Bruce Lawson, Remy Sharp

dpi

HTML5

ADDISON-WESLEY

HTML5

KAPITEL 3

Formulare

Bruce Lawson

EINES DER PROBLEME mit HTML 4-Formularen ist, dass es einfach dumme Felder sind. Die Validierung muss natürlich auf dem Server erfolgen, aber man muss das im Browser des Nutzers mit JavaScript duplizieren, damit sie die gute Erfahrung einer flüssigen Anwendung bekommen, die sie verdienen. Praktisch jede Webseite arbeitet mit irgendeiner Art von Formular (Suchfelder, Formulare zum Suchen oder Anmelden etc.). Wäre es da nicht toll, wenn bei Browsern die Validierung für einige der am häufigsten vorkommenden Datentypen, die wir abfragen, bereits integriert ist? Genau, richtig geraten … das geht mit HTML5-Formularen.

Wir ♥ HTML und HTML ♥ uns auch

HTML5 beschleunigt die Erstellung von Formularen. Es gibt ein paar schöne Schmankerl wie die beiden HTTP-Typen der Formularaktion (`update` und `delete`), die sich zu den aktuellen `get` and `post` gesellen.

Was noch attraktiver ist: Ein paar Elemente, die vorher innerhalb eines Formularelements stehen mussten (`<button>`, `<fieldset>`, `<input>`, `<label>`, `<select>`, `<textarea>` sowie `<object>` und die neuen Elemente wie `<keygen>`, `<meter>`, `<output>` und `<progress>`), können nun irgendwo auf der Seite stehen und dennoch mit einem Formular verknüpft sein, und zwar anhand des `form`-Attributs, das auf den `id` seines `form`-*Eigentümers* zeigt.

Schauen wir uns mal dieses Beispiel an:

```
<form id=foo>
<input type="text">
...
</form>
<textarea form=foo></textarea>
```

Das Formular `foo` ist Eigentümer von `<input>`, weil dieses Element darin enthalten ist und es kein `form`-Attribut hat, das diese Eigentümerschaft überschreibt. Die `<textarea>` ist außerhalb des Formulars, gehört ihm aber noch, weil dessen `form`-Attribut auf die `id` seines `form`-Eigentümers zeigt.

So bekommen Sie eine deutlich größere Flexibilität bei der Formatierung, wenn Sie wollen, dass diese Elemente optisch (und strukturell) außerhalb der übergeordneten `form`s erscheinen.

Doch der große Vorteil sind die neuen Formulartypen von HTML5 und die integrierte Validierung, die sie mitbringen. Endlich wird keine JavaScript-Validierung mehr für diese fundamentalen Datentypen benötigt, allerdings können Sie Ihre Scripts noch nicht einmotten: Die neuen Eingabetypen schalten geschmeidig einen Gang herunter, aber Ihr JavaScript wird noch gebraucht, bis eines schönen Tages alle einen HTML5-Browser haben (oder Ihr Chef sagt, dass die Nutzer altertümlicher Browser sich ab jetzt mit serverseitiger Formularprüfung abfinden müssen).

Neue Eingabetypen

Die neuen Formularfelder waren der Ursprung der Spezifikation, aus der dann HTML5 wurde, und hier sehen wir das Prinzip der abwärtskompatiblen Erweiterung in voller Aktion. Diese Erweiterungen sind weitgehend neue Werte des `type`-Attributs des Eingabeelements. Da alle Browser den Standard `<input type=text>` verwenden, werden Legacy-Browser, die die neuen Erweiterungen nicht verstehen, auf den Standard zurückgreifen. Dann können Nutzer Daten in einem einfachen Textfeld eingeben.

ANMERKUNG Die Spezifikation macht keine Vorgaben, wie Browser den Nutzern die neuen Eingabetypen präsentieren oder wie Fehler gemeldet werden sollen etc. Unterschiedliche Browser und Geräte werden verschiedene Benutzerschnittstellen präsentieren; vergleichen Sie beispielsweise die verschiedenen Wege, wie ein Auswahlfeld bei Safari auf dem Desktop und bei Safari auf dem iPhone gezeigt wird (**Abbildung 3.1**).

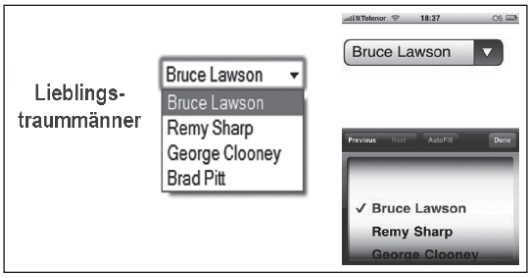


ABBILDUNG 3.1: Die gleiche Auswahlbox – links in Safari (Windows) und rechts Safari (iPhone). Die meisten Screenshots wurden mit Opera gemacht, weil dieser Browser zum Zeitpunkt dieses Schreibens (Mai 2010) die vollständigste Implementierung aufweist, aber wo etwas in einem anderen Browser implementiert ist, nutzen wir das.

Der Eingabetyp email

Dem Markup `<input type=email>` entnimmt der Browser, dass er das Formular nicht übermitteln soll, falls der Nutzer eine ungültige E-Mail-Adresse eingetragen hat – d.h., er prüft nicht, ob diese Adresse existiert, sondern nur, ob ihr Format gültig ist. So wie bei allen Eingabetypen kann der Nutzer das Formular auch übermitteln, wenn dieses Feld nicht ausgefüllt ist – außer das `<required>`-Attribut ist vorhanden.

Wie der Browser Fehler meldet, ist nicht definiert. Die (experimentelle) Implementierung in Opera 10.50 wird in **Abbildung 3.2** gezeigt.

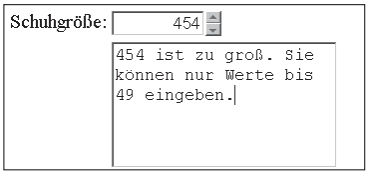


ABBILDUNG 3.2: Eine automatisch generierte Fehlermeldung in Opera 10.50

Das erlaubte `<multiple>`-Attribut bedeutet, dass als Wert dieses Felds eine durch Komma getrennte Liste von gültigen E-Mail-Adressen erlaubt ist. Natürlich braucht nun der Nutzer seine kommaseparierte Liste nicht manuell einzutippen. Dafür könnte der Browser z.B. auch eine Liste der Kontakte des Nutzers aus seinem Mail-Client oder der Handy-Kontaktliste mit Auswahlkästchen erscheinen lassen.

Das experimentelle Firefox-Addon Contacts (http://mozilla-labs.com/blog/2010/03/contacts-in-the-browser) sammelt Kontaktdaten aus verschiedenen Quellen, aus denen die Adressen angezeigt werden, sobald ein Nutzer auf ein `<input type=email>` stößt. Diese Kontaktinformationen stellt das Addon über die Contacts-API des W3C-Drafts auch Website-Scripts zur Verfügung (http://www.w3.org/2009/dap/).

Der Eingabetyp URL

Mit `<input type=url>` prüft der Browser, ob im Formularfeld eine korrekte URL eingetragen wurde. Dabei kann der Browser dem Nutzer auch Hilfe anbieten. Opera zeigt beispielsweise eine Liste der kürzlich besuchten URLs aus der Chronik des Browsers und stellt automatisch den URLs, die mit „www.“ beginnen, ein „http://“ voran. (Eine URL muss nicht unbedingt eine Web-URL sein. Die Seite könnte z.B. auch ein webbasierter HTML-Editor sein, in dem der Nutzer das Pseudoprotokoll `tel:` verwendet.)

Der Eingabetyp date

Der `date`-Typ gehört zu meinen Favoriten. Wir kennen alle solche Webseiten, in denen man Daten für einen Flug, ein Konzertticket etc. eingeben muss. Weil Datumsangaben gar nicht so einfach sind (soll man TT-MM-JJJJ oder MM-TT-JJJJ oder TT-MMM-JJ eingeben?), nehmen Entwickler für die Datumsauswahl meist JavaScript-Widgets (engl. *datepicker*), die auf den verschiedenen Sites sehr vielfältig erscheinen und auch bei Usability und Accessibility große Unterschiede aufweisen.

`<input type=date>` löst dieses Problem. Bei Opera wird beispielsweise ein Kalender-Widget geöffnet (**Abbildung 3.3**).

Beim BlackBerry-Browser in der BlackBerry Device Softwareversion 5.0 ist die Datumseingabesteuerung, mit der das Datumseingabefeld implementiert wird, die gleiche Java-Komponente, die bei der nativen BlackBerry-Kalender-App verwendet wird (obwohl sie nicht in der Kalender-App integriert ist) – siehe **Abbildung 3.4**.

ABBILDUNG 3.3: Opera 10.50 stellt ein Kalender-Widget dar.

ABBILDUNG 3.4: `<input type=date>` im BlackBerry-Browser

Natürlich steckt HTML5 hier noch in den Kinderschuhen. Der Browser könnte auch die native Kalenderapplikation aufrufen, damit Sie dort bereits vorhandene Termine einsehen können. Der Punkt ist, dass der Browser jetzt versteht, was Sie meinen. Vorher waren Datepicker aus Perspektive des Browsers nur `<div>`s, `<span>`s und Links mit einer Menge angehängter JavaScript-Verhaltensweisen. Nun weiß der Browser, dass Sie wirklich eine echte Datums- bzw. Zeitangabe eintippen, und kann vielfältigere Steuerungsmöglichkeiten und die Integration mit anderen Datums-/Zeitinformationen anbieten.

Der Eingabetyp time

Mit `<input type=time>` wird eine Zeit im 24-Stunden-Format angegeben und validiert. Auch hier bleibt die eigentliche Benutzerschnittstelle dem Browser überlassen. Sie kann ganz einfach im Eintippen von Zahlen bestehen. Wenn der Nutzer bei der Uhrzeit eine Zahl größer als 24 für Stunden oder 59 für Minuten eintippt, wird ein Fehler ausgegeben. Es könnte aber auch deutlich ausgefeilter sein: z.B. ein Zifferblatt mit per Maus verschiebbaren Zeigern. Die Benutzerschnittstelle kann auch einen Eintrag für eine Zeitzonenverschiebung ermöglichen.

Der Eingabetyp datetime

Mit `date` und `time` wird eine präzise Datums- und Zeitangabe validiert. Ein lokales Datum bzw. eine Zeitangabe funktioniert mit `datetime`, außer dass der Browser es dem Nutzer nicht erlaubt, eine Zeitzonenverschiebung einzutragen (oder zu ändern).

Der Eingabetyp month

Mit `<input type=month>` wird ein Monat eingetragen und validiert. Obwohl dies intern als Zahl zwischen 1 und 12 gespeichert wird, kann der Browser eine Auswahlmethode über den Monatsnamen anbieten. Das kann man anhand einer Auswahlbox mit den zwölf Optionen für Januar bis Dezember lösen, aber das ist nicht lokalisierbar. Wenn man den HTML5-Eingabetyp für den Monat verwendet, kann die französische Lokalisierung eines Browsers z.B. eine Drop-down-Liste mit „Janvier“ statt Januar anbieten. So hat der Browser mehr zu tun, *aber das bedeutet weniger Arbeit für Sie* und so soll es ja auch sein.

Der Eingabetyp week

Mit `<input type=week>` wird eine Kalenderwoche eingetragen und validiert. Eigentlich würde ein einfaches Eingabefeld reichen, in dem der Nutzer eine Zahl eintippt, aber in der Praxis ist es etwas komplexer: Manche Jahre haben 53 Wochen. Somit steht das Format 2010-W07 für die siebte Woche des Jahres 2010.

Opera bietet ein UI mit Datumswahl, in dem ins Eingabefeld die Kalenderwoche eines beliebigen Datums eingetragen wird (anstatt im Format YYYY-MM-DD) – **siehe Abbildung 3.5**.

ABBILDUNG 3.5: So rendert Opera 11.01 `<input type=week>`.

Der Eingabetyp number

Beim Eingabetyp `number` wird (wenig überraschend) ein Fehler ausgegeben, wenn der Nutzer etwas anderes als Zahlen eintippt. Er funktioniert perfekt mit den Attributen `min`, `max` und `step`. Bei Opera wird er als Drehfeld (*spinner*) dargestellt, das die obere und untere Grenze (wenn sie angegeben sind) nicht über- bzw. unterschreitet und anhand des in `step` angegebenen Inkrements hochgezählt wird. Allerdings kann der Nutzer den Wert auch eintippen (**Abbildung 3.6**).

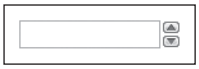


ABBILDUNG 3.6: So rendert Opera 10.50 `<input type=number>`.

Der Eingabetyp range

`<input type=range>` wird als Schieberegler (*slider*) gerendert. **Abbildung 3.7** zeigt die Darstellung in Google Chrome.

ABBILDUNG 3.7: So rendert Opera 10.50 `<input type=range>`.

Früher mussten diese Schieberegler nachgeahmt werden, indem man eine Eingabe mit JavaScript und Bildern der Zeiger zusammenbaut. Weil das nicht nativ im Browser geschieht, musste man sehr sorgfältig arbeiten (und noch mehr Code schreiben), damit die Accessibility per Tastatur möglich ist. Nun sind Schieberegler nativer Teil von HTML5 und die Verantwortung liegt nicht mehr beim Entwickler, was zu schlankerem Code und einer besseren Accessibility für Tastaturnutzer führt.

Weitere Informationen finden Sie im Beispiel des Abschnitts „Alles zusammenführen“.

Der Eingabetyp search

Dieser Eingabetyp erwartet einen Suchbegriff. Bei Safari gibt es außerdem ein unspezifiziertes Attribut, das anhand des Attributs `results=n` einen Verlauf kürzlich ausgegebener Resultate einfügt. Die Eingabetypen `search` und `text` unterscheiden sich nur stilistisch und bei Safari auf dem Mac werden für `search` die abgerundeten Standardecken des Betriebssystems verwendet, aber das kann man auch mit ein bisschen CSS überschreiben (Dank an Wilf Nas für diesen Tipp):

```
input[type="search"] {-webkit-appearance: textfield;}
```

Der Eingabetyp tel

Dieser Eingabetyp erwartet eine Telefonnummer. Es gibt keine besondere Validierung und man wird nicht einmal gezwungen, nur Zahlen einzugeben, weil viele Telefonnummern üblicherweise mit Zusatzzeichen geschrieben werden, z.B. +44 (0) 208 123 1234.

Da viele Handys ihre eigene Telefonnummer „kennen“, ist zu erwarten, dass die meisten Handys in der Lage sein werden, diese Eingabefelder automatisch zu vervollständigen. Aktuell macht das kein Handy, aber das iPhone zeigt ein Eingabefeld für Telefonnummern an (**Abbildung 3.8**).

ABBILDUNG 3.8: Die Wähltastatur des iPhone zur Vervollständigung von `<input type=tel>`

Der Eingabetyp color

Mit `<input type=color>` kann der Nutzer über einen Farbselektor einen Farbwert eingeben. Bisher ist das nur im BlackBerry implementiert (**Abbildung 3.9**).

ABBILDUNG 3.9: `<input type=color>` im BlackBerry

Neue Attribute

Neben den neuen Eingabetypen hat das `<input>`-Element auch mehrere neue Attribute, um ein bestimmtes Verhalten und Grenzwerte festzulegen: `autocomplete`, `min`, `max`, `multiple`, `pattern` und `step`. Es gibt auch das neue Attribut `list`, das mit einem der neuen Elemente verbunden für neue Dateneingabemethoden sorgt.

Das Attribut list

Das `<datalist>` erinnert an die Auswahlbox, aber nun können die Nutzer auch eigenen Text eingeben, wenn sie keine der vordefinierten Optionen wählen wollen. Die Liste ist in dem neuen `<datalist>`-Element enthalten, auf dessen `id` im Wert des `list`-Attributs verwiesen wird:

```
<input id=form-person-title type=text list=mylist>
      <datalist id=mylist>
          <option label=Mr value=Mr>
          <option label=Ms value=Ms>
          <option label=Prof value="Mad Professor">
      </datalist>
```

`<datalist>` selbst wird nicht gerendert, sondern als Wert in einem auswahlähnlichen Feld gezeigt.

Das vorige Beispiel nutzt `type=text`, um freie Eingaben zu ermöglichen, aber Sie können `<datalist>` auch mit einem der bereits erwähnten Eingabetypen verwenden, z.B. `url`, `mail` etc. Es ist auch möglich, die Optionen noch während der Eingabe des Nutzers dynamisch neu zu füllen – so wie bei Suggest, der Vorschlagsfunktion von Google. Weitere Details finden Sie unter **http://dev.opera.com/articles/view/an-html5-style-google-suggest/**.

Vielfach wurde gefragt, warum das `<input>`/`<datalist>`-Paar nicht zu einem neuen Element wie `<select>` kombiniert wurde. Die Antwort liegt in der Abwärtskompatibilität: Bei Legacy-Browsern wird `<input>`/`<datalist>` zu `<input type=text>` herabgestuft. So kann der Nutzer zumindest etwas eingeben und für solche Browser können Sie die vollständige Implementierung dann wenigstens mit JavaScript faken.

Das Attribut autofocus

Das boolesche Attribut `autofocus` ermöglicht einen deklarativen Weg, um den Fokus beim Laden der Seite auf eine Formularsteuerung zu setzen. Früher musste ein Entwickler noch JavaScript mit `control.focus()` bemühen. Durch diesen neuen Weg können Browser schlaue Dinge tun, z.B. den Fokus nicht wirklich auf die Steuerung setzen, wenn der Nutzer bereits woanders etwas eintippt (ein häufiges Problem von JavaScript-Scripts der alten Schule mit `onload focus`).

Es sollte pro Seite nur ein solches Eingabefeld geben. Aus Sicht der Usability sollte man dieses Attribut mit Sorgfalt behandeln. Wir empfehlen den Einsatz nur auf Seiten, deren Hauptzweck ein Formularfeld ist, z.B. ein Suchformular.

Das Attribut placeholder

Ein häufig von Entwicklern eingesetzter Usability-Trick ist, für den Nutzer schon mal einen Hinweistext ins Eingabefeld zu setzen. Wenn dieser dann den Fokus auf das Feld setzt, verschwindet der Text und kehrt zurück, wenn der Fokus weg ist. Dafür brauchte man JavaScript. Nun kann das aber deklarativ mit dem Attribut `placeholder` gemacht werden. Die Spezifikation sagt dazu: „Für einen umfangreicheren Hinweis oder andere textliche Ratschläge ist das Attribut `title` passender."

Wenn Browser mal eine helfende Hand brauchen

Bei älteren – und auch manchen neueren – Browsern sind einige dieser Features nicht nativ verfügbar. Das ist aber kein sonderliches Problem, weil die *meisten* der neuen HTML5-Features mit JavaScript (oder anderen Technologien wie Flash und Silverlight) *gepatcht* werden können, um dem Browser hier auf die Sprünge zu helfen. Das ist weder Graceful Degradation noch eine progressive Verbesserung, sondern etwas anderes. Meine Kollegen und ich nennen es *Polyfilling*, wobei wir uns in unserem Code auf eine native Funktionalität verlassen und die Lücken dann anhand einer externen Bibliothek füllen. Diese Bibliothek kann generisch oder auch spezifisch sein. Wenn Sie beispielsweise die Attribute `autofocus` oder `placeholder` einsetzen wollen, können Sie eine kleine JavaScript-Bibliothek einbauen und alle aktuellen Browser von IE 6 bis Firefox 3.6 unterstützen nun diese beiden Attribute (http://introducinghtml5.com/examples/ch03/polyfilling.html).

Dieses Attribut wird aktuell von auf WebKit basierenden Browsern dargestellt (ursprünglich war es eine proprietäre HTML-Extension von Apple) und Firefox 4 Beta: http://blog.oldworld.fr/index.php?post/2010/03/19/Placeholder-attribute-in-Firefox-3.7-alphas.

Das Attribut required

Das neue Attribut `required` kann man bei `<textarea>` und den meisten Eingabefeldern nutzen (außer wenn das `type`-Attribut `hidden` oder `image` ist oder bei solchen Button-Typen wie `submit`). Der Browser erlaubt die Übermittlung des Formulars nicht, wenn diese erforderlichen Felder leer sind. Wir empfehlen, bei solchen Eingabefeldern auch das ARIA-Attribut `aria-required` einzufügen (siehe die Ausführungen zu ARIA in Kapitel 2).

Das Attribut multiple

In HTML5 ist `<input type=file>` nicht neu, aber in Kombination mit dem neuen Attribut `multiple` kann der Nutzer nun mehrere Dateien hochladen:

```
<input type=file multiple>
```

Das geht auch gemeinsam mit anderen Eingabetypen: Mit `<input type=email multiple>` kann der Nutzer beispielsweise mehrere E-Mail-Adressen eingeben. Aktuell ist dies nur in WebKit-Browsern implementiert.

Das Attribut pattern

Einige der bereits erwähnten Eingabetypen (`email`, `number`, `url` etc.) sind sozusagen eingebackene reguläre Ausdrücke (*regular expressions*), weil der Browser nur prüft, ob die eingegebenen Werte so aussehen, wie sie sein sollen.

> **ANMERKUNG** Wenn Sie sich vor regulären Ausdrücken fürchten, aber mehr wissen möchten oder Ihr Fachwissen darüber erweitern wollen, dann werfen Sie doch einen Blick auf Steven Levithans Blog, das sich fast ausschließlich mit diesem Thema beschäftigt: **http://blog.stevenlevithan.com/**

Angenommen, Sie wollen, dass der reguläre Ausdruck einem anderen Muster entspricht. Mit dem `pattern`-Attribut können Sie einen maßgeschneiderten regulären Ausdruck angeben, zu dem die Eingabe passen muss. Falls also der Nutzer immer nur eine Zahl plus drei Großbuchstaben eingeben muss, wäre der reguläre Ausdruck eine Zahl [0-9] und drei Großbuchstaben [A-Z]{3}. Das Eingabefeld wird wie folgt codiert:

```
<input pattern="[0-9][A-Z]{3}" name=part
        title="Eine Teilenummer ist eine Zahl,
        ¬gefolgt von drei Großbuchstaben.">
```

Sie könnten auch einen `placeholder="9AAA"` o.Ä. als kurzen Hinweis einfügen.

In der Spezifikation wird erklärt, dass die Syntax der regulären Ausdrücke im Attribut `pattern` mit der Syntax der regulären Ausdrücke in JavaScript übereinstimmt, es sei denn, dass am Anfang ein implizites `^(:?` und am Ende ein `)$` steht.

Wenn Sie es also gewohnt sind, reguläre Ausdrücke zu nutzen, ist Ihnen ja bereits klar, was Sie tun müssen. Falls nicht, steht Ihnen nun die wunderbare Welt der regulären Ausdrücke zum Erforschen offen!

Das Internet steckt voller regulärer Ausdrücke in JavaScript, die zu diesem oder jenem passen. Also finden Sie wahrscheinlich schnell, wonach Sie suchen. Wenn reguläre Ausdrücke einfach gehalten werden, bringt man sie ganz leicht zum Funktionieren.

Damit beispielsweise ein ZIP-Code (Postleitzahl der USA) im Format 99999 oder 99999-9999 (wobei 9 für alle möglichen Zahlen steht) angegeben wird, können Sie Folgendes schreiben:

```
<input pattern="[0-9]{5}(\-[0-9]{4})?" title="Ein ZIP-
¬Code im Format 99999 oder 99999-9999">
```

Dieser reguläre Ausdruck sucht nach einer fünfstelligen numerischen Sequenz mit einem Bindestrich und dann eine weitere Folge mit vier Zahlen als optionales Suffix.

Wir könnten das auch erweitern, damit es zu britischen Postleitzahlen passt (unter Verwendung eines vereinfachten Postleitzahlencodes), und das Muster wird komplizierter:

```
<input required pattern="[0-9]{5}(\-[0-9]
¬{4})?|[a-zA-Z]{1,2}\d{1,2}\s?\d[a-zA-Z]{1,2}"
¬name=part title="Ein gültiger ZIP-Code oder
¬britischer Postcode">
```

Nun ist der reguläre Ausdruck deutlich komplizierter und Tests dieses Musters können ziemlich vertrackt werden. Weil der reguläre Ausdruck von `pattern` zur Syntax eines regulären Ausdrucks von JavaScript passt, können wir das in einer Browser-Konsole wie Firebug oder Opera Dragonfly testen und wir prüfen mit reinem JavaScript, ob das Muster funktionieren wird. Im folgenden Beispiel teste ich nur, ob ein britischer Postcode eingegeben wurde, und nehme die JavaScript-Testmethode zum Experimentieren. Beachten Sie, dass ich meinen Tests ein `^(:?` voranstelle und dahinter ein `)$` schreibe, so wie es die HTML5-Spezifikation vorschreibt:

```
/^(:?[a-zA-Z]{1,2}\d{1,2}\s?\d[a-zA-Z]{1,2})$/.
¬test("bn14 8px")
> true
/^(:?[a-zA-Z]{1,2}\d{1,2}\s?\d[a-zA-Z]{1,2})$/.
¬test("bn149 8px")
> false
```

Diese Ergebnisse sind korrekt, weil „bn149“ kein gültiger Teil eines (britischen) Postcodes ist, nicht mal in diesem erfundenen Beispiel. Schließlich bleibt noch anzumerken, dass das Attribut `pattern` *case sensitive* ist, und weil es keine Möglichkeit gibt, in einen Modus zu wechseln, bei dem die Schreibweise egal ist, müssen wir in diesem Beispiel explizit darauf achten, ob darin Groß- bzw. Kleinbuchstaben vorkommen.

Das Attribut autocomplete

Die meisten Browser haben irgendeine Art der automatischen Vervollständigung. Mit dem neuen Attribut `autocomplete` können Sie steuern, wie das funktioniert.

Der Standardstatus ist, dass die Eingabe den `autocomplete`-Status des `form`-Eigentümers erbt. In Formularen ist `autocomplete` standardmäßig aktiviert.

Wenn das Attribut `autocomplete` eines Formularelements aktiviert ist, kann das Feld mit der automatischen Vervollständigung arbeiten.

Ich zitiere die Beschreibung des `off`-Zustands aus der Spezifikation mit ihrem schrägen Humor: „Der `off`-Zustand zeigt entweder an, dass die Eingabedaten des Steuerungselements besonders sensibel sind (z.B. der Aktivierungscode für eine Nuklearwaffe) oder dass es sich um einen Wert handelt, der nie wieder verwendet wird (z.B. ein einmaliger Schlüssel für das Login beim Online-Banking). Der Nutzer muss von daher die Daten explizit jedes Mal eingeben.“

Die Attribute min und max

Wie schon bei `<input type=number>` erwähnt, begrenzen diese Attribute den eingebbaren Wertebereich. Sie können das Formular nicht mit einer Zahl übermitteln, die kleiner als `min` oder größer als `max` ist. Aber man kann sie auch für andere Eingabetypen nehmen. So akzeptiert `<input type=date min=2010-01-01 max=2010-12-31>` kein Datum, das sich nicht im Jahr 2010 befindet. Es ist trivial, den Server HTML ausgeben zu lassen, dessen `min` das heutige Datum ist, wodurch nur zukünftige Termine erlaubt sind (z.B. bei einer Online-Flugbuchung) oder ein `max` mit dem heutigen Datum (wenn z.B. das Geburtsdatum eingetragen werden soll).

Das Attribut step

Mit `step` werden die Schrittstufen einer Eingabe (also die Granularität) gesteuert. Wenn der Nutzer also in Fünferschritten einen Prozentwert zwischen 0 und 100 eingeben soll, schreiben Sie:

```
<input type=number min=0 max=100 step=5>
```

und das Drehfeld wird jeweils um 5 inkrementiert.

Bei einer Zeitsteuerung können Sie auch `step=any` schreiben. So kann jede Tageszeit mit beliebiger Genauigkeit gewählt werden (z.B. mit einer Tausendstelsekunde oder mehr). Normalerweise beschränkt man Zeitsteuerungen auf die Genauigkeit von einer Minute:

```
<input name=favtime type=time step=any>
```

Alles zusammenführen

Es ist ganz schön verwirrend, wenn man den Überblick behalten will, welche Attribute zu welchen Eingabetypen gehören, wenn man sie so wie hier alle auf einmal serviert bekommt. Aber wenn man sich erst einmal näher damit beschäftigt, ist alles eigentlich recht unkompliziert. Man kann `min` und `max` natürlich beispielsweise nicht bei einem `<textarea>` nehmen, weil es nicht sinnvoll wäre, aber `required` geht.

Ein Formular für Blog-Kommentare

Schauen wir uns ein klassisches Formularbeispiel an, mit dem die meisten wohl schon vertraut sind. Fast alle Blogs haben einen Kommentarbereich, in dem Felder für den Namen des Kommentators (erforderlich), die E-Mail-Adresse (erforderlich), URL (optional) und der Kommentar (erforderlich) stehen. Dazu müsste ganz schön viel JavaScript aufgefahren werden, wenn wir die Formularvalidierung per Hand machen wollten.

Bei HTML5 sind nur ein paar neue Formulartypen erforderlich, jeder mit einem `name`-Attribut, die von der Spezifikation her erforderlich sind, um die automatische Validierung durchzuführen. Wir bauen auch einen Submit-Button ein. Aktuell validiert Opera die Felder nur, wenn ein Formular tatsächlich gesendet wurde.

```
<form>
 <label for=form-name>Name</label>
 <input name=form-name id=form-name type=text required>
 <label for=form-email>E-Mail</label>
 <input name=form-email id=form-email type=email
 ¬required>
 <label for=form-url>URL</label>
 <input name=form-url id=form-url type=url>
 <label for=form-comment>Kommentar</label>
 <textarea name=form-comment id=form-comment required>
 </textarea>
 <input type=submit>
</form>
```

Schwupps, fertig! Braucht gar kein JavaScript!

Ein Schieberegler mit gescriptetem Output

Wir haben `<input type=range>` kennengelernt. Nun werden wir also ein Beispiel programmieren, das dem Nutzer auf dem Schieberegler automatisch die erlaubten Minimal- und Maximalwerte anzeigt und den aktuellen Wert des Schiebereglers dynamisch ausgibt.

Der Schieberegler wird von 1 bis 11 gehen, wie das bei allen guten Steuerungen der Fall sein sollte (z.B. für Gitarrenverstärker oder anderes). Es wird standardmäßig immer 1 hochgezählt, also können wir dieses Attribut weglassen.

```
<input type=range min=1 max=11 name=tap value=5>
```

Damit der Nutzer die Minimal- und Maximalwerte sehen kann, nehmen wir generierten Inhalt (was bei Schiebereglern in WebKit-Browsern nicht funktioniert):

```
input[type=range]::before {content: attr(min);}
input[type=range]::after {content: attr(max);}
```

Das zeigt Werte, die dann wie im CSS definiert formatiert werden. Dies hier wird z.B. wie in **Abbildung 3.10** dargestellt:

```
input[type=range]{width:500px; color:orange; font-
¬weight: bold; font-size:1em;}
```

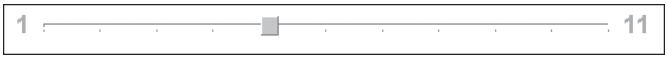


ABBILDUNG 3.10: So rendert Opera `<input type=range>` mit generierten `min-` und `max-`Werten.

Wir geben den aktuellen Wert des Schiebereglers mit dem neuen `output`-Element zurück.

Das <output>-Element

Mit dem `<output>`-Element werden die Ergebnisse einer Berechnung oder eines anderen Script-Vorgangs gezeigt. Es kann einen `form`-Eigentümer haben, entweder dadurch, dass es sich im Formular selbst befindet oder über ein Formularattribut. Die neuen Elemente `<progress>` und `<meter>` können genauso mit einem Formular verknüpft werden, damit der Nutzer Feedback bekommt.

Wir verbinden es mit dem Schieberegler über dessen Namen (`name=tap`) und das `onforminput`-Event. Wenn der `form`-Eigentümer des `output` eine Eingabe bekommt (wenn also der Schieberegler bewegt wird), geben wir den Wert dieses Inputs wieder zurück:

```
<output onforminput="value=tap.value">5</output>
```

Der eigentliche Inhalt des `<output>`-Elements (in diesem Fall „5“) wird nur gezeigt, bevor der Regler verändert wird. Sie sollten gut aufpassen, dass dieser Wert der gleiche ist wie die Wertattribute des damit verknüpften Eingabefelds. In diesem Beispiel setzt der `value=…`-Teil den sichtbaren Wert immer dorthin, worauf Sie den Regler gesetzt haben. Das könnte man als ähnlich wie `this.innerHTML` betrachten und `value` fungiert nur als Shortcut.

Das `<output>`-Element kann mit CSS formatiert werden (obwohl nur Opera das aktuell unterstützt).

Mit WAI-ARIA einen Übergang für die Erreichbarkeit schaffen

Zwar haben wir gesagt, dass durch `<input type=range>` die Verantwortung für die Accessibility dem Entwickler von den Schultern genommen wurde, aber das gilt nur, wenn HTML5 allgemein unterstützt wird und die assistive Technologie diese neue Art der Formulareingabe auch versteht.

Wenn Sie in der Zwischenzeit mit HTML5-Schiebereglern arbeiten wollen, sollten Sie auch WAI-ARIA-Informationen einbauen (was momentan dann noch zu einer Doppelung führt):

```
<input id=tap
       name=tap
       type=range
       min=1
       max=10
       value=0
       role=slider
       aria-valuemin=1
       aria-valuemax=11
       aria-valuenow=0>
```

Durch `role=slider` weiß eine assistive Technologie, wie die Steuerung den Steuerelementen des Betriebssystems zugeordnet werden soll. Sie sollten `aria-valuenow` mit JavaScript aktualisieren, wenn der Schieberegler die Position ändert. In diesem Fall sollten Sie das `change`-Event an den Schieberegler binden. In unserem Beispiel nehmen wir einfach das Attribut `onchange`. Leider können wir für ein Update des Werts `aria-valuenow` nicht die Eigenschaftssyntax verwenden. Wir müssen das DOM-Attribut aktualisieren, damit der Wert korrekt aktualisiert wird.

```
<input id=tap
       name=tap
       type=range
       min=1
       max=10
       value=0
       role=slider
       aria-valuemin=1
       aria-valuemax=11
       aria-valuenow=0
       onchange="this.setAttribute('aria-valuenow',
       ¬ this.value)">
```

So wird der Wert des Attributs `aria-valuenow` aktualisiert und kann abgefragt werden, wenn Sie das Element mit einem DOM-Inspector untersuchen.

Abwärtskompatibilität bei Legacy-Browsern

ANMERKUNG Alternativ können Sie das auch von der Modernizr-Bibliothek erledigen lassen. Dabei handelt es sich um eine kleine Library (aktuell 7K), die eine Feature Detection durchführt und JavaScript-Booleans wie `Modernizr.inputtypes[email]` zurückgibt, die auf `true` oder false gesetzt sind. Ein paar Haken hat die Sache noch (siehe „Achtung bei WebKit!“), aber es wird aktiv weiterentwickelt. Sie können es von www.modernizr.com herunterladen.

Die große Frage lautet: Was stellen wir mit den Legacy-Browsern an? Die Antwort ist, dass Sie Ihre bereits vorhandene JavaScript-Validierung noch nicht einzumotten brauchen, sondern sie als Plan B behalten, nachdem eine Feature Detection durchgeführt wurde. Um beispielsweise zu erkennen, ob `<input type=email>` unterstützt wird, erstellen Sie mit JavaScript ein neues `<input type=email>`, aber fügen es nicht der Seite hinzu. Dann befragen Sie Ihr neues Element, welches Typattribut es hat. Wenn hier nun `email` zurückgemeldet wird, unterstützt der Browser das neue Feature – also lassen Sie ihn in Ruhe seine Arbeit machen und bringen Sie keine JavaScript-Validierung mit rein. Lautet die Rückmeldung `text`, dann wird auf Plan B zurückgegriffen, weil das andere ja nicht unterstützt wird. Also sollte Ihr Code in einer alternativen Validierungsbibliothek ideal anhand einer *Lazy Load*-Technik geladen werden. Somit brauchen HTML5-konforme Browser kein JavaScript herunterzuladen und zu verarbeiten, das gar nicht erforderlich ist.

```
var i = document.createElement("input");
i.setAttribute("type", "email");
return i.type !== "text";
```

Man kann auch Attribute testen:

```
return ,autofocus' in document.createElement("input");
```

Achtung bei WebKit!

Aktuell behaupten die Desktop-basierten WebKit-Browser, namentlich Safari und Google Chrome, die Eingabetypen `email`, `url`, `tel` und `number` anhand dieser Erkennungsmethoden zu unterstützen. Tatsächlich unterstützen die veröffentlichten Browser über diese Eingabetypen noch gar nichts und es wird nur deswegen so etwas gemeldet, weil Mobile Safari (auch WebKit) diese Eingabetypen unterstützt, um abhängig vom Typ eigene Tastaturen zu ermöglichen (siehe das iPhone in Abbildung 3.8). Hier bietet der Browser einen gerätespezifischen Support, üblicherweise per Tastatur (vor allem auf dem iPhone) und abhängig vom Eingabetyp.

Was bringt Ihnen das alles? Nun, zuerst und am wichtigsten sichern Sie Ihren Code für die Zukunft ab und sind auf die Zeit vorbereitet, wenn alle Browser diese äußerst nützlichen Ergänzungen bei Formularen unterstützen. Außerdem gewinnen Sie Bonuspunkte in Sachen Usability und Accessibility.

Neue Formularfelder und Fehlermeldungen formatieren

Immer wenn wir diese neuen, intelligenten Formularfelder auf Konferenzen vorstellen, steht jemand auf und will wissen, wie man sie formatiert. Bei den meisten dieser neuen Steuerelemente kann man einfache Formatierungen vornehmen: Schrift, Farbe und dergleichen. Manche Aspekte des Moduls CSS Basic User Interface (http://www.w3.org/TR/css3-ui/) sind auf diese Elemente anwendbar, z.B. die Pseudoklassen `:invalid` und `:required`. Aber wenn man alle Wochenenden in der Datumsauswahl grün oder die Fehlermeldungen rosa machen wollte, ginge das nicht. Die Selektoren und CSS-Ansatzpunkte für diese Teile der neuen Formularsteuerungen wurden noch nicht spezifiziert.

Das ist keine schlechte Sache. Die Leute aus Ihrer Branding-Abteilung werden natürlich lamentieren, dass der Platzhaltertext nicht in den Firmenfarben erscheint. Aber in Sachen Usability und Accessibility ist das ein Gewinn. Obwohl es verführerisch ist, die „Füllung“ der Formularfelder zu formatieren (was übrigens in den Wahnsinn führen kann – siehe http://meyerweb.com/eric/thoughts/2007/05/15/formal-weirdness/), sollten Sie nicht auf die Branding-Leute hören und Formulare möglichst in den Browser-Standardeinstellungen belassen. Schieberegler und Datepicker unterscheiden sich dann auf verschiedenen Sites nicht und sind somit für die Nutzer deutlich einfacher verständlich. Es ist viel besser, wenn die Datumsauswahl auf Site X so aussieht und sich so verhält wie eine auf Site Y oder Z.

Und wenn Sie die native Steuerung verwenden, anstatt Schieberegler und Datumsauswahl mit JavaScript zu faken, sind Ihre Formulare für Nutzer assistiver Technologien höchstwahrscheinlich viel zugänglicher.

Standardeinstellungen der Browser überschreiben

Wenn wir die neuen Steuerungstypen für Formulare unter HTML5 oder das Attribut `required` einsetzen, gibt uns der Browser eine vorgefertigte Validierung mit seinen eigenen integrierten Fehlermeldungen. Das ist prima, doch was machen Sie, wenn diese Fehlermeldungen angepasst werden sollen? Was ist, wenn heute „Sprich wie ein Pirat“-Tag ist? Dann will ich vielleicht auch alle Validierungsmeldungen auf „Piratisch“ ausgeben.

Mit ein bisschen JavaScript geht das. Es ist nicht möglich, über Inhaltsattribute eigene Validierungsmeldungen zu setzen (die sind im Markup fest kodiert), aber wir können sie in die DOM-Attribute einfügen, weil man per JavaScript darauf zugreifen kann. Also setzen wir sie entweder, bevor die Validierung startet, oder geben unsere eigene Validierungsmeldung an, wenn das Formular übermittelt wird, und erstellen an diesem Punkt der Ausführung dynamisch eine Validierungsmeldung, die den Wert enthält, der den Fehler verursacht hat.

Das Folgende soll also nun nicht mehr

Bitte geben Sie eine gültige E-Mail-Adresse an

heißen, sondern wir ändern die Validierung, damit in „traditionellem Piratisch“ Folgendes erscheint:

bei uns an Bord gibt’s kein humptydumpty – aye!

Mit der Methode `setCustomValidity` kann ich meine eigene Validierungsmeldung machen:

```
var email = document.getElementById('email');
email.form.onsubmit = function () {
  email.setCustomValidity("bei uns an Bord gibt's kein
  ¬" + email.value + " - aye!");
};
```

Wie Opera diese selbst gemachten Validierungsmeldungen rendert, sehen Sie in den **Abbildungen 3.11** und **3.12**.

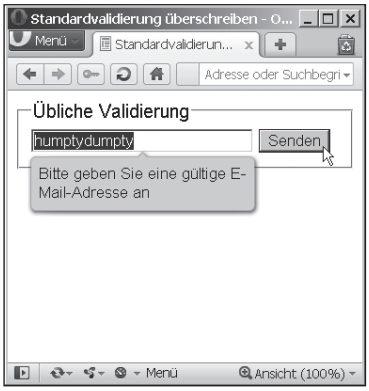


ABBILDUNG 3.11: Opera stellt die Standardvalidierungsmeldung für email dar.

ABBILDUNG 3.12: So rendert Opera unsere Validierung auf „Piratisch".

Opera ist aktuell der einzige Browser, der diese nativen Validierungsmeldungen unterstützt und somit gibt es dafür keine Benchmarks. Ich sage das, weil Sie je nach Version des Opera-Browsers merken werden, dass Sie vor der eigenen Validierungsmeldung das Präfix „The value humptydumpty is not allowed by a script on the page!" (Der Wert humptydumpty wurde von einem Skript auf der Seite nicht erlaubt!) bekommen. Das bleibt weiter in der Hand des Browsers, wogegen wir momentan (zum Zeitpunkt dieses Schreibens) nichts machen können. Darauf müssen Sie also aufpassen, falls Sie eigene Validierungsmeldungen schreiben wollen.

Wenn es Ihnen allerdings nicht liegt, wie diese eigenen Validierungsmeldungen funktionieren, gibt es noch einen Weg, um ein eigenes Validierungsverhalten zu bauen, damit es sich besser in Ihre Applikation integriert. Wenn wir `setCustomValidity` ausführen, setzt es das Nur-lesen-DOM-Attribut `validationMessage`. Das können wir nutzen, wenn wir die Validierung selbst verwalten. Schauen wir uns das im nächsten Abschnitt einmal an.

Eigene Validierung per JavaScript

Neben den Inhaltsattributen, mit denen eine clientseitige Validierung und Webformular-Widgets aktiviert werden, gibt es wie meist bei den HTML5-Spezifikationen eine begleitende JavaScript-API, über die umfassend geregelt wird, wie ein Formular validiert und übermittelt wird.

Mit der JavaScript-API steuern wir, wie unsere Besucher ein Validierungsfeedback bekommen, aber wir können den neuen Formular-APIs auch den gesamten Validierungscode überlassen. Wir können anhand der API auch bestimmen, warum genau ein bestimmtes Formularfeld nicht validiert.

Elementvalidierung erzwingen

Alle Formular- und Eingabeelemente (einschließlich `<select>` und `<textarea>`) enthalten im DOM-Knoten die Methode `checkValidity`. Wir sehen es Ihnen nach, falls Sie annehmen, dies sei die Methode, mit der man die standardmäßige Validierung und den Feedback-Prozess durch den Browser überschreibt.

Die Methode `checkValidity` gibt abhängig vom Erfolg der Validierung `true` oder `false` zurück, befiehlt aber gleichzeitig dem Browser, diese Prüfungen zu durchlaufen. Wenn Sie vorhaben, die Präsentation des Validierungsfeedbacks zu steuern, lassen Sie die Finger von dieser Methode.

Elementvalidität

Einzelne Formularfelder haben neben der Methode `checkValidity` auch ein DOM-Attribut, das ein `ValidityState`-Objekt zurückgibt. Im `validity`-Objekt gibt es mehrere `state`-Attribute, aber das einfachste und wichtigste ist das Attribut `valid`. Dieser Wert kann mit JavaScript getestet werden, um ein maßgeschneidertes Validierungsfeedbacksystem anzutreiben.

Wenn wir uns in den `submit`-Eventhandler des Formulars einklinken, können wir alle Eingabefelder manuell durchlaufen und das `validity`-Attribut prüfen. Was passiert aber, wenn das Feld gar keine Validierungsregeln hat? Sie haben Glück: Die API liefert ein `willValidate`-Attribut, mit dem wir testen können, ob wir dieses spezielle Feld validieren sollten oder nicht. Hier ein (gestelltes) Beispiel:

ANMERKUNG Zu beachten ist, dass auch `<fieldset>`-Elemente das `validity`-Attribut haben, aber leider machen diese Elemente gar nichts: Das Attribut `valid` ist immer `true`. Sie können die Methode zwar auch für `fieldsets` aufrufen, aber auch dann passiert bei den aktuellen Browsern rein gar nichts in Richtung einer eigenen Validierung.

```
var email = document.getElementById('email');
if (email.willValidate) {
  if (!email.validity.valid) {
    alert("Arrg, deine olle E-Mail ist ungültig");
  }
}
```

Wenn Sie erst einmal den Validierungsstatus des jeweiligen Formularfelds haben, können Sie per `element.validationMessage` beliebige eigene Meldungen formulieren oder die verschiedenen `validity`-Status prüfen, zu denen `valueMissing`, `typeMismatch`, `patternMismatch`, `tooLong`, `rangeUnderflow`, `rangeOverflow`, `stepMismatch` und `customError` gehören.

Validierung vermeiden

Die letzte zu beantwortende Frage wäre: Wie übermittelt man das Formular, *ohne* dass der Browser es validiert? Auch das geht. Aber warum um alles in der Welt wollen Sie das machen? Was ist, wenn Sie ein mehrstufiges Registrierungsformular haben, entweder für eine Anmeldung oder für das Übermitteln bestimmter Inhalte? Bei langen Formularen können Sie es in Abschnitte aufteilen (so wie bei eBay, wenn Sie dort einen Artikel einstellen). Sie können es Ihren Besuchern sogar erlauben, den Status ihrer übermittelten Inhalte zu speichern, auch wenn das Formular noch nicht vollständig ausgefüllt oder ungültig ist.

Es gibt für eine *Nicht*-Validierung zwei Steuerungsebenen. Das kann für die einzelne Eingabesteuerung gelten oder das gesamte Formular. Das Attribut `novalidate` kann nur für ein Formularelement gesetzt werden und verhindert die Validierung für das jeweilige Feld. Bisher gibt es für dieses Feld noch keinen praktischen, sondern nur einen theoretischen Use Case. Sie könnten sich beispielsweise in der Situation befinden, `type="email"` verwenden zu wollen, und vielleicht werden zukünftige Browser ein `type="email"` akzeptieren, mit denen die Nutzer etwas in ihrem Adressbuch nachschlagen können. Aber Sie brauchen sich nicht damit abzuquälen, den manuellen Input des Nutzers tatsächlich zu validieren, weil dessen E-Mail z.B. auch Teil eines Intranets sein könnte und dann nicht wie eine typische E-Mail-Adresse aussieht. Mit dem Attribut `novalidate` für das Formularelement bekommen Sie das hin.

Die zweite Methode `formnovalidate`, sehr praktisch und auch jetzt schon verfügbar, ist für einzelne Eingabe- und Schaltflächenelemente erlaubt (ist aber wahrscheinlich nur für `type="submit"` und `type="button"` sinnvoll). Das Attribut `formnovalidate` erlaubt es, dass das Formular übermittelt und die gesamte Validierung umgangen wird, die für die Formularfelder gesetzt wurde. Im folgenden Beispiel-Codesnippet kommt ein „Session speichern"-Button für jedes `fieldset` vor, mit dem der Nutzer seinen Fortschritt speichern kann, ohne die Validierungsregeln zu aktivieren, bis er dann schließlich den Senden-Button anklickt:

```
<form>
  <fieldset>
    <legend>Ein paar Angaben über Sie</legend>
    <div>
      <label for="email">E-Mail:</label>
      <input id="email" name="email" type="email"
      ¬required />
    </div>
    <div>
      <label for="url">Homepage:</label>
      <input id="url" type="url" name="url" />
    </div>
    <input type="submit" value="Session speichern"
    ¬ formnovalidate />
  </fieldset>
```

Sie können sich sogar in den Button „Session speichern" einklinken, um über die Eigenschaft `HTMLFieldSetElement.elements` eine JavaScript-basierte Validierung auszulösen, die nur auf den Feldern im fieldset basiert (allerdings ist das in der HTML5-Spezifikation eine neue Eigenschaft, von daher müssen Sie u.U. mit `fieldset.getElementsByTagName` arbeiten, um alle Formularfelder zu finden).

Das „Falls sich was ändert"-Event

Eine beinahe unerhebliche Änderung für das `<form>`-Element ist ein neues Event namens `oninputchange`. Tatsächlich ist dies ein sehr nützliches Event, das das Formularelement feuern lässt, sobald bei einem der Felder im Formular etwas geändert wird. Dadurch bleibt es Ihnen erspart, jeder Formularsteuerung eine Menge `onchange`-Handler einzubauen.

Wenn man beispielsweise einen Farbwähler (engl. *colour picker*) erstellen soll, der sowohl RGBA als auch HSLA ausgibt, müsste man normalerweise Event-Listener in alle Schieberegler der Werte einklinken. Aber mit dem Event `oninputchange` kann ich für das ganze Formular einen einzigen Event-Listener einklinken und anhand nur einer Methode meine RGBA- und HSLA-Werte neu berechnen.

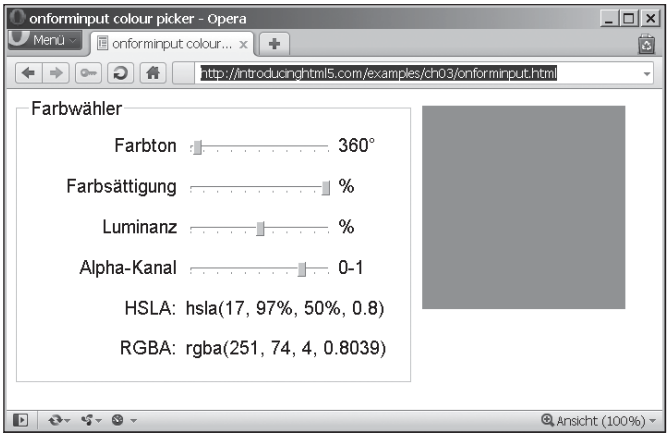
Das Ergebnis bleibt das Gleiche, egal ob ich viele oder nur einen Event-Listener anhänge. Allerdings wirkt Letzteres deutlich sauberer und besser designt, weil es keine Doppelung des Einklinkens in Events gibt.

Wenn ein Schieberegler sich ändert, wird er das RGBA und HSLA generieren und die Vorschaufarbe aktualisieren. In diesem Listing steht das erforderliche JavaScript:

```
form.onforminput = function () {
  var i = this.length, values = [], value = 0;
  while (i--, value = this[i].value) {
    if (this[i].type == 'range') {
      switch (this[i].name) {
        // Alpha-Kanal ist zwischen 0-1
        case 'alpha_channel': values.push(value /
        ¬ 100); break;
        // Farbton ist ein einfacher Wert zwischen
        ¬ 0-360
        case 'hue': values.push(value); break;
        // zum Standard gehören Farbsättigung und
        ¬ Luminanz in Prozent
        default: values.push(value + ,%');
      }
    }
  }
  hsla.value = 'hsla(' + values.reverse().join(', ')
  ¬ + ')';
  preview.style.backgroundColor = hsla.value;
  rgba.value = getComputedStyle(preview, null).
  ¬ backgroundColor;
};
```

Der endgültige Farbwähler zeigt den Wert anhand des Eingabetyps `range`, des neuen `onforminput`-Events und der neuen Output-Elemente (obwohl der Wert auch ganz einfach `.innerHTML` verwenden könnte). Das Ergebnis sehen Sie in **Abbildung 3.13**.

ABBILDUNG 3.13:
Ein HSLA-Farbwähler mit `onforminput`-Event



Zusammenfassung

Hoffentlich können Sie nun nachvollziehen, welch mächtigen Produktivitätsschub HTML5-Formulare für Entwickler bieten und was für eine einheitliche Benutzererfahrung sie für Anwender bringen. Hier stehen direkt aus dem Stand eine Riesenmenge Features zur Verfügung, die früher noch viel Programmierarbeit wie Formularvalidierung oder die Erstellung von Schiebereglersteuerungen erfordert hätten. Die Implementierung ist in Opera und den WebKit-Browsern wie Safari und Google Chrome unterschiedlich, wird aber stetig ausgeweitet und beginnt nun auch bei Firefox. Die fehlende Implementierung im IE 9 kann man mit JavaScript faken, weil die neuen Features stufenweise auch zurückgeschaltet werden können.

Wenden wir uns nun Themen zu, die mehr sexy sind.

Copyright