) mindestens ein UI-Theme einbinden müssen. Dies geschieht vor dem Einbinden der JavaScript-Dateien.

Bei diesem ist, ebenso wie beim Einsatz von Plugins (auch UI besteht aus Plugins), eine Reihenfolge zu beachten – zuerst jQuery, dann jQuery UI.

```
<!DOCTYPE HTML>
<html lang="de-DE">
<head>
   <meta charset="UTF-8">
   <title>jQuery UI einbinden</title>
   <link rel="stylesheet"
    href="../css/smoothness/jquery-ui. css">
   <link rel="stylesheet"
    href="../css/smoothness/jquery-ui.theme.css">
   <script type="text/javascript"
    src="../js/jquery-3.2.1.js"></script>
   <script type="text/javascript"
    src="../js/jquery-ui.js"></script>
   <script type="text/javascript">
      // Ihr Code hier ...
   </script>
</head>
<body>
   ...
</body>
</html>
```

6.5 CSS-Klassen eines UI-Widgets

Betrachten wir zunächst die *Widgets*, da sie als sichtbare Oberflächenelemente den spektakulärsten Aspekt des UI-Frameworks darstellen. Alle Widgets besitzen einen annähernd gleichen Aufbau, der durch eine HTML-Struktur bestimmt wird, in der *festgelegte CSS-Klassen* vergeben sind. Diese Klassen legen die Rolle fest, die die betreffenden HTML-Elemente innerhalb des Widgets einnehmen. Daraus folgt auch eine bestimmte *Optik*.

```
<div class="ui-widget">
  <div class="ui-widget-header">
    <p>Dies ist ein Widget-Header</p>
  </div>
  <div class="ui-widget-content">
    <p>Dies ist der Contentbereich des Widgets, umgeben
    von einem Element der Klasse ui-widget-content.</p>
  </div>
</div>
```

Hierbei sind folgende drei Basisklassen beteiligt:

- **ui-widget**
  Das Widget selbst bzw. der Widget-Container.
- **ui-widget-header**
  Der Container, der den Header-Bar des Widgets generiert.
- **ui-widget-content**
  Der Container, der die Inhalte des Widgets umschließt.

Mit Zusatzklassen können weitere Grundstylings wie Eckradien der Header- und des Content-Containers gesetzt werden.

- **ui-corner-all**
  Alle vier Ecken des Containers erhalten den `border-radius` des Themes.
- **ui-corner-top, ui-corner-bottom**
  Die beiden oberen respektive unteren Ecken werden gerundet.
- **ui-corner-left, ui-corner-right**
  Die beiden linken respektive rechten Ecken werden gerundet.

Die nicht betroffenen Ecken behalten den Defaultradius 0. Die Klassen können nicht auf den Widget-Container selbst angewendet werden.

Mit vergebenen Corner-Klassen sieht die Widget-Struktur so aus:

```
<div class="ui-widget">
  <div class="ui-widget-header ui-corner-top">
    <p>Dies ist ein Widget-Header</p>
```



```
  </div>
  <div class="ui-widget-content ui-corner-bottom">
    <p>Dies ist der Contentbereich des Widgets, umgeben
    von einem Element der Klasse ui-widget-content.</p>
  </div>
</div>
```

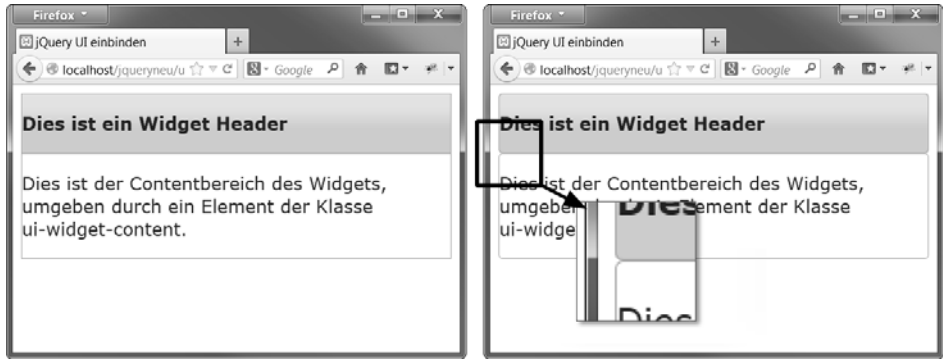

Abbildung 6.9 HTML-Struktur mit Widget-Klassen, rechts mit ui-corner-radius

Weitere Klassen dienen dem Zuweisen von Zuständen (»inaktiv« bzw. »aktiv«), wobei dies optisch durch ein Icon unterstützt werden kann. Diese Klassen werden bei Interaktion mit dem Widget automatisch vergeben:

- **ui-state-default**
  Definiert den Defaultzustand eines Containers oder UI-Elements (»nicht ausgewählt«, »inaktiv«).
- **ui-state-active**
  Definiert den aktiven Zustand des UI-Elements.
- **ui-state-hover**
  Diese Klasse definiert den Hover-Zustand des UI-Elements.

Alle drei Klassen sind im *ThemeRoller* konfigurierbar (siehe Abbildung 6.10).

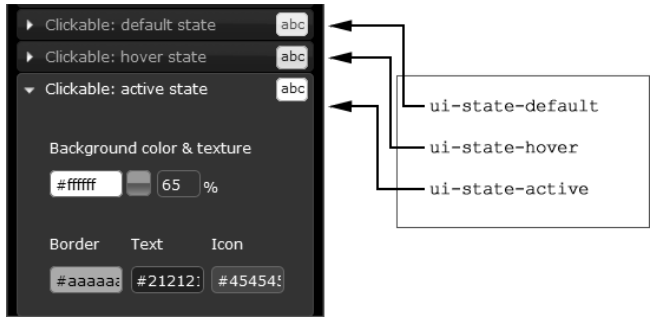

Abbildung 6.10 Der ThemeRoller als Interface für die Interaktionsklassen

Beispiel: Der Contentbereich eines Widgets wird klickbar gemacht und wechselt seine Interaktionsklasse. Im Container befindet sich ein weiteres Element, das den Bereich eines Icons definiert.

```
<div class="ui-widget">
  <div class="ui-widget-header ui-corner-top">
    <p>Dies ist ein Widget-Header</p>
  </div>
  <div class="ui-widget-content
              ui-state-default
              ui-corner-bottom">
    <p class="ui-icon ui-icon-circle-plus"
       style="float:left;margin:21px 10px"></p>
    <p>Dieser Content ist durch Klick aktivierbar.</p>
  </div>
</div>
```

Dies geht natürlich nicht ganz von selbst – fügen Sie ein Script ein, das am Contentbereich die Klasse `ui-state-active` toggelt:

```
$(document).ready(function(){
   $('.ui-widget-content').click(function(){
         $(this).toggleClass('ui-state-active');
   });
});
```

Abbildung 6.11 Aktivierbarer Content, links Default, rechts aktiviert

Beachten Sie, dass das Icon seine Darstellung ebenfalls anpasst.

6.6 Layout-Widgets aus jQuery UI

Zur Erstellung von Layout-Elementen, aber auch zur allgemeinen Gestaltung von User Interfaces und dem Handling von Interaktionen enthält jQuery UI eine Reihe vordefinierter, aber in hohem Maße konfigurierbarer Widgets. Die Gestaltung all dieser Widgets wird durch das jeweilige Theme bestimmt.

- Das *Dialog-Widget* ersetzt die modalen Dialogboxen des Betriebssystems durch konfigurierbare modale Dialoge.
- Die *Progressbar* ist ein modales Widget, das den Verlauf eines Ladevorgangs signalisieren kann.
- Das *Accordion-Widget* bietet den bekannten Akkordeon-Effekt beim Umschalten zwischen verschiedenen Inhaltsbereichen (Content-Panes).
- Das *Tab-Widget* funktioniert analog zum Accordion-Widget, verwendet aber ein Tab-Menü zur Steuerung.

6.6.1 Dialog-Widget

Ein Dialog wird auf Basis einer HTML-Struktur generiert, die »jQueryfiziert« wird – dies geschieht durch Selektion und anschließende Anwendung einer UI-Methode auf die Collection.

Beginnen wir mit einer einfachen Struktur. Die ID hilft lediglich bei der Selektion – wichtiger ist das `title`-Attribut:

```
<div id="meindialog" title="Einfacher Dialog">
  <p>Ein Dialog wird auf Grundlage dieser einfachen
   Struktur generiert und kann durch ein von jQuery UI
   erzeugtes Icon geschlossen werden.</p>
</div>
```

Der Code zur eigentlichen Erzeugung eines Defaultdialogs ist simpel:

```
$(document).ready(function() {
   $("#meindialog").dialog();
});
```

Dies bringt folgendes Ergebnis (hier mit Theme »redmond«):

Abbildung 6.12 Der Dialog ist skalier- und verschiebbar.

Um einen Standarddialog zu erzeugen, braucht auf das selektierte Element lediglich die UI-Methode `dialog()` angewendet zu werden. Ein solcher Dialog besitzt ein paar Grundeigenschaften:

- nicht modal (Seite bleibt interaktiv)
- skalierbar und verschiebbar
- öffnet sich beim Laden automatisch und zentriert

Konfigurieren des Dialogs

Um ein vom Standarddialog abweichendes Verhalten festzulegen, muss dem Dialog ein *Konfigurationsobjekt* übergeben werden:

```
$(document).ready(function() {
   $("#meindialog").dialog( {
         resizable: false,
         modal: true,
         height:150
      } );
});
```

Ein solcher Dialog ist *modal* und besitzt eine feste Größe, die vom User nicht verändert werden kann.

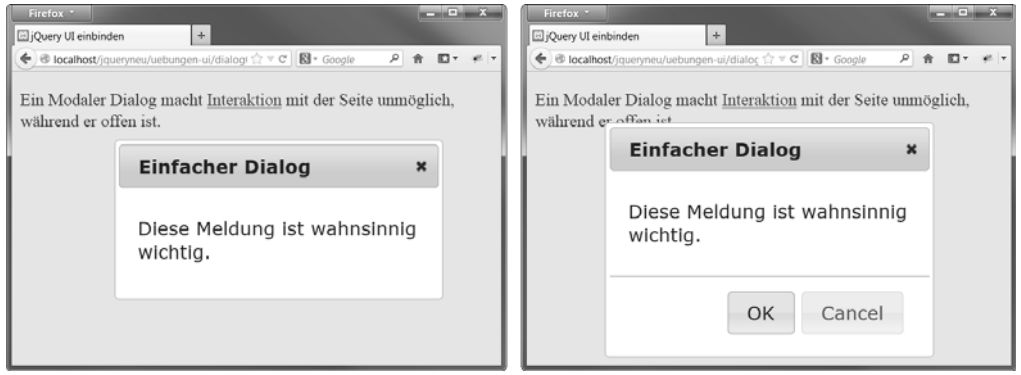

Abbildung 6.13 Konfigurierte Dialoge (rechts mit OK-/Cancel-Buttons)

Ein Dialog mit Buttons

Auch das Hinzufügen von Buttons ist möglich:

```
$(document).ready(function() {
   $("#meindialog").dialog({
         resizable: false,
         modal: true,
         height:215,
```



```
         buttons: {
            OK: function() {
               $(this).dialog("close");
            },
            Cancel: function() {
               $(this).dialog("close");
            }
         }
      });
});
```

In diesem Fall dienen die Buttons nur dem Schließen des Dialogs – hierfür wird das Dialog-Widget *this* mit der Option »close« aufgerufen:

```
$(this).dialog("close");
```

Auf einen Dialog kann die Methode `dialog()` angewendet werden.

Statt `Ok` und `Cancel` können Buttons beliebige Beschriftungen erhalten. Auch die Zahl der Buttons ist wählbar:

```
buttons: {
   "Ja": function() {
      $(this).dialog("close");
   },
   "Nein": function() {
      $(this).dialog("close");
   },
   "Vielleicht": function() {
      $(this).dialog("close");
   }
}
```

Konfigurationseigenschaften des Dialogs

Im Rahmen der Konfiguration können verschiedene Parameter gesetzt werden. Eine Auswahl sehen Sie in Tabelle 6.2.

Parameter	Wert (Default)	Erläuterung
autoOpen	boolean (*true*)	Verhindert automatisches Öffnen des Dialogs.

Tabelle 6.2 Konfigurationsparameter des Dialogs

Parameter	Wert (Default)	Erläuterung
buttons	Object ({})	Konfiguriert Buttons des Dialogs als Button-Array in einem Unterobjekt.
draggable	boolean (*true*)	Macht Dialog verschiebbar.
height, width	Number (»auto«)	Setzt Höhe und Breite des Dialogs als Ausgangswert.
hide	String (null)	Nennt den Effekt beim Schließen des Dialogs: »explode«, »bounce«, »drop« ...
maxHeight, maxWidth minHeight, minWidth	Number (*false*)	Minimal- und Maximalgrößen des Dialogs, abhängig von den Inhalten.
modal	boolean (*false*)	Setzt Dialog auf »modal« und verhindert Interaktion mit dem Rest des Dokuments.
resizable	boolean (*true*)	Größenveränderbarkeit durch den User.
title	String (–)	Setzt den Titeltext des Dialogs.

Tabelle 6.2 Konfigurationsparameter des Dialogs (Forts.)

Die API des Dialogs

Eine Übersicht über die komplette API des Dialogs finden Sie hier:

http://api.jqueryui.com/dialog/

Getter und Setter für Optionen

Alle Optionen können nachträglich gesetzt werden, indem einfach ein weiteres Mal ein Konfigurationsobjekt übergeben wird.

Alternativ kann der Dialog-Methode das Keyword `option` übergeben werden, das zusammen mit einem Parameternamen als Getter fungiert und als Setter, wenn zusätzlich noch ein Wert hineingereicht wird:


```
// Getter
var draggable =
meinDialog.dialog("option", "draggable");  //-> true
// Setter
meinDialog.dialog("option", "draggable", false);
```

Öffnen per Button: dialog(»open«)

Interessant ist es, das Öffnen des Dialogs steuerbar zu machen. Hierzu muss zum einen in der Konfiguration die `autoOpen`-Option *abgewählt* werden. Zum anderen muss ein Interface-Element (z. B. ein Button) das Öffnen programmatisch vornehmen:

```
<p><button id="meinbutton">Dialog öffnen</button></p>
<div id="meindialog" title="Einfacher Dialog">
   <p>Diese Meldung ist wahnsinnig wichtig.</p>
</div>
```

Das Script dazu:

```
$(document).ready(function() {
   $("#meindialog").dialog({ autoOpen: false,
                             modal:true });
   $( "#meinbutton" ).click(function() {
      $( "#meindialog" ).dialog("open");
   });
});
```

Dialog programmatisch erzeugen und befüllen

Die HTML-Struktur, die den Dialog aufspannt, muss nicht bereits im Dokument sein – sie kann ebenso programmatisch erzeugt werden.

Hierfür wird einfach die `$()`-Funktion als Factory eingesetzt:

```
var meinDialog = $('<div></div>');
meinDialog.dialog({autoOpen: false});
```

Dies geschieht nur aus Platzgründen in zwei Zeilen – Sie können die Dialog-Methode auch unmittelbar auf das erzeugte jQuery-Objekt anwenden. Die Übergabe von `autoOpen:false` verhindert, dass der Dialog sofort geöffnet wird.

Das (unsichtbare) Widget kann jetzt weiter konfiguriert werden. Dies geschieht durch wiederholte Anwendung der Dialog-Methode mit erneuter Übergabe eines Konfigurationsobjekts:

```
meinDialog.dialog({ modal:true,
                    title:'Generierter Dialog' });
```

In der Tat wird die bisherige Konfiguration hierdurch *ergänzt* und nicht überschrieben. Um einen Parameter zu überschreiben, müssen Sie ihn explizit neu setzen!

Sie können den Dialog jetzt öffnen – die Aktion wird an den Klick eines Buttons gebunden:

```
$("#meinbutton").click(function() {
   meinDialog.dialog("open");
});
```

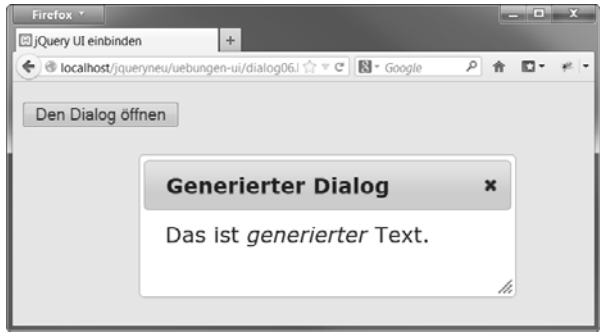

Abbildung 6.14 Generierter Dialog – sieht aus wie ein »normaler« Dialog

Der Dialog besäße jetzt allerdings noch keinen Inhalt und muss daher noch »befüllt« werden – hier mit einem einfachen HTML-String. Dies kann im Verlauf des Öffnens geschehen (ob davor oder danach, ist gleichgültig):

```
meinDialog.html('Das ist <i>generierter</i> Text.')
          .dialog("open");
```

Die Widget-Struktur ist irrelevant für den Einsatz

Sie brauchen nichts über die HTML-Struktur des Widgets zu wissen, um den Dialoginhalt zu verändern.

Methoden des Dialog-Widgets

Ein Dialog verfügt über eine Reihe von Methoden (siehe Tabelle 6.3).

Methode	Erläuterung
`open`	Öffnet den Dialog.
`close`	Schließt den Dialog.

Tabelle 6.3 Methoden des Dialog-Widgets

Methode	Erläuterung
`destroy`	Schließt den Dialog, wenn er offen ist, und entfernt das Widget-HTML aus dem Dokument.
`isOpen`	Returns: *boolean*; prüft, ob das Widget offen oder geschlossen ist.
`moveToTop`	Legt den Dialog bei mehreren geöffneten Dialogen in den obersten Z-Index.

Tabelle 6.3 Methoden des Dialog-Widgets (Forts.)

Die Methoden werden nicht direkt am Widget aufgerufen – dies geschieht stets über die Dialog-Methode, wobei der Methodenname als String übergeben wird:

```
meinDialog.dialog("open"); // öffnet den Dialog
meinDialog.dialog("close"); // schließt ihn
```

Falsch ist hingegen:

```
meinDialog.open() oder meinDialog.dialog().open();
```

6.6.2 Progressbar

Das Progressbar-Widget erzeugt einen Fortschrittsbalken, wie er als Feedback während eines Ladevorgangs o. Ä. eingesetzt wird. Es kann ganz einfach auf ein bestehendes HTML-Element aufgesetzt werden, auf das dann die Progressbar-Methode angewendet wird:

```
<div id="pbar"></div>
$(document).ready(function() {
    $("#pbar").progressbar();
});
```

Ohne Konfiguration ist das Widget allerdings nicht nützlich, da der Balken komplett leer ist. Immerhin füllt er seinen Elterncontainer in der Horizontalen voll aus, ohne dass hierfür Angaben erforderlich sind.

Parameter	Wert (Default)	Erläuterung
`disabled`	boolean (*false*)	Deaktivierung
`value`	Number (0)	Stand des Fortschrittszeigers

Tabelle 6.4 Konfigurationsparameter der Progressbar

In der Regel wird man aber einen Wert `value` übergeben, der den Stand des Fortschrittsbalkens in Prozent angibt – ein Wert von »0« bis »100« wird akzeptiert:

```
$(document).ready(function() {
    $("#pbar").progressbar({ value:50 });
});
```

Dies erweitert die HTML-Struktur wie folgt (ARIA-Klassen weggelassen):

```
<div id="pbar"
     class="ui-progressbar ui-widget
            ui-widget-content ui-corner-all">
   <div style="width:50 %;"
        class="ui-progressbar-value
               ui-widget-header ui-corner-left">
   </div>
</div>
```

Die Farbe des Balkens wird im Theme festgelegt. Um mehr Leben ins Spiel zu bringen, kann per CSS das `background-image`-Property gesetzt werden, das ein animiertes GIF einbindet. Als Selektoren werden die Klassen `ui-progressbar` und `ui-progressbar-value` eingesetzt (siehe vorausgehendes Codebeispiel):

```
<style>
.ui-progressbar .ui-progressbar-value {
    background-image: url(../css/images/pbar-ani.gif);
 }
</style>
```

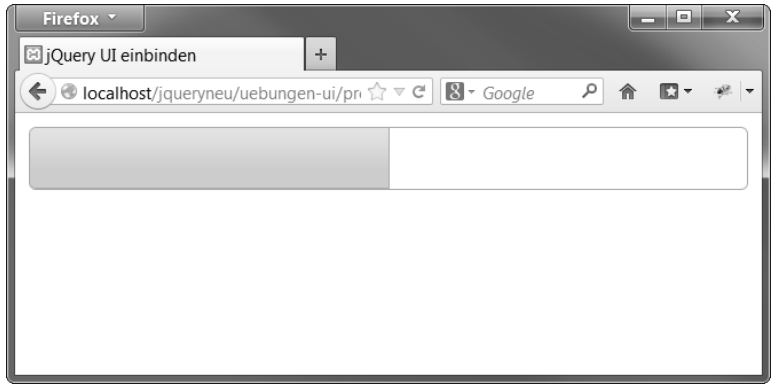

Abbildung 6.15 Einfache Progressbar mit value 50 %

Events für die Progressbar

Die Progressbar besitzt drei spezifische Arten von Event-Handlern, an die man bei der Konfiguration Callback-Funktionen binden kann.

Ereignis	Erläuterung
`create`	Das Widget wird initialisiert.
`change`	Der Wert des `value`-Parameters ändert sich.
`complete`	Der Wert des `value`-Parameters hat 100 erreicht.

Tabelle 6.5 Events der Progressbar

Der Einsatz geschieht am übersichtlichsten, indem man das Konfigurationsobjekt *getrennt* von der Initialisierung der Progressbar definiert:

```
var pconf = {
   value:50,
   create: function(){
      console.log('Progressbar initialisiert.')
   };
var change: function(){
      console.log('Der value hat sich geändert.')
   };
var complete: function(){
      console.log('Vorgang abgeschlossen.')
   }
};
var pbar = $("#pbar").progressbar(pconf);
```

Ajax-gesteuerte Progressbar

In diesem Beispiel wird die Progressbar durch ein JSON-Objekt gesteuert, das durch eine PHP-Datei generiert wird.

Hier der Scriptcode:

```
$(function(){
  var pconf = {
     value:0,
     complete: function(){
        console.log('Vorgang abgeschlossen.')
     }
  };
  // Progessbar initialisieren
  var pbar = $("#pbar").progressbar(pconf);
  // AIF-Function (startet automatisch):
  (function simulateprogress() {
    $.ajax({
```

```
        // Server liefert JSON
        dataType: "json",
        // URL erhält stets aktuellen Status
        url: './progress03_ajax.php?status=' +
             pbar.progressbar("option", "value" ),
        // Nach erfolgreichem Abschluss des Requests
        success: function(data) {

          // PHP liefert in 'data' ein JSON-Objekt mit
          // den Properties 'status' und 'message' zurück

          // Überprüfen, ob ein JSON-Objekt da ist.
          if(data.status && data.message) {

              // Updaten der Progressbar
              pbar.progressbar("option",
                               "value",
                               data.status);

              // Meldung darstellen
              $("#msg").html(data.message);

              // Solange 100 % nicht erreicht sind ...
              if(data.status!=100) {
                  simulateprogress();
              }
          }
        }); // Ende $.ajax
      })() // Start der Funktion simulateprogress als AIF
    });
```

Die PHP-Datei, die das JSON-Objekt zurückgibt, finden Sie als *progress03_ajax.php* im Onlinebereich dieses Buchs.

Hier ein Einblick in den Code:

```
$zufall = rand(1,3); // Zufallszahl generieren
sleep($zufall);
// Prozessfortschritt als Prozentwert
$php_array['status'] = $_GET['status']+($zufall * 3);
// Bei 100 % ist Schluss
if($php_array['status'] > 100) {
    $php_array['status'] = 100;
}
```



```
// Ausgabe der Nachricht
if($php_array['status'] != 100) {
    $php_array['message'] = 'Aktueller Status:'
         .$php_array['status']
         .' von 100, Countdown: '
         .(100 - $php_array['status']);
} else {
    $php_array['message'] = 'Aktueller Status: '
                            .'Ladevorgang abgeschlossen.';
}

// Ausgabe des PHP-Arrays als JSON-Objekt
echo json_encode($php_array);
```

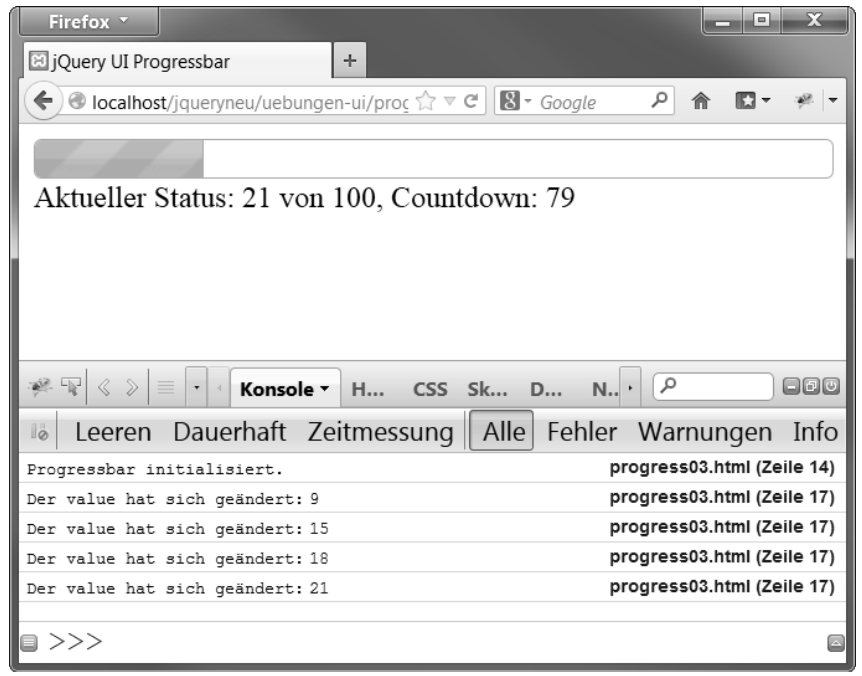

Abbildung 6.16 Ajax-Progressbar mit animiertem Hintergrund und Events

Die Konfiguration kann noch durch Callbacks bei Initialisierung und Wertänderung ergänzt werden:

```
var pconf = {
   value:0,
   create: function(){
```

```
      console.log('Progressbar initialisiert.');
   },
   change: function(){
      console.log('Der value hat sich geändert:',
                  pbar.progressbar("option", "value"))
   },
   complete: function(){
      console.log('Vorgang abgeschlossen.');
   }
};
```

6.6.3 Akkordeon-Widget

Das Akkordeon-Widget gehört zu den Content-Widgets. Es handelt sich um ein Layout-Element, das auf begrenzter und definierbarer Fläche mehrere alternative Inhalte (in Form so genannter *Content-Panes*) zeigen kann, von denen gewöhnlich jeweils nur einer aktiv ist.

Ein *Content-Pane* besteht aus einem Überschriftenelement (*Header Bar*) und dem eigentlichen Inhaltsbereich.

Der *Header Bar* dient als Klickfläche, um den Inhalt zu zeigen (und gegebenenfalls wieder zu verstecken). Je nach Konfiguration schließt sich ein geöffneter Inhalt beim Öffnen eines anderen Panes automatisch. Die Inhaltsfläche kann mit beliebigem HTML-Inhalt gefüllt werden.

Die HTML-Struktur folgt einem Muster:

```
<div id="accordioncontainer">
   <h3>Header Bar 1</h3>
   <div>Contentbereich 1</div>
   <h3>HeaderBar 2</h3>
   <div>Contentbereich 2</div>
   ...
</div>
```

Im HTML muss also stets ein äußeres Div-Element die Rolle des Widget-Containers übernehmen. In diesem Container folgen Paare aus Überschriftencontainern und wiederum Div-Containern aufeinander.

Jedes dieser Paare erzeugt gemeinsam ein Content Pane, das aus Header Bar und Contentbereich besteht.

Abbildung 6.17 Einfaches Akkordeon ohne weitere Konfiguration

Es kommt allein auf die Struktur an

Allein die paarige Struktur ist entscheidend. Es ist nicht erforderlich, dass der Header-Bar-Container wirklich eine Überschrift ist. Jedes beliebige Blockelement funktioniert, also auch Div- oder P-Container ebenso wie jede Überschrift von H1 bis H6.

Auch muss der Contentbereich nicht notwendigerweise ein Div-Container sein – dies ist allerdings dann erforderlich, wenn der Inhalt eine Struktur (z. B. mehrere Absätze o. Ä.) erhalten soll. Soll der Inhalt lediglich einen Absatz umfassen, so genügt ein P-Container für den Contentbereich.

```
<div id="accordion">
   ...
</div>
```

Geht man von einem Widget-Container mit `id="accordion"` aus (siehe oben), der eine geeignete Innenstruktur besitzt, so genügt folgender Aufruf:

```
$(document).ready( function() {
    $("#accordion").accordion();
});
```

Konfiguration des Akkordeons

Auch das Akkordeon kann über ein Konfigurationsobjekt konfiguriert werden. Folgende Eigenschaften existieren (siehe Tabelle 6.6).

Parameter	Wert (Default)	Erläuterung
active	Number (0)	Nennt das aktuell offene Panel (Array-Index; oberstes Panel hat Index 0). Mit `active:false` sind alle Panels beim Laden geschlossen. Gleichzeitig muss `collapsible:true` stehen.
animate	String oder Object ({})	Effekt beim Panel-Wechsel, z. B. `animate: "bounceslide"`. Anmerkung: Option derzeit buggy!
event	String (»click«)	Nennt das Ereignis (oder die Ereignisse), das (die) den Panel-Wechsel steuert (steuern): `event: "mouseover"`
collapsible	boolean (*false*)	Erlaubt mit *true* das Schließen *aller* Panels.
header	Selector	Nennt das Element, das im Markup die Rolle des Header-Containers spielt.
heightstyle	String (»auto«)	Die *Höhe* des Widgets: »auto«: Höhe des höchsten Panels »fill«: Höhe des Parent-Containers »content«: pro Panel individuell

icons	Object	Nennt das Icon im Header-Bar:
		`icons: {`
		`"header": "ui-icon-plus",`
		`"headerSelected": "ui-icon-minus"`
		`}`

Tabelle 6.6 Konfigurationseigenschaften des Akkordeons

6.6.4 Tab-Widget

Ebenso wie das Accordion-Widget gehört das Tab-Widget zu den Layout-Widgets. Auch hier werden Content-Panes gezeigt, zwischen denen gewechselt wird, jedoch erfolgt die Steuerung über eine Karteireiter-Navigation (»Tabs«).

Zum Erstellen genügt die Anwendung der Methode `.tabs()` auf dem Widget-Container (üblicherweise ein Div-Container):

```
$(document).ready( function() {
      $("#tabs").tabs();
});
```


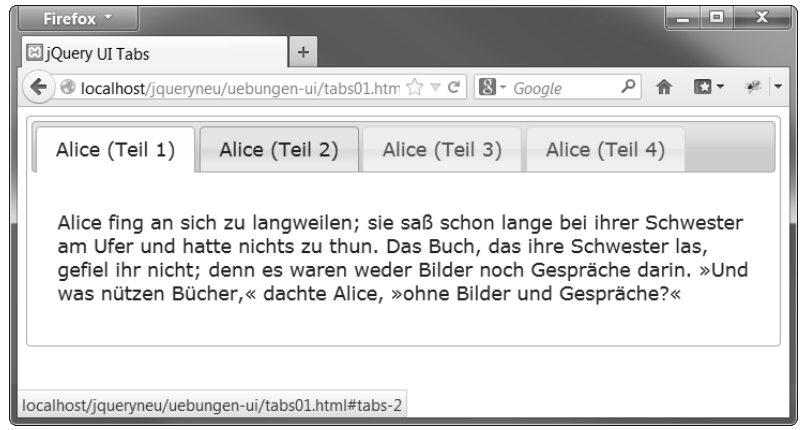

Abbildung 6.18 Tab-Widget mit vier Tabs

Die HTML-Struktur ist etwas komplexer als beim Akkordeon – die Tab-Navigation besteht aus einer `<ul>`-Liste, die sich als erstes Child im Widget-Container befindet:

```
<div id="tabs">
   <ul>
         <li><a href="#tabs-1">Alice (Teil 1)</a></li>
         <li><a href="#tabs-2">Alice (Teil 2)</a></li>
         ...
   </ul>
...
</div>
```

Pro Content-Pane muss ein List-Item mit Link vorhanden sein, das einen Fragment-Identifier besitzt, der auf ein Pane mit entsprechender ID zielt.

```
<div id="tabs">
   <ul>
         <li><a href="#tabs-1">Alice (Teil 1)</a></li>
         <li><a href="#tabs-2">Alice (Teil 2)</a></li>
         ...
   </ul>
   <div id="tabs-1">
...
   </div>
   <div id="tabs-2">
...
   </div>
...
</div>
```

Tab mit externen Inhalten per Ajax

Die Content-Panes eines Tab-Widgets können auf einfachste Weise per Ajax befüllt werden. Im Prinzip genügt es, statt des Fragment-Identifier-Links einen Link auf eine externe Ressource in die Tab-Navigation zu platzieren:

```
<div id="tabs">
   <ul>
     <li><a href="#tabs-1">Alice (Teil 1)</a></li>
     <li><a href="alice01.html">Alice (Teil 2)</a></li>
     <li><a href="alice02.html">Alice (Teil 3)</a></li>
     <li><a href="alice03.html">Alice (Teil 4)</a></li>
   </ul>
   <div id="tabs-1">
     <p>Ich komme nicht per Ajax</p>
   </div>
</div>
```

Für die per Ajax geladenen Tabs braucht kein korrespondierendes Content-Pane im Markup eingefügt zu werden!

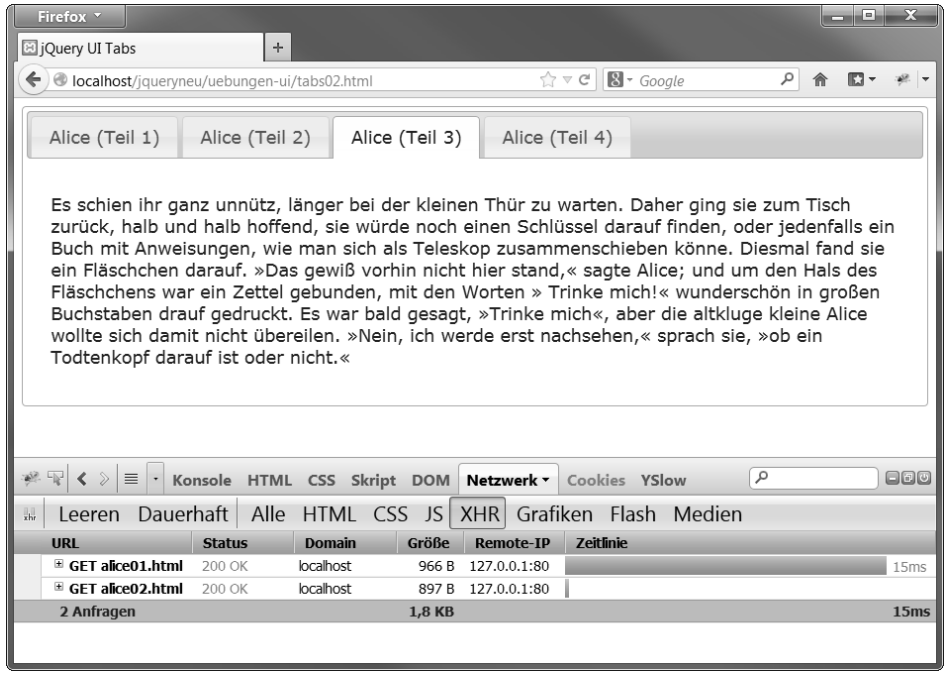

Abbildung 6.19 Tab-Inhalte werden per Ajax geladen

6.7 Formular-Widgets aus jQuery UI

Speziell für *Formulare* hält jQuery UI einige Widgets bereit, die immer dort gelegen kommen, wo korrespondierende HTML5-Formularelemente nicht verfügbar sind (ältere Browser) oder aus anderen Gründen nicht erwünscht sind (ungenügend konfigurierbar oder nicht ausreichend stylebar). Auch in Zeiten, in denen beispielsweise der Datepicker-Input nativ verfügbar ist, haben diese Widgets nach wie vor einige Berechtigung.

Im folgenden Abschnitt wird das Gewicht auf Datepicker, Slider und den erweiterten Button sowie die Autocomplete-Funktionalität gelegt. Es stehen jedoch noch weitere Widgets wie z. B. der Spinner für Zahleneingabe zur Verfügung.

6.7.1 Datepicker

Ein spezielles Formularelement zur *Eingabe von Datumswerten* verfügbar zu haben, birgt enorme Vorteile. Zwei davon stehen im Vordergrund:

- **Reglementierung des Eingabeformats**
  Der Datums-String liegt in einem Format vor, wie es für die Verarbeitung erforderlich ist.
- **Intuitive Bedienung**
  Der User wählt das Datum in einer übersichtlichen Kalendermetapher.

Das Fehlen eines entsprechenden nativen Elements führte in HTML5 zur Einführung des Input-Typs »date«. Seitdem braucht man (in der Theorie) eigentlich nicht mehr als dies hier:

```
<input type="date" name="datum">
```

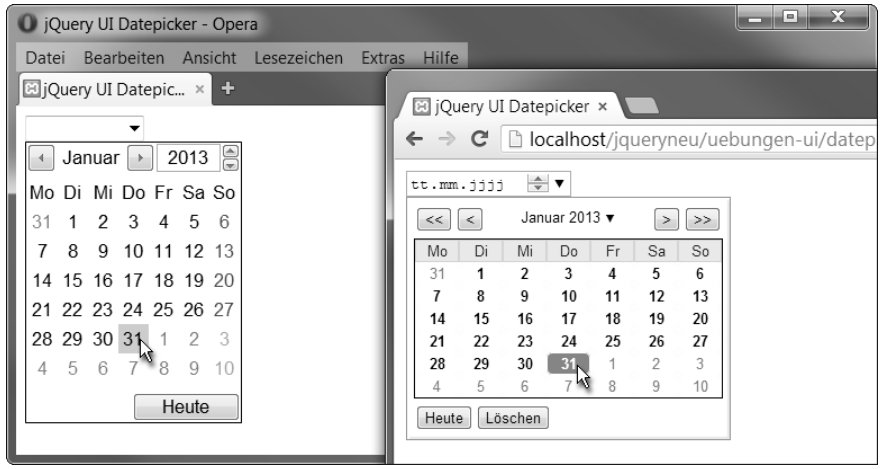

Abbildung 6.20 Nativer HTML5-Date-Input in Opera (links) und Chrome (rechts)

Eigentlich hat man hier mehrere *Vorteile*:

- Unabhängigkeit von JavaScript
- native Validierung im Rahmen der HTML5-Formularvalidierung

Den Vorteilen stehen aber auch gravierende *Nachteile* gegenüber:

- **Mangelhafte Unterstützung:**
  Implementierung bislang nicht im Internet Explorer und nur in ganz aktuellen Firefox-, Edge-, Opera- und Chrome-Browsern (siehe: *http://caniuse.com/#feat=input-datetime*).
- **Mangelhafte Gestaltungsmöglichkeiten:**
  Das Widget Interface wird durch den Browser im Rahmen der vom Betriebssystem vorgegebenen Gestalt erzeugt. Ein Einfluss (*Theming*) per CSS ist nicht möglich.

Das jQuery-Widget Datepicker

Der jQuery-Weg ist bereits altbekannt – ein HTML-Element (sinnvollerweise ein Formular-Input) wird per ID ausgewählt und die UI-Methode `datepicker()` darauf angewendet:

```
<input type="text" id="datepicker">
```

Hier der Code:

```
$(document).ready( function() {
    $("#datepicker").datepicker();
});
```

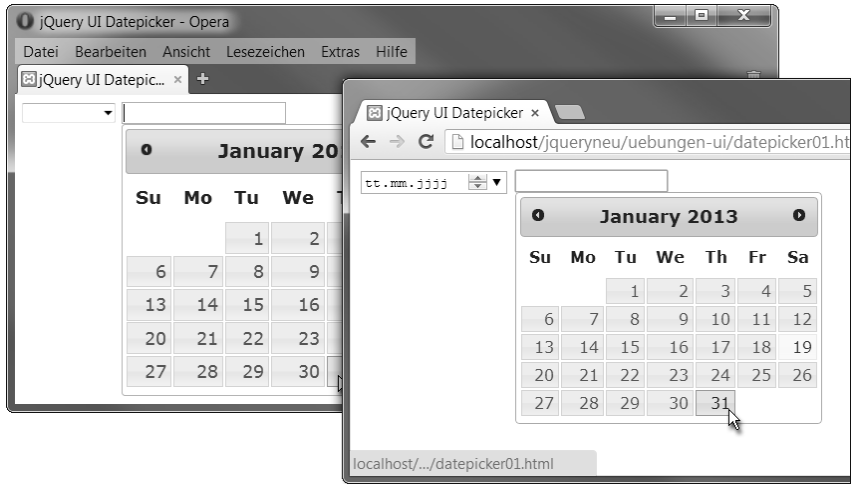

Abbildung 6.21 jQuery UI Datepicker in Opera (links) und Chrome (rechts)

Lokalisierung des Datepickers

Die Oberfläche des Widgets ist per Default in Englisch gehalten. Eine andere Lokalisierung erfordert das Einbinden einer passenden Lokalisierungsdatei. Da derartige Dateien in der Version 10.12 nicht mehr Teil des UI-Development-Pakets sind, müssen sie gesondert bei Github heruntergeladen werden. Sie finden sie dort unter

https://github.com/jquery/jquery-ui/releases/tag/1.12.1

Sie erhalten eine gepackte Datei mit allen Entwicklerkomponenten wie Testdateien, Demonstrationen der Widgets und allen kompatiblen jQuery-Versionen. Die Lokalisierungsdateien finden Sie im Unterordner *ui/il18n/*. Kopieren Sie von dort das File *datepicker-de.js* (deutsche Lokalisierung) aus dem Unterverzeichnis, und legen Sie es in Ihr *js*-Verzeichnis, am besten in ein neues Unterverzeichnis *i18n/*.

Binden Sie die Datei wie folgt ein:

```
<script type="text/javascript"
        src="../js/jquery.js"></script>
<script type="text/javascript"
        src="../js/jquery-ui.js"></script>
<script type="text/javascript"
        src="../js/i18n/datepicker-de.js"></script>
```

Achtung: Die Lokalisierung ist nicht automatisch im UI-Script inkludiert!

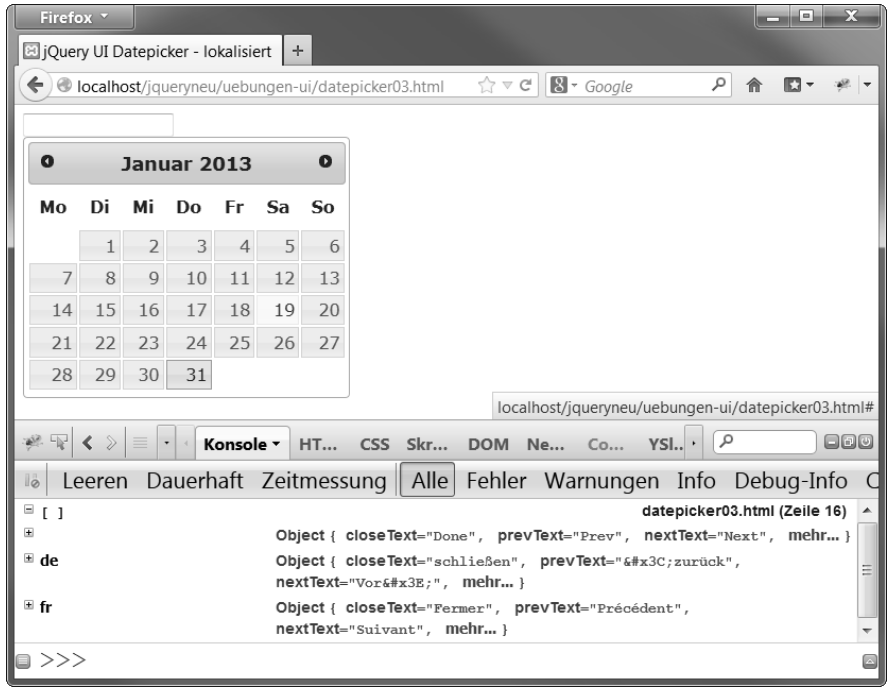

Abbildung 6.22 Datepicker, deutsch lokalisiert

Legen Sie noch eine weitere Datei, *jquery.ui.datepicker-fr.js*, in das lokale *i18n*-Verzeichnis. Nun können Sie die Lokalisierung zwischen deutscher und französischer Ausgabe umstellen.

Der Datepicker besitzt ein Objektattribut `$.datepicker.regional`, das die geladenen Lokalisierungen enthält. Die Defaultlokalisierung befindet sich in einem Property mit dem Key `""` (leerer String).

Solange nur eine zusätzliche Lokalisierung geladen ist, befindet sich nur ein weiteres Property im Objekt (derzeit `"de"` für Deutsch). Laden Sie nun die französische Lokalisierung dazu:

```
...
<script type="text/javascript"
        src="../js/i18n/datepicker-de.js">
</script>
<script type="text/javascript"
        src="../js/i18n/datepicker-fr.js">
</script>
...
```

Sie sehen, dass der Datepicker nun eine *französische* Oberfläche besitzt – binden Sie einzelne Lokalisierungen ein, so gilt stets die letzte.

Um die *deutsche* Lokalisierung (wieder) zu erhalten, ohne die Reihenfolge der Scripteinbindung zu verändern, können Sie die Lokalisierung als Argument bei der Generierung des Widgets übergeben:

```
$("#datepicker").datepicker(
      $.datepicker.regional["de"]
);
```

Achtung: Hier wird kein Konfigurationsobjekt übergeben! Die Konfiguration gilt obendrein nur für das aktuelle Widget.

Wollen Sie alle Datepicker des Dokuments gleichmäßig lokalisieren, so geschieht dies über die Option `$.datepicker.setDefaults()`:

```
$.datepicker.setDefaults($.datepicker.regional["de"]);
```

Dies setzt alle Datepicker-Widgets auf die deutsche Lokalisierung. Möchten Sie auf die Defaultlokalisierung zurück, so übergeben Sie statt `"de"` den leeren String `""`:

```
$.datepicker.setDefaults($.datepicker.regional[""]);
```


Noch mehr Lokalisierungen

Um *allumfassende* Lokalisierungsmöglichkeiten verfügbar zu haben, binden Sie den kompletten Ordner *i18n* ein. Sie müssen dann leider dennoch alle Lokalisierungsdateien der Sprachen, die Sie zur Verfügung stellen wollen, einzeln einbinden.

Nun können Sie die Lokalisierung (wie auch immer) anpassen – hier geschieht dies über folgende Select-Option:

```
<select id="locale">
   <option value="da">Danish (Dansk)</option>
   <option value="en-GB">English/UK</option>
   <option value="fi">Finnish (suomi)</option>
   <option value="de">German (Deutsch)</option>
   <option value="fr" >French (Français)</option>
   <option value="no">Norwegian (Norsk)</option>
</select>
```

Die aktuelle Lokalisierung des Datepicker-Widgets geschieht über folgende Funktion (setzt den Parameter über die Setter-Funktion der Optionen):

```
$("#locale").change(function() {
   $("#datepicker" ).datepicker(
      "option", $.datepicker.regional[$(this).val()]
   );
});
```

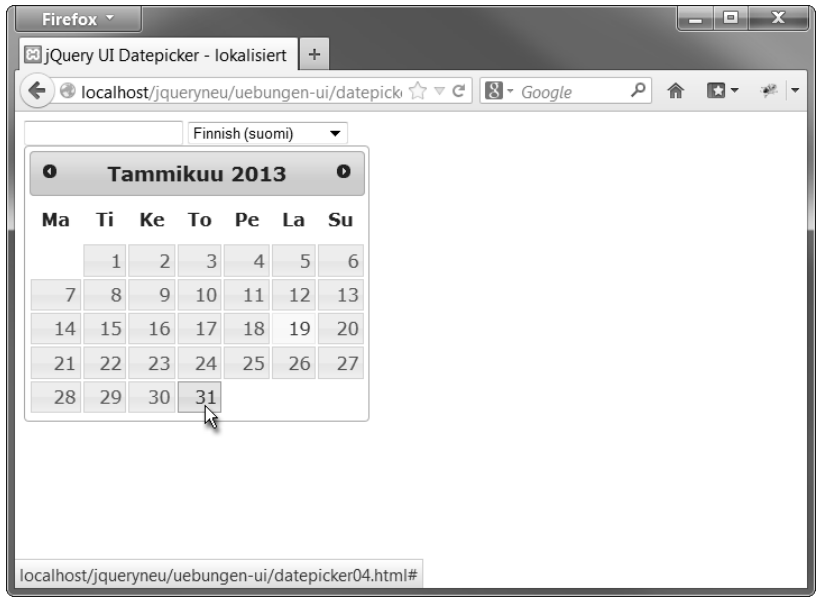

Abbildung 6.23 Datepicker nach Umschaltung auf finnische Lokalisierung

Konfiguration des Datepickers

Optionen für einen Datepicker können (derzeit) nicht sinnvoll im Rahmen eines Konfigurationsobjekts während der Widget-Erstellung gesetzt werden.

Setzen Sie sie stattdessen nachträglich über den Option-Setter:

```
// Widget erstellen:
var dp = $("#datepicker").datepicker();
// Widget konfigurieren:
dp.datepicker("option", "showAnim", "slideDown");
```

Dies ändert die Einblendeanimation eines Datepickers auf *SlideDown*.

Auf folgendem Weg erreichen Sie, dass für Monat und Jahr je ein Dropdown-Menü im Widget erscheint:

```
var dp = $("#datepicker").datepicker();
dp.datepicker("option", {
       changeMonth: true,
       changeYear:true
});
```

Sie können dem Option-Setter auch ein Objekt übergeben!

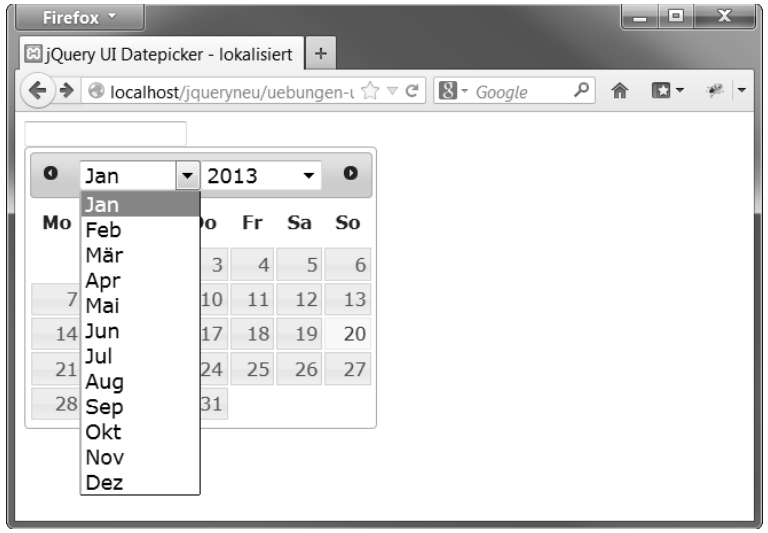

Abbildung 6.24 Datepicker mit changeYear- und changeMonth-Option

Das Ausgabeformat des Datums steuern Sie über die `dateFormat`-Option:

```
var dp = $("#datepicker").datepicker();
dp.datepicker( "option", "dateFormat", "yy-mm-dd" );
```


API des Datepickers

Eine Übersicht über die API des Datepickers erhalten Sie hier:

http://api.jqueryui.com/datepicker

6.7.2 Slider

Das *Slider-Widget* stellt eine intuitiv nutzbare Eingabemöglichkeit für numerische Werte dar, die vom Nutzer einfach durch Ziehen eines Schiebereglers vorgenommen werden kann.

Das Basiselement eines Sliders ist gewöhnlich ein Div-Container, auf den die UI-Methode `slider()` angewendet wird:

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

Dieses wird einfach zum Slider gemacht:

```
$(document).ready( function() {
   $("#slider").slider();
});
```

Ein Slider ist kein Range-Input!

Ein *Enhancement* eines Inputs, um den HTML5-Input »range« zu simulieren, ist nicht möglich. Basiselement muss ein Container sein!

Am Slider treten bei Betätigung Ereignisse auf, an die per Konfiguration Handler-Funktionen attacht werden können. Hier ein Beispiel für das *Change*-Ereignis, das bei Verschieben des Reglers auftritt. Es wird der Inhalt eines Span-Containers mit dem aktuellen Wert nach Betätigen gefüllt:

```
<div id="slider"></div>
<div>Slider steht auf: <span id="output">0</span></div>
```

Hier der Code:

```
var out = $('#output');
$("#slider").slider({
      change:function(evt, ui){
               console.log(ui.value);
               out.html(ui.value);
      }
});
```

Der Callback erhält als erstes Argument `evt` ein Event-Objekt, als zweites Argument `ui` den Slider, dessen `value` direkt auslesbar ist.

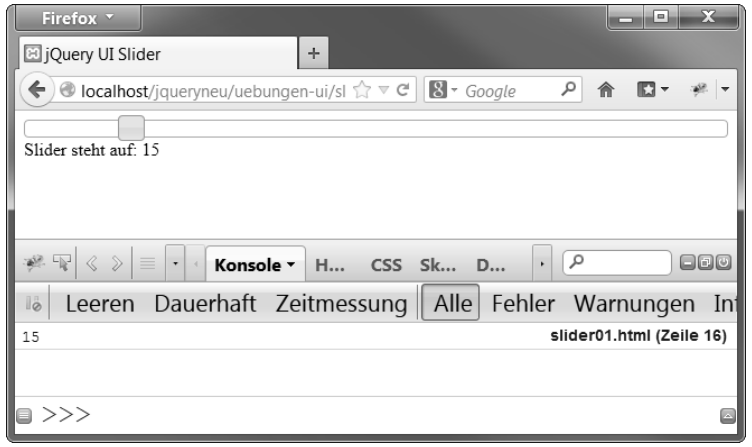

Abbildung 6.25 Slider mit Wertausgabe

Möchten Sie eine kontinuierliche Ausgabe bereits während der Bewegung (*Change* feuert erst nach Abschluss der Bewegung), so verwenden Sie das Event `slide`:

```
var out = $('#output');
$("#slider").slider({
      slide:function(evt, ui){
               out.html(ui.value);
      }
});
```

Über den `step`-Parameter kann eine Rasterung des Regelbereichs (von 0 bis 100) erreicht werden:

```
var out = $('#output');
$("#slider").slider({
      step : 10,
      slide:function(evt, ui){
               out.html(ui.value);
      }
});
```

Mit Hilfe der Optionen `min` und `max` lässt sich der Regelbereich beliebig definieren:

```
var out = $('#output');
$("#slider").slider({
      min  : 10,
      max  : 20,
      step : 2,
```



```
      slide:function(evt, ui){
               out.html(ui.value);
      }
});
```

Abbildung 6.26 Slider mit min, max (zwischen 10 und 20) sowie step

Mit der `range`-Option lässt sich ein Slider mit zwei Reglern definieren. Der `value` nimmt dann die Form eines Arrays an:

```
var out = $('#output');
$("#slider").slider({
      range: true,
      min: 0,
      max: 250,
      values: [ 50, 150 ],  // Startwerte
            slide:function(ev, ui){
            out.html(ui.values[0] +
                     '€ bis ' +
                     ui.values[1] +'€.');
      }
});
```

API des UI-Sliders

Die komplette API des UI-Sliders finden Sie hier:

http://jqueryui.com/slider

Abbildung 6.27 Ein Slider als Range-Regler mit zwei Eckwerten

6.7.3 Button

Das *Button-Widget* von jQuery UI erzeugt klickbare Button-Elemente auf der Grundlage unterschiedlicher HTML-Markups, die über das jeweilige UI-Theme gestaltet sind. Mit dem Widget können Einzel-Buttons, aber auch Buttonarrays (analog zu Checkboxen oder Radiobuttons) realisiert werden, die auf Wunsch auch mit einem Icon versehen werden können.

Das unterliegende Markup kann sein:

- ein Button-Element `<button>`
- ein Input des Typs »button«, »submit« oder »refresh«
- ein beliebiger Link in Form eines `<a>`-Elements

Auf das entsprechend ausgewählte Element ist lediglich die UI-Methode `button()` anzuwenden.

Einfacher Button

Einen einfachen Button erreichen Sie, indem Sie die UI-Methode auf ein passendes HTML-Element anwenden – hier ist es ein Button-Container:

```
<button id="meinButton">Ein einfacher Button</button>
```


Hier der Code dazu:

```
$(document).ready( function() {
      $("#meinButton")
            .button()
            .click(function( event ) {
                  console.log("Button geklickt.");
            });
});
```

Sie können ein Icon über das Konfigurationsobjekt hinzufügen – die Option `icons` erhält ein Objekt, das die Properties `primary` und `secondary` für bis zu zwei Icons enthält:

```
$("#meinButton")
            .button( {
                icons: {
                  primary: "ui-icon-folder-open"
                }
            });
```

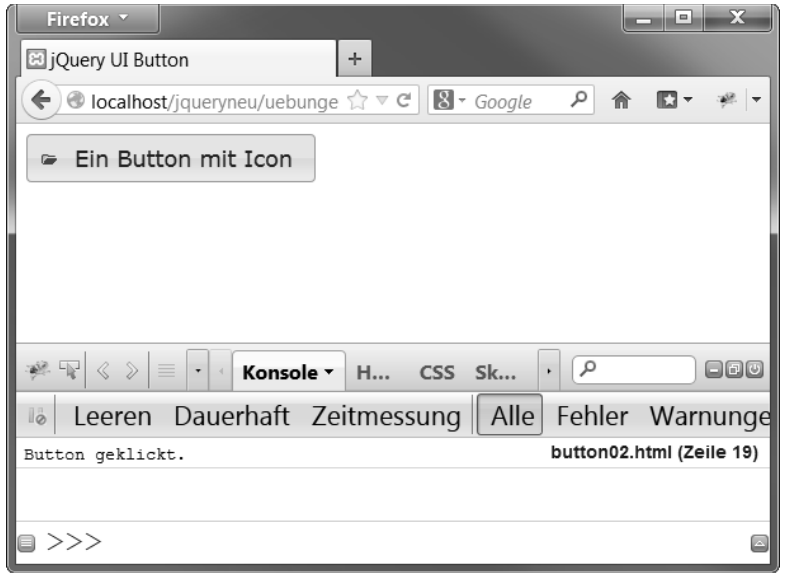

Abbildung 6.28 Ein Button-Widget mit Icon

Eine Übersicht über die verfügbaren Icons und ihre Namen erhalten Sie im Theme-Roller (Ansicht gekürzt, siehe Abbildung 6.29).

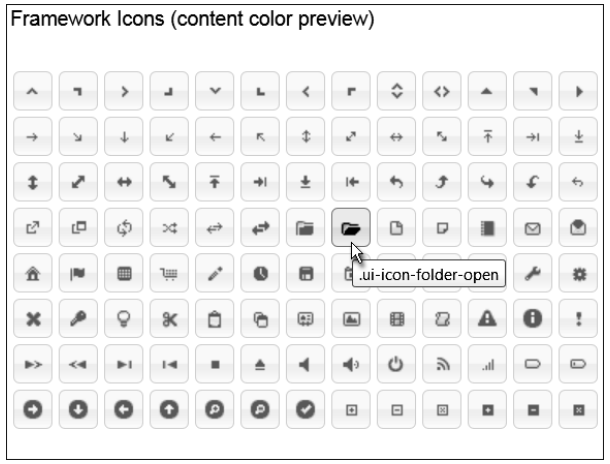

Abbildung 6.29 Icons für jQuery UI

Togglebutton

Ein *Togglebutton* ist ein Button, der seinen Zustand nach Klick wechselt. Grundlage für ihn ist ein Checkbox-Input. Die Button-Beschriftung wird dem korrespondierenden Label-Container entnommen:

```
<label for="cb">Toggle-Button</label>
<input type="checkbox" id="cb">
```

Folgender Scriptcode ist erforderlich:

```
$("#cb")
   .button( { icons: {
      primary: "ui-icon-folder-collapsed"
     }
});
```

Abbildung 6.30 Togglebutton (im Toggle-Zustand)

Buttonarrays mit buttonset()

Mehrere Checkboxen oder Radiobuttons können gemeinsam als Array eingesetzt werden. Hierfür ist die Methode `buttonset()` auf die Container-Box der Inputs anzuwenden:

```
<div id="array">
   <label for="cb1">Rosen</label>
   <input type="checkbox" id="cb1">
   <label for="cb2">Tulpen</label>
   <input type="checkbox" id="cb2">
   <label for="cb3">Nelken</label>
   <input type="checkbox" id="cb3">
</div>
```

Es genügt die einfache Anweisung:

```
$("#array").buttonset();
```

Analog können *Radiobuttons* eingesetzt werden. In diesem Fall kann nur eine Option selektiert werden.

Abbildung 6.31 Buttonarray aus Checkboxen (zweiter Button selektiert)

API des UI-Buttons

Die komplette API des UI-Buttons finden Sie hier:

http://jqueryui.com/button

6.7.4 Autocomplete

Das *Autocomplete-Widget* dient als Vorschlagsliste für Text-Inputs, aus der der User einen Vorschlag durch Klick auswählen kann.

Als zu Grunde liegendes HTML-Element dient ein *Text-Input*, das von einem Div-Container mit der Klasse `ui-widget` umgeben sein muss (ohne dies funktioniert es nicht):

```
<div class="ui-widget">
   <label for="prog">Ihre Lieblingssprache: </label>
   <input type="text" id="prog">
</div>
```

Hier ist die Datenquelle ein Array:

```
var sprachen = [ "ActionScript", "AppleScript", "Asp", "BASIC", "C", "C++",
"Clojure", "COBOL", "ColdFusion", "Erlang", "Fortran", "Groovy", "Haskell",
"Java", "JavaScript",  "Lisp", "Perl", "PHP", "Python", "Ruby", "Scala",
"Scheme"];
$( "#prog" ).autocomplete({source: sprachen});
```

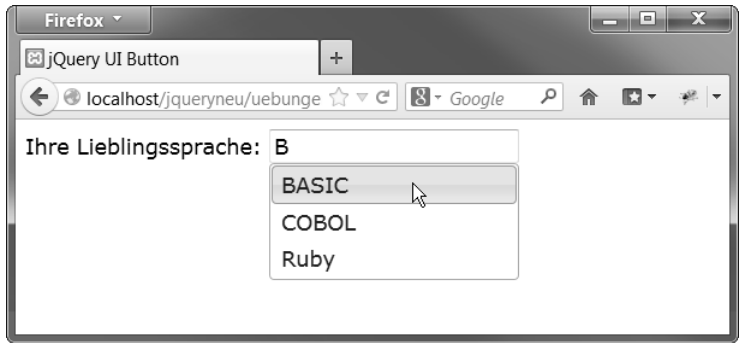

Abbildung 6.32 Autocomplete – alle Items mit »B« werden gezeigt

API des Autocomplete-Widgets

Die komplette API des Autocomplete-Widgets finden Sie hier:

http://api.jqueryui.com/autocomplete

6.8 Effekte und Interaktionen aus jQuery UI

Eine breite Palette der Funktionalitäten aus jQuery UI betrifft nicht Widgets, sondern ist den Kategorien *Effekte* und *Verhalten/Interaktion* zuzuordnen. Die meisten von ihnen sind jedoch auch auf Widgets anwendbar bzw. in deren Defaultverhalten bereits eingebaut.

Ergänzungen

Im Rahmen von jQuery UI werden die auf *Selektionen* anwendbaren jQuery-Methoden durch weitere aus dem Fundus von jQuery UI ergänzt. Mit ihrer Hilfe erstrecken