Tauchen Sie ein in die Webentwicklung der Zukunft

Durchstarten mit HTML5

Mark Pilgrim
Deutsche Übersetzung von Lars Schulten

O'REILLY®

Durchstarten mit HTML5

Durchstarten mit HTML5

Mark Pilgrim

Deutsche Übersetzung von Lars Schulten

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
E-Mail: kommentar@oreilly.de

1. Auflage 2011

Die Originalausgabe erschien 2010 unter dem Titel
HTML5: Up and Running bei O'Reilly Media, Inc.

Bibliografische Information Der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

Lektorat: Inken Kiupel, Köln
Korrektorat: Sibylle Feldmann, Düsseldorf
Satz: Thilo Bollmann, Reemers Publishing Services GmbH, Krefeld, www.reemers.de
Umschlaggestaltung: Karen Montgomery, Boston
Produktion: Geesche Kieckbusch, Hamburg
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; www.koeselbuch.de

ISBN 978-3-89721-571-9

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort

Einstieg

HTML5 ist die nächste Generation von HTML, die HTML 4.01, XHTML 1.0 und XHTML 1.1 ablösen wird. HTML5 bietet neue Funktionen, die für moderne Webanwendungen erforderlich sind. Außerdem standardisiert es viele Funktionen der Webplattform, die Entwickler seit Jahren nutzen, ohne dass sie je von einer Standardisierungsorganisation dokumentiert oder geprüft worden wären. (Würde es Sie überraschen, wenn ich Ihnen mitteilte, dass das `Window`-Objekt noch nie formell dokumentiert wurde? Über die ganzen neuen Funktionen hinaus ist HTML5 auch der erste Versuch, viele der »De-facto-Standards« formell zu dokumentieren, die Webbrowser bereits seit Jahren unterstützen.)

Wie seine Vorgänger soll HTML5 plattformübergreifend sein. Sie müssen nicht unter Windows, Mac OS X, Linux, Multics oder irgendeinem anderen bestimmten Betriebssystem arbeiten, um HTML5 nutzen zu können. Das Einzige, was Sie brauchen, ist ein moderner Webbrowser, der für alle wichtigen Betriebssysteme kostenlos zur Verfügung steht. Vielleicht haben Sie ja sogar bereits einen Webbrowser, der bestimmte HTML5-Funktionen unterstützt. Die jüngsten Versionen von Apple Safari, Google Chrome, Mozilla Firefox und Opera unterstützen bereits viele HTML5-Funktionen. (Ausführlichere Informationen finden Sie in den Browserkompatibilitätstabellen in diesem Buch.) Die mobilen Webbrowser, die auf iPhones, iPads und Android-Handys von Haus aus installiert sind, bieten exzellente Unterstützung für HTML5. Sogar Microsoft hat angekündigt, dass die anstehende Version 9 des Internet Explorer einige HTML5-Funktionalitäten unterstützen wird.

Dieses Buch wird sich auf acht Themen konzentrieren:

- Neue semantische Elemente wie `<header>`, `<footer>` und `<section>` (Kapitel 3).
- Canvas, eine 2-D-Zeichenfläche, die Sie mit JavaScript programmieren können (Kapitel 4).
- Videos, die Sie in Ihre Webseiten einbetten können, ohne dass zum Abspielen externe Plug-ins erforderlich sind (Kapitel 5).

- Geolocation, mit deren Hilfe Besucher beschließen können, Ihrer Webanwendung ihren physischen Aufenthaltsort mitzuteilen (Kapitel 6).
- Persistenter lokaler Speicher ohne externe Plug-ins (Kapitel 7).
- Offline-Webanwendungen, die auch funktionieren, wenn keine Netzwerkverbindung besteht (Kapitel 8).
- Verbesserungen für HTML-Webformulare (Kapitel 9).
- Mikrodaten, mit denen Sie eigene über HTML5 hinausgehende Vokabulare erstellen können, um Ihre Webseiten mit zusätzlichen semantischen Informationen auszustatten (Kapitel 10).

HTML5 wurde so entworfen, dass es so weit möglich mit älteren Browsern kompatibel ist. Seine neuen Funktionen setzen auf bestehenden Features auf und ermöglichen Ihnen so, Ersatzinhalte für ältere Browser anzubieten. Wenn Sie das im Einzelfall genauer steuern müssen, können Sie die Unterstützung einzelner HTML5-Funktionen (Kapitel 2) mit ein paar Zeilen JavaScript prüfen. Verlassen Sie sich nicht auf ein fehleranfälliges Browser-Sniffing, wenn Sie prüfen müssen, ob ein Browser HTML5 unterstützt! Prüfen Sie stattdessen, ob die erforderlichen Features vorhanden sind, indem Sie HTML5 selbst nutzen.

Die in diesem Buch verwendeten Konventionen

In diesem Buch werden die folgenden typografischen Konventionen genutzt:

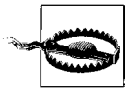
Kursiv
: Zeigt neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateinamenserweiterungen an.

`Nicht-Proportionalschrift`
: Wird für Codeauszüge verwendet und dient im Fließtext dazu, auf Codeelemente wie Variablen- und Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter zu verweisen.

`Nicht-Proportionalschrift fett`
: Zeigt Befehle oder anderen Text an, der genau so vom Benutzer eingegeben werden muss.

`Nicht-Proportionalschrift kursiv`
: Zeigt Text, der vom Benutzer durch eigene oder aus dem Kontext erschlossene Werte ersetzt werden sollte.

Dieses Symbol zeigt einen Tipp, einen Vorschlag oder eine allgemeine Anmerkung an.



Dieses Symbol zeigt eine Warnung an.

Codebeispiele

Dieses Buch soll Ihnen dabei helfen, Ihre Arbeit zu erledigen. Im Allgemeinen können Sie den Code aus diesem Buch in Ihren Programmen und Dokumentationen verwenden. Sie müssen O'Reilly nur dann um Erlaubnis fragen, wenn Sie einen signifikanten Teil des Codes verwenden wollen. Wenn Sie z.B. ein Programm schreiben, das verschiedene Codefragmente aus diesem Buch nutzt, benötigen Sie keine gesonderte Erlaubnis. Der Verkauf oder die Distribution einer CD-ROM mit Codebeispielen aus O'Reilly-Büchern hingegen erfordert eine Genehmigung. Die Beantwortung einer Frage durch Zitieren dieses Buchs und des Beispielcodes bedarf keiner Erlaubnis. Die Einbindung einer signifikanten Menge des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts aber erfordert eine Genehmigung.

Wir begrüßen es, wenn wir als Quelle genannt werden, verlangen es aber nicht. Eine Nennung enthält normalerweise den Titel, den Autor, den Verlag und die ISBN. Zum Beispiel: »*Durchstarten mit HTML5* von Mark Pilgrim. Copyright 2011 O'Reilly Verlag, 978-3-89721-571-9«.

Falls Sie nicht sicher sind, ob die Nutzung der Codebeispiele über die hier erteilte Genehmigung hinausgeht, nehmen Sie bitte unter der Adresse *permissions@oreilly.com* Kontakt mit uns auf.

Die englische Website zum Buch

Dieses Buch ist eine leicht veränderte Fassung der Onlineversion, die Sie unter *http://diveintohtml5.org/* finden und die vom Autor gepflegt wird. Da die (englisch sprachige) Site *http://diveintohtml5.org/* selbst in HTML5 gepflegt wird, enthält sie Live-Beispiele für den in diesem Buch beschriebenen Code, der in Teilen für den Druck modifiziert werden musste. Statten Sie *http://diveintohtml5.org/* einen Besuch ab, um sich diese Beispiele anzusehen. Beachten Sie, dass die Darstellung der Site browserabhängig ist, nicht gleich gut von allen Browsern unterstützt wird.

KAPITEL 1

Wie wir an diesen Punkt gelangt sind

Einstieg

Kürzlich bin ich auf eine Aussage eines Mozilla-Entwicklers zu den Spannungen, die mit der Entwicklung von Standards einher gehen, gestoßen (*http://lists.w3.org/Archives/Public/public-html/2010Jan/0107.html*):

> Implementierungen und Spezifikationen stehen in einer fragilen Wechselbeziehung. Einerseits möchte man, dass möglichst keine Implementierungen erscheinen, bevor die Spezifikation fertig ist, weil sich dann alle auf die speziellen Aspekte dieser Implementierung stützen und damit die Spezifikation einschränken. Andererseits will man die Spezifikation nicht fertigstellen, bevor es Implementierungen und Erfahrungen mit diesen Implementierungen gibt, weil man diese Rückmeldungen benötigt. Diese Spannung ist unvermeidbar. Wir müssen uns unseren Weg durch sie bahnen.

Behalten Sie dieses Zitat im Gedächtnis und lassen Sie mich erläutern, wie es zu HTML5 kam.

MIME-Typen

In diesem Buch geht es um HTML5, es geht weder um die vorangegangenen Versionen von HTML noch um eine der Versionen von XHTML. Aber wenn wir die Geschichte von HTML5 und die Motivation verstehen wollen, die dahintersteht, müssen wir uns zunächst einige technische Details vor Augen führen. Genauer gesagt: MIME-Typen.

Immer wenn Ihr Browser eine Webseite anfordert, schickt der Webserver dem eigentlichen Markup der Seite einige Header voraus. Diese Header sind normalerweise unsichtbar, können aber durch einige Werkzeuge für Webentwickler sichtbar gemacht werden, falls Sie daran Interesse haben. Wichtig sind Header, weil sie dem Browser sagen, wie er das Markup behandeln soll, das hinterhergeschickt wird. Der wichtigste Header heißt `Content-Type` und sieht folgendermaßen aus:

```
Content-Type: text/html
```

`text/html` bezeichnet man als den »Inhaltstyp« oder »MIME-Typ« der Seite. Dieser Header ist das *Einzige*, das festlegt, was eine bestimmte Ressource tatsächlich ist, und der bestimmt damit, wie sie dargestellt werden soll. Grafiken haben eigene MIME-Typen (`image/jpeg` für JPEG-Bilder, `image/png` für PNG-Bilder und so weiter). JavaScript-Dateien haben einen eigenen MIME-Typ. CSS-Stylesheets haben einen eigenen MIME-Typ. Alles hat seinen eigenen MIME-Typ. Das Web steht auf MIME-Typen.

Die Wirklichkeit ist natürlich noch komplizierter als das. Die ersten Webserver (ich spreche hier von Webservern aus 1993) verschickten keinen `Content-Type`-Header, da es diese noch nicht gab. (Sie wurden erst 1994 erfunden.) Aus Kompatibilitätsgründen, die bis ins Jahr 1993 zurückreichen, ignorieren einige beliebte Webbrowser den `Content-Type`-Header unter bestimmten Umständen immer noch. (Das nennt man »Content-Sniffing«.) Aber im Prinzip wurde Ihnen alles, was Sie sich je im Web angesehen haben – HTML-Seiten, Bilder, Skripten, Videos, PDFs, alles mit einer URL –, mit einem bestimmten MIME-Typ im `Content-Type`-Header ausgeliefert.

Packen Sie sich das ins Hinterstübchen, denn darauf werden wir zurückkommen.

Eine weitschweifige Abschweifung zur Entwicklung von Standards

Warum wohl gibt es ein `<img>`-Element? Ich vermute, das gehört nicht zu den Fragen, die man sich häufiger stellt. Aber offensichtlich muss es sich doch irgendjemand ausgedacht haben. Solche Dinge tauchen schließlich nicht aus dem Nichts auf. Alle Elemente, alle Attribute, alle Eigenschaften von HTML, die Sie je verwendet haben, müssen von irgendwem geschaffen worden sein, der entschied, wie sie funktionieren sollen, und das für alle Zeiten festschrieb. Die, die das gemacht haben, sind keine Götter und schon gar nicht unfehlbar. Es sind Menschen. Mit Sicherheit Menschen mit Grips. Aber bloß Menschen.

Genau das ist einer der wunderbaren Aspekte von Standards, die öffentlich entwickelt werden: dass man die Zeit zurückdrehen und derartige Fragen beantworten kann. Diskussionen erfolgen auf Mailinglisten, die gewöhnlich archiviert werden und so auch später für alle zugänglich sind. Ich habe mich deswegen entschlossen, etwas E-Mail-Archäologie zu betreiben und eine Antwort auf die Frage nach dem `<img>`-Element zu suchen. Dabei musste ich in Zeiten zurückkehren, bevor es diese Organisation namens World Wide Web Consortium (W3C) gab. Ich kehrte in die ersten Tage des Webs zurück, in die Zeiten, da man die Anzahl der Webserver noch mit den Fingern zweier Hände und vielleicht noch einigen Zehen hätte abzählen können.

Am 25. Februar 1993 schrieb Marc Andreessen:[1]

> Ich möchte ein neues, optionales HTML-Tag vorschlagen:
>
> IMG
>
> Es hat das erforderliche Attribut `SRC="url"`.
>
> Es gibt eine Bitmap- oder Pixmap-Datei an, die der Browser über das Netzwerk abrufen und als Bild interpretieren soll, das an der Stelle des Tags in den Text eingefügt wird.
>
> Ein Beispiel:
>
> `<IMG SRC="file://foobar.com/foo/bar/blargh.xbm">`
>
> (Es gibt kein schließendes Tag; es ist ein eigenständiges Tag.)
>
> Das Tag kann wie alles andere in einen Anker eingebettet werden. Wenn das erfolgt, wird es zu einem Symbol, das ebenso aktiviert werden kann wie ein gewöhnlicher Textanker.
>
> Browser sollten etwas Spielraum dabei haben, welche Bildformate sie unterstützen. Xbm und Xpm sind beispielsweise gute Kandidaten. Kann ein Browser mit einem angegebenen Format nichts anfangen, kann er verfahren, wie ihm beliebt (X Mosaic wird als Platzhalter eine Standard-Bitmap anzeigen).
>
> Das ist eine für X Mosaic erforderliche Funktion; wir haben eine funktionsfähige Implementierung und werden diese zumindest intern nutzen. Natürlich bin ich allen Vorschlägen dazu offen, wie das innerhalb von HTML ablaufen soll. Wenn jemand eine Idee hat, die besser als das ist, was ich hier vorschlage, teilt sie mir mit. Mir ist klar, dass das in Bezug auf das Bildformat schwammig ist, aber ich sehe keine Alternative dazu, den Browser »nach eigenem Gusto« verfahren zu lassen und darauf zu warten, dass eine bessere Lösung vorhanden ist (vielleicht irgendwann MIME).

Dieses Zitat erfordert einige Erläuterungen. Xbm und Xpm waren beliebte Grafikformate auf Unix-Systemen.

Mosaic war einer der ersten Webbrowser. (»X Mosaic« war die Version, die auf Unix-Systemen lief.) Als er diese Nachricht Anfang 1993 schrieb, hatte Marc weder das Unternehmen gegründet, Mosaic Communications Corporation, das ihn bekannt gemacht hat, noch die Arbeit am Markenzeichenprodukt des Unternehmens, »Mosaic Netscape«, begonnen. (Möglicherweise sind Ihnen die späteren Namen von Unternehmen und Browser vertrauter: »Netscape Corporation« und »Netscape Navigator«.)

»Vielleicht irgendwann MIME« ist ein Verweis auf die sogenannte Inhaltsaushandlung (*Content Negotiation*), eine HTTP-Funktion, bei der ein Client (wie ein Webbrowser) einem Server (wie einem Webserver) sagt, welche Arten von Ressourcen (wie `image/jpeg`) er unterstützt, damit der Server etwas in einem Format liefern kann, das der Client versteht. »Das ursprüngliche HTML in der 1991 definierten Form« (die einzige Form,

1 *http://1997.webhistory.org/www.lists/www-talk.1993q1/0182.html*. Den englischen Thread, der auf den folgenden Seiten beschrieben wird und der sinngemäß ins Deutsche übertragen wurde, können Sie über die »Next message«- und »Previous message«-Links verfolgen.

die im Februar 1993 implementiert war) bot Clients keine Möglichkeit, Servern mitzuteilen, welche Grafikformate sie unterstützten. Daher das Dilemma, vor dem Marc stand.

Einige Stunden später antwortete Tony Johnson:

> In Midas 2.0 (den wir hier am SLAC nutzen und der in Kürze veröffentlicht werden soll) haben wir etwas Vergleichbares. Allerdings verwenden wir andere Namen und ein zusätzliches Argument: `NAME="name"`. Die Funktionalität deckt sich fast vollständig mit der des von Ihnen vorgeschlagenen `IMG`-Tags, z.B.:
>
> ```
> <ICON name="NoEntry" href="http://note/foo/bar/NoEntry.xbm">
> ```
>
> Der Gedanke hinter dem Parameter `name` war, dass es Browsern möglich sein sollte, einen Satz »eingebauter« Bilder vorzuhalten. Entspricht `name` einem der »eingebauten« Bilder, soll der Browser dieses nutzen und sich nicht daran machen, das Bild abzurufen. Der Name würde auch als Hinweis für »Line Mode Browser« dienen, der ihnen anzeigt, was anstelle des Bilds eingesetzt wird.
>
> Die Parameter- und Tag-Namen sind mir ziemlich wurscht, aber ich hielte es für vernünftig, wenn wir die gleichen Dinge nutzen würden. Abkürzungen finde ich persönlich nicht so gut, warum also nicht `IMAGE=` und `SOURCE=`. `ICON` sagt mir etwas mehr zu, da das `IMAGE` klein gehalten werden soll. Aber vielleicht ist das Wort `ICON` auch zu überladen?

Midas war ein anderer früher Webbrowser zur Zeit von X Mosaic. Er war plattformübergreifend und lief unter Unix und VMS. »SLAC« steht für das Stanford Linear Accelerator Center, heutzutage das SLAC National Accelerator Laboratory, das den ersten Webserver in den Vereinigten Staaten beherbergte (genauer, den ersten Webserver außerhalb Europas). Als Tony diese Nachricht schrieb, war das SLAC bereits ein alteingesessener WWW-Bewohner, der auf seinem Webserver seit unvorstellbaren 441 Tagen fünf Webseiten betrieb.

Tony fuhr folgendermaßen fort:

> Da wir uns gerade mit Tags befassen: Ich habe hier noch ein anderes ähnliches Tag, das ich gerne in Midas 2.0 unterstützen würde. Im Prinzip ist das:
>
> ```
> <INCLUDE HREF="...">
> ```
>
> Die Absicht hier ist, dass das zweite Dokument an der Stelle in das erste Dokument eingeführt werden soll, an der das Tag stand. Im Grunde könnte das referenzierte Objekt alles sein. Der eigentliche Zweck aber war, Bilder (hier jedoch beliebiger Größe) in Dokumente einzubetten. Auch hier ist beabsichtigt, dass das Format des eingeschlossenen Dokuments, wenn HTTP2 verfügbar ist, separat ausgehandelt wird.

»HTTP2« verweist auf das in 1992 definierte Basic HTTP. Zu jenem Zeitpunkt, Anfang 1993, war es größtenteils noch nicht implementiert. Der als »HTTP2« bekannte Entwurf reifte und wurde später als »HTTP 1.0« standardisiert. HTTP 1.0 umfasste Request-Header zur Inhaltsaushandlung, aka »vielleicht irgendwann MIME.«

Tony fuhr fort:

> Eine Alternative, die ich erwog, war:
>
> ```
> <A HREF="..." INCLUDE>Siehe Foto</A>
> ```
>
> Eigentlich sagt es mir nicht sonderlich zu, die Funktion des <A>-Tags auf diese Weise zu erweitern, aber hier hat es den Zweck, die Kompatibilität mit Browsern zu wahren, die den INCLUDE-Parameter nicht unterstützen. Der Gedanke ist, dass Browser, die INCLUDE verstehen, den Ankertext (hier »Siehe Foto«) durch das eingeschlossene Dokument (Bild) ersetzen, während ältere oder dümmere Browser das INCLUDE-Tag vollständig ignorieren.

Dieser Vorschlag wurde nie implementiert, aber der Gedanke, einen Text anzubieten, wenn ein Bild fehlt, ist eine wichtige Bedienungshilfe, die in Marcs ursprünglichem <IMG>-Vorschlag fehlte. Viele Jahre später wurde diese Funktion als das <img alt>-Attribut umgesetzt, das von Netscape dann auch gleich missbraucht wurde, indem es als Tooltip behandelt wurde.

Wenige Stunden, nachdem Tony seine Nachricht geschrieben hatte, antwortete Tim Berners-Lee:

> Ich hätte gedacht, dass Abbildungen als
>
> ```
> <a name=fig1 href="fghjkdfghj" REL="EMBED, PRESENT">Figure </a>
> ```
>
> repräsentiert würden. Die Werte für REL bedeuten dabei:
>
> ```
> EMBED    Folgendes bei der Darstellung hier einbetten
> PRESENT  Das anzeigen, wenn das Quelldokument angezeigt wird
> ```
>
> Wichtig ist, dass man mehrere Kombinationen davon haben kann. Unterstützt der Browser keine davon, führt das trotzdem nicht zu Problemen.
>
> [Ich] sehe, dass es zu verschachtelten Ankern führt, wenn es als Methode für wählbare Symbole verwendet würde. Hmm. Aber ein spezielles Tag hätte ich gerne vermieden.

Dieser Vorschlag wurde nie implementiert, aber das rel-Attribut gibt es immer noch (siehe dazu den Abschnitt »Link-Relationen« auf Seite 38).

Jim Davis ergänzte:

> Es wäre gut, wenn es eine Möglichkeit gäbe, den Inhaltstyp anzugeben, z.B.
>
> ```
> <IMG HREF="http://nsa.gov/pub/sounds/gorby.au" CONTENT-TYPE=audio/basic>
> ```
>
> Aber ich bin absolut bereit, mit der Anforderung zu leben, dass ich den Inhaltstyp über die Dateierweiterung angebe.

Dieser Vorschlag wurde ebenfalls nie implementiert, aber Netscape ergänzte später Unterstützung für die Einbettung beliebiger Objektmedien mit dem Element <embed>.

Jay C. Weber stellte die Frage:

> Bilder stehen zwar ganz oben in meiner Liste wünschenswerter Medientypen in WWW-Browsern, aber ich glaube dennoch nicht, dass wir nach und nach Strukturen für spezifische

> Medientypen schaffen sollten. Was bitte ist aus dem Enthusiasmus für den MIME-Typ-Mechanismus geworden?

Marc Andreessen antwortete:

> Das soll kein Ersatz für die kommende Verwendung von MIME als Standard sein. Es bietet eine notwendige und einfache Implementierung von Funktionalität, die unabhängig von MIME erforderlich ist.

Jay C. Weber antwortete:

> Klammern wir MIME vorübergehend aus, wenn es das Problem verdeckt. Mein Einwand war, dass wir hier erörtern, »wie wir eingebettete Bilder unterstützen«, statt uns zu fragen, »wie wir eingebettete Objekte unterschiedlicher Medientypen unterstützen«.
>
> Andernfalls schlägt nächste Woche einer vor, »ein neues Tag für Audio einzuführen: `<AUD SRC="file://foobar.com/foo/bar/blargh.snd">`«.
>
> Es sollte doch nicht so viel Aufwand sein, etwas zu wählen, das allgemeiner verwendbar ist.

Im Rückblick wissen wir heute, dass Jays Bedenken wohlbegründet waren. Es dauerte etwas länger als eine Woche, aber jetzt ergänzt HTML5 schließlich doch `<video>`- und `<audio>`- Elemente.

In Reaktion auf Jays ursprüngliche Meldung schrieb Dave Raggett:

> Wohl wahr! Ich möchte eine ganze Spanne möglicher Bild-/Grafiktypen berücksichtigt sehen sowie die Möglichkeit einer Formataushandlung. Tims Bemerkung zur Unterstützung klickbarer Bereiche in Bildern ist ebenfalls wichtig.

Später im Jahr 1993 schlug Dave HTML+ als Weiterentwicklung des HTML-Standards vor. Der Vorschlag wurde nie implementiert und von HTML 2.0 überholt. HTML 2.0 war eine nachgeholte Spezifikation, d.h., sie formalisierte Funktionen, die bereits verbreitet verwendet wurden. »Diese Spezifikation sammelt, klärt und formalisiert eine Gruppe von Funktionen, die ungefähr den Fähigkeiten des HTML entsprechen, das im Juni 1994 verbreitet eingesetzt wurde.«

Später formulierte Dave HTML 3.0 auf Basis seines früheren HTML+-Entwurfs. Von der Referenzimplementierung des W3C abgesehen, Arena, gab es keine einzige HTML-3.0-Implementierung. Es wurde von HTML 3.2 überholt, das ebenfalls eine »nachgeschobene Spezifikation« war: »HTML 3.2 ergänzt verbreitete Funktionen wie Tabellen, Applets und Textfluss um Bilder, bewahrt aber vollständige Rückwärtskompatibilität mit dem bestehenden Standard HTML 2.0.«

Dave hat später an HTML 4.0 mitgeschrieben, HTML Tidy entwickelt und anschließend an XHTML, XForms, MathML und anderen modernen W3C-Spezifikationen mitgewirkt.

Kehren wir ins Jahr 1993 zurück. Marc antwortete auf Dave:

> Vielleicht sollten wir besser über eine allgemeine prozedurale Sprache zur Grafikbeschreibung nachdenken, die wir in beliebige Hyperlinks einbetten, die an Symbole, Bilder oder Text oder irgendetwas anderes gebunden sind. Hat sich schon mal jemand Intermedias Fähigkeiten in dieser Hinsicht angesehen?

Intermedia war ein Hypertext-Projekt der Brown-Universität. Es wurde von 1985 bis 1991 entwickelt und lief auf A/UX, einem Unix-ähnlichen Betriebssystem für frühe Macintosh-Computer.

Der Gedanke einer »allgemeinen prozeduralen Sprache zur Grafikbeschreibung« wurde später aufgenommen. Moderne Browser unterstützen sowohl SVG (deklaratives Markup mit eingebettetem Scripting) als auch `<canvas>` (eine prozedurale Direct-Mode-Grafik-API) – obwohl Zweiteres als proprietäre Erweiterung begann, bevor es »nachträglich« durch die WHAT Working Group »spezifiziert« wurde.

Bill Janssen antwortete:

> Weitere betrachtenswerte Systeme, die dieses (recht wertvolle) Konzept kennen, sind Andrew und Slate. Andrew basiert auf _Insets_, die jeweils einen relevanten Typ wie Text, Bitmap, Zeichnung, Animation, Nachricht, Tabelle usw. haben. Das Konzept rekursiver Einbettung ist vorhanden, sodass jede Art von Inset in eine andere eingebettet werden kann, die Einbettung unterstützt. Beispielsweise kann ein Inset an jedem Punkt eines Text-Widgets oder in einen rechteckigen Bereich in ein Zeichnungs-Widget oder in eine beliebige Zelle einer Tabelle eingefügt werden.

»Andrew« bezieht sich auf das Andrew User Interface System, das zu jener Zeit allerdings einfach unter dem Namen Andrew Project bekannt war.

Inzwischen hatte Thomas Fine eine andere Idee:

> Hier ist meine Meinung. Das beste Verfahren zur Integration von Bildern im WWW ist MIME. Ich bin sicher, dass Postscript in MIME bereits ein unterstützter Subtyp ist, und es kommt sehr gut mit der Kombination aus Text und Grafik klar.
>
> Aber man kann es nicht anklicken, sagt ihr? Stimmt. Ich vermute, dass es in Display Postscript dafür bereits eine Lösung gibt. Und selbst wenn das nicht der Fall ist, sollte eine entsprechende Ergänzung des Postscript-Standards trivial sein. Man definiert einen Anker, der die URL angibt, und nutzt den aktuellen Pfad als einen geschlossenen Bereich für den Button. Da Postscript sich in Bezug auf Pfade so gut verhält, macht das beliebige Button-Formen zu einem Kinderspiel.

Display PostScript war eine Technologie zur Darstellung auf dem Bildschirm, die gemeinsam von Adobe und NeXT entwickelt wurde.

Dieser Vorschlag wurde nie implementiert, aber der Gedanke, dass man HTML am besten damit repariert, dass man es durch etwas ganz anderes ersetzt, wird auch heute gelegentlich noch geäußert.

Am 2. März 1993 kommentierte Tim Berners-Lee:

> HTTP2 gestattet Dokumenten, beliebige Typen zu enthalten, von denen der Client sagt, dass er mit ihnen umgehen kann, nicht nur registrierte MIME-Typen. Man kann also experimentieren. Ja, ich denke, es gibt Argumente für Postscript mit Hypertext. Ich weiß nicht, ob Display Postscript genug bietet. Ich weiß, dass Adobe versucht, das eigene, Postscript-basierte »PDF« zu etablieren, das Links bieten soll und von ihren proprietären Readern gelesen werden können soll.

> Ich dachte, dass eine allgemeine Overlay-Sprache für Anker (HyTime basiert?) den Standards für Hypertext einerseits und Grafik?Video andererseits eine voneinander unabhängige Entwicklung ermöglichen würde. Das wäre für beide von Vorteil.
>
> Machen wir das `IMG`-Tag zu `INCLUDE` und lassen wir es auf beliebige Dokumenttypen verweisen. Oder `EMBED`, sollte `INCLUDE` zu sehr nach einem C++-Include klingen und die Erwartung wecken, dass man damit SGML-Code anbietet, der inline geparst wird – was so ja nicht beabsichtigt ist.

HyTime war ein frühes, SGML-basiertes Hypertext-Dokumentensystem. Es spielte in den frühen Diskussionen über HTML und später XML eine wichtige Rolle.

Auch Tims Vorschlag für ein `<INCLUDE>`-Tag wurde nie implementiert, aber einen Widerhall können Sie heute noch in den `<object>`-, `<embed>`- und `<iframe>`-Elementen sehen.

Schließlich kehrte am 12. März 1993 Marc Andreessen in den Thread zurück:

> Wieder zurück im Thread zu eingebetteten Bildern – ich stehe kurz davor, Mosaic v0.10 zu veröffentlichen, der Inline-GIF- und -XBM-Bilder/-Bitmaps unterstützen wird, wie ich bereits erwähnte. [...]
>
> Zurzeit sind wir nicht bereit, `INCLUDE/EMBED` zu unterstützen. [...] Wir werden wahrscheinlich also bei `<IMG SRC="url">` bleiben (nicht `ICON`, da man nicht alle eingebundenen Bilder vernünftigerweise als Icon bezeichnen kann). Aktuell erhalten eingebundene Bilder keinen expliziten Inhaltstyp. Wir planen aber, das in Zukunft zu unterstützen (gemeinsam mit einer allgemeinen Adaption von MIME). Tatsächlich ermitteln die von uns zurzeit verwendeten Routinen zum Lesen der Bilder das Format beim Lesen. Auch die Dateinamenserweiterung spielt also keine Rolle.

Eine kontinuierliche Linie

Ich finde fast alle Aspekte dieser etwa 17 Jahre alten Unterhaltung außerordentlich faszinierend, die zur Schaffung eines HTML-Elements führte, das es auf beinahe jeder Webseite gibt, die je veröffentlicht wurde. Bedenken Sie:

- HTTP gibt es immer noch. Es reifte erfolgreich von 0.9 zu 1.0 und später 1.1 und entwickelt sich auch heute noch (*http://www.ietf.org/dyn/wg/charter/httpbis-charter.html*).
- Und HTML gibt es weiterhin. Dieses rudimentäre Datenformat (das nicht einmal eingebettete Bilder unterstützte!) entwickelte sich in kontinuierlicher Linie zu 2.0, 3.2 und 4.0. Eine verwickelte, verknotete, verworrene Linie – sicher. Es gab Unmengen »toter Zweige« im Entwicklungsbaum, Punkte, an denen standardbewusste Menschen die eigenen Prinzipien vergaßen (und die Autoren und Implementierer hinter sich ließen), aber selbst heute, im Jahr 2010, werden Webseiten aus dem Jahr 1990 von modernen Browsern noch dargestellt. Ich habe gerade eine im Browser meines brandaktuellen Android-Handys geladen, und ich wurde nicht einmal aufgefordert, doch bitte zu warten, »während ein veraltetes Format importiert wird ...«.
- HTML war immer eine Unterhaltung zwischen Browserherstellern, Standardfetischisten und anderen Leuten, die sich einfach einklinkten und über spitze Klammern

sprechen wollten. Die meisten der erfolgreichen Versionen von HTML waren »nachgeschobene Spezifikationen«, die versuchten, das Leben einzuholen und dabei gleichzeitig in die richtige Richtung zu lenken. Alle, die Ihnen sagen, dass HTML »rein« gehalten werden sollte (vermutlich indem man Browserhersteller oder Webautoren oder gleich beides ignoriert), haben schlicht keine Ahnung. HTML war nie rein, und alle Versuche, es zu reinigen, endeten in spektakulären Misserfolgen, Misserfolgen, die nur mit denen der Versuche mithalten können, HTML vollständig zu ersetzen.

- Keinen der Browser, die 1993 in Gebrauch waren, gibt es heute noch in erkennbarer Form. Der Netscape Navigator wurde 1998 aufgegeben und von Grund auf neugeschrieben, um die Mozilla Suite zu schaffen, aus der dann Firefox hervorging. Der Internet Explorer fand seine bescheidenen »Anfang« in »Microsoft Plus! für Windows 95«, wo er mit einigen Desktop-Themes und einem Flipper-Spiel verbunden war. Aber natürlich lassen sich auch die Spuren dieses Browsers weiter zurückverfolgen.
- Einige der 1993 genutzten Betriebssysteme gibt es immer noch. Doch keins davon ist im modernen Web relevant. Heutzutage nutzen die meisten Menschen, die sich im Web herumtreiben, einen PC, auf dem Windows 2000 oder jünger läuft, einen Mac mit Mac OS X, einen PC mit irgendeiner Linux-Variante oder ein Handgerät wie das iPhone. 1993 stand Windows bei Version 3.1 (und konkurrierte mit OS/2), Macs liefen unter System 7, und Linux wurde über das Usenet verteilt. (Möchten Sie sich einen Spaß erlauben? Suchen Sie sich einen graubärtigen Technologiefreak, flüstern Sie ihm »Trumpet Winsock« oder »MacPPP« ins Ohr und schauen Sie sich die Reaktion an.)
- Einige der genannten Personen sind immer noch da und immer noch an dem beteiligt, was wir heute einfach »Webstandards« nennen. Und das nach fast 20 Jahren. Einige von ihnen waren bereits an den Vorläufern von HTML beteiligt, die bis in die 1980ger-Jahre und noch weiter zurückreichen.
- Da wir gerade von Vorläufern sprechen ... die irgendwann aufkommende Popularität von HTML und Internet lässt einen leicht die zeitgenössischen Systeme und Formate vergessen, die ihre Gestaltung beeinflussten. Hatten Sie, bevor Sie dieses Kapitel lasen, je von Andrew, Intermedia oder HyTime gehört? Und HyTime war nicht irgendein mickriges akademisches Projekt: Es war ein ISO-Standard, der für die militärische Verwendung zugelassen war. Das war Big Business. Und unter *http://www.sgmlsource.com/history/hthist.htm* können Sie die Geschichte selbst nachlesen.

Aber nichts davon beantwortet unsere ursprüngliche Frage: Warum gibt es ein `<img>`-Element? Warum nicht ein `<icon>`-Element? Oder ein `<include>`-Element? Warum nicht ein Hyperlink mit einem `include`-Attribut oder eine Kombination aus `rel`-Werten? Warum ein `<img>`-Element? Ganz einfach, weil Marc Andreessen eins auslieferte und veröffentlichter Code gewinnt.

Das soll nicht heißen, dass veröffentlichter Code immer gewinnt; schließlich veröffentlichten auch Andrew, Intermedia und HyTime Code. Code ist für den Erfolg notwendig, aber nicht ausreichend. Und ich möchte keinesfalls sagen, dass es immer zu den besten

Ergebnisse führt, wenn der Code vor dem Standard veröffentlicht wird. Marcs `<img>`-Element erforderte kein gemeinsames Grafikformat; es definierte nicht, wie Text es umfloss, es bot keine Unterstützung für Alternativtexte oder Ersatzinhalte für ältere Browser. Und auch 17 Jahre später schlagen wir uns immer noch mit Content-Sniffing herum, und es ist immer noch eine Quelle wahnwitziger Sicherheitslücken. Das können Sie über die großen Browserkriege zurückverfolgen bis zu jenem 25. Februar 1993, an dem Marc Andreessen nebenbei »vielleicht irgendwann MIME« anmerkte und seinen Code dann doch einfach so auslieferte.

Die HTML-Entwicklung von 1997 bis 2004

Im Dezember 1997 veröffentlichte das World Wide Web Consortium (W3C) HTML 4.0 und schloss sofort darauf die HTML Working Group. Weniger als zwei Monate später veröffentlichte eine andere W3C Working Group XML 1.0. Und wieder weniger als drei Monate später hielt das W3C einen Workshop namens »Shaping the Future of HTML«, um folgende Frage zu beantworten: »Hat das W3C die Weiterentwicklung von HTML aufgegeben?« Folgendermaßen lautete die Antwort:

> Man kam in Diskussionen überein, dass eine weitere Erweiterung von HTML 4.0 nicht einfach wäre und Gleiches für die Umwandlung von 4.0 in eine XML-Anwendung gelte. Es wurde vorgeschlagen, dass man sich aus diesen Beschränkungen am besten löst, indem man mit der nächsten Generation von HTML einen ganz neuen Anfang auf Basis eines Satzes von XML-Tag-Mengen macht.

Das W3C setzte die HTML Working Group erneut daran, diesen »Satz von XML-Tag-Mengen zu gestalten«. Der erste Schritt ihrer Mitglieder im Dezember 1998 war, eine vorläufige Spezifikation zu skizzieren, die HTML als XML neu formulierte, ohne neue Elemente oder Attribute zu ergänzen. Diese Spezifikation wurde später als »XHTML 1.0« bekannt. Es definierte einen neuen MIME-Typ für XHTML-Dokumente, `application/xhtml+xml`. Um die Migration vorhandener HTML 4-Seiten zu vereinfachen, schloss es allerdings auch Anhang C ein, der »Gestaltungsrichtlinien für Autoren zusammenfasst, die ihre XHTML-Dokumente von den bestehenden HTML-User-Agents darstellen lassen möchten«. Anhang C sagte, dass es zulässig sei, sogenannte »XHTML-Seiten« zu schreiben, diese aber immer noch mit dem MIME-Typ `text/html` auszuliefern.

Das nächste Ziel waren Webformulare. Im August 1999 veröffentlichte eben diese HTML Working Group einen ersten Entwurf für XHTML Extended Forms. Ihre Mitglieder formulierten in den allerersten Sätzen dieses Entwurfsdokuments folgende Erwartungen (*http://www.w3.org/TR/1999/WD-xhtml-forms-req-19990830#intro*):

> Nach sorgfältigen Erwägungen hat die HTML Working Group beschlossen, dass die Ziele für die nächste Generation von Formularen nicht mit der Bewahrung der Rückwärtskompatibilität mit Browsern vereinbar ist, die für frühere Versionen von HTML entworfen wurden. Wir haben uns zum Ziel gesetzt, ein sauberes neues Modell für Formulare (»XHTML Extended Forms«) auf Basis eines Satzes wohldefinierter Anforderungen zu entwerfen. Die in diesem Dokument beschriebenen Anforderungen basieren auf Erfahrungen mit einem sehr breiten Spektrum von Formularanwendungen.

Einige Monate später wurde »XHTML Extended Forms« in »XForms« umgetauft und in eine eigene Arbeitsgruppe ausgelagert. Diese Arbeitsgruppe arbeitete parallel zur HTML Working Group und veröffentlichte im Oktober 2003 schließlich die erste Version von XForms 1.0.

Inzwischen, der Umstieg auf XML war vollzogen, richteten die Mitglieder der HTML Working Group ihr Augenmerk darauf, »die nächste Generation von HTML« zu schaffen. Im Mai 2001 veröffentlichten sie die erste Version von XHTML 1.1, die XHTML 1.0 nur um einige kleinere Funktionen ergänzte, aber das Schlupfloch »Anhang C« entfernte. Ab Version 1.1 sollten alle XHTML-Dokumente mit dem MIME-Typ `application/xhtml+xml` ausgeliefert werden.

Alles, was Sie über XHTML wissen, ist falsch

Warum sind MIME-Typen wichtig? Warum komme ich immer wieder auf sie zurück? In zwei Worten: drakonische Fehlerbehandlung. Bei HTML waren Browser stets »nachsichtig«. Vergessen Sie beim Erstellen einer HTML-Seite das `<title>`-Tag, wird sie vom Browser trotzdem angezeigt, obwohl das `<title>`-Element in allen Versionen von HTML erforderlich war. Bestimmte Tags sind in anderen Tags nicht erlaubt, aber wenn Sie eine Seite erstellen, die das ignoriert, kommen Browser damit (irgendwie) klar und zeigen die Seite an, ohne einen Fehler zu melden.

Wie zu erwarten war, führte der Umstand, dass »fehlerhaftes« HTML-Markup in Webbrowsern trotzdem funktioniert, dazu, das Autoren fehlerhafte HTML-Seiten erstellten. Eine Menge fehlerhafter Seiten. Gemäß einiger Schätzungen enthalten über 99 Prozent aller HTML-Seiten, die sich heute im Web befinden, mindestens einen Fehler. Aber weil diese Fehler nicht dazu führen, dass Browser eine sichtbare Fehlermeldung anzeigen, werden sie nie repariert.

Das W3C betrachtete das als ein grundlegendes Problem des Webs und begann damit, es zu beheben. XML, das 1997 veröffentlicht wurde, brach mit der Tradition nachsichtiger Clients und verlangte, dass alle Programme, die mit XML arbeiteten, Fehler in der sogenannten »Wohlgeformtheit« als fatale Fehler betrachten müssen. Die Konzeption, die Auswertung beim ersten Fehler abzubrechen, wurde unter dem Namen »drakonische Fehlerbehandlung« bekannt. Namenspate war der griechische Gesetzgeber Drakon, der die Todesstrafe bereits für recht bescheidene Verletzungen seiner Gesetze vorschrieb. Als das W3C HTML dann als XML-Vokabular neu formulierte, verlangten die Verantwortlichen, dass alle Dokumente, die mit dem neuen MIME-Typ `application/xhtml+xml` ausgeliefert werden, der drakonischen Fehlerbehandlung unterliegen. Enthielte eine XHTML-Seite auch nur einen einzigen Fehler, hätte ein Webbrowser keine andere Wahl, als die Verarbeitung abzubrechen und dem Benutzer eine Fehlermeldung anzuzeigen.

Nicht alle waren von dieser Vorstellung begeistert. Bei einer geschätzten Fehlerquote von 99 Prozent bei den bestehenden Webseiten führten die stets gegebene Gefahr, dass dem Endnutzer Fehlermeldungen präsentiert werden, und der Mangel an neuen Funktionalitäten in XHTML 1.0 und 1.1, die den Aufwand gerechtfertigt hätten, dazu, dass Web-

autoren `application/xhtml+xml` schlicht ignorierten. Was aber nicht heißt, dass sie XHTML vollständig ignoriert hätten. Nein, keinesfalls. Anhang C der Spezifikation für XHTML 1.0 gab den Webautoren der Welt ein Schlupfloch: »Nutzt etwas, dass nach XHTML-Syntax aussieht, aber liefert es weiterhin mit dem MIME-Typ `text/html`.« Und genau das machten Tausende von Webentwicklern: Sie »rüsteten« auf XHTML-Syntax auf, lieferten es aber weiterhin mit dem MIME-Typ `text/html`.

Bis zum heutigen Tag behaupten viele Webseiten, XHTML zu sein – sie beginnen in der ersten Zeile mit dem Doctype XHTML, nutzen Tag-Namen in Kleinbuchstaben, umgeben Attributwerte mit Anführungszeichen und schließen leere Elemente wie `<br />` und `<hr />` mit einem Schrägstrich. Doch nur ein winziger Bruchteil dieser Seiten wird mit dem MIME-Typ `application/xhtml+xml` ausgeliefert, der die drakonische Fehlerbehandlung von XML in Gang setzen würde. Jede Seite, die mit dem MIME-Typ `text/html` ausgeliefert wird, wird unabhängig von Doctype, Syntax oder Programmierweise vom »nachsichtigen« HTML-Parser geparst, der stillschweigend alle Markup-Fehler ignoriert und weder Endbenutzer noch sonst irgendwen alarmiert, wenn die Seite technisch fehlerhaft ist.

XHTML 1.0 enthielt dieses Schlupfloch, aber XHTML 1.1 schloss es, und das nie fertiggestellte XHTML 2.0 setzte die Tradition fort, eine drakonische Fehlerbehandlung zu verlangen. Das ist der Grund dafür, dass es Milliarden von Webseiten gibt, die behaupten, XHTML 1.0 zu sein, und nur eine Handvoll, die von sich sagen, sie seien XHTML 1.1 (oder XHTML 2.0). Und wie sieht das bei Ihnen aus? Nutzen Sie tatsächlich XHTML? Überprüfen Sie Ihren MIME-Typ. (Und sollten Sie nicht wissen, welchen MIME-Typ Sie nutzen, kann ich Ihnen quasi garantieren, dass Sie immer noch `text/html` verwenden.) Liefern Sie Ihre Seiten nicht mit dem MIME-Typ `application/xhtml+xml`, ist Ihr »sogenanntes XHTML« nur dem Namen nach XML.

Eine konkurrierende Vision

Im Juni 2004 hielt das W3C den Workshop »Web Applications and Compound Documents«. Bei diesem Workshop waren Vertreter mehrerer Browserhersteller, Webentwicklungsunternehmen und andere W3C-Mitglieder anwesend. Eine Gruppe der betroffenen Teilnehmer, unter anderem die Mozilla Foundation und Opera Software, präsentierte ihre konkurrierenden Vorstellungemn von der Zukunft des Internets: eine Evolution des existierenden HTML 4-Standards, die neuen Funktionalitäten für moderne Webanwendungen einschließt (*http://www.w3.org/2004/04/webapps-cdf-ws/papers/opera.html*):

> Die folgenden sieben Grundsätze repräsentieren das, was unserer Ansicht nach die entscheidendsten Anforderungen für diese Arbeit sind:
>
> *Rückwärtskompatibilität, eindeutiger Migrationsweg*
>
> Technologien für Webanwendungen sollten auf Technologien basieren, mit denen Autoren vertraut sind, einschließlich HTML, CSS, DOM und JavaScript.
>
> Elementare Funktionalitäten für Webanwendungen sollten mit Behaviorn, Scripting und Stylesheets heute im IE6 implementierbar sein, damit Autoren einen klaren Migrationsweg

haben. Jede Lösung, die mit dem aktuell dominanten User-Agent ohne binäre Plug-ins nicht realisierbar ist, hat kaum Aussichten auf Erfolg.

Wohldefinierte Fehlerbehandlung
: Die Fehlerbehandlung in Webanwendungen muss so detailliert definiert sein, dass User-Agents (UAs) keine eigenen Mechanismen zur Fehlerbehandlung erfinden oder die anderer User-Agents nachbauen müssen.

Benutzer sollten Autorenfehlern nicht ausgesetzt werden
: Spezifikationen müssen ein festgelegtes Verhalten zur Wiederherstellung für jede Fehlermöglichkeit bieten. Die Fehlerbehandlung sollte größtenteils als nachsichtige Wiederherstellung definiert sein (wie bei CSS) anstatt als offensichtliches und fatales Scheitern (wie bei XML).

Praktische Relevanz
: Jede Funktion in einer Spezifikation für Webanwendungen muss durch einen tatsächlichen Anwendungsfall gerechtfertigt sein. Das Gegenteil muss nicht unbedingt zutreffen: Nicht jeder Anwendungsfall macht notwendigerweise eine neue Funktion erforderlich.

  Anwendungsfälle sollten vorzugsweise auf existierenden Sites basieren, bei denen die Autoren bislang eine schlechte Lösung wählten, um die Beschränkung zu umgehen.

Scripting gibt es und das soll auch so bleiben
: Aber es sollte vermieden werden, wenn praktischeres deklaratives Markup verwendet werden kann. Scripting sollte geräte- und präsentationsneutral sein, es sei denn, es ist auf gerätespezifische Weise eingeschränkt (z.B. wenn es in XBL eingeschlossen ist).

Gerätespezifische Profile sollten vermieden werden
: Autoren sollten sich darauf verlassen können, dass von den Desktop- und Mobilversionen eines UA die gleichen Funktionalitäten implementiert werden.

Offener Prozess
: Das Web hat von der Entwicklung in einer offenen Umgebung profitiert. Webanwendungen werden das Herz des Webs sein, und ihre Entwicklung sollte offen erfolgen. Mailinglisten, Archive und Spezifikationsentwürfe sollten permanent für die Öffentlichkeit einsehbar sein.

In einer unverbindlichen Abstimmung wurden die Teilnehmer des Workshops gefragt, ob »das W3C deklarative Erweiterungen zu HTML und CSS und imperative Erweiterungen zu DOM entwickeln soll, um den mittleren Anforderungen von Webanwendungen (im Gegensatz zu den ausgefeilten, ausgewachsenen APIs auf Betriebssystemebene) zu genügen«. Das Ergebnis war 11 zu 8. In ihrer Zusammenfassung des Workshops (*http://www.w3.org/2004/04/webapps-cdf-ws/summary*) schrieben die W3C-Mitglieder: »Zurzeit beabsichtigt das W3C nicht, Ressourcen in das dritte Thema der unverbindlichen Abstimmung zu stecken: Erweiterungen zu HTML und CSS für Webanwendungen über die Technologien hinaus, die unter dem Dach der aktuellen W3C-Arbeitsgruppen entwickelt werden.«

Angesichts dieser Entscheidung hatten die Gruppen, die eine Weiterentwicklung von HTML und HTML-Formularen anstrebten, nur zwei Möglichkeiten: aufzugeben oder die eigene Arbeit außerhalb des W3C fortzusetzen. Sie entschieden sich für Letzteres, registrierten die Domain *whatwg.org*, und im Juni 2004 erblickte die WHAT Working Group das Licht der Welt.

Was? Die WHAT Working Group

Aber was ist nun eigentlich die WHAT Working Group? Das lasse ich sie selbst erläutern (*http://www.whatwg.org/news/start*):

> Die Web Hypertext Applications Technology Working Group ist ein lockerer, inoffizieller und offener Zusammenschluss von Browserherstellern und anderen betroffenen Parteien. Die Gruppierung beabsichtigt, Spezifikationen auf Basis von HTML und verwandten Technologien zu entwickeln, die die Entwicklung interaktiver Webanwendungen vereinfachen, und plant, die Ergebnisse bei einer Standardisierungsorganisation einzureichen. Die so eingereichten Ergebnisse sollen die Arbeitsgrundlage einer formellen Erweiterung von HTML im Rahmen eines Standards bilden.
>
> Die Bildung dieses Forums folgt auf mehrere Monate Arbeit über private E-Mails zu Spezifikationen für derartige Technologien. Das Hauptanliegen bis zu diesem Zeitpunkt war, HTML4-Formulare so zu erweitern, dass sie die Funktionen bieten, die Autoren fordern, ohne dass dabei die Kompatibilität mit bestehenden Inhalten gebrochen wird. Die Gruppe wurde gebildet, um zu sichern, dass die zukünftige Entwicklung dieser Spezifikationen vollkommen offen und über eine öffentlich archivierte und offene Mailingliste erfolgt.

Die Schlüsselformulierung hier ist: »ohne die Rückwärtskompatibilität zu brechen«. XHTML (vom Schlupfloch Anhang C abgesehen) ist nicht rückwärtskompatibel zu HTML. Es verlangt einen vollständig neuen MIME-Typ und ordnet eine drakonische Fehlerbehandlung für alle Inhalte an, die mit diesem MIME-Typ ausgeliefert werden. XForms ist nicht rückwärtskompatibel zu HTML-Formularen, weil sie nur in Dokumenten angeboten werden können, die mit dem neuen XHTML-MIME-Typ ausgeliefert werden. Das bedeutet, dass auch XForms eine drakonische Fehlerbehandlung anordnet. Alle Wege führen über MIME!

Statt alles, was in über einem Jahrzehnt in HTML investiert wurde, auf den Müll zu werfen, und 99 Prozent der existierenden Webseiten unbrauchbar zu machen, entschloss sich die WHAT Working Group, einen anderen Ansatz zu wählen: die Algorithmen für die »nachsichtige Fehlerbehandlung« zu dokumentieren, die Browser tatsächlich nutzten. Webbrowser waren HTML-Fehlern gegenüber immer nachsichtig. Aber es hatte sich nie jemand die Mühe gemacht, aufzuschreiben, wie genau sie das machten. NCSA Mosaic hatte einen eigenen Algorithmus für den Umgang mit defekten Seiten, und Netscape versuchte gleichzuziehen. Anschließend versuchte der Internet Explorer, mit Netscape gleichzuziehen. Dann versuchten Opera und Firefox, mit dem Internet Explorer gleichzuziehen. Nun versuchte Safari, mit Firefox gleichzuziehen. Und so weiter – bis zum heutigen Tag. Auf der Strecke blieb dabei eine Heerschaar von Entwicklern, die versuchten, ihre Produkte mit denen ihrer Konkurrenten kompatibel zu machen.

Das klingt nicht nur nach einer unglaublichen Menge Arbeit, es ist eine unglaubliche Menge Arbeit. Oder, besser, war eine unglaubliche Menge Arbeit. Es nahm einige Jahre in Anspruch, aber es gelang der WHAT Working Group erfolgreich (ein paar obskure Grenzfälle ausgenommen), zu dokumentieren, wie man HTML auf eine Weise parst, die mit bestehenden Webinhalten kompatibel ist. Im endgültigen Algorithmus gibt es keinen

Schritt, der anordnet, dass der HTML-Verbraucher die Verarbeitung abbrechen und dem Endnutzer eine Fehlermeldung anzeigen soll.

Während die dazu erforderlichen Rekonstruktionsarbeiten erfolgten, hat die WHAT Working Group still auch noch an einigen anderen Dingen gearbeitet. Eins davon war eine Spezifikation, die zunächst Web Forms 2.0 getauft wurde, die HTML-Formularen neuen Steuerelemente hinzufügte. (Mehr zu Web Forms werden Sie in Kapitel 9 erfahren.) Das andere war ein Spezifikationsentwurf namens »Web Applications 1.0«, der wichtige neue Funktionen, wie eine Direct Mode-Zeichenumgebung (siehe Kapitel 4), sowie native Unterstützung für Audio und Video ohne Plug-ins (siehe Kapitel 5) beinhaltete.

Zurück zum W3C

Einige Jahre ignorierten W3C und WHAT Working Group einander größtenteils. Während sich die WHAT Working Group auf Webformulare und neue HTML-Funktionen konzentrierte, war die W3C HTML Working Group mit Version 2.0 von XHTML beschäftigt. Aber im Oktober 2006 war klar, dass die WHAT Working Group gewaltig in Schwung gekommen war, während XHTML 2 immer noch im Entwurfsstadium hing und von keinem wichtigen Browser implementiert war. In eben diesem Monat verkündete Tim Berners-Lee, der Gründer des W3C höchstpersönlich, dass das W3C mit der WHAT Working Group zusammenarbeiten werde (*http://dig.csail.mit.edu/breadcrumbs/node/166*), um HTML weiterzuentwickeln:

> Manche Dinge machen einige Jahre Erfahrung klarer. Es ist notwendig, dass HTML inkrementell entwickelt wird. Der Versuch, die Welt dazu zu bringen, auf XML umzusteigen, und ihr auf einmal Anführungszeichen um Attribute, Schrägstriche in leeren Tags und Namensräume aufzuerlegen, ist gescheitert. Die große Gemeinschaft derer, die HTML generieren, hat sich nicht bewegt, hauptsächlich weil sich die Browser nicht beschwerten. Einige größere Gruppen stiegen um und genießen jetzt die Früchte wohlgeformter Systeme, aber eben nicht alle. Es ist wichtig, HTML inkrementell zu pflegen und gleichzeitig den Übergang zu einer wohlgeformten Welt fortzusetzen und mehr Kraft in diese Richtung zu entwickeln.
>
> Der Plan ist, eine vollständig neue HTML-Arbeitsgruppe zu bilden. Im Unterschied zur Vorgängergruppe erhält diese den Auftrag, HTML inkrementell zu verbessern und parallel dazu auch XHTML. Sie wird eine neue Leitung und einen neuen Ansprechpartner haben. Sie wird an HTML und XHTML gemeinsam arbeiten. Viele Kreise, einschließlich Browserherstellern, mit denen wir gesprochen haben, unterstützen diese Gruppe.
>
> Es wird auch Arbeit an Formularen geben. Das ist ein komplexes Feld, da sowohl die bestehenden HTML-Formulare als auch XForms Formularsprachen sind. HTML-Formulare sind allgegenwärtig, aber es gibt auch viele Implementierungen und Nutzer von XForms. Inzwischen hat der Webforms-Beitrag einige sinnvolle Erweiterungen für HTML-Formulare vorgeschlagen. Der Plan ist, unter Rückgriff auf Webforms HTML-Formulare zu erweitern.

Eine der ersten Amtshandlungen der neu gebildeten W3C HTML Working Group war, »Web Applications 1.0« in »HTML5« umzutaufen. Und da sind wir – bereit zum Einstieg in HTML5.

Nachtrag

Im Oktober 2009 schloss das W3C die XHTML 2 Working Group und gab folgende Erklärung ab, um die Entscheidung zu erläutern (*http://www.w3.org/2009/06/xhtml-faq.html*):

> Als das W3C im März 2007 die HTML and XHTML 2 Working Groups einsetzte, haben wir darauf hingewiesen, dass wir den Markt für XHTML 2 weiterhin beobachten würden. Das W3C erkennt nun, dass es immens wichtig ist, der Community ein eindeutiges Signal über die Zukunft von HTML zu geben.
>
> Obwohl wir den Wert der Beiträge der XHTML 2 Working Group über die Jahre anerkennen, hat das W3C-Management nach Gesprächen mit den Teilnehmern beschlossen, das Statut der Working Group zum Ende des Jahres 2009 auslaufen zu lassen und nicht zu erneuern.

Es gewinnen die, die veröffentlichen!

Weitere Lektüre

- »The History of the Web« (*http://hixie.ch/commentary/web/history*), ein alter Entwurf von Ian Hickson
- »HTML/History« (*http://esw.w3.org/topic/HTML/history*), von Michael Smith, Henri Sivonen und anderen
- »A Brief History of HTML« (*http://www.atendesigngroup.com/blog/brief-history-of-html*), von Scott Reynen

KAPITEL 2

HTML5-Funktionen abprüfen

Einstieg

»Wie bitte kann ich HTML5 einsetzen, wenn ältere Browser es nicht unterstützen?« Ist das die Frage, die Ihnen auf der Zunge liegt? Jedoch wäre die Formulierung der Frage bereits irreführend. HTML5 ist nicht eine große Sache, es ist eine Sammlung separater Funktionalitäten. Es ergäbe also überhaupt keinen Sinn, zu prüfen, ob »HTML5 insgesamt« unterstützt wird. Dagegen können Sie aber prüfen, ob bestimmte Features wie Canvas, Video oder Geolocation unterstützt werden.

Erkennungstechniken

Wenn Ihr Browser eine Webseite darstellt, konstruiert er ein Document Object Model (DOM), eine Sammlung von Objekten, die die HTML-Elemente auf der Seite darstellen. Jedes Element – jedes `<p>`, jedes `<div>`, jedes `<span>` – wird im DOM durch ein anderes Objekt dargestellt. (Es gibt auch globale Objekt wie `window` und `document`, die nicht an spezifische Elemente gebunden sind.)

Alle DOM-Objekte teilen einen Satz gemeinsamer Eigenschaften, aber einige Objekte haben zusätzliche Eigenschaften, die andere nicht haben. In Browsern, die HTML5-Funktionen unterstützen, besitzen bestimmte Objekte bestimmte eindeutige Eigenschaften. Ein kurzer Blick auf das DOM sagt Ihnen also, welche Funktionen unterstützt werden.

Es gibt vier grundlegende Techniken, um zu erkennen, ob ein Browser eine bestimmte Funktion unterstützt. In aufsteigender Komplexität sind das:

1. Prüfen, ob ein globales Objekt (wie `window` oder `navigator`) eine bestimmte Eigenschaft unterstützt.

   Ein Beispiel für eine derartige Prüfung der Geolocation-Unterstützung finden Sie im Abschnitt »Geolocation« auf Seite 27.
2. Ein Element erstellen und dann prüfen, ob dieses Element eine bestimmte Eigenschaft bietet.

Ein Beispiel für eine derartige Prüfung der Canvas-Unterstützung finden Sie im Abschnitt »Canvas« auf Seite 19.

3. Ein Element erstellen und prüfen, ob dieses Element eine bestimmte Methode bietet, und gegebenenfalls dann die Methode aufrufen, um den Wert zu prüfen, den sie liefert.

   Ein Beispiel für eine derartige Erkennung der unterstützten Videoformate finden Sie im Abschnitt »Videoformate« auf Seite 22.

4. Ein Element erstellen, eine bestimmte Eigenschaft auf einen bestimmten Wert setzen und dann prüfen, ob die Eigenschaft den Wert bewahrt hat.

   Ein Beispiel für eine derartige Prüfung der unterstützten `<input>`-Typen finden Sie im Abschnitt »input-Typen« auf Seite 28.

Modernizr: Eine Bibliothek zur HTML5-Erkennung

Modernizr (*http://www.modernizr.com*) ist eine unter der MIT-Lizenz stehende Open Source-JavaScript-Bibliothek, die die Unterstützung für viele HTML5- und CSS3-Funktionen prüft. Als dies geschrieben wurde, war 1.1 die aktuelle Version. Sie sollten immer die aktuellste Version einsetzen. Fügen Sie dazu zu Beginn Ihrer Seite folgendes `<script>`-Element ein:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Einstieg in HTML5</title>
  <script src="modernizr.min.js"></script>
</head>
<body>
  ...
</body>
</html>
```

Modernizr wird automatisch ausgeführt. Es gibt keine `modernizr_init()`-Funktion, die aufgerufen werden muss. Bei der Ausführung erstellt Modernizr ein globales Objekt namens `Modernizr`, das einen Satz Boolescher Eigenschaften für jede Funktion enthält, die erkannt wurde. Unterstützt Ihr Browser die Canvas-API (siehe Kapitel 4), wird beispielsweise die Eigenschaft `Modernizr.canvas` auf `true` gesetzt. Unterstützt er die Canvas-API nicht, wird `Modernizr.canvas` auf `false` gesetzt:

```
if (Modernizr.canvas) {
  // Zeichnen wir ein paar Figuren!
} else {
  // Keine native Canvas-Unterstützung verfügbar :(
}
```

Canvas

HTML5 definiert das <canvas>-Element (*http://bit.ly/9JHzOf*) als »auflösungsunabhängige Zeichenfläche, die zur Erstellung von Diagrammen, Grafiken für Spiele oder anderen visuellen Bildern genutzt werden kann.« Ein Canvas ist ein Rechteck in Ihrer Seite, in das Sie mit JavaScript alles zeichnen können, was Ihnen beliebt. HTML5 definiert eine Funktionsmenge (die Canvas-API) zum Zeichnen von Figuren, Definieren von Pfaden, Erstellen von Verläufen und Anwenden von Transformationen.

Die Prüfung auf Canvas-API-Unterstützung nutzt Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser die Canvas-API unterstützt, hat das von ihr erstellte DOM-Objekt zur Repräsentation eines <canvas>-Elements eine getContext()-Methode (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60). Unterstützt er die Canvas-API nicht, hat das für ein <canvas>-Element erstellte DOM-Objekt nur einen Satz allgemeiner Eigenschaften, keine Canvas-spezifischen Eigenschaften. Canvas-Unterstützung können Sie mit folgender Funktion prüfen:

```
function supports_canvas() {
  return !!document.createElement('canvas').getContext;
}
```

Diese Funktion erstellt zunächst ein ungenutztes <canvas>-Element:

```
return !!document.createElement('canvas').getContext;
```

Dieses Element wird nie in Ihre Seite eingebunden und deswegen von keinem gesehen. Es geistert bloß im Speicher herum, bewegt sich nicht und tut nichts – wie ein Kanu auf einem trägen Fluss.

Nachdem Sie dieses <canvas>-Element erstellt haben, prüfen Sie, ob es die Methode getContext() bietet. Diese Methode gibt es nur, wenn Ihr Browser die Canvas-API unterstützt:

```
return !!document.createElement('canvas').getContext;
```

Schließlich nutzen Sie den Trick mit der doppelten Negation, um das Ergebnis in einen Booleschen Wert (true oder false) umzuwandeln:

```
return !!document.createElement('canvas').getContext;
```

Diese Funktion erkennt die Unterstützung für die meisten Funktionen der API, einschließlich Figuren (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60), Pfaden (siehe dazu den Abschnitt »Pfade« auf Seite 63), Verläufen (siehe dazu den Abschnitt »Verläufe« auf Seite 70) und Mustern. Die externe explorercanvas-Bibliothek (siehe dazu den Abschnitt »Was ist mit dem IE?« auf Seite 76), die die Canvas-API in Microsofts Internet Explorer bereitstellt, erkennt es nicht.

Statt diese Funktion selbst zu schreiben, können Sie Modernizr (das wir im letzten Abschnitt vorstellten) nutzen, um die Unterstützung für die Canvas-API zu prüfen:

```
if (Modernizr.canvas) {
  // Zeichnen wir ein paar Figuren!
```

```
} else {
  // Keine native Canvas-Unterstützung verfügbar :(
}
```

Es gibt eine eigene Prüfung für die Canvas-Text-API, die wir Ihnen gleich vorstellen werden.

Canvas-Text

Selbst wenn Ihr Browser die Canvas-API unterstützt, kann es sein, dass er die Canvas-Text-API nicht unterstützt. Die Canvas-API ist mit der Zeit gewachsen, und die Textfunktionen wurden recht spät eingeführt. Einige Browser boten Canvas-Unterstützung, bevor die Text-API vollständig war.

Die Prüfung für die Canvas-Text-API-Unterstützung nutzt erneut Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Falls Ihr Browser die Canvas-API unterstützt, bietet das DOM-Objekt, das zur Repräsentation eines `<canvas>`-Elements genutzt wird, eine `getContext()`-Methode (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60). Sollte er die Canvas-API nicht unterstützen, hat das für ein `<canvas>`-Element erstellte DOM-Objekt nur allgemeine Eigenschaften, keine Canvas-spezifischen. Canvas-Text-Unterstützung können Sie mit folgender Funktion prüfen:

```
function supports_canvas_text() {
  if (!supports_canvas()) { return false; }
  var dummy_canvas = document.createElement('canvas');
  var context = dummy_canvas.getContext('2d');
  return typeof context.fillText == 'function';
}
```

Zunächst prüft diese Funktion mit der im vorangehenden Abschnitt vorgestellten Funktion `supports_canvas()` die Canvas-Unterstützung:

```
if (!supports_canvas()) { return false; }
```

Unterstützt Ihr Browser die Canvas-API nicht, unterstützt er mit Sicherheit auch die Canvas-Text-API nicht!

Anschließend wird ein ungenutztes `<canvas>`-Element erstellt und sein Zeichenkontext abgerufen. Da `supports_canvas()` bereits geprüft hat, ob das Canvas-Objekt die Methode `getContext()` bietet, ist gesichert, dass das funktioniert:

```
var dummy_canvas = document.createElement('canvas');
var context = dummy_canvas.getContext('2d');
```

Abschließend prüfen Sie, ob Ihr Zeichenkontext eine `fillText()`-Funktion bietet. Tut er das, ist die Canvas-Text-API verfügbar:

```
return typeof context.fillText == 'function';
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für die Canvas-Text-API zu prüfen:

```
if (Modernizr.canvastext) {
 // Zeichnen wir etwas Text!
} else {
  // Keine native Unterstützung für Canvas-Text verfügbar :(
}
```

Video

HTML5 definiert ein neues Element namens `<video>`, über das Videos in Ihre Webseiten eingebettet werden können. Die Einbettung von Videos war zuvor ohne externe Plug-ins wie Apple QuickTime oder Adobe Flash nicht möglich.

Das `<video>`-Element wurde so entworfen, dass es ohne Erkennungsskripten genutzt werden kann. Sie können mehrere Videodateien angeben, und Browser, die HTML5-Video unterstützen, wählen auf Basis der von ihnen unterstützten Videoformate eine davon aus.[1]

Browser, die HTML5-Video nicht unterstützen, ignorieren das `<video>`-Element einfach. Das können Sie zu Ihrem Vorteil nutzen, indem Sie sie dann einfach anweisen, das Video stattdessen über ein externes Plug-in abzuspielen. Kroc Camen hat eine Lösung entworfen, die sich Video for Everybody! (*http://camendesign.com/code/video_for_everybody*) nennt und HTML5-Video nutzt, so es verfügbar ist, in älteren Browsern aber auf QuickTime oder Flash ausweicht. Diese Lösung verlangt keinerlei JavaScript und funktioniert in praktisch jedem Browser, auch in den Browsern von Mobilgeräten.

Wollen Sie Ihr Video nicht einfach nur in die Seite packen und abspielen, müssen Sie JavaScript nutzen. Die Prüfung für Videounterstützung nutzt Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser HTML5-Video unterstützt, hat das zur Repräsentation von `<video>`-Elementen erstellte DOM-Objekt eine `canPlayType()`-Methode. Unterstützt Ihr Browser HTML5-Video nicht, besitzt das für `<video>`-Elemente erstellte DOM-Objekt nur die Eigenschaften, die allen Elementen gemeinsam sind. Videounterstützung können Sie mit folgender Funktion prüfen:

```
function supports_video() {
  return !!document.createElement('video').canPlayType;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für HTML5-Video zu prüfen:

1 Siehe »A gentle introduction to video encoding, part 1: container formats« (*http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats*) und »part 2: lossy video codecs« (*http://diveintomark.org/archives/2008/12/19/give-part-2-lossy-video-codecs*), wenn Sie mehr über die unterschiedlichen Videoformate erfahren wollen.

```
if (Modernizr.video) {
  // Spielen wir ein Video!
} else {
  // Keine native Videounterstützung verfügbar :(
  // Vielleicht prüfen Sie stattdessen auf QuickTime oder Flash!
}
```

Es gibt einen eigenen Test, mit dem Sie prüfen können, welche Videoformate Ihr Browser abspielen kann. Diesen werden wir Ihnen im nächsten Abschnitt vorführen.

Videoformate

Videoformate sind wie Sprachen. Auch wenn eine englische und eine spanische Zeitung die gleichen Informationen enthalten, bringt Ihnen nur eine der beiden Zeitungen etwas, solange Sie lediglich eine der beiden Sprachen beherrschen! Wenn ein Browser ein Video abspielen können soll, muss er die »Sprache« beherrschen, in der es geschrieben wurde.

Die »Sprache« eines Videos nennt man »Codec« –das ist der Algorithmus, der eingesetzt wurde, um das Video zu einem Bit-Strom zu kodieren. Es gibt auf der Welt Dutzende von Codecs. Welchen davon sollen Sie also nutzen? Die unglückselige Realität von HTML5-Video ist, dass sich Browser einfach nicht auf einen einzigen Codec einigen können. Es scheint allerdings, als hätte man die Sache jetzt auf zwei eingeschränkt. Der eine Codec kostet Geld (aufgrund der Patentlizenzen) und wird von Safari und dem iPhone unterstützt. (Er funktioniert auch in Adobe Flash, falls Sie eine Lösung wie Video for Everybody! nutzen.) Der andere Codec ist frei und funktioniert in Open Source-Browsern wie Chromium und Mozilla Firefox.

Die Prüfung auf die Videoformatunterstützung nutzt Erkennungstechnik Nummer 3 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Falls Ihr Browser HTML5-Video unterstützt, bietet das zur Repräsentation des `<video>`-Elements erstellte DOM-Objekt eine `canPlayType()`-Methode. Diese Methode sagt Ihnen, ob der Browser ein bestimmtes Videoformat unterstützt.

Folgende Funktion prüft auf das patentbelastete Format, das von Macs und iPhones unterstützt wird:

```
function supports_h264_baseline_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

Zunächst prüft die Funktion die HTML5-Video-Unterstützung mit der `supports_video()`-Funktion aus dem vorangegangenen Abschnitt:

```
if (!supports_video()) { return false; }
```

Unterstützt Ihr Brower HTML5-Video nicht, unterstützt er natürlich auch keinerlei Videoformate!

Anschließend erstellt die Funktion ein ungenutztes <video>-Element (bindet es aber nicht in die Seite ein, damit es nicht sichtbar wird) und ruft die Methode canPlayType() auf. Diese Methode muss vorhanden sein, da wir mit supports_video() gerade geprüft haben, ob dem so ist:

```
var v = document.createElement("video");
return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
```

Eigentlich ist ein »Videoformat« eine Kombination aus mehreren unterschiedlichen Dingen. Technisch ausgedrückt, fragen Sie den Browser damit, ob er H.264 Baseline-Video und AAC LC-Audio in einem MPEG-4-Container abspielen kann.[2]

Die Funktion canPlayType() liefert nicht einfach true oder false. Videoformate sind eine komplexe Angelegenheit! Deswegen liefert die Funktion einen String, der anzeigt, was der Browser vom jeweiligen Format hält:

"probably"
: Falls der Browser recht sicher ist, dass er das Format abspielen kann.

"maybe"
: Sollte der Browser meinen, dass er dazu in der Lage sein könnte, dieses Format abzuspielen.

"" *(ein leerer String)*
: Wenn der Browser sicher ist, dass er dieses Format nicht abspielen kann.

Eine zweite Funktion prüft das offene Videoformat, das von Mozilla Firefox und anderen Open Source-Browsern unterstützt wird. Das Vorgehen ist genau das gleiche. Der einzige Unterschied ist der String, der der Funktion canPlayType() übergeben wird. Hier fragen Sie den Browser technisch ausgedrückt, ob er Theora-Video und Vorbis-Audio in einem Ogg-Container abspielen kann:

```
function supports_ogg_theora_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/ogg; codecs="theora, vorbis"');
}
```

Außerdem gibt es mit WebM (*http://www.webmproject.org*) einen weiteren, kürzlich zu Open Source gemachten (und nicht patentbelasteten) Videocodec, der in die nächsten Versionen der wichtigeren Browser, einschließlich Chrome, Firefox und Opera, eingebaut werden wird. Sie können die gleiche Technik für die Prüfung von WebM-Video nutzen:

```
function supports_webm_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/webm; codecs="vp8, vorbis"');
}
```

2 Was all das heißt, werde ich in Kapitel 5 erklären. Ergänzend können Sie dazu auch »A gentle introduction to video encoding« (*http://diveintomark.org/tag/give*) lesen.

Statt eigenhändig diese Funktionen zu schreiben, nutzen Sie Modernizr, um die Unterstützung verschiedener HTML5-Videoformate zu prüfen (beachten Sie, dass Modernizr aktuell noch keine Unterstützung für das Erkennen des WebM-Videoformats bietet):

```
if (Modernizr.video) {
 // Film ab! Aber in welchem Format?
  if (Modernizr.video.ogg) {
    // Probieren wir Ogg Theora + Vorbis in einem Ogg-Container!
  } else if (Modernizr.video.h264){
    // Probieren wir H.264-Video + AAC-Audio in einem MP4-Container!
  }
}
```

Local Storage

HTML5 Storage (*http://dev.w3.org/html5/webstorage/*) oder Local Storage (lokaler Speicher) bietet Webseiten eine Möglichkeit, Daten auf Ihrem Rechner zu speichern und später wieder abzurufen. Das Konzept ist mit dem von Cookies vergleichbar, aber für größere Datenmengen gedacht. Die Größe von Cookies ist beschränkt, und Ihr Browser sendet sie mit jeder Anfrage nach einer neuen Seite (was zusätzliche Zeit und wertvolle Bandbreite in Anspruch nimmt). HTML5 Storage bleibt auf Ihrem Rechner, und Websites können darauf mit JavaScript zugreifen, nachdem die Seite geladen wurde.

Fragen an Professor Markup

F: Ist HTML5 Storage tatsächlich Teil von HTML5? Warum steckt es dann in einer eigenständigen Spezifikation?

A: Die kurze Antwort ist: Ja. HTML5 Storage ist Teil von HTML5. Die etwas längere Antwort ist, dass lokaler Speicher ursprünglich Teil der eigentlichen HTML5-Spezifikation war, aber in eine separate Spezifikation ausgelagert wurde, weil sich einige Mitglieder der HTML5 Working Group beschwerten, dass HTML5 zu umfangreich werde. Wenn das für Sie so klingt, als wollte man die Kalorienmenge reduzieren, indem man das Törtchen in mehrere Teile zerlegt, dann willkommen in der verdrehten Welt der Standards.

Die Prüfung der Unterstützung für HTML5 Storage nutzt Erkennungstechnik Nummer 1 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser HTML5 Storage unterstützt, besitzt das globale `window`-Objekt eine `localStorage`-Eigenschaft. Unterstützt er HTML5 Storage nicht, ist die Eigenschaft `localStorage` undefiniert. Mit folgender Funktion können Sie prüfen, ob lokaler Storage unterstützt wird:

```
function supports_local_storage() {
  return ('localStorage' in window) && window['localStorage'] !== null;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für HTML5 Local Storage zu prüfen:

```
if (Modernizr.localstorage) {
  // window.localStorage ist verfügbar!
} else {
  // Keine native Unterstützung für Local Storage :(
  // Vielleicht nehmen Sie Gears oder eine andere externe Lösung!
}
```

Beachten Sie, dass JavaScript Groß-/Kleinschreibung berücksichtigt. Das Modernizr-Attribut heißt `localstorage` (alles Kleinbuchstaben), die DOM-Eigenschaft hingegen `window.localStorage` (Groß- und Kleinbuchstaben).

Fragen an Professor Markup

F: Wie sicher ist meine HTML5 Storage-Datenbank? Können andere sie lesen?

A: Jeder, der sich an Ihren Rechner setzen kann, kann vermutlich auf Ihre HTML5 Storage-Datenbank zugreifen (oder sie gar ändern). Im Browser kann jede Website die eigenen Werte lesen und schreiben, aber nicht auf die Werte zugreifen, die von anderen Sites gespeichert wurden. Das ist die sogenannte Same-Origin-Restriction (*http://bit.ly/9YyPpj*) (d.h. eine Beschränkung auf den gleichen Urheber).

Web Worker

Web Worker (*http://bit.ly/9jheof*) bietet ein Standardverfahren für die Ausführung von JavaScript im Hintergrund. Web Worker ermöglicht Ihnen, »Threads« in Gang zu setzen, die mehr oder minder parallel ausgeführt werden. (Überlegen Sie, auf welche Weise Ihr Computer mehrere Anwendungen ausführen kann, und Sie haben die Sache schon fast verstanden.) Diese »Hintergrund-Threads« können komplexe mathematische Berechnungen anstellen, Netzwerkanfragen durchführen und auf lokalen Speicher zugreifen, während die eigentliche Webseite weiterhin reagiert, wenn der Benutzer scrollt, klickt oder tippt.

Die Prüfung für Web Worker nutzt Erkennungstechnik Nummer 1 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser die Web Worker-API unterstützt, bietet das globale `window`-Objekt eine `Worker`-Eigenschaft. Unterstützt Ihr Browser die Web Worker-API nicht, ist die Eigenschaft `Worker` undefiniert. Folgende Funktion prüft die Web Worker-Unterstützung:

```
function supports_web_workers() {
  return !!window.Worker;
}
```

Anstatt diese Funktion selbst zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für Web Worker zu prüfen:

```
if (Modernizr.webworkers) {
  // window.Worker ist verfügbar!
} else {
  // Keine native Unterstützung für Web Worker :(
  // Vielleicht nehmen Sie Gears oder eine andere externe Lösung!
}
```

Beachten Sie, dass JavaScript Groß-/Kleinschreibung berücksichtigt. Das Modernizr-Attribut heißt `webworkers` (alles Kleinbuchstaben), das DOM-Objekt hingegen `window.Worker` (mit dem großen »W« von »Worker«).

Offline-Webanwendungen

Statische Webseiten offline zu lesen, ist leicht: Stellen Sie eine Verbindung mit dem Internet her, laden Sie eine Webseite, beenden Sie die Verbindung mit dem Internet, fahren Sie in ein einsames Häuschen am See und lesen Sie in aller Ruhe die Webseite. (Wenn Sie Zeit sparen wollen, können Sie den Schritt mit dem Häuschen am See weglassen.) Aber was ist mit Webanwendungen wie Gmail oder Google Docs, wenn Sie offline sind? Dank HTML5 kann jetzt jeder (nicht nur Google) Webseiten bauen, die auch offline funktionieren.

Offline-Webanwendungen beginnen als gewöhnliche Webanwendungen. Wenn Sie das erste Mal eine offlinefähige Website besuchen, sagt der Webserver Ihrem Browser, welche Dateien er für die Offlinearbeit benötigt. Diese Dateien können beliebiger Natur sein – HTML, JavaScript, Bilder, selbst Videos (siehe dazu den Abschnitt »Video« auf Seite 21). Hat Ihr Browser einmal alle erforderlichen Dateien heruntergeladen, können Sie die Webseite wieder besuchen, selbst wenn Sie nicht mit dem Internet verbunden sind. Ihr Browser bemerkt, dass Sie offline sind, und nutzt die Dateien, die er bereits heruntergeladen hat. Sobald Sie wieder online sind, werden alle Änderungen, die Sie vorgenommen haben, zurück auf den entfernten Webserver geladen.

Offlineunterstützung prüfen Sie mit Erkennungstechnik Nummer 1 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser Offline-Webanwendungen unterstützt, bietet das globale `window`-Objekt eine `applicationCache`-Eigenschaft. Unterstützt Ihr Browser Offline-Webanwendungen nicht, ist die Eigenschaft `applicationCache` undefiniert. Offlineunterstützung können Sie mit der folgenden Funktion prüfen:

```
function supports_offline() {
  return !!window.applicationCache;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für Offline-Webanwendungen zu prüfen:

```
if (Modernizr.applicationcache) {
 // window.applicationCache ist verfügbar!
} else {
  // Keine native Offline-Unterstützung :(
  // Vielleicht probieren Sie Gears oder eine andere externe Lösung!
}
```

Beachten Sie, dass JavaScript Groß-/Kleinschreibung berücksichtigt. Das Modernizr-Attribut heißt `applicationcache` (alles Kleinbuchstaben), das DOM-Objekt hingegen `window.applicationCache` (gemischte Groß-/Kleinschreibung).

Geolocation

Bei Geolocation geht es darum, festzustellen, wo Sie sich auf der Welt befinden, und diese Information (wenn Sie möchten) mit Menschen zu teilen, denen Sie vertrauen. Es gibt viele Möglichkeiten, herauszufinden, wo Sie stecken – Ihre IP-Adresse, Ihre drahtlose Netzwerkverbindung, die Zelle, mit der Ihr Handy verbunden ist, oder spezielle GPS-Hardware, die Längen- und Breitengradangaben von Satelliten im All empfängt.

Fragen an Professor Markup

F: Ist Geolocation ein Teil von HTML5? Warum erwähnen Sie es dann?

A: Geolocation-Unterstützung wird zurzeit in Browser integriert, gemeinsam mit der Unterstützung für andere HTML5-Funktionen. Genau genommen, wird Geolocation von der Geolocation Working Group (*http://www.w3.org/2008/geolocation/*) standardisiert, die unabhängig von der HTML5 Working Group ist. Aber ich werde in diesem Buch trotzdem über Geolocation sprechen, weil sie Teil der Evolution des Webs ist, die aktuell stattfindet.

Die Prüfung der Geolocation-Unterstützung nutzt Erkennungstechnik Nummer 1 (siehe Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser die Geolocation-API unterstützt, bietet das globale `navigator`-Objekt eine `geolocation`-Eigenschaft. Unterstützt er die Geolocation-API nicht, ist die Eigenschaft `geolocation` undefiniert. Folgendermaßen prüfen Sie die Geolocation-Unterstützung:

```
function supports_geolocation() {
  return !!navigator.geolocation;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für die Geolocation-API zu prüfen:

```
if (Modernizr.geolocation) {
  // Schauen wir, wo Sie stecken!
} else {
  // Keine native Geolocation-Unterstützung verfügbar :(
  // Vielleicht probieren Sie Gears oder eine andere externe Lösung!
}
```

Auch wenn Ihr Browser keine native Unterstützung für die Geolocation-API bietet, ist nicht alle Hoffnung verloren. Gears (*http://tools.google.com/gears/*) ist ein Open Source-Browser-Plug-in von Google, das unter Windows, Mac, Linux, Windows Mobile und Android läuft. Es bietet eine Reihe von Funktionen für ältere Browser, die den ganzen ausgefallenen Kram, den wir in diesem Kapitel besprochen haben, nicht unterstützen. Eine der Funktionen, die Gears bietet, ist eine Geolocation-API. Es ist nicht die gleiche wie die `navigator.geolocation`-API, aber sie erfüllt den gleichen Zweck.

Außerdem gibt es gerätespezifische Geolocation-APIs auf mehreren Handyplattformen einschließlich BlackBerry, Nokia, Palm und OMTP BONDI.

Kapitel 6 wird sich ausführlich mit allen Aspekten der Verwendung der unterschiedlichen APIs befassen.

input-Typen

Sie wissen alles über Webformulare, stimmt's? Sie erstellen ein `<form>`-Element, fügen ein paar `<input type="text">`-Elemente und vielleicht ein `<input type="password">`-Element hinzu und geben der Sache mit einem `<input type="submit">`-Button den letzten Schliff.

Das, was Sie wissen, ist nicht einmal die Hälfte dessen, was es zu wissen gibt. HTML5 definiert mehr als ein Dutzend neuer Eingabetypen, die Sie in Ihren Formularen nutzen können:

`<input type="search">`
: Mehr zu Suchfeldern finden Sie unter *http://bit.ly/9mQt5C*.

`<input type="number">`
: Mehr zu Zahlenfeldern finden Sie unter *http://bit.ly/aPZHjD*.

`<input type="range">`
: Mehr zu Schiebereglern finden Sie unter *http://bit.ly/dmLiRr*.

`<input type="color">`
: Mehr zu Farbwählern finden Sie unter *http://bit.ly/bwRcMO*.

`<input type="tel">`
: Mehr zu Telefonnummern finden Sie unter *http://bit.ly/amkWLq*.

`<input type="url">`
: Mehr zu Webadressen finden Sie unter *http://bit.ly/cjKb3a*.

`<input type="email">`
: Mehr zu E-Mail-Adressen finden Sie unter *http://bit.ly/aaDrgS*.

`<input type="date">`
: Mehr zu Datumswählern finden Sie unter *http://bit.ly/c8hL58*.

`<input type="month">`
: Mehr zu Monatsangaben finden Sie unter *http://bit.ly/cDgHRI*.

`<input type="week">`
: Mehr zur Wochenangaben finden Sie unter *http://bit.ly/bR3r58*.

`<input type="time">`
: Mehr zu Zeitstempeln finden Sie unter *http://bit.ly/bfMCMn*.

`<input type="datetime">`
: Mehr zu präzisen und absoluten Datums- und Uhrzeitangaben finden Sie unter *http://bit.ly/c46zVW*.

`<input type="datetime-local">`
: Mehr zu lokalen Datums- und Uhrzeitangaben finden Sie unter *http://bit.ly/aziNkE*.

Die Prüfung der Unterstützung der HTML5-Eingabetypen nutzt Erkennungstechnik Nummer 4 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Zunächst erstellen Sie ein ungenutztes `<input>`-Element im Speicher:

```
var i = document.createElement("input");
```

Der Standardeingabetyp für alle `<input>`-Elemente ist `"text"`. Das wird von entscheidender Bedeutung sein.

Anschließend setzen Sie das `type`-Attribut dieses `<input>`-Elements auf den Eingabetyp, den Sie prüfen wollen:

```
i.setAttribute("type", "color");
```

Wenn Ihr Browser einen bestimmten Eingabetyp unterstützt, bewahrt die Eigenschaft `type` den von Ihnen gesetzten Wert. Unterstützt er diesen bestimmten Eingabetyp nicht, ignoriert er den von Ihnen gesetzten Wert, und die Eigenschaft `type` behält den Wert `"text"`:

```
return i.type !== "text";
```

Statt eigenhändig 13 separate Funktionen zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für alle in HTML5 neu definierten Eingabetypen zu prüfen. Modernizr nutzt nur ein einziges `<input>`-Element, um die Unterstützung für alle 13 Eingabetypen zu prüfen. Dann baut es einen Hash namens `Modernizr.inputtypes` auf, der 13 Schlüssel (die HTML5-`type`-Attribute) und 13 Boolesche Werte enthält (`true`, falls der Typ unterstützt wird, andernfalls `false`):

```
if (!Modernizr.inputtypes.date) {
  // Keine eingebaute Unterstützung für <input type="date"> :(
  // Vielleicht bauen Sie sich eine eigene mit
  // Dojo oder jQueryUI!
}
```

Platzhaltertext

Neben den neuen Eingabetypen enthält HTML5 einige kleinere Verbesserungen für bestehende Formulare. Eine Verbesserung ist die Möglichkeit, für ein Eingabefeld Platzhaltertext zu festzulegen. Platzhaltertext wird im Eingabefeld angezeigt, solange das Feld leer und nicht im Fokus ist. Sobald Sie in das Eingabefeld klicken (oder es per Tabulator betreten), verschwindet der Platzhaltertext. Abschnitt »Platzhaltertext« auf Seite 157 zeigt das in einem Screenshot, falls Sie Schwierigkeiten haben, sich das vorzustellen.

Die Prüfung der Platzhalterunterstützung nutzt Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser Platzhaltertext in Eingabefeldern unterstützt, enthält das zur Repräsentation eines `<input>`-Elements erzeugte DOM-Objekt eine `placeholder`-Eigenschaft (auch wenn Sie in Ihr HTML kein `placeholder`-Attribut einschließen). Unterstützt er Platzhaltertext nicht, hat das für ein `<input>`-Element erstellte DOM-Objekt keine `placeholder`-Eigenschaft. Folgendermaßen prüfen Sie die Platzhalterunterstützung:

```
function supports_input_placeholder() {
  var i = document.createElement('input');
  return 'placeholder' in i;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für Platzhaltertext zu prüfen:

```
if (Modernizr.input.placeholder) {
  // Ihr Platzhalter sollte jetzt bereits sichtbar sein!
} else {
  // Keine Platzhalterunterstützung :(
  // Weichen Sie auf ein skriptbasierte Lösung aus!
}
```

Formular-Autofokus

Viele Websites nutzen JavaScript, um dem ersten Eingabefeld eines Webformulars automatisch den Fokus zu geben. Beispielsweise setzt die Google-Homepage automatisch den Fokus in das Suchfeld, damit Sie sofort mit der Eingabe von Suchwörtern beginnen können, ohne zuvor manuell den Cursor im Eingabefeld positionieren zu müssen. Für die meisten Nutzer ist das äußerst angenehm, Benutzer mit besonderen Bedürfnissen kann es aber auch stören. Wenn Sie die Leertaste drücken, um die Seite zu scrollen, passiert nichts, weil das Eingabefeld bereits den Fokus hat. (Stattdessen wird ein Leerzeichen ins Eingabefeld eingegeben.) Setzen Sie den Fokus in ein anderes Eingabefeld, während die Seite noch geladen wird, kann es passieren, dass das »hilfsbereite« Autofokusskript der Site den Fokus wieder in ein anderes Eingabefeld setzt, sobald das Laden abgeschlossen ist, und Sie damit in Ihrer Arbeit unterbricht und an der falschen Stelle etwas eintippen lässt.

Da die Fokussteuerung über JavaScript erfolgt, kann es recht kompliziert sein, all diese Sonderfälle zu berücksichtigen. Und es gibt kaum Ausweichmöglichkeiten für diejenigen, die sich nicht daran erfreuen, dass Webseiten den Fokus »kapern«.

Um dieses Problem zu lösen, führt HTML5 ein `autofocus`-Attribut für alle Formularsteuerelemente ein. Das `autofocus`-Attribut macht genau das, was der Name verspricht: Es verschiebt den Fokus auf ein bestimmtes Eingabefeld. Aber weil es Markup statt Skript ist, ist das Verhalten auf allen Websites gleich. Außerdem können Browserhersteller (oder Autoren von Erweiterungen) den Benutzern Möglichkeiten bieten, das Autofokusverhalten abzuschalten.

Die Prüfung der Autofokusunterstützung nutzt Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser Formularsteuerelemente mit Autofokus unterstützt, bietet das zur Repräsentation eines `<input>`-Elements erzeugte DOM-Objekt eine `autofocus`-Eigenschaft (auch wenn Sie in Ihr HTML kein `autofocus`-Attribut einschließen). Bietet er keine Unterstützung für das automatische Fokussieren von Formularsteuerelementen, bietet das für `<input>`-Elemente erstellte DOM-Objekt keine `autofocus`-Eigenschaft. Autofokusunterstützung können Sie mit folgender Funktion prüfen:

```
function supports_input_autofocus() {
  var i = document.createElement('input');
  return 'autofocus' in i;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für automatisch fokussierte Formularfelder zu prüfen:

```
if (Modernizr.input.autofocus) {
// Autofokus funktioniert!
} else {
  // Keine Autofokus-Unterstützung :(
  // Weichen Sie auf eine skriptbasierte Lösung aus!
}
```

Mikrodaten

Mikrodaten (*http://bit.ly/ckt9Rj*) sind ein standardisiertes Verfahren, auf Ihren Webseiten zusätzliche semantische Informationen anzubieten. Beispielsweise können Sie Mikrodaten nutzen, um zu deklarieren, dass ein Foto unter einer bestimmten Creative Commons-Lizenz verfügbar ist. Wie Sie in Kapitel 10 sehen werden, können Sie Mikrodaten dazu verwenden, eine »Info«-Seite auszuzeichnen. Browser, Browsererweiterungen und Suchmaschinen können Ihr HTML5-Mikrodaten-Markup in vCard, ein Standardformat zum Austausch von Kontaktdaten, umwandeln. Außerdem können Sie Ihre eigenen Mikrodatenvokabulare definieren.

Der HTML5-Mikrodaten-Standard enthält sowohl HTML-Markup (im Wesentlichen für Suchmaschinen) als auch einen Satz von DOM-Funktionen (hauptsächlich für Browser).

Es schadet nicht, Mikrodaten-Markup in Webseiten einzuschließen. Es sind lediglich ein paar gut platzierte Attribute, und Suchmaschinen, die die Mikrodatenattribute nicht verstehen, ignorieren sie einfach. Aber wenn Sie über das DOM auf Mikrodaten zugreifen oder Mikrodaten manipulieren wollen, müssen Sie prüfen, ob der Browser die DOM-API für Mikrodaten unterstützt.

Die Prüfung auf Unterstützung der HTML5-Mikrodaten-API nutzt Erkennungstechnik Nummer 1 (siehe dazu den Abschnitt »Erkennungstechniken« auf Seite 17). Wenn Ihr Browser die HTML5-Mikrodaten-API unterstützt, bietet das globale `document`-Objekt eine `getItems()`-Funktion. Wenn er Mikrodaten nicht unterstützt, ist diese `getItems()`-Funktion undefiniert. Die Mikrodatenunterstützung können Sie folgendermaßen prüfen:

```
function supports_microdata_api() {
  return !!document.getItems;
}
```

Modernizr bietet aktuell noch keine Unterstützung für die Prüfung der Mikrodaten-API. Sie müssen also eine Funktion wie die obige nutzen.

Weitere Lektüre

Spezifikationen und Standards:

- Das `<canvas>`-Element (*http://bit.ly/9JHzOf*)
- Das `<video>`-Element (*http://bit.ly/a3kpiq*)
- `<input>`-Typen (*http://bit.ly/akweH4*)
- Das `<input placeholder>`-Attribut (*http://bit.ly/caGl8N*)
- Das `<input autofocus>`-Attribut (*http://bit.ly/db1Fj4*)
- HTML5 Storage (*http://dev.w3.org/html5/webstorage/*)
- Web Workers (*http://bit.ly/9jheof*)
- Offline-Webanwendungen (*http://bit.ly/d8ZgzX*)
- Die Geolocation-API (*http://www.w3.org/TR/geolocation-API/*)

JavaScript-Bibliotheken:

- Modernizr (*http://www.modernizr.com/*), eine HTML5-Erkennungsbibliothek
- *geo.js* (*http://code.google.com/p/geo-location-javascript/*), ein Geolocation-API-Wrapper

Andere Artikel und Einführungen:

- Video for Everybody! (*http://camendesign.com/code/video_for_everybody*)
- »A gentle introduction to video encoding« (*http://diveintomark.org/tag/give*)
- »Video type parameters« (*http://wiki.whatwg.org/wiki/Video_type_parameters*)
- Der Anhang zu diesem Buch

KAPITEL 3

Was all das bedeutet

Einstieg

Dieses Kapitel wird sich eine HTML-Seite vornehmen, die vollkommen in Ordnung ist, und sie verbessern. Einige Teile werden kürzer werden, andere länger. Und alles wird so viel semantischer werden. Es wird Sie umwerfen.

Hier ist die entsprechende Seite: *http://diveintohtml5.org/examples/blog-original.html*. Studieren Sie sie. Leben Sie sie. Lieben Sie sie. Öffnen Sie sie in einem neuen Tab und kehren Sie nicht zurück, bevor Sie nicht wenigstens einmal auf *Quelltext anzeigen* geklickt haben.

Die Doctype-Deklaration

Beginnen wir mit der ersten Zeile:

```
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Dieses Ding nennt man die Doctype-Deklaration. Dahinter verbirgt sich eine lange Geschichte und etwas schwarze Magie. Während der Entwicklung des Internet Explorer 5 für den Mac stand Microsoft vor einem überraschenden Problem. Die anstehende Version des Browsers hatte ihre Standardunterstützung so stark verbessert, dass ältere Seiten nicht mehr ordentlich dargestellt wurden. Besser gesagt, sie wurden korrekt (d.h. der Spezifikation gemäß) dargestellt, aber man erwartete, dass sie *inkorrekt* dargestellt würden. Die Seiten selbst wurden unter Berücksichtigung der Macken der vorherrschenden Browser der Zeit, allen voran Netscape 4 und Internet Explorer 4, geschrieben. IE5/Mac war so fortgeschritten, dass er das Web zerstörte.

Microsoft fand eine innovative Lösung. Bevor der IE5/Mac die Seite darstellte, warf er einen Blick auf die Doctype-Deklaration, die üblicherweise die erste Zeile in einer HTML-Quelle bildet (und sogar vor dem `<html>`-Element selbst steht). Ältere Seiten (die sich auf die Mängel älterer Browser stützten) enthielten üblicherweise überhaupt keine Doctype-Deklaration. Diese Seiten stellte der IE5/Mac wie ältere Browser dar. Um die neue

Standardunterstützung zu »aktivieren«, mussten die Autoren von Webseiten vor dem `<html>`-Element den richtigen Doctype deklarieren.

Dieses Konzept verbreitete sich wie ein Lauffeuer, und schon bald boten alle wichtigeren Browser zwei Modi an: den »Quirks-Modus« (der auf den Macken basierte) und den »Standards-Modus«. Wie Sie sich denken können – schließlich reden wir hier ja über das Internet –, geriet die Sache schnell außer Kontrolle. Als Mozilla versuchte, Version 1.1 des eigenen Browsers zu veröffentlichen, stellte man fest, dass sich einige im Standards-Modus dargestellte Seiten eigentlich auf bestimmte Macken verließen. Um diese Macken zu beheben, reparierte Mozilla seine Rendering-Engine und zerstörte damit auf einen Schlag Tausende von Seiten. Und dies führte – ungelogen – zum »Almost Standards-Modus«.

In seiner grundlegenden Schrift, »Activating Browser Modes with Doctype« (*http://hsivonen.iki.fi/doctype/*), fasst Henri Sivonen die verschiedenen Modi zusammen:

Quirks-Modus
: Im Quirks-Modus verletzen Browser die zeitgenössischen Webformatspezifikationen, um zu vermeiden, dass Seiten »beschädigt« werden, die gemäß den Verfahren geschrieben waren, die Ende der 1990er-Jahre vorherrschend waren.

Standards-Modus
: Im Standards-Modus versuchen Browser, den Spezifikationen entsprechende Dokumente spezifikationsgemäß zu behandeln, sofern der Browser diese unterstützt. HTML5 nennt diesen Modus den »No Quirks-Modus«.

Almost Standards-Modus
: Firefox, Safari, Chrome, Opera (seit 7.5) und IE8 bieten einen »Almost Standards-Modus« genannten Modus, der die vertikale Größenermittlung für Tabellenzellen traditionell und nicht streng gemäß der CSS2-Spezifikation durchführte. Diesen Modus bezeichnet HTML5 als den »eingeschränkten Quirks-Modus«.

Sie sollten den Rest von Henris Artikel lesen, weil ich hier erheblich vereinfache. Selbst im IE5/Mac gab es einige ältere Doctype-Deklarationen, die nicht reichten, um die Standardunterstützung zu aktivieren. Mit der Zeit wuchs die Liste der Macken und ebenso die Liste der Doctypes, die den Quirks-Modus aktivierten. Bei meiner letzten Zählung gab es 5 Doctypes, die den Almost Standards-Modus aktivierten, und 73, die den Quirks-Modus aktivierten. Aber dabei habe ich sicherlich einige vergessen – von dem, was der Internet Explorer 8 macht, um zwischen seinen vier – Sie haben richtig gehört, vier – Rendering-Modi umzuschalten, ganz zu schweigen. Unter *http://hsivonen.iki.fi/doctype/ie8-mode.png* finden Sie ein Flussdiagramm. Vernichten Sie es. Überantworten Sie es dem Feuer.

Also ... Wo waren wir noch? Stimmt, beim Doctype:

```
<!DOCTYPE html
          PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Das ist einer der 15 Doctypes, die in allen modernen Browsern den Standards-Modus aktivieren. Er ist vollkommen in Ordnung. Wenn Sie möchten, können Sie ihn lassen, wie er ist. Oder ändern Sie ihn in den HTML5-Doctype, der kürzer und süßer ist und ebenfalls in allen modernen Browsern den Standards-Modus aktiviert.

Das ist der HTML5-Doctype:

```
<!DOCTYPE html>
```

Das ist alles. Nur 15 Zeichen. Das ist so einfach, dass man es mit der Hand tippen kann, ohne es zu verbocken.

Professor Markup sagt

Die Doctype-Deklaration muss die erste Zeile Ihrer HTML-Datei sein. Steht irgendetwas davor – *selbst eine einzige leere Zeile* –, behandeln manche Browser sie so, als enthielte sie überhaupt keine Doctype-Deklaration. Gibt es sie nicht, stellt der Browser Ihre Seite im Quirks-Modus dar. Dieser Fehler kann sehr schwer zu entdecken sein. Zusätzlicher Withespace spielt in HTML üblicherweise keine Rolle. Man neigt also dazu, ihn einfach zu übersehen. Aber an dieser Stelle ist er äußerst wichtig!

Das Wurzelelement

Eine HTML-Seite ist eine Folge geschachtelter Elemente. Die vollständige Struktur der Seite ist wie ein Baum. Einige Elemente sind »Geschwister«, wie Zweige, die aus dem gleichen Baumstamm wachsen. Einige Elemente sind »Kinder« anderer Elemente, vergleichbar mit einem dünneren Zweig, der aus einem dickeren Zweig wächst. (Umgekehrt funktioniert das auch: Ein Element, das andere Elemente enthält, wird als »Elternknoten« seines unmittelbaren Kindelements und als »Vorfahr« seiner Enkel bezeichnet.) Elemente, die keine Kinder haben, bezeichnet man als »Blattknoten«. Das äußerste Element, das Vorfahr aller anderen Elemente auf der Seite ist, bezeichnet man als das »Wurzelelement« oder den »Wurzelknoten«. Das Wurzelelement einer HTML-Seite ist immer `<html>`.

In unserer Beispielseite (*http://diveintohtml5.org/examples/blog-original.html*) sieht das Wurzelelement so aus:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      lang="en"
      xml:lang="en">
```

Dieses Markup ist vollständig in Ordnung. Wieder können Sie es lassen, wie es ist – wenn es Ihnen gefällt. Es ist gültiges HTML5. Aber Teile davon sind in HTML5 nicht mehr erforderlich. Sie können also ein paar Bytes sparen, indem Sie sie entfernen.

Das Erste, was wir besprechen müssen, ist das `xmlns`-Attribut. Es ist ein Überbleibsel aus XHTML 1.0. Es sagt, dass sich die Elemente in dieser Seite im XHTML-Namensraum befinden, `http://www.w3.org/1999/xhtml`. Aber Elemente in HTML5 sind immer in diesem

Namensraum. Sie müssen ihn also nicht mehr explizit deklarieren. Ihre HTML5-Seite funktioniert in allen modernen Browsern, ob dieses Attribut vorhanden ist oder nicht.

Lassen wir das xmlns-Attribut weg, bleibt uns folgendes Wurzelelement:

```
<html lang="en" xml:lang="en">
```

Die Attribute lang und xml:lang definieren beide die Sprache dieser HTML-Seite. en steht für »Englisch«.[1] Warum aber zwei Attribute für die gleiche Sache? Auch das ist ein XHTML-Erbe. In HTML5 hat nur das lang-Attribut Auswirkungen. Sie können das xml:lang-Attribut stehen lassen, wenn Sie wollen. Doch wenn Sie das tun, müssen Sie sicherstellen, dass es den gleichen Wert enthält wie das lang-Attribut:

> Zur Erleichterung des Umstiegs von und nach XHTML können Autoren ein Attribut in keinem Namensraum, ohne Präfix und mit dem literalen lokalen Namen xml:lang auf HTML-Elementen in HTML-Dokumenten definieren. Ein solches Attribut darf allerdings nur definiert sein, wenn ebenfalls ein lang-Attribut ohne Namensraum angegeben wird. Beide Argumente müssen den gleichen Wert enthalten, wenn sie ohne Berücksichtigung von Groß-/Kleinschreibung im ASCII-Bereich verglichen werden. Das namensraum- und präfixlose Attribut mit dem literalen lokalen Namen xml:lang hat keine Auswirkungen auf die Sprachverarbeitung.

Sind Sie bereit, es fallen zu lassen? Das ist in Ordnung, lassen Sie es langsam aus Ihrem Leben verschwinden ... und weg! Wir verbleiben bei diesem Wurzelelement:

```
<html lang="en">
```

Und das ist alles, was ich dazu zu sagen habe.

Das <head>-Element

Das erste Kind des Wurzelelements ist üblicherweise das <head>-Element. Das <head>-Element enthält Metadateninformationen über die Seite, nicht den Inhalt der Seite selbst. (Der steht im darauf folgenden <body>-Element.) Das <head>-Element selbst ist ziemlich langweilig und hat sich in HTML5 kaum auf interessante Weise geändert. Das Gute ist das, was im <head>-Element steht. Und dazu wenden wir uns wieder unserer Beispielseite zu:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>My Weblog</title>
  <link rel="stylesheet" type="text/css" href="style-original.css" />
  <link rel="alternate" type="application/atom+xml"
                       title="My Weblog feed"
                       href="/feed/" />
  <link rel="search" type="application/opensearchdescription+xml"
                       title="My Weblog search"
                       href="opensearch.xml"  />
  <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

Erster Schritt: das <meta>-Element.

1 Sie schreiben gar nicht in Englisch? Weitere Sprachcodes finden Sie unter *http://www.w3.org/International/questions/qa-choosing-language-tags*.

Zeichenkodierung

Wenn Sie an »Text« denken, denken Sie wahrscheinlich »Zeichen und Symbole, die ich auf meinem Bildschirm sehe«. Aber Computer befassen sich nicht mit Zeichen und Symbolen. Computer befassen sich mit Bits und Bytes. Jedes bisschen Text, das Ihnen je auf einem Bildschirm vor Augen trat, wird eigentlich in einer bestimmten *Zeichenkodierung* gespeichert. Es gibt unzählige unterschiedliche Zeichenkodierungen. Einige von ihnen sind für bestimmte Sprachen wie Russisch oder Chinesisch oder Englisch gedacht, andere können für viele Sprachen verwendet werden. Grob formuliert, könnte man sagen, dass die Zeichenkodierung die Zuordnung zwischen dem ist, was Sie auf dem Bildschirm sehen, und dem, was Ihr Computer im Speicher und auf der Festplatte speichert.

In Wirklichkeit ist das natürlich etwas komplizierter. Viele Zeichen tauchen in verschiedenen Kodierungen auf, aber jede dieser Kodierungen kann eine andere Folge von Bytes nutzen, um diese Zeichen tatsächlich im Speicher oder auf der Festplatte zu speichern. Sie können sich eine Zeichenkodierung also als eine Art Entschlüsselungsmechanismus für Text vorstellen. Gibt Ihnen jemand eine Folge von Bytes und behauptet, es sei »Text«, müssen Sie wissen, welche Zeichenkodierung genutzt wurde, damit Sie die Bytes wieder in Zeichen umrechnen und anzeigen (oder irgendwie verarbeiten) können.

Wie also ermitteln Browser tatsächlich die Zeichenkodierung der Byte-Streams, die der Server sendet? Ich bin froh, dass Sie das fragen. Wenn Sie mit HTTP-Headern vertraut sind, haben Sie vielleicht schon einmal einen Header dieser Form gesehen:

```
Content-Type: text/html; charset="utf-8"
```

Kurz und knapp sagt diese Zeile, dass der Server denkt, er sende Ihnen ein HTML-Dokument, das die Zeichenkodierung `UTF-8` nutzt. Unglücklicherweise haben im großen Brei des World Wide Web nur äußerst wenige Autoren die Kontrolle über ihre HTTP-Server. Denken Sie an Blogger (*http://www.blogger.com*): Der Inhalt wird von den unterschiedlichsten Personen geschaffen, die Server werden von Google gesteuert. Deswegen bot HTML 4 eine Möglichkeit, die Zeichenkodierung im HTML-Dokument selbst anzugeben. Auch das haben Sie wahrscheinlich schon gesehen:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Dieses sagt, dass der Webautor meint, er habe ein HTML-Dokument unter Verwendung der Zeichenkodierung `UTF-8` geschrieben.

Beide Techniken funktionieren auch in HTML5 noch. Der HTTP-Header ist die bevorzugte Methode und überschreibt ein eventuell vorhandenes `<meta>`-Tag. Aber da nicht jeder HTTP-Header setzen kann, gibt es das `<meta>`-Tag immer noch. Und es ist in HTML5 sogar noch etwas einfacher geworden. Es sieht jetzt folgendermaßen aus:

```
<meta charset="utf-8" />
```

Das funktioniert in allen Browsern. Wie es zu dieser verkürzten Syntax gekommen ist? Hier ist die beste Erklärung, die ich finden konnte (*http://lists.w3.org/Archives/Public/public-html/2007Jul/0550.html*):

Der Grund für die <meta charset="">-Attributkombination ist, dass UAs sie bereits implementieren, weil viele dazu neigen, die Dinge folgendermaßen ohne Anführungszeichen anzugeben:

```
<META HTTP-EQUIV=Content-Type CONTENT=text/html; charset=ISO-8859-1>
```

Es gibt sogar einige <meta charset>-Testfälle (*http://simon.html5.org/test/html/parsing/encoding*), wenn Sie sich nicht vorstellen können, dass Browser das bereits tun.

Fragen an Professor Markup

F: Ich habe noch nie seltsame Zeichen verwendet. Muss ich trotzdem eine Zeichenkodierung angeben?

A: Ja! Sie sollten immer eine Zeichenkodierung für alle HTML-Seiten angeben, die Sie ausliefern. Geben Sie keine Kodierung an, kann das zu Sicherheitslücken führen (*http://code.google.com/p/doctype/wiki/ArticleUtf7*).

Zusammengefasst: Die Sache mit der Zeichenkodierung ist eine komplizierte Angelegenheit, die durch Jahrzehnte schlecht geschriebener Software, die von Autoren verwendet wird, deren bevorzugte Arbeitstechnik immer noch Copy-and-Paste ist, nicht gerade vereinfacht wird. Sie sollten bei jedem HTML-Dokument grundsätzlich eine Zeichenkodierung angeben, da es andernfalls zu üblen Folgen kommen kann. Sie können das über den HTTP-Content-Type-Header, die <meta http-equiv>-Deklaration oder die kürzere <meta charset>-Deklaration tun. Aber vergessen Sie sie nicht. Das Web wird es Ihnen danken.

Link-Relationen

Gewöhnliche Links (<a href>) zeigen einfach auf eine andere Seite. Link-Relationen sind ein Mittel, zu erläutern, warum Sie auf eine andere Seite verweisen. Sie beenden den Satz: Ich verweise auf diese andere Seite, weil ...

- ... sie ein Stylesheet mit CSS-Regeln ist, die Ihr Browser auf dieses Dokument anwenden soll.
- ... sie ein Feed ist, der den gleichen Inhalt wie diese Seite in einem standardkonformen, abonnierbaren Format enthält.
- ... sie eine Übersetzung dieser Seite in eine andere Sprache ist.
- ... sie das Gleiche beinhaltet wie diese Seite, aber im PDF-Format.
- ... weil sie das nächste Kapitel eines Onlinebuchs ist, zu dem auch diese Seite gehört.

Und so weiter. HTML5 teilt Link-Relationen in zwei Kategorien ein (*http://bit.ly/d2cbiR*):

> Zwei Kategorien von Links können mit dem Linkelement erstellt werden. **Links auf externe Ressourcen** sind Links auf Ressourcen, die genutzt werden, um das aktuelle Dokument zu bereichern, und **Hyperlink-Links** sind Links auf andere Dokumente. [...]
>
> Das genaue Verhalten von Links auf externe Ressourcen hängt von der spezifischen Beziehung ab, die für den entsprechenden Linktyp definiert ist.

Von den Beispielen, die ich Ihnen gegeben hatte, ist nur das erste (`rel="stylesheet"`) ein Link auf eine externe Ressource. Der Rest sind Hyperlinks auf andere Dokumente. Wenn Sie wollen, können Sie diesen Links folgen oder auch nicht. Aber sie sind nicht erforderlich, um die aktuelle Seite zu betrachten.

Link-Relationen tauchen am häufigsten in `<link>`-Elementen im `<head>`-Element einer Seite auf. Einige Link-Relationen können auch auf `<a>`-Elementen genutzt werden, aber selbst wenn das erlaubt ist, ist es nicht sehr üblich. HTML5 gestattet ebenfalls einige Relationen auf `<area>`-Elementen, aber das ist sogar *noch weniger* üblich. (HTML 4 gestattete keine `rel`-Attribute auf `<area>`-Elementen.) Anhand der vollständigen Liste der Link-Relationen (*http://bit.ly/a3nsqi*) können Sie prüfen, wo Sie bestimmte `rel`-Werte nutzen können.

Fragen an Professor Markup

F: Kann ich eigene Link-Relationen definieren?

A: Es scheint eine endlose Liste an neuen Ideen für Link-Relationen zu geben. Im Versuch, zu verhindern, dass einfach welche erfunden werden, unterhält die WHAT Working Group eine Registrierung der vorgeschlagenen `rel`-Werte (*http://wiki.whatwg.org/wiki/RelExtensions*) und definiert den Akzeptierungsprozess (*http://bit.ly/da3pse*).

rel = stylesheet

Schauen wir uns die erste Link-Relation in unserer Beispielseite an:

```
<link rel="stylesheet" href="style-original.css" type="text/css" />
```

Das ist die am häufigsten genutzte Link-Relation auf der Welt (buchstäblich). `<link rel="stylesheet">` dient dazu, auf CSS-Regeln zu verweisen, die in einer eigenen Datei gespeichert sind. Eine kleine Optimierung, die Sie in HTML5 vornehmen können, ist, das `type`-Attribut wegzulassen. Es gibt nur eine Stylesheet-Sprache für das Web, CSS, das ist deswegen der Standardwert für das `type`-Attribut:

```
<link rel="stylesheet" href="style-original.css" />
```

Das funktioniert in allen Browsern. (Vielleicht erfindet jemand irgendwann eine neuen Stylesheet-Sprache. Aber wenn das passiert, können Sie das `type`-Attribut einfach wieder einfügen.)

rel = alternate

Fahren wir mit unserer Beispielseite fort:

```
<link rel="alternate"
       type="application/atom+xml"
       title="My Weblog feed"
       href="/feed/" />
```

Auch diese Link-Relation ist recht häufig. `<link rel="alternate">` in Kombination mit dem RSS- oder Atom-Medientyp im `type`-Attribut aktiviert die sogenannte »automatische Feederkennung«. Sie ermöglicht Feedreadern wie dem Google Reader, zu erkennen, dass die Site einen Newsfeed mit den neuesten Artikeln hat. Die meisten Browser unterstützen die automatische Feederkennung zusätzlich damit, dass sie ein spezielles Symbol neben der URL anzeigen. (Anders als bei `rel="stylesheet"` ist das `type`-Attribut hier relevant. Lassen Sie es nicht weg!)

Die Link-Relation `rel="alternate"` war schon in HTML 4 ein seltsamer Zwitter von Anwendungsfällen. In HTML5 wurde ihre Definition klarer gemacht und erweitert, um bestehende Webinhalte präziser zu beschreiben. Wie Sie gerade sahen, zeigt die Verbindung von `rel="alternate"` und `type=application/atom+xml` einen Atom-Feed für die aktuelle Seite an. Aber Sie können `rel="alternate"` gemeinsam mit anderen `type`-Attributen nutzen, um Inhalte in anderen Formaten wie PDF anzuzeigen.

HTML5 legt auch eine weitere lang währende Verwirrung über Links auf Übersetzungen von Dokumenten bei. HTML 4 sagt, dass das `lang`-Attribut gemeinsam mit `rel="alternate"` angegeben werden muss, um die Sprache des verlinkten Dokuments anzugeben, aber das ist falsch. Die HTML 4-Errata-Liste führt unumwunden vier Fehler in der HTML 4-Spezifikation (mit anderen editorischen Kleinigkeiten) auf. Einer dieser Fehler betrifft die Angabe der Sprache eines mit `rel="alternate"` eingebundenen Dokuments. (Das richtige Verfahren, das im HTML 4-Errata-Dokument und jetzt in HTML5 beschrieben wird, ist die Verwendung des Attributs `hreflang`.) Unglücklicherweise wurden diese Errata nie rückwirkend in die HTML 4-Spezifikation eingebaut, weil niemand in der W3C HTML Working Group mehr an HTML arbeitete.

Andere Link-Relationen in HTML5

`rel="archives"`
: (*http://bit.ly/clzlyG*) »zeigt an, dass das angegebene Dokument eine Sammlung von Datensätzen, Dokumenten oder anderem Material von historischem Interesse ist. Die Indexseite eines Blogs könnte auf einen Index der vergangenen Blog-Einträge mit `rel="archives"` verweisen.«

`rel="author"`
: wird genutzt, um Informationen zum Autor der Seite anzugeben. Das kann eine `mailto`:-Adresse sein, muss es aber nicht. Es könnte auch einfach auf ein Kontaktformular oder eine »Über mich«-Seite verweisen.

`rel="external"`
: (*http://bit.ly/dBVO09*) »zeigt an, dass der Link auf ein Dokument zeigt, das nicht Teil der Site ist, zu der das aktuelle Dokument gehört.« Ich glaube, es wurde zuerst durch WordPress populär gemacht, das es für Links verwendet, die in Kommentaren hinterlassen werden.

`rel="start"`, `rel="prev"` *und* `rel="next"`
: (*http://www.w3.org/TR/html401/types.html#type-links*) werden genutzt, um Relationen zwischen Seiten zu definieren, die Teil einer Serie sind (wie die Kapitel eines Buchs oder auch die Einträge eines Blogs). Das Einzige davon, das je korrekt genutzt wurde, ist `rel="next"`. Man nutzte `rel="previous"` statt `rel="prev"`; man nutzte `rel="begin"` und `rel="first"` statt `rel="start"`; man nutzte `rel="end"` statt `rel="last"`. Und dann dachte man sich auch noch `rel="up"` aus, um auf eine »Elternseite« zu verweisen.

  HTML5 schließt `rel="first"` ein, das am häufigsten verwendet wurde, um auf die »erste Seite einer Reihe« zu verweisen (`rel="start"` ist ein nicht standardkonformes Synonym, das die Rückwärtskompatibilität gewährleisten soll). Es schließt wie HTML 4 `rel="prev"` und `rel="next"` ein (und unterstützt `rel="previous"` im Dienste der Rückwärtskompatibilität) und auch `rel="last"` (das Letzte in einer Reihe und passende Gegenstück zu `rel="first"`) und `rel="up"`.

  Am einfachsten machen Sie sich die Bedeutung von `rel="up"` klar, wenn Sie sich die Wegweiser in Ihrer Site- Navigation vor Augen führen (oder zumindest vorstellen). Ihre Homepage ist wahrscheinlich die erste Seite unter Ihren Wegweisern und die aktuelle Seite die letzte. `rel="up"` weist auf die vorletzte Seite unter Ihren Wegweisern.

`rel="icon"`
: (*http://bit.ly/diAJUP*) ist die zweitbeliebteste Link-Relation (*http://code.google.com/webstats/2005-12/linkrels.html*) nach `rel="stylesheet"`. Üblicherweise wird sie mit `shortcut` kombiniert, wie hier:

  ```
  <link rel="shortcut icon" href="/favicon.ico">
  ```

  Alle wichtigeren Browser unterstützen diese Verwendung, um ein kleines Symbol mit der Seite zu verbinden. Üblicherweise wird es in der Adressleiste des Browsers neben der URL angezeigt oder im Browser-Tab oder in beidem.

  Ebenfalls neu in HTML5: Das `sizes`-Attribut kann gemeinsam mit der `icon`-Relation genutzt werden, um die Größe des referenzierten Symbols anzuzeigen (*http://bit.ly/diAJUP*).

`rel="license"`
: (*http://bit.ly/9n9Xfv*) wurde von der Mikroformat-Gemeinschaft erfunden. Es »zeigt an, dass das referenzierte Dokument die Copyright-Lizenzbedingungen angibt, unter der das aktuelle Dokument bereitgestellt wird«.

`rel="nofollow"`
: (*http://bit.ly/cGjSPi*) »zeigt an, dass der Link vom ursprünglichen Autor oder Herausgeber der Seite nicht befürwortet wird oder dass der Link auf das referenzierte Dokument im Wesentlichen auf einer kommerziellen Beziehung zwischen den Urhebern der beiden Seiten beruht.« Sie wurde von Google erfunden und in der Mikroformat-Gemeinschaft standardisiert. Die Überlegung war, dass Spammer es ufgeben würden, Spam-Kommentare in Blogs zu hinterlassen, wenn `nofollow`-Links keinen

Einfluss auf das Seiten-Ranking haben. Das geschah nicht. Aber `rel="nofollow"` blieb. Viele beliebte Blogging-Systeme hängen an Links in Kommentaren standardmäßig `rel="nofollow"` an.

`rel="noreferrer"`
: (*http://bit.ly/cQMSJg*) »zeigt an, dass keine Referrer-Daten weitergegeben werden sollen, wenn dem Link gefolgt wird.« Das wird von keinem der aktuell veröffentlichten Browser unterstützt, wurde aber in aktuelle WebKit-Nightlies eingebaut und wird irgendwann in Safari, Google Chrome und anderen WebKit-basierten Browsern auftauchen. Einen `rel="noreferrer"`-Testfall finden Sie unter *http://wearehugh.com/public/2009/04/rel-noreferrer.html*.

`rel="pingback"`
: (*http://bit.ly/cIAGXB*) gibt die Adresse eines »Pingback-Servers« an. Wie in der Pingback-Spezifikation (*http://hixie.ch/specs/pingback/pingback-1.0*) steht: »Das Pingback-System ist ein Mittel, Blogs automatisch zu benachrichtigen, wenn andere Websites auf es verlinken [...] Es ermöglicht umgekehrtes Linking – eine Möglichkeit, in einer Kette von Links wieder nach oben zu steigen, statt einfach in ihr abzusteigen.« Blogging-Systeme, insbesondere WordPress, implementieren den Pingback-Mechanismus, um Autoren zu benachrichtigen, dass Sie auf sie verlinkt haben, als Sie einen neuen Blog-Eintrag erstellten.

`rel="prefetch"`
: (*http://bit.ly/9o0nMS*) »zeigt an, dass es vorteilhaft sein kann, die angegebene Ressource vorab abzurufen und zu cachen, da es äußerst wahrscheinlich ist, dass der Benutzer sie benötigt.« Suchmaschinen fügen Suchergebnissen `<link rel="prefetch" href="<emphasis>URL OF TOP SEARCH RESULT</emphasis>">` hinzu, wenn sie meinen, dass das Topergebnis erheblich populärer ist als alle anderen. Ein Beispiel: Nehmen Sie Firefox, suchen Sie mit Google nach CNN, schauen Sie sich den Quelltext der Seite an und suchen Sie nach dem Schlüsselwort `prefetch`. Mozilla Firefox ist aktuell der einzige Browser, der `rel="prefetch"` unterstützt.

`rel="search"`
: (*http://bit.ly/aApkaP*) »zeigt an, dass das referenzierte Dokument eine Schnittstelle bietet, die speziell dem Durchsuchen des Dokuments und verwandter Ressourcen dient.« Genauer gesagt, wenn `rel="search"`etwas Nützliches tun soll, sollte es auf ein OpenSearch-Dokument verweisen, das beschreibt, wie ein Browser eine URL aufbauen muss, um die aktuelle Site nach einem bestimmten Suchwort zu durchsuchen. OpenSearch (und `rel="search"`-Links, die auf OpenSearch-Beschreibungsdokumente verweisen) wird von Microsoft Internet Explorer seit Version 7 und von Mozilla Firefox seit Version 2 unterstützt.

`rel="sidebar"`
: (*http://bit.ly/azTA9D*) »zeigt an, dass das referenzierte Dokument, wenn es abgerufen wird, in einem sekundären Browsing-Kontext angezeigt werden soll (wenn möglich) statt im aktuellen Browsing-Kontext.« Was bedeutet das? In Opera und Mozilla Firefox bedeutet es: »Wenn ich auf diesen Link klicke, werde ich aufgefordert, ein

Lesezeichen anzulegen, das, wenn es im Lesezeichen-Menü angewählt wird, das verlinkte Dokument in einer Browser-Sidebar anzeigt.« (Opera nennt es »Panel« statt »Sidebar«.) Internet Explorer, Safari und Chrome ignorieren `rel="sidebar"` und behandeln es als gewöhnlichen Link. Einen `rel="sidebar"`-Testfall finden Sie unter *http://wearehugh.com/public/2009/04/rel-sidebar.html*.

`rel="tag"`
: (*http://bit.ly/9bYlfa*) »zeigt an, dass das Tag, das das referenzierte Dokument repräsentiert, für das aktuelle Dokument gilt.« Die Auszeichnung von »Tags« (Kategorieschlüsselwörtern) mit dem `rel`-Attribut wurde von Technorati erfunden, um die Kategorisierung von Blog-Einträgen zu vereinfachen. Frühe Blogs und Tutorials bezeichneten sie deswegen als »Technorati-Tags«. (Das haben Sie richtig verstanden: Ein kommerzielles Unternehmen hat die gesamte Welt davon überzeugt, Metadaten zu verwenden, die ihm das Leben erleichtern. Gute Arbeit, wenn man so etwas schafft!) Die Syntax wurde später in der Mikroformat-Gemeinschaft standardisiert, wo sie einfach zu `rel="tag"` wurde. Die meisten Blog-Systeme, die es ermöglichen, Kategorien, Schlüsselwörter oder Tags mit individuellen Einträgen zu versehen, zeichnen sie mit `rel="tag"`-Links aus. Browser machen nichts Spezielles damit; eigentlich sollen sie Suchmaschinen signalisieren, worum es in der Seite geht.

Neue semantische Elemente in HTML5

HTML5 dreht sich nicht nur darum, bestehendes Markup zu verkürzen (obwohl es in dieser Beziehung einiges leistet). Es definiert außerdem einige neue semantische Elemente. Das sind die Elemente, die die HTML5-Spezifikation definiert:

`<section>`
: Das `section`-Element repräsentiert einen allgemeinen Abschnitt in einem Dokument oder einer Anwendung. Ein Abschnitt ist in diesem Kontext eine thematische Gruppierung von Inhalten, die üblicherweise unter einer Überschrift stehen. Beispiele für Abschnitte wären Kapitel, die verschiedenen Tabs in einem Dialog mit Tabs oder die nummerierten Abschnitte einer wissenschaftlichen Arbeit. Die Homepage einer Website könnte mehrere Abschnitte für eine Einführung, die eigentlichen Nachrichten sowie die Kontaktdaten enthalten.

`<nav>`
: Das `nav`-Element repräsentiert einen Abschnitt einer Seite, der auf andere Seiten oder Teile in der Seite verlinkt: ein Abschnitt mit Navigationslinks. Nicht alle Linkgruppen auf einer Seite müssen in einem `nav`-Element stehen – nur die Abschnitte, die aus wichtigen Navigationsblöcken bestehen, sind für das `nav`-Element gedacht. Die häufig anzutreffenden kurzen Linklisten in den Fußleisten von Webseiten, die auf verschiedene Seiten innerhalb der Site verweisen, wie Geschäftsbedingungen, Homepage oder Copyright, zählen beispielsweise nicht dazu. Das `footer`-Element allein, ohne eingebettetes `nav`-Element, reicht in solchen Fällen aus.

`<article>`
: Das `article`-Element repräsentiert eine abgeschlossene Einheit in einem Dokument, einer Anwendung oder einer Site, die unabhängig verbreitet oder wiederverwendet werden kann, z.B. in RSS-Feeds. Es könnte beispielsweise ein Forenbeitrag, ein Zeitschriften- oder Zeitungsartikel, ein Blog-Eintrag, ein Benutzerkommentar, ein interaktives Widget oder Gadget oder ein Element mit unabhängigem Inhalt enthalten.

`<aside>`	Das `aside`-Element repräsentiert einen Abschnitt einer Seite, der Inhalte enthält, die sich zwar auf den das `aside`-Element umgebenden eigentlichen Inhalt der Seite beziehen, aber als von ihm unabhängig betrachtet werden können. In Druckwerken werden derartige Abschnitte häufig als Seitenleisten dargestellt. Das Element kann für typografische Effekte wie herausgehobene Zitate oder Seitenleisten, für Werbung, für Gruppen von `nav`-Elementen und andere Inhalte verwendet werden, die als vom eigentlichen Inhalt der Seite getrennt betrachtet werden können.
`<hgroup>`	Das `hgroup`-Element repräsentiert die Überschrift eines Abschnitts. Das Element wird genutzt, um einen Satz von `h1`–`h6`-Elementen zu gruppieren, wenn die Überschrift mehrere Ebenen wie Untertitel, alternative Titel oder Schlagzeigen enthält.
`<header>`	Das `header`-Element repräsentiert eine Gruppe von Einführungselementen oder Navigationshilfen. Ein `header`-Element soll üblicherweise die Überschrift eines Abschnitts (ein `h1`–`h6`-Element oder ein `hgroup`-Element) enthalten, aber erforderlich ist das nicht. Das `header`-Element kann auch verwendet werden, um das Inhaltsverzeichnis eines Abschnitts, ein Suchformular oder eventuelle Logos einzuschließen.
`<footer>`	Das `footer`-Element repräsentiert die Fußleiste für das unmittelbar vorausgehende abschnittsartige Element oder abschnittsartige Wurzelelement. Eine Fußleiste enthält üblicherweise Informationen zu einem Abschnitt, beispielsweise wer ihn geschrieben hat, verweist auf verwandte Dokumente, bietet Copyright-Informationen und Ähnliches. Fußleisten müssen nicht notwendigerweise am Ende eines Abschnitts erscheinen, tun das aber meist. Wenn das `footer`-Element vollständige Abschnitte enthält, repräsentieren diese Anhänge, Indizes, lange Kolophone, umfangreiche Lizenzbedingungen und andere derartige Inhalte.
`<time>`	Das `time`-Element repräsentiert eine Zeit auf einer 24-Stunden-Uhr oder ein genaues Datum auf dem proleptischen gregorianischen Kalender, optional mit einer Zeit und einer Zeitzonenverschiebung.
`<mark>`	Das `mark`-Element repräsentiert einen Textverlauf in einem Dokument, der zu Verweiszwecken markiert oder vorgehoben ist.

Ich weiß, Sie hungern danach, gleich mit der Verwendung dieser neuen Elemente loszulegen. Sonst würden Sie dieses Kapitel sicher nicht lesen. Aber erst müssen wir einen kleinen Umweg nehmen.

Wie Browser mit unbekannten Elementen umgehen

Jeder Browser besitzt eine Stammliste der HTML-Elemente, die er unterstützt. Bei Mozilla Firefox ist diese beispielsweise in *nsElementTable.cpp* (*http://mxr.mozilla.org/seamonkey/source/parser/htmlparser/src/nsElementTable.cpp*) gespeichert. Elemente, die sich nicht in dieser Liste befinden, werden als »unbekannte Elemente« behandelt. Es gibt zwei grundlegende Fragen in Bezug auf unbekannte Elemente:

Wie soll das Element dargestellt werden?
: Standardmäßig erhält `<p>` oberhalb und unterhalb Freiraum, `<blockquote>` wird mit einem linken Außenabstand eingerückt, und `<h1>` wird in einer größeren Schrift dargestellt.

Wie sollte das DOM des Elements aussehen?
: Mozillas *nsElementTable.cpp* enthält Informationen dazu, welche Arten anderer Elemente die einzelnen Elemente enthalten können. Wenn Sie Markup wie dieses einschließen, `<p><p>`, schließt das zweite Paragraf-Element implizit das erste. Die Elemente werden also zu Geschwistern, nicht zu Eltern und Kindern. Aber wenn Sie `<p><span>`

schreiben, schließt das span den Absatz nicht, weil Firefox weiß, dass <p> ein Blockelement ist, das das Inline-Element enthalten kann. Das wird im DOM also zu einem Kind des <p>-Elements.

Die verschiedenen Browser beantworten diese Fragen auf unterschiedliche Weise. (Erschreckend, ich weiß.) Unter den wichtigsten Browsern beantwortet Microsofts Internet Explorer diese Frage am problematischsten.

Die erste Frage lässt sich recht leicht beantworten: Unbekannte Elemente erhalten keine spezielle Darstellung. Sie erben einfach die CSS-Eigenschaften, die an der Stelle in Kraft sind, an der sie in der Seite erscheinen. Es bleibt dem Seitenautor überlassen, alle Darstellungsstyles mit CSS anzugeben. Unglücklicherweise erlaubt der Internet Explorer (vor Version 9) kein Styling für unbekannte Elemente. Nehmen wir an, Sie hätten eine Seite mit folgendem Markup:

```
<style type="text/css">
  article { display: block; border: 1px solid red }
</style>
...
<article>
<h1>Willkommen bei Initech</h1>
<p>Das ist Ihr <span>erster Tag</span>.</p>
</article>
```

Der Internet Explorer (bis einschließlich IE8) würde diesen Artikel nicht mit einem roten Rand darstellen. Während ich dies schrieb, befand der Internet Explorer 9 immer noch im Betastadium, aber Microsoft hat gesagt (und Entwickler haben das bestätigt), dass der Internet Explorer 9 dieses Problem nicht mehr beinhalten wird.

Das zweite Problem ist das DOM, das der Browser erstellt, wenn er ein unbekanntes Element antrifft. Auch hier weist der Internet Explorer die größten Probleme auf. Wenn der IE einen Elementnamen nicht explizit erkennt, fügt er das Element *als leeren Knoten ohne Kinder* in das DOM ein. Alle Elemente, die Sie als unmittelbare Kinder des unbekannten Elements erwarten würden, werden tatsächlich also als seine Geschwister eingefügt.

Hier ist etwas ASCII-Kunst, die den Unterschied vorführt. Das ist das DOM, das HTML5 vorschreibt:

```
article
|
+--h1 (Kind von article)
|  |
|  +--Textknoten "Willkommen bei Initech"
|
+--p (Kind von article, Geschwisterknoten von h1)
   |
   +--Textknoten "Das ist Ihr "
   |
   +--span
   |  |
   |  +--Textknoten "erster Tag"
   |
   +--Textknoten "."
```

Aber das ist das DOM, das der Internet Explorer tatsächlich erstellt:

```
article (keine Kinder)
h1 (Geschwisterknoten von article)
|
+--Textknoten "Willkommen bei Initech"
p (Geschwisterknoten von h1)
|
+--Textknoten "Das ist Ihr "
|
+--span
|  |
|  +--Textknoten "erster Tag"
|
+--Textknoten "."
```

Es gibt einen wunderlichen Workaround für dieses Problem. Wenn Sie ein leeres <article>-Element mit JavaScript erstellen, bevor Sie es in der Seite verwenden, erkennt der Internet Explorer auf magische Weise das <article>-Element und ermöglicht Ihnen, es mit CSS zu stylen. Sie müssen das Scheinelement nie in das DOM einfügen. Sie müssen das Element nur einmal pro Seite erstellen, um dem IE beizubringen, das Styling für ein Element zuzulassen, das er nicht kennt, zum Beispiel so:

```
<html>
<head>
<style>
  article { display: block; border: 1px solid red }
</style>
<script>document.createElement("article");</script>
</head>
<body>
<article>
<h1>Willkommen bei Initech</h1>
<p>Das ist Ihr <span>erster Tag</span>.</p>
</article>
</body>
</html>
```

Das funktioniert in allen Versionen des Internet Explorer bis zurück zum IE6! Wir können diese Technik nutzen, um Scheinelemente für alle neuen HTML5-Elemente auf einmal zu erstellen – wieder ohne sie in das DOM einzufügen. Sie werden diese Scheinelemente nie sehen und können sie dann einfach verwenden, ohne sich über Nicht-HTML5-fähige Browser Gedanken machen zu müssen.

Remy Sharp hat genau das mit seinem »HTML5-Aktivierungsskript« (*http://remysharp.com/2009/01/07/html5-enabling-script*) gemacht. Das Skript hat schon mehrere Überarbeitungen erfahren, aber hier ist der grundlegende Gedanke:

```
<!--[if lt IE 9]>
<script>
  var e = ("abbr,article,aside,audio,canvas,datalist,details," +
    "figure,footer,header,hgroup,mark,menu,meter,nav,output," +
    "progress,section,time,video").split(',');
```

```
  for (var i = 0; i < e.length; i++) {
    document.createElement(e[i]);
  }
</script>
<![endif]-->
```

Die `<!--[if lt IE 9]>`- und `<![endif]-->`-Teile sind bedingte Kommentare. Der Internet Explorer interpretiert sie als `if`-Anweisung: »Ist der aktuelle Browser eine Version des Internet Explorers vor Version 9 führe diesen Block aus.« Alle anderen Browser behandeln den gesamten Block als HTML-Kommentar. Die Folge ist, dass der Internet Explorer (bis einschließlich Version 8) das Skript ausführt, andere Browser es aber vollständig ignorieren. Das sorgt dafür, dass die Seite in Browsern, die diesen Hack nicht benötigen, schneller geladen wird.

Der JavaScript-Code selbst ist recht gradlinig. Die Variable `e` wird zu einem Array mit Strings wie `"abbr"`, `"article"`, `"aside"` und so weiter. Dann durchlaufen wir dieses Array und erstellen jedes der angeführten Elemente, indem `document.createElement()` aufgerufen wird. Aber da wir den Rückgabewert ignorieren, werden die Elemente nie in das DOM eingefügt. Es reicht jedoch, um den Internet Explorer dazu zu bringen, diese Elemente so zu behandeln, wie wir es wollen, wenn wir sie später in der Seite verwenden.

Dieser »später«-Aspekt ist wichtig. Das Skript muss am Anfang Ihrer Seite stehen – vorzugsweise im `<head>`-Element –, nicht am Ende. Das sorgt dafür, dass der Internet Explorer das Skript ausführt, *bevor* er Ihre Tags und Attribute parst. Wenn Sie dieses Skript am Ende Ihrer Seite angeben, ist es zu spät. Der Internet Explorer hat Ihr Markup bereits falsch interpretiert und das falsche DOM aufgebaut, und er wird das wegen Ihres Skripts nicht rückgängig machen.

Remy Sharp hat dieses Skript »geschrumpft« und stellt es auf Google Project Hosting (*http://code.google.com/p/html5shiv/*) bereit. (Sollte Ihnen diese Frage auf der Zunge liegen: Das Skript ist Open Source und steht unter der MIT-Lizenz. Sie können es also in Ihrem Projekt verwenden.) Wenn Sie möchten, können Sie das Skript sogar einbinden, indem Sie unmittelbar auf die gehostete Version verlinken. So beispielsweise:

```
<head>
  <meta charset="utf-8" />
  <title>My Weblog</title>
  <!--[if lt IE 9]>
  <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
  <![endif]-->
</head>
```

Jetzt sind wir bereit, die neuen semantischen Elemente in HTML5 auch zu benutzen.

Kopfleisten und Überschriften

Kehren wir zu unserer Beispielseite zurück und schauen wir uns zunächst die Überschriften an:

```
<div id="header">
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
</div>

...

<div class="entry">
  <h2>Travel day</h2>
</div>

...

<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

Dieses Markup ist vollständig in Ordnung. Wenn es Ihnen gefällt, können Sie es lassen, wie es ist. Es ist gültiges HTML5. Aber HTML5 bietet einige zusätzliche semantische Elemente für Überschriften und Abschnitte.

Befreien wir uns zunächst von diesem `<div id="header">`. Das ist ein sehr verbreitetes Muster. Das `div`-Element hat keine eigene Semantik, ebenso das `id`-Attribut. (Clients dürfen aus dem Wert des `id`-Attributs keine Bedeutung ableiten.) Sie könnten stattdessen `<div id="shazbot">` verwenden, und es hätte den gleichen semantischen Wert, also gar keinen.

HTML5 definiert für diesen Zweck ein `<header>`-Element. Die HTML5-Spezifikation bietet eine Reihe lebensnaher Beispiele für die Verwendung des `<header>`-Elements. So würde das bei unserer Beispielseite aussehen:

```
<header>
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
  ...
</header>
```

Das ist gut. Es sagt allen, die es wissen wollen, dass das eine Kopfleiste (ein Header) ist. Aber was ist mit diesem Slogan? Ein weiteres verbreitetes Muster, für das es bislang kein Standard-Markup gab. So etwas ist schwer auszuzeichnen. Ein Slogan ist so etwas wie ein Untertitel, der mit der eigentlichen Überschrift verknüpft ist. Es ist also eine Zwischenüberschrift, die keinen eigenen Abschnitt öffnet.

Überschriftenelemente wie `<h1>` und `<h2>` geben Ihrer Seite eine Struktur. Gemeinsam bilden sie eine Gliederung, die Sie nutzen können, um sich die Seite vorzustellen (oder durch die Seite zu navigieren). Bildschirmleser nutzen Dokumentgliederungen, um blinden Nutzern die Navigation der Seite zu erleichtern. Es gibt Onlinewerkzeuge

(*http://gsnedders.html5.org/outliner/*) und Browsererweiterungen (*http://chrispederick.com/work/web-developer/*), mit deren Hilfe Sie die Gliederung Ihres Dokuments visualisieren können.

In HTML 4 waren die Elemente `<h1>`–`<h6>` die einzige Möglichkeit, eine Dokumentgliederung zu erstellen. Die Gliederung unserer Beispielseite sieht so aus:

```
My Weblog (h1)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

Das ist in Ordnung, bedeutet aber, dass es keine Möglichkeit gibt, den Slogan »A lot of effort went into making this effortless.« auszuzeichnen. Versuchen wir, ihn als `<h2>`-Element zu markieren, würden wir damit einen Phantomknoten in die Dokumentgliederung einfügen:

```
My Weblog (h1)
|
+--A lot of effort went into making this effortless. (h2)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

Das entspricht jedoch nicht der Struktur des Dokuments. Der Slogan entspricht keinem Abschnitt. Er ist lediglich ein Untertitel.

Vielleicht könnten wir den Slogan als `<h2>` auszeichnen und alle Artikeltitel als `<h3>`? Aber das ist sogar noch schlimmer:

```
My Weblog (h1)
|
+--A lot of effort went into making this effortless. (h2)
   |
   +--Travel day (h3)
   |
   +--I'm going to Prague! (h3)
```

Der Phantomknoten befindet sich immer noch in unserer Dokumentgliederung, hat aber Kinder »gestohlen«, die eigentlich unter den Wurzelknoten gehören. Und hier liegt das Problem: HTML 4 bietet keine Möglichkeit, einen Untertitel auszuzeichnen, ohne ihn der Dokumentgliederung hinzuzufügen. Ganz gleich, wie wir die Dinge herumschieben, »A lot of effort went into making this effortless« landet in unserer Gliederung. Und deswegen landeten wir bei semantisch bedeutungslosem Markup wie `<p class="tagline">`.

HTML5 bietet jetzt eine Lösung dafür: das `<hgroup>`-Element. Das `<hgroup>`-Element fungiert als Verpackung für zwei oder mehr aufeinander bezogene Überschriftenelemente. Was »aufeinander bezogen« heißt? Es bedeutet, dass sie gemeinsam einen einzigen Knoten in der Dokumentgliederung bilden.

In HTML5 nutzen Sie folgendes Markup:

```
<header>
  <hgroup>
    <h1>My Weblog</h1>
    <h2>A lot of effort went into making this effortless.</h2>
  </hgroup>
  ...
</header>

...

<div class="entry">
  <h2>Travel day</h2>
</div>

...

<div class="entry">
  <h2>I'm going to Prague!</h2>
</div>
```

So sieht die Dokumentgliederung aus, die erstellt wird:

```
My Weblog (h1 of its hgroup)
|
+--Travel day (h2)
|
+--I'm going to Prague! (h2)
```

Sie können Ihre eigenen Seiten im HTML5 Outliner testen, um zu sichern, dass Sie die Überschriftenelemente ordentlich verwenden.

Artikel

Fahren wir mit unserer Beispielseite fort und schauen wir uns an, was wir mit diesem Markup machen können:

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="link to this post">
       Travel day
    </a>
  </h2>
  ...
</div>
```

Auch das ist gültiges HTML5. Aber HTML5 bietet ein spezifischeres Element für den häufigen Fall der Auszeichnung eines Artikels auf einer Seite – das entsprechend benannte `<article>`-Element:

```
<article>
  <p class="post-date">October 22, 2009</p>
  <h2>
    <a href="#"
       rel="bookmark"
       title="link to this post">
       Travel day
    </a>
  </h2>
  ...
</article>
```

Aber ganz so einfach ist das nicht. Sie sollten noch eine weitere Änderung vornehmen. Ich werde es Ihnen erst zeigen und dann erläutern:

```
<article>
  <header>
    <p class="post-date">October 22, 2009</p>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  ...
</article>
```

Haben Sie das verstanden? Ich habe das <h2>-Element in ein <h1> umgewandelt und in ein <header>-Element gepackt. Sie haben bereits gesehen, wie das <header>-Element verwendet wird. Es soll alle Elemente zusammenfassen, die die Überschrift des Artikels bilden (in diesem Fall sind das das Veröffentlichungsdatum und der Titel). Aber ... sollte man nicht nur ein <h1> pro Dokument haben? Beschädigt das nicht die Dokumentstruktur? Nein. Um das zu verstehen, müssen wir einen Schritt zurückgehen.

In HTML 4 konnte man die Dokumentgliederung nur mit den Elementen <h1>–<h6> gestalten. Sollte Ihre Gliederung lediglich einen Wurzelknoten enthalten, mussten Sie sich in Ihrem Markup auf ein einziges <h1>-Element beschränken. Aber die HTML5-Spezifikation definiert einen Algorithmus zur Erstellung einer Dokumentgliederung, die die neuen semantischen Elemente von HTML5 einschließt. Der HTML5-Algorithmus sagt, dass ein <article>-Element einen neuen Abschnitt erzeugt, d.h. einen neuen Knoten in der Dokumentgliederung. Und in HTML5 kann jeder Abschnitt sein eigenes <h1>-Element haben.

Das ist eine erhebliche Änderung im Vergleich zu HTML 4, und hier sind die Gründe dafür, dass das so gut ist. Viele Webseiten werden eigentlich auf Basis von Schablonen erstellt. Etwas Inhalt wird von dieser Quelle bezogen und oben in die Seite hineingepfropft. Etwas Inhalt wird von jener Quelle abgerufen und unten in die Seite hineingepfropft. Viele Einführungen sind auf vollkommen gleiche Weise strukturiert. »Hier ist etwas HTML-Markup. Kopieren Sie es und fügen Sie es in Ihre Seite ein.« Bei kleinen Datenmengen ist das vollkommen ausreichend, aber was ist, wenn das Markup, das so

eingefügt wird, einen ganzen Abschnitt umfasst? Dann würde jene Einführung Folgendes sagen: »Hier ist etwas HTML-Markup. Kopieren Sie es, fügen Sie es in einem Texteditor in ein Dokument ein und passen Sie die Tags für die Überschriften so an, dass sie den Schachtelungsebenen der entsprechenden Überschriften-Tags in der Seite entsprechen, in die das Markup eingefügt wird.«

Formulieren wir die Sache anders. HTML 4 bietet kein *allgemeines* Überschriftenelement. Es bietet sechs streng nummerierte Überschriftenelemente, `<h1>`–`<h6>`, die genau in dieser Reihenfolge geschachtelt werden müssen. Das ist recht nervig, vor allem wenn Ihre Seite eher »zusammengestückelt« als »geschrieben« wird. Und das ist das Problem, das HTML5 mit den neuen abschnittsbildenden Elementen und den neuen Regeln für die vorhandenen Überschriftenelemente löst. Wenn Sie die neuen abschnittsbildenden Elemente nutzen, können Sie Markup wie dieses nutzen:

```
<article>
  <header>
    <h1>A syndicated post</h1>
  </header>
  <p>Lorem ipsum blah blah...</p>
</article>
```

Sie können es einfach kopieren und an völlig beliebiger Stelle in Ihre Seite einfügen, ohne daran Änderungen vornehmen zu müssen. Dass es ein `<h1>`-Element ist, ist kein Problem, weil das Ganze in ein `<article>`-Element eingeschlossen ist. Das `<article>`-Element definiert einen in sich abgeschlossenen Knoten in der Dokumentgliederung, das `<h1>`-Element bietet einen Titel für diesen Gliederungsknoten, und alle anderen abschnittsbildenden Elemente in der Seite bleiben genau auf der Schachtelungsebene, auf der sie sich auch vorher befanden.

Professor Markup sagt

Wie im Web üblich, ist die Realität wieder einmal komplizierter, als ich hier durchblicken lasse. Die neuen »explizit« abschnittsbildenden Elemente (wie `<h1>` in einem `<article>`) können auf unerwartete Weise mit den alten »implizit« abschnittsbildenden Elemente (`<h1>`–`<h6>` allein) kollidieren. Sie machen sich das Leben erheblich leichter, wenn Sie nur eines von beidem nutzen, nicht beides gleichzeitig. Müssen Sie in einer Seite beides nutzen müssen, sollten Sie sich das Ergebnis unbedingt im HTML5 Outliner anschauen, um zu prüfen, ob Ihre Dokumentgliederung vernünftig ist.

Datum und Uhrzeit

Aufregend, nicht wahr? Nicht »Nacht vor der Hochzeit«- oder »Abend vor Weihnachten«-aufregend natürlich, aber schon spannend, was semantisches Markup betrifft. Fahren wir mit unserer Beispielseite fort. Als Nächstes wollen wir uns folgende Zeile vornehmen:

```
<div class="entry">
  <p class="post-date">October 22, 2009</p>
  <h2>Travel day</h2>
</div>
```

Wieder die gleiche alte Geschichte? Ein verbreitetes Muster – das Veröffentlichungsdatum eines Artikels –, für das es kein semantisches Markup gibt. Also griffen Autoren auf generisches Markup zurück und nutzen selbst definierte `class`-Attribute. Auch das ist gültiges HTML5. Sie müssen es nicht ändern. Aber HTML5 bietet eine spezifische Lösung für diesen Fall – das `<time>`-Element:

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Das `<time>`-Element hat drei Bestandteile:

- einen maschinenlesbaren Zeitstempel,
- einen für Menschen lesbaren Textinhalt sowie
- ein optionales `pubdate`-Flag.

In diesem Beispiel gibt das `datetime`-Attribut nur ein Datum, keine Uhrzeit an. Das Format ist eine vierstellige Jahresangabe sowie zweistellige Monats- und Tagesangaben, jeweils getrennt durch Bindestriche:

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Wenn Sie auch eine Zeit einfügen wollen, können Sie nach dem Datum den Buchstaben `T`, auf ihn folgend die Zeit (24-Stunden-Format) und eine Zeitzonenverschiebung angeben:

```
<time datetime="2009-10-22T13:59:47-04:00" pubdate>
  October 22, 2009 1:59pm EDT
</time>
```

Das Format für die Datum/Zeit-Angabe ist ziemlich flexibel. Die HTML5-Spezifikation enthält eine Reihe von Beispielen für gültige Datum/Zeit-Strings.

Beachten Sie, dass ich den Textinhalt zwischen `<time>` und `</time>` so geändert habe, dass er dem Zeitstempel entspricht. Das ist nicht vorgeschrieben. Der Textinhalt kann beliebige Form haben, solange dass `datetime`-Attribut eine gültige Datum/Zeit-Angabe enthält. (Die zulässigen Formate für den `datetime`-Wert sind eingeschränkt, was im Tag steht, das bleibt natürlich Ihnen überlassen. Dass oben eine Zeitangabe in dem im angelsächsischen Raum üblichen Format steht, liegt daran, dass die Webseite selbst aus diesem Sprachraum kommt.) Auch das ist also gültiges HTML5:

```
<time datetime="2009-10-22">letzten Donnerstag</time>
```

Und das ebenfalls:

```
<time datetime="2009-10-22"></time>
```

Das letzte Puzzleteil ist das `pubdate`-Attribut. Es ist ein Boolesches Attribut. Sie müssen es also nur einfügen, wenn Sie es benötigen:

```
<time datetime="2009-10-22" pubdate>October 22, 2009</time>
```

Sollte Ihnen dieses »nackte« Attribut nicht gefallen, können Sie folgendes Äquivalent nutzen:

```
<time datetime="2009-10-22" pubdate="pubdate">October 22, 2009</time>
```

Was bedeutet dieses pubdate-Attribut? Zwei Dinge: Wenn das <time>-Element in einem <article>-Element steht, bedeutet es, dass das der Zeitstempel des Publikationsdatums des Artikels ist. Steht das <time>-Element nicht in einem <article>-Element, bedeutet es, dass der Zeitstempel das Publikationsdatum des vollständigen Dokuments ist.

Hier ist der vollständige Artikel, und zwar so umformuliert, dass er alle Vorteile von HTML5 nutzt:

```
<article>
  <header>
    <time datetime="2009-10-22" pubdate>
      October 22, 2009
    </time>
    <h1>
      <a href="#"
         rel="bookmark"
         title="link to this post">
         Travel day
      </a>
    </h1>
  </header>
  <p>Lorem ipsum dolor sit amet...</p>
</article>
```

Navigation

Einer der wichtigsten Teile jeder Website ist die Navigationsleiste. spiegel.de hat Tabs – Register – oben auf jeder Seite, die auf die verschiedenen Nachrichtenbereiche verweisen – »Politik«, »Wirtschaft«, »Sport« usw. Google-Suchergebnisse bieten oben eine ähnliche Leiste, über die Sie Ihre Suche auf den verschiedenen Google-Diensten durchführen können – »Bilder«, »Videos«, »Maps« usw. Und unsere Beispielseite hat eine Navigationsleiste in der Kopfleiste, die Links auf die verschiedenen Bereiche unserer hypothetischen Site enthält – »home«, »blog«, »gallery« und »about«.

So wurde diese Navigationsleiste ursprünglich ausgezeichnet:

```
<div id="nav">
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</div>
```

Auch das ist gültiges HTML5. Aber an dieser Liste aus vier Elementen ist nichts, das Ihnen mitteilt, dass sie Teil der Navigation der Site ist. Visuell können Sie das erraten, weil sie

Teil der Kopfleiste ist und der Text der Links darauf hindeutet. Aber semantisch gibt es nichts, was diese Liste von jeder anderen Liste in der Seite unterscheidet.

Wen die Semantik der Site-Navigation interessiert? Leute mit Behinderungen (*http://diveintoaccessibility.org*) beispielsweise. Warum das? Betrachten Sie folgendes Szenario: Sie können sich nur eingeschränkt bewegen. Eine Maus zu verwenden, fällt Ihnen schwer oder ist vollkommen unmöglich. Um das zu kompensieren, nutzen Sie eventuell eine Browsererweiterung, die es Ihnen ermöglicht, zu den wichtigen Navigationslinks zu springen (oder über sie hinaus). Oder überlegen Sie das: Ihr Sehvermögen ist eingeschränkt, und Sie nutzen ein spezielles Programm, einen sogenannten »Screenreader«, der Text in Sprache umwandelt und Webseiten zusammenfasst. Wenn Sie den Seitentitel hinter sich haben, sind die nächsten wichtigen Informationen die Navigationslinks. Wenn Sie schnell navigieren wollen, sagen Sie dem Bildschirmleser, dass er zur Navigationsleiste springen und mit dem Lesen beginnen soll. Wenn Sie schnell lesen wollen, sagen Sie Ihrem Bildschirmleser vielleicht, dass er die Navigationsleiste überspringen und damit beginnen soll, den eigentlichen Inhalt der Seite vorzulesen. In beiden Fällen ist es äußerst wichtig, dass man die Navigationslinks programmtechnisch ermitteln kann.

Die Auszeichnung der Site-Navigation mit `<div id="nav">` ist also nicht falsch, allerdings auch nicht sonderlich richtig. Dieser Mangel wirkt sich auf die Handhabung aus. HTML5 bietet ein semantisches Mittel, Navigationsabschnitte auszuzeichnen – das `<nav>`-Element:

```
<nav>
  <ul>
    <li><a href="#">home</a></li>
    <li><a href="#">blog</a></li>
    <li><a href="#">gallery</a></li>
    <li><a href="#">about</a></li>
  </ul>
</nav>
```

Fragen an Professor Markup

F: Sind Sprunglinks mit dem `<nav>`*-Element kompatibel? Brauche ich Sprunglinks in HTML5 noch?*

A: Sprunglinks ermöglichen Lesern, Navigationsabschnitte zu überspringen. Sie sind für behinderte Nutzer hilfreich, die zusätzliche Programme verwenden, um sich eine Webseite laut vorlesen zu lassen und ohne Maus in ihr zu navigieren. Warum und wie Sie Sprunglinks anbieten sollten, erfahren Sie unter *http://www.webaim.org/techniques/skipnav*.

Wenn Screenreader gelernt haben, das `<nav>`-Element zu erkennen, werden Sprunglinks überflüssig werden, da Screenreader dann automatisch anbieten können, einen Navigationsabschnitt zu überspringen, der mit dem `<nav>`-Element ausgezeichnet ist. Es wird allerdings eine Weile dauern, bis alle behinderten Benutzer auf HTML5-fähige Screenreader-Software umgestiegen sind. Sie sollten also weiterhin Sprunglinks anbieten, um Ihre `<nav>`-Abschnitte zu überspringen.

Fußleisten

Endlich sind wir am Ende unserer Beispielseite angelangt. Das Letzte, worüber wir sprechen wollen, ist auch das letzte Element der Seite: die Fußleiste. Das entsprechende Markup sah ursprünglich so aus:

```
<div id="footer">
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</div>
```

Das ist gültiges HTML5. Wenn es Ihnen gefällt, können Sie es so lassen. Aber HTML5 bietet ein spezifischeres Element dafür – das `<footer>`-Element:

```
<footer>
  <p>&#167;</p>
  <p>&#169; 2001&#8211;9 <a href="#">Mark Pilgrim</a></p>
</footer>
```

Was man in ein `<footer>`-Element stecken sollte? Wahrscheinlich alles, was Sie bislang in ein `<div id="footer">` gepackt haben. Gut, diese Antwort enthält vermutlich wenig Neues für Sie, aber genau das ist die Antwort. Die HTML5-Spezifikation sagt: »Eine Fußleiste enthält üblicherweise Informationen über den Abschnitt, dem sie zugeordnet ist, beispielsweise wer ihn geschrieben hat, Links auf darauf bezogene Dokumente, Copyright-Angaben und Ähnliches.« Genau das steht auch in der Fußleiste der Beispielseite: ein kurzer Copyright-Hinweis und ein Link auf die Informationsseite zum Seitenautor. Wenn ich mich auf anderen populären Sites umschaue, sehe ich eine Menge Möglichkeiten für das `footer`-Element:

- Der Spiegel hat eine Fußleiste, die Links zu Onlinepartnern und Links auf weitere Site-Inhalte, unter anderem zur Hilfe-Seite, zur Kontakt-Seite oder zum Impressum enthält. Alles höchst passendes `<footer>`-Material.
- Google hat die berühmte spartanische Homepage, aber unten finden sich Links auf »Werben mit Google«, »Unternehmensprogramm«, »Über Google« sowie ein Copyright-Hinweis und die Datenschutzerklärung. All das könnte in ein `<footer>`-Element gepackt werden.
- Mein Weblog (*http://diveintomark.org*) hat eine Fußleiste mit Links auf meine anderen Sites und einem Copyright-Hinweis. Auf alle Fälle für ein `<footer>`-Element geeignet. (Beachten Sie, dass die Links selbst nicht in ein `<nav>`-Element gepackt werden sollten, weil es keine Links für die Navigation der Site sind. Sie stellen lediglich eine Sammlung von Links auf meine anderen Projekte auf anderen Sites dar.)

Fat Footers (*http://ui-patterns.com/pattern/FatFooter*) (umfangreiche Fußleisten) sind aktuell der letzte Schrei. Schauen Sie sich nur die Fußleiste der W3C-Site (*http://www.w3.org*) an. Sie enthält drei Spalten mit den Überschriften »Navigation«, »Contact W3C« und »W3C Updates«. Das Markup sieht ungefähr so aus:

```
<div id="w3c_footer">
  <div class="w3c_footer-nav">
    <h3>Navigation</h3>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>Contact W3C</h3>
    <ul>
      <li><a href="/Consortium/contact">Contact</a></li>
      <li><a href="/Help/">Help and FAQ</a></li>
      <li><a href="/Consortium/sup">Donate</a></li>
      <li><a href="/Consortium/siteindex">Site Map</a></li>
    </ul>
  </div>
  <div class="w3c_footer-nav">
    <h3>W3C Updates</h3>
    <ul>
      <li><a href="http://twitter.com/W3C">Twitter</a></li>
      <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
    </ul>
  </div>
  <p class="copyright">Copyright © 2009 W3C</p>
</div>
```

Um das in semantisches HTML5 umzuwandeln, würde ich die folgenden Änderungen vornehmen:

- Ich würde das äußere `<div id="w3c_footer">` in ein `<footer>`-Element umwandeln.
- Ich würde die ersten beiden `<div class="w3c_footer-nav">` in `<nav>`-Elemente umwandeln und das dritte in ein `<section>`-Element.
- Ich würde die `<h3>`-Überschriften in `<h1>`-Überschriften umwandeln, da sie dann alle in abschnittsbildenden Elementen stünden. Das `<nav>`-Element erstellt einen Abschnitt in der Dokumentgliederung, genau wie das `<article>`-Element (siehe dazu den Abschnitt »Artikel« auf Seite 50).

Das endgültige Markup könnte folgendermaßen aussehen:

```
<footer>
  <nav>
    <h1>Navigation</h1>
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/standards/">Standards</a></li>
      <li><a href="/participate/">Participate</a></li>
      <li><a href="/Consortium/membership">Membership</a></li>
      <li><a href="/Consortium/">About W3C</a></li>
    </ul>
  </nav>
```

```
    <nav>
      <h1>Contact W3C</h1>
      <ul>
        <li><a href="/Consortium/contact">Contact</a></li>
        <li><a href="/Help/">Help and FAQ</a></li>
        <li><a href="/Consortium/sup">Donate</a></li>
        <li><a href="/Consortium/siteindex">Site Map</a></li>
      </ul>
    </nav>
    <section>
      <h1>W3C Updates</h1>
      <ul>
        <li><a href="http://twitter.com/W3C">Twitter</a></li>
        <li><a href="http://identi.ca/w3c">Identi.ca</a></li>
      </ul>
    </section>
    <p class="copyright">Copyright © 2009 W3C</p>
  </footer>
```

Weitere Lektüre

Die in diesem Kapitel genutzten Beispielseiten:

- Original (HTML 4) (*http://diveintohtml5.org/examples/blog-original.html*)
- Überarbeitet (HTML5) (*http://diveintohtml5.org/examples/blog-html5.html*)

Zu Zeichenkodierungen:

- »The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)« (*http://www.joelonsoftware.com/articles/Unicode.html*) von Joel Spolsky

Zur Aktivierung der neuen HTML5-Elemente im Internet Explorer:

- »How to style unknown elements in IE« (*http://xopus.com/devblog/2008/style-unknown-elements.html*) von Sjoerd Visscher
- HTML5 shiv (*http://ejohn.org/blog/html5-shiv/*) von John Resig
- HTML5 enabling script (*http://remysharp.com/2009/01/07/html5-enabling-script/*) von Remy Sharp

Zu Standards-Modi und Doctype-Sniffing:

- »Activating Browser Modes with Doctype« (*http://hsivonen.iki.fi/doctype/*) von Henri Sivonen. Das ist der einzige Artikel, den Sie zu diesem Thema lesen sollten. Es gibt viele andere, aber die sind entweder veraltet, unvollständig oder falsch.

HTML5-fähiger Validierer:

- Validator.nu (X)HTML5 Validator (*http://html5.validator.nu*)

KAPITEL 4

Zeichenstunde

Einstieg

HTML5 definiert das <canvas>-Element (*http://bit.ly/9JHzOf*) als »eine auflösungsabhängige Bitmap-Leinwand zur unmittelbaren Zeichnung von Diagrammen, Spielgrafiken und anderen visuellen Bildern«. Ein Canvas ist ein Rechteck in Ihrer Seite, in das Sie mit JavaScript nach Belieben zeichnen können. Die folgende Tabelle zeigt, welche Browser eine elementare Canvas-Unterstützung boten, als dies geschrieben wurde:

IE[1]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Wie also sieht ein Canvas aus? Nach nichts eigentlich. Ein <canvas>-Element an sich hat keinen Inhalt oder Rahmen. Das Markup sieht so aus:

```
<canvas width="300" height="225"></canvas>
```

Abbildung 4-1 zeigt das Canvas mit einem gepunkteten Rahmen, damit Sie sehen können, wovon wir reden.

Eine Seite kann mehrere <canvas>-Elemente enthalten. Jedes Canvas erscheint im DOM, und jedes Canvas hat seinen eigenen Zustand. Wenn Sie jedem Canvas ein id-Attribut geben, können Sie darauf zugreifen wie auf jedes andere Element in einer Seite.

Erweitern wir das Markup um ein id-Attribut:

```
<canvas id="a" width="300" height="225"></canvas>
```

Jetzt ist es ein Kinderspiel, das <canvas>-Element im DOM zu finden:

```
var a_canvas = document.getElementById("a");
```

1 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.

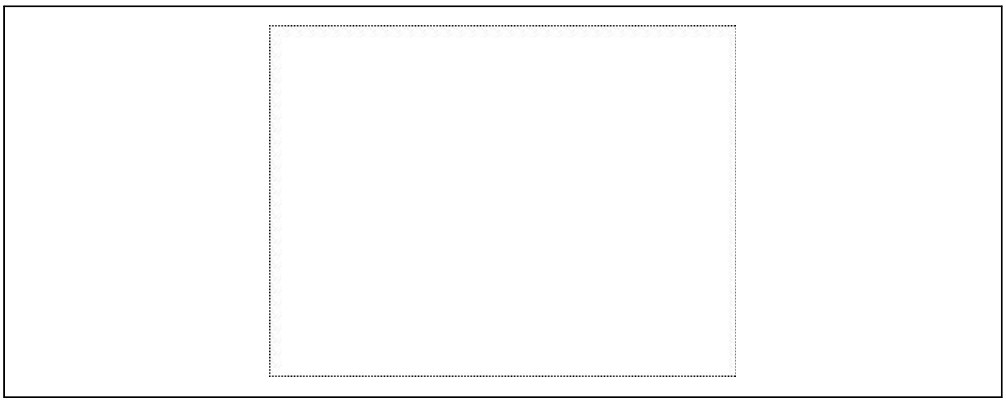


Abbildung 4-1: Canvas mit Rahmen

Einfache Figuren

IE[2]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Zu Beginn ist ein Canvas leer. Wie langweilig! Schaffen wir Abhilfe. Zeichnen wir etwas. Sie können den `onclick`-Handler einsetzen, um eine Funktion aufzurufen, die ein Rechteck zeichnet (*http://diveintohtml5.org/canvas.html* präsentiert Ihnen ein interaktives Beispiel):

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

Die erste Zeile der Funktion ist nichts Besonderes. Sie sucht im DOM nach dem `<canvas>`-Element. In der zweiten Zeile wird die Angelegenheit schon interessanter. Ein Canvas hat einen Zeichenkontext (Drawing-Kontext), und das ist der Ort, an dem all die interessanten Dinge geschehen. Haben Sie das `<canvas>`-Element (über `document.getElementById()` oder eine beliebige andere Methode) im DOM gefunden, können Sie seine `getContext()`-Methode aufrufen. Sie müssen dabei den String `"2d"` an `getContext()` übergeben:

```
function draw_b() {
  var b_canvas = document.getElementById("b");
  var b_context = b_canvas.getContext("2d");
  b_context.fillRect(50, 25, 150, 100);
}
```

2 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.

Fragen an Professor Markup

F: Gibt es ein 3-D-Canvas?

A: Noch nicht. Einzelne Hersteller haben mit eigenen dreidimensionalen Canvas-APIs experimentiert, aber keine wurde standardisiert. Die HTML5-Spezifikation sagt dazu: »Eine zukünftige Version dieser Spezifikation wird wahrscheinlich einen 3-D-Kontext definieren.«

Sie haben also ein `<canvas>`-Element und seinen Zeichenkontext. Der Zeichenkontext ist das, worin alle Zeichenmethoden und Eigenschaften definiert sind. Es gibt eine große Menge an Eigenschaften und Methoden, die dem Zeichnen von Rechtecken gewidmet sind:

- Die `fillStyle`-Eigenschaft kann eine CSS-Farbe, ein Muster oder ein Verlauf sein. (Mehr zu Verläufen in Kürze.) Der Standard für `fillStyle` ist Schwarz, aber Sie können die Eigenschaft auf einen beliebigen Wert setzen. Jeder Zeichenkontext hält seine Eigenschaften fest, solange die Seite offen ist, oder bis Sie etwas unternehmen, dass sie neu setzt.
- `fillRect(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`)` zeichnet ein Rechteck, das mit dem aktuellen Füllstil gefüllt wird.
- Die `strokeStyle`-Eigenschaft gleicht `fillStyle`, d.h., sie kann eine CSS-Farbe, ein Muster oder einen Verlauf enthalten.
- `strokeRect(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`)` zeichnet ein Rechteck mit dem aktuellen Strichstil. `strokeRect` füllt das Rechteck nicht, sondern zeichnet nur den Rahmen.
- `clearRect(`*`x`*`, `*`y`*`, `*`width`*`, `*`height`*`)` leert die Pixel im angegebenen Rechteck.

Fragen an Professor Markup

F: Kann ich ein Canvas »zurücksetzen«?

A: Ja. Das Setzen der Breite oder Höhe eines `<canvas>`-Elements löscht seinen Inhalt und setzt alle Eigenschaften auf seinem Zeichenkontext auf ihre Standardwerte zurück. Sie müssen die Breite nicht ändern, es reicht, wenn Sie sie folgendermaßen wieder auf ihren aktuellen Wert setzen:

```
var b_canvas = document.getElementById("b");
b_canvas.width = b_canvas.width;
```

Kehren wir zum Code aus unserem vorangegangenen Beispiel zurück:

```
var b_canvas = document.getElementById("b");
var b_context = b_canvas.getContext("2d");
b_context.fillRect(50, 25, 150, 100);
```

Der Aufruf von `fillRect()` zeichnet ein Rechteck und füllt es mit dem aktuellen Füllstil, d.h. mit Schwarz, da wir den Füllstil nicht geändert haben. Das Rechteck wird von seiner oberen linken Ecke (`50, 25`), seiner Breite (`150`) und seiner Höhe (`100`) definiert. Schauen wir uns das Canvas-Koordinatensystem an, damit wir uns die Sache besser vorstellen können.

Canvas-Koordinaten

Das Canvas ist ein zweidimensionales Raster. Die Koordinaten (0, 0) bilden die obere linke Ecke des Canvas. Entlang der x-Achse erhöhen sich die Werte in Richtung des rechten Rands des Canvas, entlang der y-Achse erhöhen sich die Werte in Richtung seines unteren Rands.

Das Koordinatendiagramm in Abbildung 4-2 wurde in einem `<canvas>`-Element gezeichnet. Sie besteht aus:

- einem Satz vertikaler cremefarbener Linien,
- einem Satz horizontaler cremefarbener Linien,
- zwei horizontalen schwarzen Linien,
- zwei kleinen diagonalen schwarzen Linien, die einen Pfeil bilden,
- zwei vertikalen schwarzen Linien,
- zwei kleinen diagonalen schwarzen Linien, die einen weiteren Pfeil bilden,
- dem Buchstaben »x«,
- dem Buchstaben »y«,
- dem Text »(0, 0)« nahe der oberen linken Ecke,
- dem Text »(500, 375)« nahe der unteren linken Ecke sowie
- einem Punkt in der oberen linken Ecke und einem anderen in der unteren rechten Ecke.

In den folgenden Abschnitten werden wir uns ansehen, wie man die Effekte erzeugt, die in dieser Abbildung gezeigt werden, aber erst müssen wir das `<canvas>`-Element selbst definieren. Folgendes `<canvas>`-Element definiert mit den Eigenschaften `width` und `height` die Breite der Zeichenfläche und die `id`, damit wir es später finden können:

```
<canvas id="c" width="500" height="375"></canvas>
```

Dann benötigen wir ein Skript, um das `<canvas>`-Element im DOM zu finden und seinen Zeichenkontext abzurufen:

```
var c_canvas = document.getElementById("c");
var context = c_canvas.getContext("2d");
```

Jetzt können wir damit beginnen, Linien zu zeichnen.

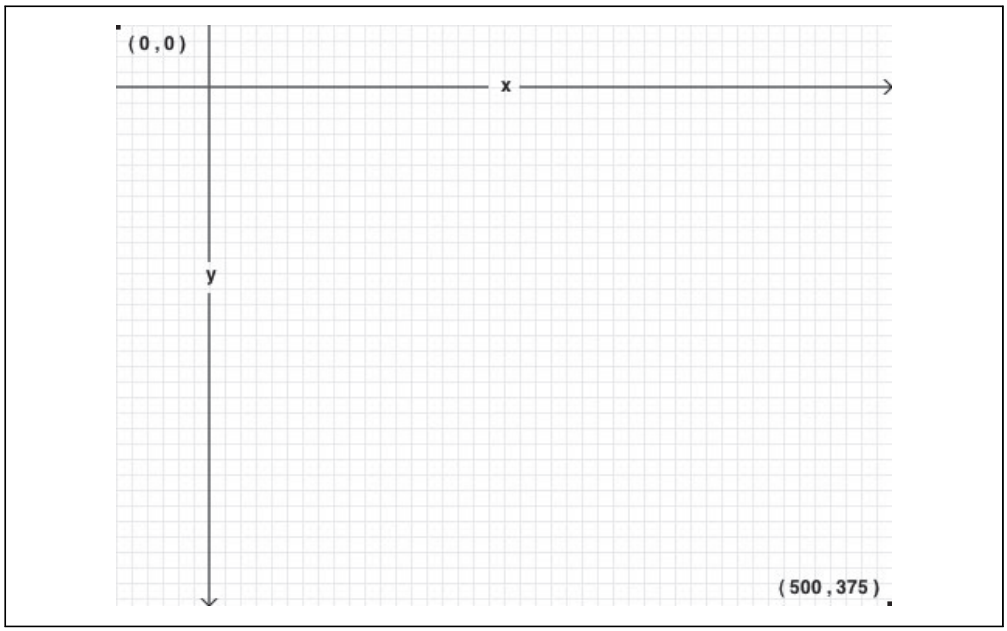


Abbildung 4-2: Ein Canvas-Koordinatendiagramm

Pfade

IE[3]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Mal angenommen, Sie möchten eine Tintenzeichnung erstellen. Normalerweise fangen Sie nicht sofort an, mit Tinte zu zeichnen, weil Sie eventuelle Fehler dann nicht so leicht beheben können. Stattdessen werden Sie die Linien und Kurven zunächst mit einem Bleistift skizzieren und diese Skizze erst dann mit Tinte überzeichnen, wenn Sie mit Ihrem Entwurf zufrieden sind.

Jedes Canvas hat einen *Pfad*. Die Definition des Pfads ist wie das Zeichnen mit einem Bleistift. Sie können zeichnen, was Sie wollen, aber Teil des fertigen Produkts wird es erst, wenn Sie zur Feder greifen und Ihren Pfad mit Tinte überzeichnen.

Die folgenden beiden Methoden nutzen Sie, um zwei gerade Linien mit Bleistift zu zeichnen:

- `moveTo(x, y)` bewegt den Bleistift zum angegebenen Startpunkt.
- `lineTo(x, y)` zeichnet eine Linie zum angegebenen Endpunkt.

3 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.

Je häufiger Sie `moveTo()` und `lineTo()`aufrufen, umso umfangreicher wird der Pfad. Diese beiden Methoden sind »Bleistiftmethoden«. Sie können sie so oft aufrufen, wie Sie wollen, aber Sie werden so lange nichts sehen, bis Sie eine der »Tintenmethoden« aufrufen.

Beginnen wir damit, das cremefarbene Raster zu zeichnen:

```
for (var x = 0.5; x < 500; x += 10) {
  context.moveTo(x, 0);
  context.lineTo(x, 375);
}

for (var y = 0.5; y < 375; y += 10) {
  context.moveTo(0, y);
  context.lineTo(500, y);
}
```

Da das alles »Bleistiftmethoden« waren, wurde noch nichts auf das Canvas gezeichnet. Wir müssen es erst mit einer »Tintenmethode« festigen:

```
context.strokeStyle = "#eee";
context.stroke();
```

`stroke()` ist eine der »Tintenmethoden«. Sie übernimmt den komplexen Pfad, den Sie mit diesen ganzen `moveTo()`- und `lineTo()`-Aufrufen erzeugt haben, und zeichnet ihn tatsächlich auf das Canvas. `strokeStyle` steuert die Farbe Ihrer Linien. Abbildung 4-3 zeigt das Ergebnis.

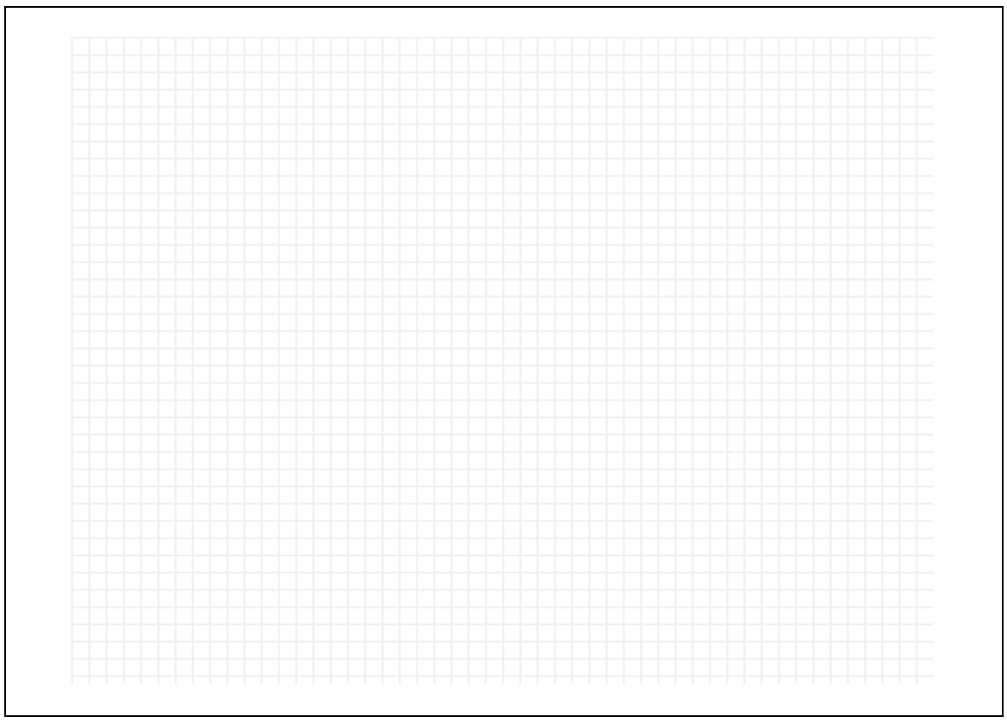


Abbildung 4-3: Ein auf ein Canvas gezeichnetes Raster

Fragen an Professor Markup

F: Warum haben Sie `x` und `y` bei `0.5` beginnen lassen? Warum nicht bei `0`?

A: Stellen Sie sich die einzelnen Pixel als große Quadrate vor. Die ganzzahligen Koordinaten (`0`, `1`, `2`, ...) sind die Ränder der Quadrate. Wenn Sie auf einer ganzzahligen Koordinate eine Linie zeichnen, die eine Einheit breit ist, reicht diese zur Hälfte jeweils in die Pixelquadrate auf beiden Seiten. Da der Rechner keine halben Pixel zeichnen kann, wird die Linie also mit zwei Pixeln Breite gezeichnet. Wollen Sie eine Linie zeichnen, die nur ein Pixel breit ist, müssen Sie die Koordinaten lotrecht zur Richtung der Linie um 0,5 verschieben.

Zeichnen wir jetzt den horizontalen Pfeil. Alle Linien und Kurven auf einem Pfad werden in der gleiche Farbe gezeichnet (stimmt, Verläufe, mit denen wollten wir uns auch noch befassen). Den Pfeil wollen wir in einer anderen Farbe zeichnen – Tintenschwarz statt cremefarben –, also müssen wir dazu einen neuen Pfad beginnen:

```
context.beginPath();
context.moveTo(0, 40);
context.lineTo(240, 40);
context.moveTo(260, 40);
context.lineTo(500, 40);
context.moveTo(495, 35);
context.lineTo(500, 40);
context.lineTo(495, 45);
```

Der vertikale Pfeil sieht ganz ähnlich aus. Da er die gleiche Farbe wie der horizontale Pfeil hat, müssen wir diesmal keinen neuen Pfad beginnen. Die beiden Pfeile werden Teil des gleichen Pfads sein:

```
context.moveTo(60, 0);
context.lineTo(60, 153);
context.moveTo(60, 173);
context.lineTo(60, 375);
context.moveTo(65, 370);
context.lineTo(60, 375);
context.lineTo(55, 370);
```

Ich sagte, dass diese Pfeil schwarz werden sollen, aber unser `strokeStyle` ist immer noch cremefarben. (`fillStyle` und `strokeStyle` werden nicht zurückgesetzt, wenn ein neuer Pfad begonnen wird.) Bislang war das kein Problem, da wir nur »Bleistiftmethoden« genutzt haben. Aber bevor wir den neuen Pfad tatsächlich, in »Tinte«, zeichnen lassen, müssen wir `strokeStyle` auf Schwarz setzen. Sonst werden die Pfeile cremefarben und folglich kaum erkennbar sein! Die folgenden Zeilen ändern die Farbe in Schwarz und zeichnen die Linien auf das Canvas:

```
context.strokeStyle = "#000";
context.stroke();
```

Abbildung 4-4 zeigt das Ergebnis.

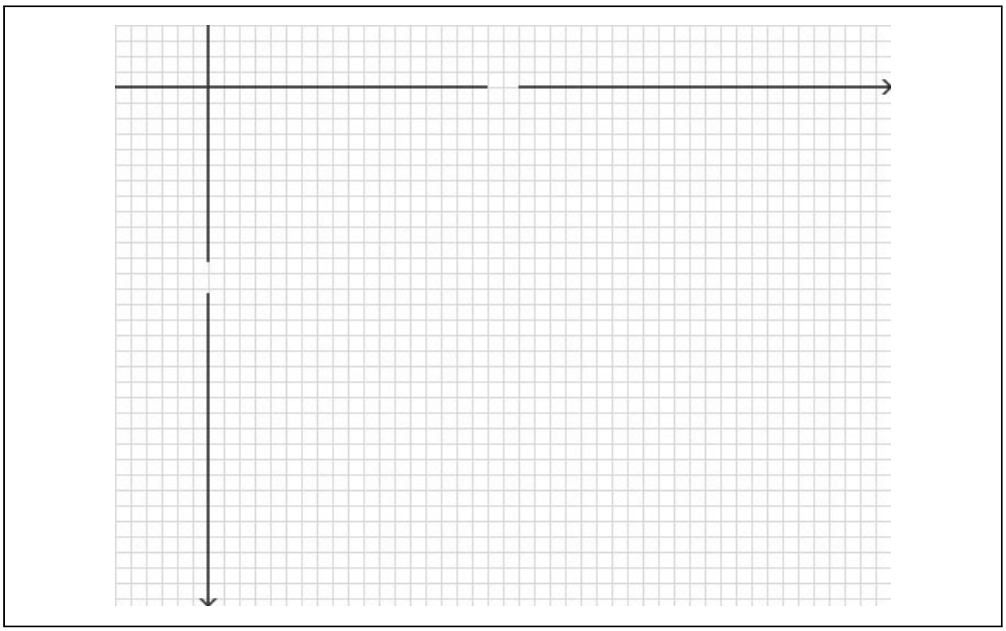


Abbildung 4-4: Unbeschriftete Achsen auf einem Canvas

Text

IE[4]	Firefox[5]	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Auf ein Canvas können Sie nicht nur Linien, sondern auch Text zeichnen. Anders als beim Text in der umgebenden Webseite gibt es kein Box-Modell. Das heißt, dass keine der vertrauten CSS-Layouttechniken verfügbar sind: keine schwebenden Elemente, keine Außenabstände, keine Innenabstände, kein Zeilenumbruch. (Vielleicht sind Sie ja der Meinung, das sei gar nicht so verkehrt!) Sie können ein paar Schriftattribute setzen, und anschließend wählen Sie sich einen Punkt auf Ihrem Canvas, um dort Ihren Text zu zeichnen.

Die folgenden Schriftattribute sind auf dem Zeichenkontext verfügbar (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60):

- `font` kann ein beliebiger Wert sein, den Sie auch in einer CSS-`font`-Regel angeben. Das schließt den Schriftschnitt, die Schriftvariante, das Schriftgewicht, die Schriftgröße, die Zeilenhöhe und die Schriftfamilie ein.
- `textAlign` steuert die Textausrichtung. Es ähnelt der CSS-`text`-Regel `align` (ist mit ihr aber nicht identisch). Mögliche Werte sind `start`, `end`, `left`, `right` und `center`.

4 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.
5 Mozilla Firefox 3.0 benötigt eine Kompatibilitätsschicht.

- `textBaseline` steuert, auf welcher Höhe im em-Quadrat der Text gezeichnet wird. Mögliche Werte sind `top`, `hanging`, `middle`, `alphabetic`, `ideographic` und `bottom`.

`textBaseline` ist verzwickt, weil Text verzwickt ist. (Na gut, englischer Text ist kein Problem, aber auf ein Canvas kann man beliebige Unicode-Zeichen zeichnen, und Unicode ist ein Problem.) Die HTML5-Spezifikation erläutert die verschiedenen Basislinien für Text:[6]

> Die obere Grundlinie des Gevierts entspricht ungefähr dem oberen Rand der Glyphen (d.h. der Schriftzeichen) einer Schrift. An der hängenden Grundlinie sind einige Glyphen wie आ verankert. Die Mittellinie bezeichnet die Mitte zwischen der oberen und der unteren Begrenzung des Gevierts. Die alphabetische Grundlinie gibt die Höhe an, auf der Zeichen wie Á, ÿ, f und Ω verankert sind. Auf der ideographischen Grundlinie sind Glyphen wie 私 und 達 verankert. Auf der unteren Grundlinie des Gevierts liegen Schriftzeichen mit Unterlängen wie p oder y in etwa auf. Der Abstand zwischen der unteren bzw. der oberen Begrenzung der Zeichen-Box und den Grundlinien des Gevierts kann recht groß sein, weil manche Schriftzeichen weit über das Geviert hinaus reichen (siehe Abbildung 4-5).

Abbildung 4-5: Text-Grundlinien

Bei einfachen Alphabeten wie dem englischen oder dem deutschen können Sie sich bei der Eigenschaft `textBaseline` problemlos auf die Werte `top`, `middle` oder `bottom` beschränken.

Zeichnen wir etwas Text! Text, der in ein Canvas gezeichnet wird, erbt die Schriftgröße und den Schriftschnitt des `<canvas>`-Elements. Aber Sie können diese Werte überschreiben, indem Sie die `font`-Eigenschaft des Zeichenkontexts nutzen:

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

6 *http://bit.ly/aHCdDO*. Das Geviert (oder em-Quadrat) ist eine typografische Maßeinheit, mit der die Höhe und die Breite von Schriftzeichen festgelegt wird. Die Maßeinheit em wird verwendet, um die quadratische Fläche zu beschreiben, deren Breite gleich der Höhe des Schriftkegels ist. Das Geviert definiert also in der Höhe den Mindestzeilenabstand einer Schrift.

Die Methode `fillText()` zeichnet den eigentlichen Text:

```
context.font = "bold 12px sans-serif";
context.fillText("x", 248, 43);
context.fillText("y", 58, 165);
```

Fragen an Professor Markup

F: Kann ich relative Schriftgrößen nutzen, um Text in ein Canvas zu zeichnen?

A: Ja. Wie alle anderen HTML-Elemente in Ihrer Seite hat das `<canvas>`-Element eine auf Basis der CSS-Regeln der Seite berechnete Schriftgröße. Setzen Sie die Eigenschaft `context.font` auf eine relative Schriftgröße wie `1.5em` oder `150%`, multipliziert Ihr Browser diesen Wert mit der für das `<canvas>`-Element selbst berechneten Schriftgröße.

Nehmen wir an, wir wollen, dass der Text oben links bei `y=5` erscheint. Leider sind wir faul und haben absolut keine Lust, die Texthöhe zu messen und dann auf dieser Basis die Grundlinie zu berechnen. Stattdessen können wir `textBaseline` einfach auf `top` setzen und dann die Koordinate der Zeichen-Box übergeben:

```
context.textBaseline = "top";
context.fillText("( 0 , 0 )", 8, 5);
```

Jetzt zum Text unten rechts. Angenommen, das rechte Ende des Textes soll sich auf den Koordinaten `(492,370)` befinden – nur wenige Pixel von der unteren rechten Ecke des Canvas selbst entfernt. Wieder sind wir zu faul, die Breite oder Höhe des Texts zu messen. Stattdessen setzen wir `textAlign` auf `right` und `textBaseline` auf `bottom` und rufen mit `fillText()` den Text auf den Koordinaten an der unteren rechten Ecke der Zeichen-Box auf:

```
context.textAlign = "right";
context.textBaseline = "bottom";
context.fillText("( 500 , 375 )", 492, 370);
```

Abbildung 4-6 zeigt das Ergebnis.

Mist! Wir haben die Punkte in den Ecken vergessen. Wie man Kreise zeichnet, werden wir uns etwas später ansehen. Jetzt mogeln wir etwas und zeichnen sie einfach als Rechtecke (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60):

```
context.fillRect(0, 0, 3, 3);
context.fillRect(497, 372, 3, 3);
```

Und das war es! Abbildung 4-7 zeigt das endgültige Ergebnis.

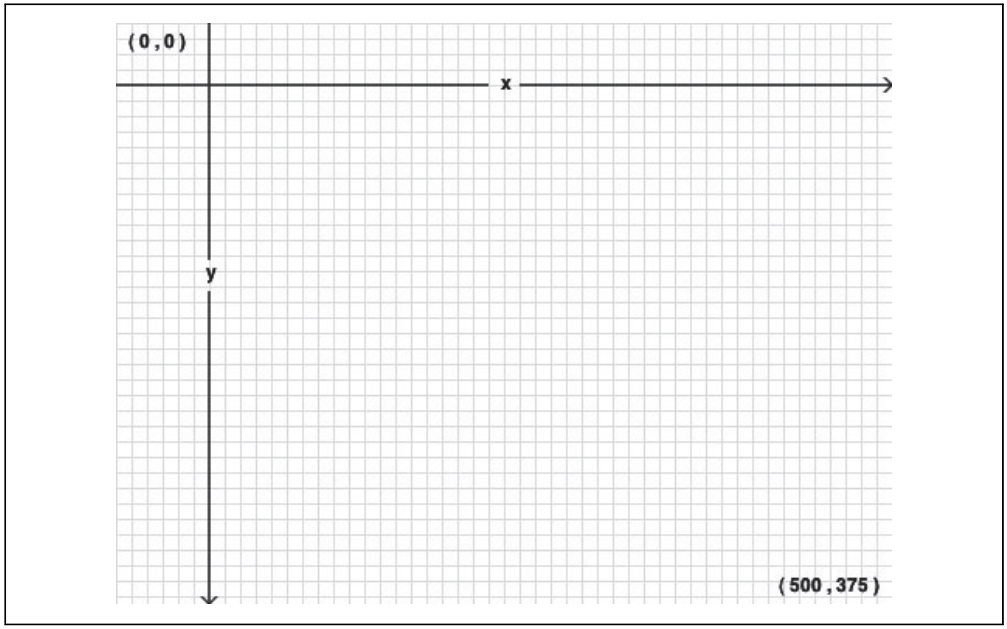


Abbildung 4-6: Beschriftete Achsen auf einem Canvas

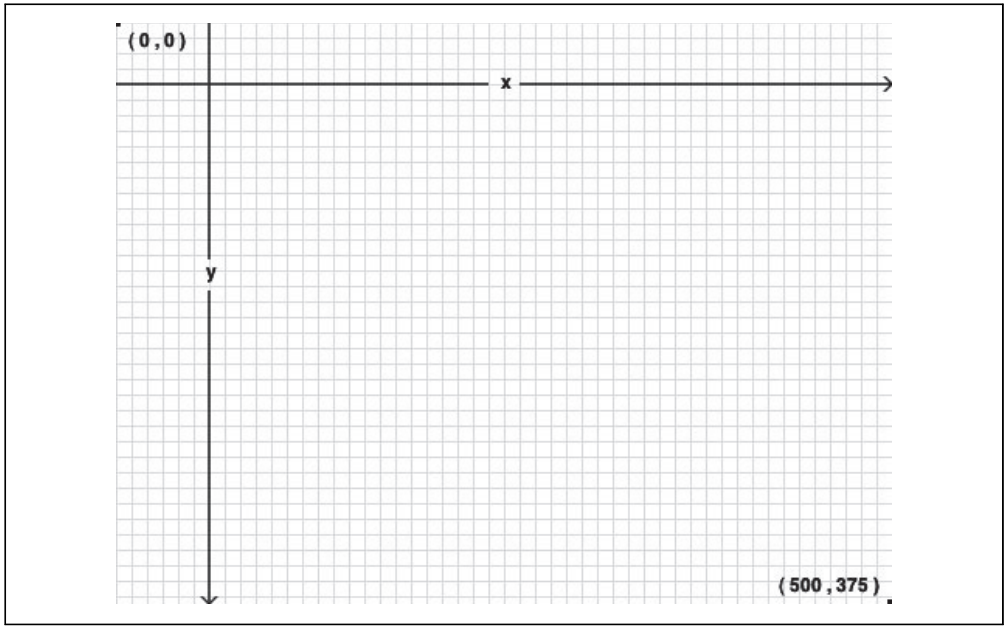


Abbildung 4-7: Ein Canvas-Koordinatensystem auf einem Canvas

Verläufe

	IE[7]	Firefox	Safari	Chrome	Opera	iPhone	Android
Lineare Verläufe	7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+
Radiale Verläufe	·	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Zu Beginn dieses Kapitels haben wir uns angesehen, wie man ein Rechteck zeichnet, das durchgängig mit einer Farbe gefüllt ist (siehe dazu den Abschnitt »Einfache Figuren« auf Seite 60), später haben wir eine ebenso gefüllte Linie gezeichnet (siehe Abschnitt »Pfade« auf Seite 63). Aber bei beidem sind wir nicht auf Einfarbigkeit beschränkt. Mit Verläufen können wir die vielfältigsten Zauberkunststücke vollführen. Abbildung 4-8 zeigt Ihnen ein Beispiel.

Das Markup sieht aus wie bei jedem anderen Canvas:

```
<canvas id="d" width="300" height="225"></canvas>
```

Erst müssen wir das <canvas>-Element und seinen Zeichenkontext abrufen:

```
var d_canvas = document.getElementById("d");
var context = d_canvas.getContext("2d");
```

Abbildung 4-8: Ein linearer Verlauf von links nach rechts

Sobald wir den Zeichenkontext haben, können wir mit der Definition des Verlaufs loslegen. Ein Verlauf ist ein sanfter Wechsel von zwei oder mehr Farben. Der Canvas-Zeichenkontext unterstützt zwei Arten von Verläufen:

- `createLinearGradient(`*x0*`,` *y0*`,` *x1*`,` *y1*`)` malt entlang einer Linie von (*x0*, *y0*) nach (*x1*, *y1*).
- `createRadialGradient(`*x0*`,` *y0*`,` *r0*`,` *x1*`,` *y1*`,` *r1*`)` malt entlang eines Kegels zwischen zwei Kreisen. Die ersten drei Parameter repräsentieren den Ausgangskreis mit dem Ursprung (*x0*, *y0*) und dem Radius *r0*. Die letzten drei Parameter repräsentieren den Zielkreis mit dem Ursprung (*x1*, *y1*) und dem Radius *r1*.

7 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.

Erstellen wir einen linearen Verlauf. Verläufe können beliebige Größen haben. Diesen werden wir 300 Pixel breit machen, genauso breit wie das Canvas also:

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

Da die y-Werte (der zweite und der vierte Parameter) beide `0` sind, fließt dieser Verlauf gleichförmig von links nach rechts.

Haben wir unseren Verlauf erzeugt, können wir die Farben definieren. Ein Verlauf hat zwei oder mehr Farbstopps. Diese können an beliebigen Punkten entlang des Verlaufs liegen. Wollen Sie einen Farbstopp einfügen, müssen Sie seine Position auf dem Verlauf angeben. Diese Position wird durch einen beliebigen Wert zwischen 0 und 1 bestimmt.

Definieren wir einen Verlauf von Schwarz nach Weiß:

```
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
```

Durch die Definition eines Verlaufs wird noch nichts auf das Canvas gezeichnet. Er ist einfach ein Objekt, das irgendwo im Speicher sitzt. Wollen Sie ihn zeichnen, setzen Sie `fillStyle` auf den Verlauf und zeichnen eine Figur wie ein Rechteck oder eine Linie:

```
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Abbildung 4-9 zeigt das Ergebnis.

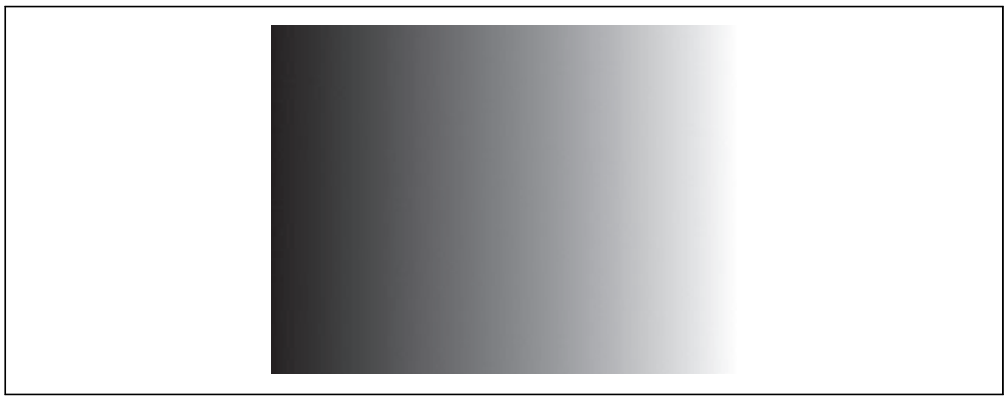


Abbildung 4-9: Ein linearer Verlauf von links nach rechts

Angenommen, Sie wollen aber einen Verlauf erzeugen, der von oben nach unten fließt. Halten Sie dazu bei der Erstellung des Verlaufs die x-Werte (den ersten und dritten Parameter) konstant und lassen Sie die y-Werte (den zweiten und vierten Parameter) von 0 bis zur Höhe des Canvas reichen:

```
var my_gradient = context.createLinearGradient(0, 0, 0, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Abbildung 4-10 zeigt das Ergebnis.

Abbildung 4-10: Ein linearer Verlauf von oben nach unten

Sie können auch diagonal verlaufende Verläufe erstellen, so zum Beispiel:

```
var my_gradient = context.createLinearGradient(0, 0, 300, 225);
my_gradient.addColorStop(0, "black");
my_gradient.addColorStop(1, "white");
context.fillStyle = my_gradient;
context.fillRect(0, 0, 300, 225);
```

Abbildung 4-11 zeigt das Ergebnis.

Abbildung 4-11: Ein diagonaler linearer Verlauf

Bilder

IE[8]	Firefox	Safari	Chrome	Opera	iPhone	Android
7.0+	3.0+	3.0+	3.0+	10.0+	1.0+	1.0+

Abbildung 4-12 zeigt das Bild einer Katze, das mit einem `<img>`-Element angezeigt wird.

Abbildung 4-12: Katze mit einem <img>-Element

Abbildung 4-13 zeigt die gleiche Katze, aber auf ein Canvas gezeichnet.

Abbildung 4-13: Katze mit einem <canvas>-Element

Der Canvas-Zeichenkontext definiert verschiedene Methoden, um Bilder auf ein Canvas zu zeichnen:

- `drawImage(`*`Bild, dx, dy`*`)` erwartet ein Bild und zeichnet es auf das Canvas. Die übergebenen Koordinaten (*dx, dy*) repräsentieren die obere linke Ecke des Bilds. Bei den Koordinaten (0, 0) würde das Bild in die obere linke Ecke des Canvas gezeichnet.
- `drawImage(`*`Bild, dx, dy, dw, dh`*`)` erwartet ein Bild, skaliert es auf eine Breite von *dw* sowie eine Höhe von *dh* und zeichnet es bei den Koordinaten(*dx, dy*) auf das Canvas.
- `drawImage(`*`Bild, sx, sy, sw, sh, dx, dy, dw, dh`*`)` nimmt ein Bild, schneidet es auf das Rechteck (*sx, sy, sw, sh*) zu, skaliert es auf die Größe (*dw, dh*) und zeichnet es bei den Koordinaten (*dx, dy*) auf das Canvas.

Die HTML5-Spezifikation erläutert die Parameter für `drawImage()` (*http://bit.ly/9WTZAp*) folgendermaßen:

> Das Quellrechteck ist das Rechteck (in der Bildquelle), dessen Ecken die Punkte (*sx, sy*), (*sx+sw, sy*), (*sx+sw, sy+sh*), (sx, sy+sh) sind.

8 Der Internet Explorer benötigt die externe Bibliothek explorercanvas.

Das Zielrechteck ist das Rechteck (im Canvas), dessen Ecken die vier Punkte (*dx, dy*), (*dx+dw, dy*), (*dx+dw, dy+dh*), (*dx, dy+dh*) sind.

Abbildung 4-14 zeigt eine visuelle Darstellung dieser Parameter.

Wenn Sie ein Bild auf ein Canvas zeichnen wollen, benötigen Sie zunächst einmal ein Bild. Das Bild kann ein vorhandenes `<img>`-Element oder ein mit JavaScript erstelltes `Image`-Objekt sein. In beiden Fällen müssen Sie sicherstellen, dass das Bild vollständig geladen ist, bevor Sie versuchen, es auf das Canvas zu zeichnen.

Nutzen Sie ein vorhandenes `<img>`-Element, können Sie es gefahrlos während des Events `window.onload` zeichnen:

```
<img id="cat" src="images/cat.png" alt="sleeping cat" width="177" height="113">
<canvas id="e" width="177" height="113"></canvas>
<script>
window.onload = function() {
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = document.getElementById("cat");
  context.drawImage(cat, 0, 0);
};
</script>
```

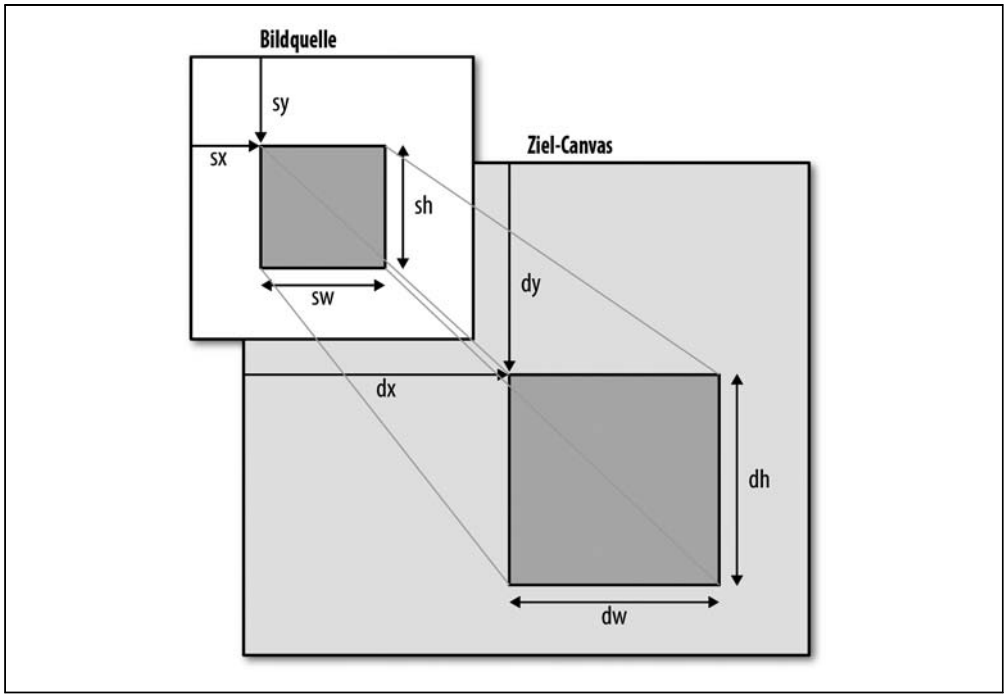


Abbildung 4-14: Wie drawImage() ein Bild in das Canvas einfügt

Möchten Sie ein Image-Objekt vollständig in JavaScript erstellen, können Sie das Bild gefahrlos während des Events `Image.onload` zeichnen:

```
<canvas id="e" width="177" height="113"></canvas>
<script>
  var canvas = document.getElementById("e");
  var context = canvas.getContext("2d");
  var cat = new Image();
  cat.src = "images/cat.png";
  cat.onload = function() {
    context.drawImage(cat, 0, 0);
  };
</script>
```

Die optionalen dritten und vierten Parameter für die Methode `drawImage()` steuern die Skalierung des Bilds. Abbildung 4-15 zeigt das gleiche Katzenbild auf die halbe Breite und Höhe skaliert und mehrfach an unterschiedlichen Koordinaten in ein einziges Canvas gezeichnet:

Hier ist das Skript, das diese »Katzenfamilie« zeichnet:

```
cat.onload = function() {
  for (var x = 0, y = 0;
       x < 500 && y < 375;
       x += 50, y += 37) {
    context.drawImage(cat, x, y, 88, 56);
  }
};
```

Die ganzen Mühen werfen natürlich eine legitime Frage auf: Warum sollte man überhaupt ein Bild in ein Canvas zeichnen wollen? Warum soll man die zusätzliche Komplexität im Vergleich zu einem gewöhnlichen `<img>`-Element und ein paar CSS-Regeln auf sich nehmen? Selbst die »Katzenfamilie« könnte man leicht mit zehn überlappenden `<img>`-Elementen nachbauen.

Die einfache Antwort ist, dass man das wohl aus demselben Grund macht, aus dem man auch Text in ein Canvas schreibt (siehe dazu den Abschnitt »Text« auf Seite 66). Unser Canvas-Koordinatendiagramm (siehe dazu den Abschnitt »Canvas-Koordinaten« auf Seite 62) schloss Text, Linien und Figuren ein. Der Text-auf-Canvas-Aspekt war nur ein Baustein in einem größeren Werk. Bei einem komplexeren Diagramm könnte man `drawImage()` beispielsweise einsetzen, um Symbole, Logos oder andere Grafiken zu zeichnen.

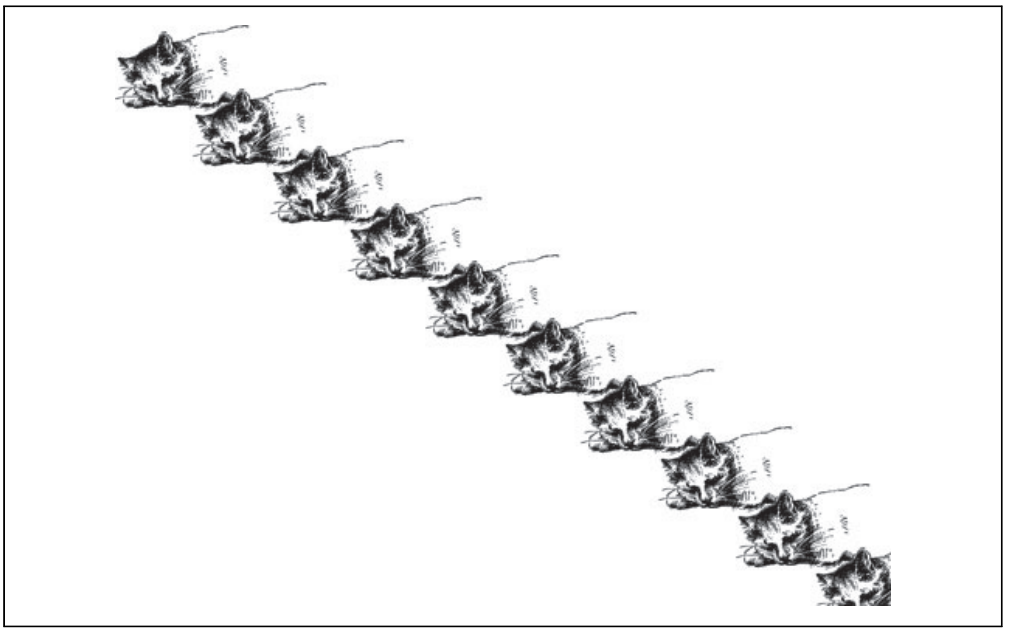


Abbildung 4-15: Eine Katzenfamilie

Was ist mit dem IE?

Microsofts Internet Explorer (bis einschließlich Version 8, der die aktuelle Version war, als dies geschrieben wurde) bietet keine Unterstützung der Canvas-API. Er unterstützt allerdings eine proprietäre Microsoft-Technologie namens VML, die ähnlich Dinge tun kann wie das `<canvas>`-Element. Und so wurde *excanvas.js* geboren.

ExplorerCanvas (*http://code.google.com/p/explorercanvas/*) – *excanvas.js* – ist eine Open Source-JavaScript-Bibliothek unter der Apache-Lizenz, die die Canvas-API im Internet Explorer nachbildet. Wenn Sie sie verwenden wollen, müssen Sie am Anfang Ihrer Seite folgendes `<script>`-Element einfügen:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Dive Into HTML5</title>
  <!--[if IE]>
    <script src="excanvas.js"></script>
  <![endif]-->
</head>
<body>
  ...
</body>
</html>
```

<!--[if IE]> und <![endif]--> sind bedingte Kommentare. Der Internet Explorer interpretiert sie als if-Anweisung: »Wenn der aktuelle Browser eine Version des Internet Explorer ist, führe diesen Block aus.« Andere Browser behandeln den gesamten Block als HTML-Kommentar. Die Folge ist, dass der Internet Explorer das Skript *excanvas.js* herunterlädt und ausführt, andere Browser es aber vollständig ignorieren (nicht herunterladen, nicht ausführen, nichts). Das bewirkt, dass Ihre Seite in Browsern, die die Canvas-API nativ unterstützen, schneller geladen wird.

Haben Sie das Skript *excanvas.js* in das <head>-Element Ihrer Seite aufgenommen, müssen Sie nichts mehr tun, um auch den Belangen des IE zu genügen. Sie können einfach <canvas>-Elemente in Ihr Markup aufnehmen oder dynamisch mit JavaScript erstellen. Folgen Sie lediglich den Anweisungen in diesem Kapitel, um den Zeichenkontext eines <canvas>-Elements abzurufen, und schon können Sie loslegen und Figuren, Text oder Muster zeichnen.

Schön wäre es. Aber leider stimmt das nicht ganz. Folgende Einschränkungen müssen Sie beachten:

- Verläufe (siehe dazu den Abschnitt »Verläufe« auf Seite 70) können nur linear sein. Radiale Verläufe werden nicht unterstützt.
- Muster müssen in beide Richtungen wiederholt werden.
- Beschneidungskanten werden nicht unterstützt.
- Nicht proportionale Skalierung skaliert Striche nicht korrekt.
- Es läuft langsam. Das sollte eigentlich niemanden sonderlich überraschen, da der JavaScript-Parser des Internet Explorer bekanntermaßen langsamer als der jedes anderen Browsers ist. Und wenn Sie dann auch noch beginnen, komplexe Figuren über eine JavaScript-Bibliothek zu zeichnen, die die Befehle für eine ganz andere Technologie übersetzt, wird das alles naturgemäß ausgebremst. Bei einfachen Beispielen, dem Zeichnen einiger Linien oder der Transformation eines Bilds, wird Ihnen der Leistungsabfall kaum auffallen. Aber er wird Ihnen sofort in die Augen springen, wenn Sie sich an Canvas-basierte Animationen und andere ausgefallene Dinge wagen.

Der Einsatz von *excanvas.js* hat noch einen weiteren Haken. Das ist ein Problem, auf das ich gestoßen bin, als ich die Beispiele für dieses Kapitel erstellte. ExplorerCanvas initialisiert seine eigene Pseudo-Canvas-Schnittstelle automatisch jedes Mal, wenn Sie das Skript *excanvas.js* in Ihre HTML-Seite einschließen. Das aber bedeutet nicht, dass der Internet Explorer sie dann auch unmittelbar nutzen kann. In bestimmten Situationen kann eine Race-Condition auftreten, bei der die Pseudo-Canvas-Schnittstelle fast, aber eben nur fast, einsatzbereit ist. Das wichtigste Symptom für diesen Zustand ist, dass sich der Internet Explorer darüber beschwert, dass das Objekt eine Eigenschaft oder Methode nicht unterstützt, wenn Sie versuchen, etwas mit einem <canvas>-Element zu tun, beispielsweise den Zeichenkontext abzurufen.

Die einfachste Lösung ist, alle Canvas-bezogenen Aktionen aufzuschieben, bis das onload-Event ausgelöst wurde. Das kann eine Weile dauern. Wenn Ihre Seite viele Bilder oder

Videos enthält, verzögern diese das `onload`-Event. Aber es gibt ExplorerCanvas Zeit, seine magischen Transformation durchzuführen.

Ein vollständiges Beispiel

Halma ist ein Jahrhunderte altes Spiel, das es in vielen Varianten gibt. Für dieses Beispiel habe ich eine Solitär-Version von Halma mit neun Spielsteinen auf einem 9 × 9 Felder großen Spielfeld erstellt. Zu Anfang des Spiels bilden die Spielsteine ein 3 × 3 Felder großes Rechteck in der unteren linken Ecke des Spielfelds. Ziel des Spiels ist es, alle Spielsteine in so wenig Zügen wie möglich so zu verschieben, dass sie ein 3 × 3 Felder großes Quadrat in der oberen rechten Ecke des Spielfelds belegen.

In Halma gibt es zwei Arten zulässiger Spielzüge:

- Einen Stein auf ein angrenzendes leeres Feld bewegen. Ein »leeres« Feld ist ein Feld, auf dem aktuell kein Stein steht. Ein angrenzendes Feld ist ein Feld, das nördlich, südlich, östlich, westlich, nordwestlich, nordöstlich, südwestlich oder südöstlich an das aktuelle Feld anschließt. (Das Spiel springt nicht von der einen Seite zur anderen über. Befindet sich ein Stein in der äußersten linken Spalte, kann er sich nicht nach Westen, Nordwesten oder Nordosten bewegen. Ist ein Stein in der untersten Reihe, kann er nicht nach Süden, Südosten oder Südwesten gehen.)
- Mit einem Stein über einen anderen Stein auf einem angrenzenden Feld springen und das so oft wiederholen, wie es möglich ist. Das bedeutet, dass es als ein Zug zählt, wenn Sie über einen Stein auf einem angrenzenden Feld springen und dann wieder über einen weiteren Stein, der sich auf einem an Ihre neue Position angrenzenden Feld befindet. Genau gesagt, eine beliebige Anzahl von Sprungen zählt als ein Zug. (Da es Ziel des Spiels ist, die Anzahl an Zügen so gering wie möglich zu halten, geht es bei Halma also darum, lange Ketten verteilter Spielsteine aufzubauen und diese dann so zu nutzen, dass andere Steine über sie hinweg mit Sprüngen eine möglichst große Strecke zurücklegen können.)

Abbildung 4-16 ist ein Screenshot des Spiels selbst. Sie können es auch online spielen (*http://diveintohtml5.org/examples/canvas-halma.html*), wenn Sie mit den Entwicklerwerkzeugen Ihres Browsers einen Blick darauf werfen wollen.

Wie das alles funktioniert? Ich kann Ihnen gar nicht sagen, wie ich mich über diese Frage freue. Ich werde Ihnen hier nicht den gesamten Code vorführen (Sie können ihn sich unter *http://diveintohtml5.org/examples/halma.js* ansehen). Den größten Teil des Codes für den Spielablauf werde ich überspringen. Aber ich möchte einige Teile daraus hervorheben, die sich um das Zeichnen auf das Canvas kümmern und auf die Mausklicks auf das `<canvas>`-Element reagieren.

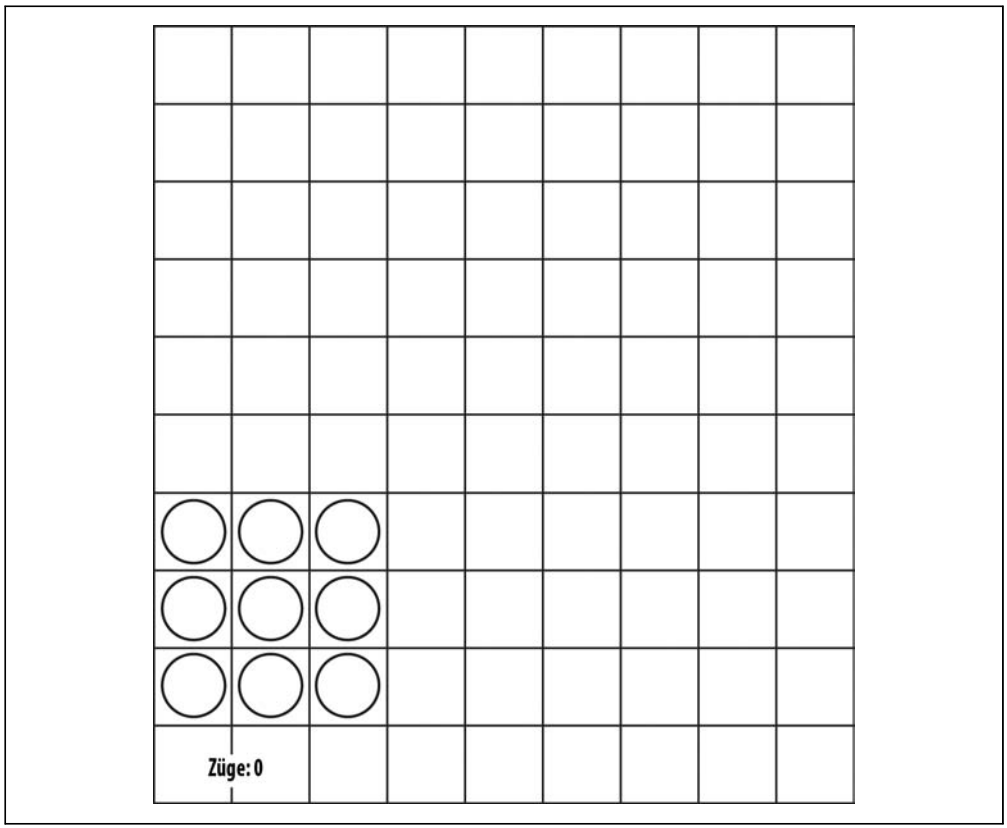


Abbildung 4-16: Startposition eines Halma-Spiels

Während die Seite geladen wird, initialisieren wir das Spiel, indem wir die Dimensionen des <canvas> selbst festlegen und eine Referenz auf seinen Zeichenkontext speichern:

```
gCanvasElement.width = kPixelWidth;
gCanvasElement.height = kPixelHeight;
gDrawingContext = gCanvasElement.getContext("2d");
```

Dann machen wir etwas, das Sie bislang noch nicht gesehen haben – wir fügen dem <canvas>-Element einen Event-Listener hinzu, der Click-Events überwacht:

```
gCanvasElement.addEventListener("click", halmaOnClick, false);
```

Die Funktion `halmaOnClick()` wird aufgerufen, wenn der Benutzer irgendwo in das Canvas klickt. Ihr Argument ist ein `MouseEvent`-Objekt, das Informationen dazu enthält, wohin der Benutzer geklickt hat:

```
function halmaOnClick(e) {
    var cell = getCursorPosition(e);

    // Was hier folgt, dient nur noch der Steuerung des Spielablaufs.
    for (var i = 0; i < gNumPieces; i++) {
```

```
            if ((gPieces[i].row == cell.row) &&
                (gPieces[i].column == cell.column)) {
                clickOnPiece(i);
                return;
            }
        }
        clickOnEmptyCell(cell);
    }
```

Der nächste Schritt ist, anhand des `MouseEvent`-Objekts zu berechnen, auf welches Feld des Halma-Bretts gerade geklickt wurde. Das Halma-Spiel nimmt das gesamte Canvas ein. Jeder Klick ist also ein Klick irgendwo auf das Spielbrett. Wir müssen herausfinden, wo. Das ist nicht ganz einfach, weil Maus-Events in beinahe allen Browsern unterschiedlich implementiert werden:

```
function getCursorPosition(e) {
    var x;
    var y;
    if (e.pageX || e.pageY) {
      x = e.pageX;
      y = e.pageY;
    }
    else {
      x = e.clientX + document.body.scrollLeft +
            document.documentElement.scrollLeft;
      y = e.clientY + document.body.scrollTop +
            document.documentElement.scrollTop;
    }
```

Die `x`- und `y`-Koordinaten, die wir jetzt haben, sind relativ zum Dokument (d.h. zur ganzen HTML-Seite). Das, was wir brauchen, sind aber Koordinaten in Bezug auf das Canvas. Diese erhalten wir so:

```
x -= gCanvasElement.offsetLeft;
y -= gCanvasElement.offsetTop;
```

Jetzt haben wir `x`- und `y`-Koordinaten in Bezug auf das Canvas (siehe dazu den Abschnitt »Canvas-Koordinaten« auf Seite 62). Das heißt, wenn `x` gleich `0` ist und `y` gleich `0` ist, wissen wir, dass der Benutzer gerade auf das Pixel oben links im Canvas geklickt hat.

Mit diesen Informationen können wir das Feld auf dem Spielbrett berechnen, auf das geklickt wurde, und dann entsprechend reagieren:

```
    var cell = new Cell(Math.floor(y/kPieceWidth),
                        Math.floor(x/kPieceHeight));
    return cell;
}
```

Wow! Maus-Events sind eine harte Nuss. Aber Sie können die gleiche Logik (eigentlich genau diesen Code) in allen Canvas-basierten Anwendungen nutzen. Denken Sie daran: Mausklick → dokumentbezogene Koordinaten → Canvas-bezogene Koordinaten → anwendungsspezifischer Code.

Okay, schauen wir uns die eigentliche Zeichenroutine an. Weil die Grafiken so einfach sind, habe ich beschlossen, dass Spielbrett vollständig zu löschen und neu zu zeichnen, wenn sich irgendetwas im Spiel ändert. Das ist nicht unbedingt erforderlich. Der Zeichenkontext hält alles fest, was zuvor auf ihn gezeichnet wurde, selbst wenn der Benutzer es außer Sicht scrollt oder den Tab wechselt und dann später zurückkehrt. Wenn Sie Canvas-basierte Anwendungen mit komplexeren Grafiken (beispielsweise ein Arkade-Spiel) erstellen, können Sie die Leistung verbessern, indem Sie festhalten, welche Bereiche des Canvas betroffen sind, und nur diese betroffenen Bereiche neu zeichnen. Aber das geht über den Horizont dieses Buchs hinaus. Hier ist der Code, der das Brett löscht:

```
gDrawingContext.clearRect(0, 0, kPixelWidth, kPixelHeight);
```

Die Routine zum Zeichnen des Spielbretts sollte vertraut aussehen. Sie hat große Ähnlichkeiten damit, wie wir unser Canvas-Koordinatendiagramm gezeichnet haben (siehe dazu den Abschnitt »Canvas-Koordinaten« auf Seite 62):

```
gDrawingContext.beginPath();

/* Vertikale Linien */
for (var x = 0; x <= kPixelWidth; x += kPieceWidth) {
    gDrawingContext.moveTo(0.5 + x, 0);
    gDrawingContext.lineTo(0.5 + x, kPixelHeight);
}

/* Horizontale Linien */
for (var y = 0; y <= kPixelHeight; y += kPieceHeight) {
    gDrawingContext.moveTo(0, 0.5 + y);
    gDrawingContext.lineTo(kPixelWidth, 0.5 +  y);
}

/* Zeichnen! */
gDrawingContext.strokeStyle = "#ccc";
gDrawingContext.stroke();
```

Richtig interessant wird es, wenn wir die einzelnen Spielsteine zeichnen müssen. Ein Spielstein ist ein Kreis, etwas also, das wir zuvor noch nicht gezeichnet haben. Und wenn der Benutzer in Vorbereitung des Zugs auf einen Spielstein klickt, soll dieser Kreis außerdem noch gefüllt werden. Hier folgt der Code, das Argument `p` repräsentiert einen Spielstein, der die Eigenschaften `row` und `column` hat, um anzuzeigen, wo er sich aktuell auf dem Spielfeld befindet. Wir nutzen einige spielspezifische Konstanten, um (`Spalte`, `Zeile`) in Canvas-bezogene (`x`, `y`)-Koordinaten zu übersetzen, und zeichnen dann einen Kreis, den wir (wenn der Spielstein ausgewählt ist) einfarbig füllen:

```
function drawPiece(p, selected) {
    var column = p.column;
    var row = p.row;
    var x = (column * kPieceWidth) + (kPieceWidth/2);
    var y = (row * kPieceHeight) + (kPieceHeight/2);
    var radius = (kPieceWidth/2) - (kPieceWidth/10);
```

Das ist das Ende des spielspezifischen Logik. Jetzt haben wir (`x`, `y`)-Koordinaten in Bezug auf das Canvas für den Mittelpunkt des zu zeichnenden Kreises. Die Canvas-API kennt

keine `circle()`-Methode. Aber es gibt eine `arc()`-Methode. Und was ist ein Kreis anderes als ein geschlossener Kreisbogen? Erinnern Sie sich noch an Ihre Geometriestunden? Die `arc()`-Methode erwartet einen Mittelpunkt (`x, y`), einen Radius, einen Start- und einen Endwinkel (beides in Radiant) sowie einen Schalter, der die Richtung anzeigt (`false` für »im Uhrzeigersinn«, `true` für »entgegen dem Uhrzeigersinn«). Wir können JavaScripts eingebautes `Math`-Modul nutzen, um die Winkel in Radiant zu berechnen:

```
gDrawingContext.beginPath();
gDrawingContext.arc(x, y, radius, 0, Math.PI * 2, false);
gDrawingContext.closePath();
```

Aber Moment! Es wurde noch nichts gezeichnet. Wie `moveTo()` und `lineTo()` ist `arc()` eine »Bleistiftmethode« (siehe dazu den Abschnitt »Pfade« auf Seite 63). Damit der Kreis tatsächlich gezeichnet und in »Tinte« verewigt wird, müssen wir `strokeStyle` setzen und `stroke()` aufrufen

```
gDrawingContext.strokeStyle = "#000";
gDrawingContext.stroke();
```

Was wir machen, wenn der Stein ausgewählt ist? Wir können den Pfad wiederverwenden, den wir für den Umriss des Steins erstellt haben, und den Kreis einfarbig füllen:

```
if (selected) {
    gDrawingContext.fillStyle = "#000";
    gDrawingContext.fill();
}
```

Und das war es so ziemlich. Der Rest des Programms ist spielspezifische Logik – gültige und ungültige Züge trennen, die Anzahl von Zügen festhalten, prüfen, ob das Spiel zu Ende ist ... Mit neuen Kreisen, ein paar geraden Linien und einem `onclick`-Handler haben wir ein vollständige Spiel in `<canvas>` erzeugt. Hurra!

Weitere Lektüre

- Canvas-Tutorial (*https://developer.mozilla.org/en/Canvas_tutorial*) im Mozilla Developer Center
- »HTML5 `canvas`–the basics« (*http://dev.opera.com/articles/view/html-5-canvas-the-basics/*) von Mihai Sucan
- Canvas Demos (*http://www.canvasdemos.com*): Demos, Tools und Einführungen zum HTML-`<canvas>`-Element
- »The `canvas` element« (*http://bit.ly/9JHzOf*) im Entwurf für den HTML5-Standard

KAPITEL 5

Video im Web

Einstieg

Jeder, der in den letzten vier Jahren irgendwann Youtube.com einen Besuch abgestattet hat, weiß, dass man Videos in Webseiten einbetten kann. Aber vor HTML5 gab es kein standardisiertes Verfahren dafür. Vermutlich wurden so gut wie alle Videos, die Sie im Web gesehen haben, mithilfe eines externen Plug-ins anzeigt – vielleicht QuickTime, vielleicht RealPlayer, vielleicht auch Flash. (YouTube nutzt Flash.) Diese Plug-ins spielen mit Ihrem Browser so gut zusammen, dass Sie häufig nicht einmal merken, dass sie genutzt werden – bis Sie versuchen, ein Video auf einer Plattform zu schauen, die das erforderliche Plug-in nicht unterstützt.

HTML5 definiert ein Standardverfahren zur Einbettung von Videos in Webseiten: das `<video>`-Element. Die Unterstützung für das `<video>`-Element ist immer noch im Reifungsprozess. Dieser Ausdruck sollte einen unerfreulichen Umstand etwas freundlicher formulieren: Noch funktioniert es nicht (zumindest nicht überall). Kein Grund zur Verzweiflung! Es gibt Unmengen an Alternativen, Ausweichmöglichkeiten und Optionen. Tabelle 5-1 zeigt, welche Browser das `<video>`-Element unterstützen, als dies geschrieben wurde.

Tabelle 5-1: Unterstützungsmatrix für das <video>-Element

IE9	IE8	IE7	Firefox 3.5	Firefox 3.0	Safari 4	Safari 3	Chrome	Opera
✓	.	.	✓	.	✓	✓	✓	✓

Aber die Unterstützung des `<video>`-Elements selbst ist nur ein Aspekt der Geschichte. Bevor wir uns HTML5-Video ansehen können, müssen Sie erst noch etwas über Video selbst lernen. (Wenn Sie sich damit bereits auskennen, können Sie gleich zum Abschnitt »Was im Web funktioniert« auf Seite 91 springen.)

Videocontainer

Vielleicht denken Sie bei Videodateien an »AVI-Dateien« oder »MP4-Dateien«. Aber eigentlich sind »AVI« und »MP4« nur Containerformate. So wie eine ZIP-Datei jede Art

von Datei enthalten kann, so definieren Videocontainerformate nur, wie etwas in ihnen gespeichert wird, nicht welche Art von Daten gespeichert werden. (Auch das ist wieder einmal komplizierter, als wir es hier beschreiben, weil nicht alle Videostreams mit allen Containerformaten kompatibel sind, aber das muss Sie für den Augenblick nicht kümmern.)

Eine Videodatei enthält in den meisten Fällen mehrere Spuren (Tracks): eine Videospur (ohne Ton) und eine oder mehrere Audiospuren (ohne Bild). Diese Spuren sind üblicherweise aufeinander bezogen. Eine Audiospur enthält Markierungen, mit deren Hilfe der Ton mit dem Bild synchronisiert wird. Einzelne Spuren können Metadaten enthalten, die beispielsweise das Seitenverhältnis für eine Videospur oder die Sprache einer Audiospur angeben. Container selbst können ebenfalls Metadaten enthalten, beispielsweise den Titel des Videos, ein Coverbild, Folgennummern (bei Fernsehserien) und so weiter.

Es gibt jede Menge Containerformate für Videos. Einige der beliebtesten sind:

MPEG-4
: Hat üblicherweise die Dateinamenserweiterung *.mp4* oder *.m4v*. Der MPEG-4-Container basiert auf Apples älterem QuickTime-Container (*.mov*). Filmtrailer auf Apples Website nutzen immer noch den älteren QuickTime-Container, aber Filme, die Sie über iTunes ausleihen, werden in einem MPEG-4-Container ausgeliefert.

Flash Video
: Hat üblicherweise die Dateinamenserweiterung *.flv*. Flash Video wird, Überraschung, von Adobes Flash genutzt. Vor Flash 9.0.60.184 (d.h. Flash Player 9, Update 3) war das das einzige Containerformat, das Flash unterstützte. Jüngere Flash-Versionen unterstützen auch den MPEG-4-Container.

Ogg
: Hat üblicherweise die Dateinamenserweiterung *.ogv*. Ogg ist ein offener Standard, der Open Source-freundlich ist und von keinen bekannten Patenten beeinträchtigt wird. Firefox 3.5, Chrome 4 und Opera 10.5 bieten native Unterstützung ohne plattformspezifische Plug-ins für das Ogg-Containerformat, für Ogg-Video (»Theora« genannt) und Ogg-Audio (»Vorbis« genannt). Auf dem Desktop wird Ogg von Haus aus von allen wichtigen Linux-Distributionen unterstützt. Auf dem Mac oder unter Windows können Sie es nutzen, indem Sie die QuickTime-Komponenten beziehungsweise die DirectShow-Filter installieren. Außerdem kann es auf allen Plattformen mit dem ausgezeichneten VLC-Medien-Player VLC (*http://www.videolan.org/vlc/*) abgespielt werden.

WebM
: Hat die Dateinamenserweiterung *.webm*. WebM ist ein neues Containerformat, das technisch gesehen große Ähnlichkeit mit einem anderen Format namens Matroska hat. WebM wurde bei der Google I/O 2010 angekündigt. Es wurde für die ausschließliche Verwendung des VP8-Videocodecs und des Vorbis-Audiocodecs konzipiert. (Mehr dazu gleich.) Es wird in den kommenden Versionen von Chromium, Google Chrome, Mozilla Firefox und Opera nativ und ohne plattformspezifische Plug-ins

unterstützt werden. Adobe hat angekündigt, dass die nächste Version von Flash WebM-Video ebenfalls unterstützen wird.

Audio Video Interleave
: Hat die Dateinamenserweiterung *.avi*. Das AVI-Containerformat wurde in einfacheren Zeiten von Microsoft entwickelt, als schon der Umstand, dass Computer Videos abspielen können, fast für Verwunderung sorgte. Offiziell unterstützt dieser Container viele der Funktionen neuerer Containerformate nicht. Offiziell bietet er keine Unterstützung für Videometadaten. Offiziell unterstützt er auch die meisten der heute gebräuchlichen modernen Video- und Audiocodecs nicht. Mit der Zeit haben verschiedene Unternehmen versucht, ihn auf in der Regel inkompatible Weise dazu zu bringen, dieses oder jenes zu unterstützen. Es ist immer noch das Standardcontainerformat für beliebte Encoder wie beispielsweise MEncoder (*http://www.mplayerhq.hu/DOCS/HTML/en/encoding-guide.html*).

Videocodecs

Wenn Sie sagen, dass Sie »ein Video schauen«, sprechen Sie wahrscheinlich von einer Kombination aus Video- und Audio-Stream. Aber Sie haben keine zwei separaten Dateien, Sie haben nur »das Video«. Vielleicht ist es eine AVI-Datei oder eine MP4-Datei. Wie im vorangegangenen Abschnitt beschrieben, sind das nur Containerformate, die, ähnlich wie eine ZIP-Datei, mehrere Dateien möglicherweise unterschiedlichen Typs enthält. Das Containerformat definiert, wie der Video- und der Audiostream in einer einzigen Datei gespeichert werden.

Wenn Sie »ein Video schauen«, macht Ihr Videoplayer mehrere Dinge gleichzeitig:

- Er interpretiert das Containerformat, um herauszufinden, welche Video- und Audiospuren verfügbar sind und wie sie in der Datei gespeichert sind, damit er die Daten finden kann, die er gleich dekodieren muss.
- Er dekodiert den Videostream und zeigt auf dem Bildschirm eine Folge von Bildern an.
- Er dekodiert den Audiostream und sendet die Geräusche an Ihre Lautsprecher.

Ein *Videocodec* ist ein Algorithmus, gemäß dem ein Videostream kodiert ist. Das heißt, er gibt an, wie Punkt 2 aus dieser Liste zu erledigen ist. (Das Wort »Codec« ist ein Kofferwort, eine Kombination den Wörtern »Coder« und »Decoder«.) Ihr Videoplayer *dekodiert* den Videostream gemäß dem Videocodec und zeigt dann eine Abfolge von Bildern oder »Frames« auf dem Bildschirm an. Die meisten modernen Videocodecs nutzen diverse Tricks, um die Menge an Informationen klein zu halten, die erforderlich ist, um nacheinander diese Frames anzuzeigen. Anstatt die Frames einzeln zu speichern (wie Screenshots), speichern sie beispielsweise nur die Unterschiede zwischen den Frames. Da sich bei den meisten Videos von Frame zu Frame kaum etwas ändert, ermöglicht das hohe Kompressionsraten, die zu kleineren Dateigrößen führen.

Es gibt *verlustbehaftete* und *verlustfreie* Videocodecs. Verlustfreies Video ist viel zu groß, als dass es sich im Web vernünftig nutzen ließe. Deswegen werde ich mich auf verlustbehaftete Codecs konzentrieren. Bei einem verlustbehafteten Videocodec gehen bei der Kodierung unwiederbringlich Informationen verloren. Wie beim Kopieren von Audiokassetten verlieren Sie bei jeder neuen Kodierung des Videos Informationen und verschlechtern damit die Qualität. Statt des Rauschens, das Sie sich dadurch bei Audiokassetten einhandeln, führt das mehrfache Kodieren von Videos zu einem würfeligen Bild, insbesondere bei stark bewegten Szenen. (Das kann auch bei der ersten Kodierung der ursprünglichen Quelle passieren, wenn Sie einen schlechten Videocodec wählen oder Ihrem Codec die falschen Parameter übergeben.) Das Gute an verlustbehafteten Codecs ist jedoch, dass sie erstaunliche Kompressionsraten ermöglichen, viele Möglichkeiten zum Schummeln bieten und die »Würfeligkeit« beim Abspielen glätten, damit sie für das menschliche Auge beim Abspielen weniger auffällig ist.

Es gibt eine Menge Videocodecs. Die drei wichtigsten sind H.264, Theora und VP8.

H.264

H.264 (*http://en.wikipedia.org/wiki/H.264*) ist auch als »MPEG-4 part 10«, oder »MPEG-4 AVC« oder »MPEG-4 Advanced Video Coding« bekannt. H.264 wurde von der MPEG-Gruppe entwickelt und 2003 standardisiert. Dabei wurde versucht, einen einzigen Codec so zu gestalten, dass er gleichermaßen für Geräte mit geringer Bandbreite und wenig Rechenkraft (Handys) geeignet ist wie für Geräte mit großer Bandbreite und großer Rechenkraft (moderne Desktopcomputer) – und natürlich auch alles dazwischen. Um das zu erreichen, wurde der H.264-Standard in »Profile« gegliedert, die jeweils einen Satz optionaler Funktionen beschreiben, die einen bestimmten Kompromiss zwischen Komplexität und Dateigröße bedingen. Höhere Profile nutzen mehr optionale Funktionen und bieten bessere visuelle Qualität bei kleineren Dateigrößen. Ihre Kodierung nimmt mehr Zeit in Anspruch, und für eine Dekodierung in Echtzeit ist mehr Rechenkraft erforderlich.

Um Ihnen eine grobe Vorstellung von der Spannbreite dieser Profile zu geben: Apples iPhone unterstützt das Baseline-Profil, AppleTV unterstützt die Profile Baseline und Main, und Adobe Flash auf Desktop-PCs unterstützt die Profile Baseline, Main und High. YouTube (das Google, meinem Arbeitgeber, gehört) nutzt H.264, um High-Definition-Videos zu kodieren, die über Adobe Flash abgespielt werden. YouTube unterstützt auch H.264-kodierte Videos für Mobilgeräte einschließlich Apples iPhone und Handys mit Googles Android-Betriebssystem. Außerdem ist H.264 einer der Codecs, die von der Blu-ray-Spezifikation vorgeschrieben werden. Blu-ray-Discs, die ihn verwenden, nutzen gewöhnlich das High-Profil.

Die meisten Nicht-PC-Geräte, die H.264-Video abspielen (einschließlich des iPhones und eigenständiger Blu-ray-Player) führen die Dekodierung auf einem speziellen Chip durch, da ihre Hauptprozesseren nicht einmal annähernd die Rechenkraft besitzen, die erforderlich ist, um Videos in Echtzeit zu dekodieren. Viele Desktopgrafikkarten unterstützen ebenfalls hardwareseitig die Dekodierung von H.264. Es gibt eine Reihe konkurrierender H.264-Encoder, unter anderem auch die Open Source-Bibliothek x264. Der H.264-Stan-

dard ist patentbelastet. Die Lizensierung läuft über die MPEG LA-Gruppe. H.264-Video kann in die meisten beliebten Containerformate (siehe dazu den Abschnitt »Videocontainer« auf Seite 83) eingebettet werden, einschließlich MP4 (das primär von Apples iTunes Store verwendet wird) und MKV (das primär von nicht kommerziellen Videoenthusiasten verwendet wird).

Theora

Der Theora-Codec (*http://en.wikipedia.org/wiki/Theora*) entwickelte sich aus dem VP3-Codec und wurde später von der Xiph.org Foundation weiterentwickelt. Theora ist ein kostenloser Codec und von keinen Patenten belastet, im Unterschied zu den ursprünglichen VP3-Patenten, die kostenlos lizenziert wurden. Obwohl der Standard seit 2004 »fixiert« ist, hat das Theora-Projekt (das einen Open Source-Referenz-Encoder und -Decoder einschließt) Version 1.0 erst im November 2008 und Version 1.1 im September 2009 veröffentlicht.

Theora-Video kann in jedes Containerformat eingebettet werden, wird aber am häufigsten in einem Ogg-Container genutzt. Alle wichtigeren Linux-Distributionen unterstützen Theora von Haus aus, und Mozilla Firefox 3.5 schließt native Unterstützung für Theora-Video in einem Ogg-Container ein. Mit »nativ« meine ich die »Unterstützung auf allen Plattformen ohne plattformspezifische Plug-ins«. Sie können Theora-Video auch auf Windows oder Mac OS X abspielen, wenn Sie die Open Source-Decoder-Software von Xiph.org installiert haben.

VP8

VP8 (*http://en.wikipedia.org/wiki/VP8*) ist ein weiterer Videocodec von On2, demselben Unternehmen, das ursprünglich VP3 (später Theora) entwickelt hatte. Technisch hat es eine ähnliche Qualität wie H.264 Baseline, bietet aber viel Potenzial für weitere Verbesserungen.

2010 erwarb Google On2 und veröffentlichte die Videocodec-Spezifikation sowie einen Beispiel-Encoder und -Decoder als Open Source. Ein Teil dieses Schritts war, dass Google gleichzeitig alle Patente »öffnete«, die On2 für VP8 beantragt hatte, indem es sie kostenfrei lizenzierte. (Mehr kann man bei Patenten nicht erwarten – sind sie einmal ausgestellt worden, kann man sie nicht einfach »freigeben« oder annullieren. Um sie Open Source-freundlich zu machen, lizenziert man sie lizenzgebührenfrei. Dann kann jeder die von den Patenten betroffenen Technologien nutzen, ohne etwas zu zahlen oder Patentlizenzen auszuhandeln.) Seit dem 19. Mai 2010 ist *VP8 ein lizenzgebührenfreier, moderner Codec, der von keinen Patenten beeinträchtigt wird*, von den Patenten abgesehen, die On2 (jetzt Google) bereits lizenzgebührenfrei lizenziert hat.

Audiocodecs

Wenn Sie sich nicht auf Filme beschränken möchten, die vor 1927 gedreht wurden, werden Sie sicherlich wollen, dass Ihre Videodatei eine Audiospur enthält. Wie Videocodecs sind *Audiocodecs* Kodierungsalgorithmen, die, in diesem Fall, für Audiostreams genutzt werden. Wie bei Videocodecs gibt es *verlustbehaftete* und *verlustfreie* Audiocodecs. Und wie verlustfreies Video ist auch verlustfreies Audio zu groß, um im Web von Nutzen zu sein. Ich werde mich also auch hier auf die verlustbehaftete Variante konzentrieren.

Tatsächlich können wir die Auswahl sogar noch weiter einschränken, weil es verschiedene Kategorien verlustbehafteter Audiocodecs gibt. Audio wird an vielen Punkten genutzt, an denen kein Video verwendet wird (beim Telefonieren beispielsweise), und es gibt eine ganze Kategorie von Audiocodecs, die für die Sprachwiedergabe optimiert sind. So einen Codec würden Sie nicht einsetzen, um eine Musik-CD zu rippen. Das Ergebnis würde klingen wie das Singen einer Vierjährigen über ein Babyphone. Aber Sie würden ihn für Asterisk-PBX nutzen, weil dabei die Bandbreite wertvoll ist und diese Codecs menschliche Sprache auf einen Bruchteil dessen komprimieren können, was mit einem allgemeineren Codec möglich wäre. Aufgrund fehlender Unterstützung in Browsern, sowohl nativ als auch über externe Plug-ins, sind sprachoptimierte Audiocodecs im Web nie in Schwung gekommen. Deswegen werde ich mich auf *allgemeine verlustbehaftete Audiocodecs* konzentrieren.

Wie ich in Abschnitt »Videocodecs« auf Seite 85 sagte, macht Ihr Computer mehrere Dinge auf einmal, wenn Sie »ein Video schauen«:

1. Das Containerformat interpretieren.
2. Den Videostream dekodieren.
3. Den Audiostream dekodieren und an Ihre Lautsprecher senden.

Der *Audiocodec* gibt an, wie Punkt 3 ausgeführt wird, d.h., wie der Audiostream dekodiert und in die digitalen Wellenformen transformiert wird, die Ihre Lautsprecher dann in Klang umwandeln. Wie bei Videocodecs gibt es alle möglichen Tricks, um die Menge an Informationen zu reduzieren, die im Audiostream gespeichert wird. Und da wir uns hier mit verlustbehafteten Audiocodecs befassen, gehen während des Aufnahme→Kodierung→Dekodierung→Hören-Prozesses Informationen verloren. Unterschiedliche Audiocodecs lassen dabei unterschiedliche Informationen fallen. Sie verfolgen dabei aber alle das gleiche Ziel: Ihre Ohren dazu zu bringen, dass sie nicht bemerken, dass da Teile fehlen.

Ein Konzept, dass es bei Audio gibt, bei Video aber nicht, sind *Kanäle*. Wir senden Ton an Ihre Lautsprecher, stimmt's? Und wie viele Lautsprecher haben Sie? Wenn Sie an Ihrem Rechner sitzen, haben Sie vielleicht nur zwei: einen auf der linken Seite, einen auf der rechten. Mein Desktop hat drei: links, rechts und einen weiteren auf dem Boden. Sogenannte Surround-Sound-Systeme können sogar sechs oder mehr Lautsprecher haben. Jeder Lautsprecher erhält einen Kanal der Aufzeichnung. Die Theorie ist, dass Sie, wenn Sie in der Mitte zwischen den sechs Lautsprechern sitzen, quasi von sechs

verschiedenen Tonkanälen umzingelt sind, die von Ihrem Gehirn so zusammengefügt werden, dass Sie den Eindruck haben, Sie befänden sich mitten im Geschehen. Ob das funktioniert? Eine viele Millionen Euro schwere Industrie scheint daran zu glauben.

Die meisten allgemeinen Audiocodecs kommen mit zwei Tonkanälen klar. Während der Aufzeichnung wird der Ton in einen rechten und einen linken Kanal getrennt. Beim Kodieren werden beide Kanäle im selben Audiostream gespeichert, beim Dekodieren werden sie dann jeweils an den entsprechenden Lautsprecher geschickt. Einige Audiocodecs kommen mit mehr als zwei Kanälen klar und sie halten nach, welcher Kanal welcher ist, damit Ihr Player den richtigen Ton an den richtigen Lautsprecher senden kann.

Es gibt eine Menge Audiocodecs. Hatte ich nicht schon gesagt, es gäbe eine Menge Videocodecs? Vergessen Sie das. Es gibt eine *Unmenge* Audiocodecs, aber im Web sind eigentlich nur drei von Interesse: MP3, AAC und Vorbis.

MPEG-1 Audio Layer 3

MPEG-1 Audio Layer 3 (*http://en.wikipedia.org/wiki/MPEG-1_Audio_Layer_3*) kürzt man üblicherweise mit »MP3« ab. Sollten Sie von MP3 noch nicht gehört haben, weiß ich wirklich nicht, wie ich Ihnen helfen soll. Aldi verkauft gelegentlich tragbare Musikplayer und nennt sie »MP3-Player.« *Aldi.* Na gut ...

MP3s können bis zu zwei Tonkanäle enthalten. Sie können mit unterschiedlichen *Bitraten* kodiert werden: 64 kbps, 128 kbps, 192 kbps und noch einigen mehr, von 32 bis 320. Je höher die Bitrate, umso größer die Datei und umso besser die Qualität – allerdings ist das Verhältnis von Bitrate und Qualität nicht linear. (128 kbps klingen mehr als doppelt so gut im Vergleich zu 64 kbps, aber 256 kbps klingen nicht doppelt so gut wie 128 kbps.) Außerdem erlaubt das (1991 standardisierte) MP3-Format *Kodierung mit variabler Bitrate*. Das bedeutet, dass einige Teile des kodierten Streams stärker komprimiert werden als andere. Beispielsweise kann die Pause zwischen zwei Noten mit einer sehr geringen Bitrate kodiert werden, und wenn später mehrere Instrumente gemeinsam einen komplexen Akkord spielen, kann sie stark erhöht werden. MP3s können auch mit einer konstanten Bitrate kodiert werden, was man wenig überraschend *Kodierung mit konstanter Bitrate* nennt.

Der MP3-Standard definiert ganz genau, wie MP3s zu kodieren sind (definiert allerdings nicht so genau, wie man sie dekodieren muss). Die verschiedenen Kodierer nutzen unterschiedliche psychoakustische Modelle, die zu äußerst unterschiedlichen Ergebnissen führen, aber alle von den gleichen Abspielgeräten dekodiert werden können. Das Open Source-Projekt LAME ist der beste freie Kodierer und vielleicht der beste Kodierer überhaupt (außer für ganz geringe Bitraten).

Das MP3-Format ist patentbelastet. Das erklärt auch, warum Linux MP3-Dateien im Grundzustand nicht abspielen kann. Fast alle tragbaren Musikspieler unterstützen eigenständige MP3-Dateien, und MP3-Audiostreams können in alle Videocontainer eingebettet werden. Adobe Flash kann eigenständige MP3-Dateien und MP3-Audiostreams in einem MP4-Videocontainer abspielen.

Advanced Audio Coding

Advanced Audio Coding (*http://en.wikipedia.org/wiki/Advanced_Audio_Coding*) bezeichnet man zärtlich als »AAC«. Das 1997 standardisierte Format sprang ins Rampenlicht, als Apple beschloss, es zum Standardformat im iTunes Store zu machen. Ursprünglich waren AAC-Dateien, die im iTunes Store »gekauft« wurden, mit Apples proprietärem DRM-Schema namens FairPlay verschlüsselt. Viele Songs im iTunes Store sind mittlerweile als ungeschützte AAC-Dateien verfügbar. Apple nennt das »iTunes Plus«, weil es so viel besser klingt, als alles andere als »iTunes Minus« zu bezeichnen. Das AAC-Format ist patentbelastet. Die Lizenzgebühren sind online einsehbar.

AAC soll bei gleicher Bitrate eine bessere Klangqualität als MP3 bieten und kann Audio mit beliebigen Bitraten kodieren. (MP3 ist auf eine begrenzte Anzahl von Bitraten mit einer Obergrenze von 320 kbps beschränkt.) AAC kann bis zu 48 Tonkanäle kodieren, was aber so gut wie niemand tut. Das AAC-Format unterscheidet sich von MP3 auch dadurch, dass es mehrere Profile definiert, auf ganz ähnlich Weise wie H.264 und aus den gleichen Gründen. Das Profil »low-complexity« soll in Echtzeit von Geräten mit geringer Rechenkraft abspielbar sein. Höhere Profile bieten bessere Klangqualität bei gleicher Bitrate zum Preis einer langsameren Kodierung und Dekodierung.

Alle aktuellen Apple-Produkte, einschließlich iPod, AppleTV und QuickTime, unterstützen bestimmte AAC-Profile in eigenständigen Audiodateien und bei Audiostreams in einem MP4-Videocontainer. Adobe Flash unterstützt alle AAC-Profile in MP4, so auch die Open Source-Videoplayer MPlayer und VLC. Für die Kodierung ist die FAAC-Bibliothek die Open Source-Option. Die Unterstützung dieser Bibliothek kann bei der Kompilierung in *mencoder* und *ffmpeg* angeschaltet werden.

Vorbis

Vorbis (*http://en.wikipedia.org/wiki/Vorbis*) wird oft »Ogg Vorbis« genannt, obgleich das technisch nicht richtig ist – »Ogg« ist lediglich ein Containerformat (siehe dazu den Abschnitt »Videocontainer« auf Seite 83), und Vorbis-Audiostreams können in andere Container eingebettet werden. Vorbis wird von keinen bekannten Patenten belastet und deswegen von Haus aus von den meisten wichtigen Linux-Distributionen und tragbaren Geräten unterstützt, die auf der Open Source-Firmware Rockbox laufen. Mozilla Firefox 3.5 unterstützt Vorbis-Audiodateien in einem Ogg-Container oder Ogg-Videos mit einer Vorbis-Audiospur. Android-Geräte können auch eigenständige Vorbis-Audiodateien abspielen. Vorbis-Audiostreams werden üblicherweise in einen Ogg- oder WebM-Container eingebettet, können aber auch in einen MP4- oder MKV-Container oder mit ein paar Tricks in AVI eingebettet werden. Vorbis unterstützt eine beliebige Anzahl von Tonkanälen.

Es gibt Open Source-Vorbis-Kodierer und -Dekodierer. Dazu zählen unter anderem der OggConvert-Dekodierer, der *ffmpeg*-Dekodierer, der aoTuV-Kodierer und der libvorbis-Dekodierer, außerdem die QuickTime-Komponenten für Mac OS X und die DirectShow-Filter für Windows.

Was im Web funktioniert

Sollten Ihnen die Augen noch nicht übergegangen sein, ergeht es Ihnen besser als den meisten. Wie Sie sehen können, ist Video (und Audio) eine komplizierte Angelegenheit – und das, obwohl ich die Sache in dieser Darstellung noch verkürzt habe! Ich bin mir sicher, dass Sie sich fragen, was all das mit HTML5 zu tun hat. HTML5 schließt ein `<video>`-Element zur Einbettung von Videos in Webseiten ein. Es gibt keine Einschränkung für den Videocodec, den Audiocodec oder das Containerformat, das Sie für Ihr Video verwenden können. Ein `<video>`-Element kann auf mehrere Videodateien verweisen, und der Browser wählt die erste Datei, die er tatsächlich abspielen kann. Welcher Browser welchen Container und welche Codecs unterstützt, das müssen Sie selbst herausfinden.

Als dies geschrieben wurde, sah die HTML5-Videolandschaft folgendermaßen aus:

- Mozilla Firefox (3.5 und später) unterstützt Theora-Video und Vorbis-Audio in einem Ogg-Container.
- Opera (10.5 und später) unterstützt Theora-Video und Vorbis-Audio in einem Ogg-Container.
- Google Chrome (3.0 und später) unterstützt Theora-Video und Vorbis-Audio in einem Ogg-Container. Er unterstützt außerdem H.264-Video (alle Profile) und AAC-Audio (alle Profile) in einem MP4-Container.
- Am 9. Juni 2010, zum Zeitpunkt des Schreibens dieses Kapitels, unterstützten die Entwicklerversionen von Google Chrome, Chromium, Mozilla Firefox und Opera VP8-Video und Vorbis-Audio in einem WebM-Container. (Aktuellere Informationen und Download-Links für WebM-kompatible Browser finden Sie bei *webmproject.org*.)
- Safari auf Macs und Windows-PCs (3.0 und höher) unterstützen alles, was QuickTime unterstützt. Theoretisch könnten Sie von Ihren Benutzern verlangen, dass sie externe QuickTime-Plug-ins installieren. Die Praxis zeigt, dass das nur wenige Benutzer machen. Ihnen bleiben also die Formate, die QuickTime »von Haus aus« unterstützt. Das ist eine lange Liste, sie schließt jedoch Theora-Video, Vorbis-Audio oder den Ogg-Container nicht ein. Aber QuickTime unterstützt H.264-Video (Main-Profil) und AAC-Audio in einem MP4-Container.
- Mobile Geräte wie Apples iPhone und Google Android-Handys unterstützen H.264-Video (Baseline-Profil) und AAC-Audio (Low-Complexity-Profil) in einem MP4-Container.
- Adobe Flash (9.0.60.184 und später) unterstützen H.264-Video (alle Profile) und AAC-Audio (alle Profile) in einem MP4-Container.
- Internet Explorer 9 wird noch nicht genau angegebene Profile von H.264-Video und AAC-Audio in einem MP4-Container unterstützen.
- Der Internet Explorer 8 bietet keine HTML5-Videounterstützung, aber so gut wie alle Internet Explorer-Benutzer haben das Adobe Flash-Plug-in. Später in diesem Kapitel werde ich Ihnen zeigen, wie Sie HTML5-Video nutzen und gegebenenfalls auf Flash ausweichen können.

Tabelle 5-2 bietet die Informationen oben in leichter verdaulicher Form.

Tabelle 5-2: Videocodec-Unterstützung in aktuellen Browsern

Codecs/Container	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	·	3.5+	·	5.0+	10.5+	·	·
H.264+AAC+MP4	·	·	3.0+	5.0+	·	3.0+	2.0+
WebM	·	·	·	·	·	·	·

In einem Jahr wird diese Landschaft ganz anders aussehen: WebM wird in mehreren Browsern implementiert sein. Diese Browser werden nicht experimentelle WebM-fähige Versionen bieten, und Benutzer werden auf die neuesten Versionen umsteigen. Die zu erwartende Codec-Unterstützung sehen Sie in Tabelle 5-3.

Tabelle 5-3: Videocodec-Unterstützung in kommenden Browsern

Codecs/Container	IE	Firefox	Safari	Chrome	Opera	iPhone	Android
Theora+Vorbis+Ogg	·	3.5+	·	5.0+	10.5+	·	·
H.264+AAC+MP4	·	·	3.0+	5.0+	·	3.0+	2.0+
WebM	9.0+[1]	4.0+	·	6.0+	11.0+	·	[2]

Und jetzt zum K.-o.-Schlag ...

Professor Markup sagt

Es gibt keine einzige Kombination von Containern und Codecs, die in allen HTML5-Browsern funktioniert.

Es ist wenig wahrscheinlich, dass sich das in der Zukunft ändern wird.

Wollen Sie Ihr Video auf allen Geräten und Plattformen verfügbar machen, müssen Sie es in unterschiedlichen Formaten kodieren.

Wenn Sie maximale Kompatibilität anstreben, wird Ihr Video-Workflow ungefähr so aussehen:

1. Sie erstellen eine Version, die Theora-Video und Vorbis-Audio in einem Ogg-Container nutzt.
2. Sie erstellen eine weitere Version, die WebM (VP8 + Vorbis) nutzt.
3. Sie erstellen noch eine Version, die H.264 Baseline-Video und AAC Low-Complexity-Audio in einem MP4-Container nutzt.
4. Sie verweisen auf alle drei Videodateien aus einem einzigen `<video>`-Element und weichen notfalls auf einen Flash-basierten Videoplayer aus.

1 Internet Explorer 9 wird WebM nur unterstützen, »wenn der Benutzer einen VP8-Codec installiert hat«, was impliziert, dass Microsoft den Codec selbst nicht ausliefern wird.

2 Google hat erklärt, dass es WebM »in einer zukünftigen Version« von Android unterstützen wird, aber es gibt noch keinen festen Zeitplan.

Lizenzprobleme bei H.264 Video

Bevor wir fortfahren, muss ich darauf hinweisen, dass diese doppelte Kodierung Ihrer Videos ihren Preis hat. Einen Preis über den offensichtlichen Preis hinaus – sie bringt doppelte Arbeit mit sich, da sie mehr Rechner und mehr Zeit in Anspruch nimmt als eine einfache Kodierung eines Videos. Einen sehr realen Preis, der mit H.264-Video verbunden ist: Lizenzgebühren.

Erinnern Sie sich noch an meine erste H.264-Videoerklärung (siehe dazu den Abschnitt »H.264« auf Seite 86). Dort erwähnte ich, dass dieser Videocodec patentbelastet ist und die Lizenzierung über das MPEG LA-Konsortium gesteuert wird. Jetzt zeigt sich, dass das nicht ganz unwichtig war. Damit Sie es verstehen, werde ich Sie durch das H.264-Lizenzlabyrinth führen:[3]

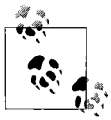
> Die MPEG LA gliedert das H.264-Lizenz-Portfolio in zwei Unterlizenzen: eine für Hersteller von Kodierern und Dekodierern und eine andere für Inhaltsanbieter. [...]
>
> Die Unterlizenz für die Anbieterseite wird noch weiter in vier Unterkategorien aufgespalten, von denen zwei (Subskription und titelbezogener Kauf oder bezahlte Verwendung) daran gebunden sind, ob der Endnutzer direkt für den Videodienst zahlt, und zwei (»freies« Fernsehen und Internetausstrahlung) an Bezahlung aus anderen Quellen als dem Endbenutzer. [...]
>
> Die Lizenzgebühr für »freies« Fernsehen basiert auf einer von zwei Gebührenoptionen. Die erste ist eine einmalige Zahlung von 2.500 Dollar pro AVC-Transmission-Kodierer. Das deckt einen AVC-Kodierer ab, »der vom oder im Dienste des Lizenznehmers zur Übermittlung von AVC-Video an den Endbenutzer genutzt wird«, der ihn dekodiert und betrachtet. Wenn Sie sich fragen, ob das eine doppelte Zahlungsforderung ist, lautet die Antwort: Ja. Eine Lizenzgebühr wurde bereits vom Hersteller des Kodierers bezahlt, eine weitere bezahlt der Ausstrahler.
>
> Die zweite Lizenzgebührenoption ist eine jährliche Ausstrahlungsgebühr. [...] Die jährliche Ausstrahlungsgebühr ist nach Publikumsgröße aufgeteilt:
>
> - 2.500 Dollar pro Kalenderjahr bei einem Ausstrahlungsmarkt von 100.000 bis 499.999 Fernsehhaushalten
> - 5.000 Dollar pro Kalenderjahr bei einem Ausstrahlungsmarkt von 500.000 bis 999.999 Fernsehhaushalten
> - 10.000 pro Kalenderjahr bei einem Ausstrahlungsmarkt von 1.000000 oder mehr Fernsehhaushalten
>
> [...] Bei allen Fragen ums »freie« Fernsehen, warum sollte man sich darüber Gedanken machen, wenn die eigene Ausstrahlungsform keine Rundfunkausstrahlung ist? Wie bereits gesagt, gelten die Teilnahmegebühren für jede Art der Auslieferung von Inhalten. Nachdem definiert wurde, dass »freies« Fernsehen nicht nur Rundfunk (im klassischen Sinne, d.h. über Antenne oder Kabel) einschließt, definiert die MPEG LA, dass auch für Internetausstrahlungen Teilnahmegebühren anfallen (im Wortlaut geht es um »AVC-Video, das über das weltweite Internet an den Endbenutzer ausgestrahlt wird, bei dem der Endbenutzer keine Gebühr für das Recht des Erhalts

3 *http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/The-H.264-Licensing-Labyrinth-65403.aspx*.

> oder der Betrachtung des Videos zahlt«). Anders gesagt, jede öffentliche Ausstrahlung, ob über Antenne, Kabel, Satellit oder Internet, unterliegt Teilnahmegebühren. [...]
>
> Die Gebühren sind bei Internetausstrahlungen potenziell größer, vielleicht unter der Annahme, dass die Internetausstrahlung schneller wächst als OTA oder »freies« Fernsehen über Kabel oder Satellit. Die Gebühr für den Ausstrahlungsmarkt »freies Fernsehen« und eine weitere Gebühr zusammenfassend, erteilt die MPEG LA während des ersten Lizenzierungszeitraums eine Gnadenfrist bis zum 31. Dezember 2010, d.h., die Ausstrahlung ist lizenzgebührenfrei, sofern der Empfang frei ist. Sie merkt an, dass »nach dem ersten Lizenzierungszeitraum die Gebühr nicht größer sein wird als das ökonomische Äquivalent dessen, was im gleichen Zeitraum für freies Fernsehen zu zahlen wäre«.

Der letzte Teil – zur Gebührenstruktur für Internetaustrahlungen – wurde bereits nachgebessert. Im Februar 2010 kündigte die MPEG LA an, dass freies Internet-Streaming auch im folgenden Lizenzierungszeitraum, bis zum 31.12.2015, gebührenfrei ist: Ende August schob sie dann in einer weiteren Pressemitteilung (*http://www.mpegla.com/Lists/MPEG%20LA%20News%20List/Attachments/231/n-10-08-26.pdf*) nach, dass diese Ausnahme für die gesamte weitere Laufzeit der Lizenz gelte, d.h. auch über den 31.12.2015 hinaus.

Ogg-Video mit Firefogg kodieren



In diesem Abschnitt werde ich »Ogg-Video« als Kurzform für »Theora-Video und Vorbis-Audio in einem Ogg-Container« verwenden. Das ist eine Kombination aus Codec und Container, die nativ in Mozilla Firefox und Google Chrome funktioniert.

Firefogg ist eine Open Source-Firefox-Erweiterung, die unter der GPL steht, zur Kodierung von Ogg-Video. Wenn Sie sie nutzen wollen, müssen Sie Mozilla Firefox 3.5 oder später installieren und dann die Firefogg-Website besuchen, die Sie in Abbildung 5-1 sehen.

Klicken Sie auf *Install Firefogg*. Firefox fragt nach, ob Sie der Site tatsächlich gestatten wollen, eine Erweiterung zu installieren. Klicken Sie aus *Erlauben*, um fortzufahren (Abbildung 5-2).

Firefox präsentiert das übliche Fenster zur Softwareinstallation. Klicken Sie auf *Jetzt installieren*, um fortzufahren (Abbildung 5-3).

Klicken Sie auf *Firefox neu starten*, um die Installation abzuschließen (Abbildung 5-4).

Nachdem Firefox neu gestartet wurde, bestätigt die Firefogg-Website, dass Firefogg erfolgreich installiert wurde (Abbildung 5-5).

Klicken Sie auf *Make web video*, um den Kodierungsprozess in Gang zu setzen (Abbildung 5-6). Klicken Sie auf der folgenden Seite auf *Deutsch*, präsentiert Ihnen Firefogg eine deutsche Schnittstelle. Klicken Sie auf *Wähle Datei*, um Ihr Quellvideo auszuwählen (Abbildung 5-7).

Abbildung 5-1: Firefogg-Homepage

Abbildung 5-2: Firefogg die Installation erlauben

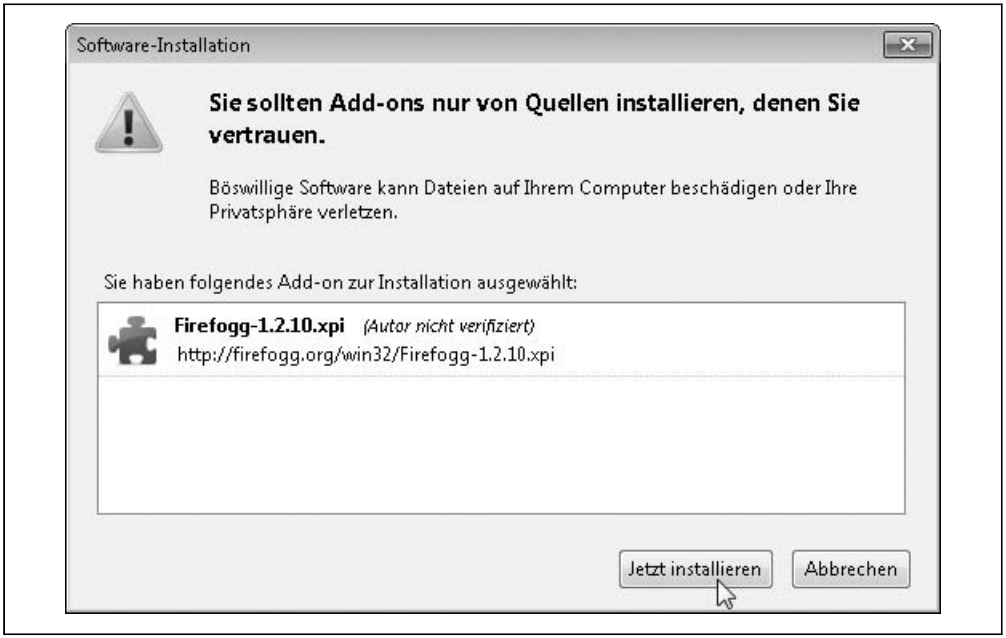


Abbildung 5-3: Firefogg installieren

Abbildung 5-4: Firefox neu starten

Abbildung 5-5: Installation erfolgreich

Abbildung 5-6: Erstellen wir ein Video!

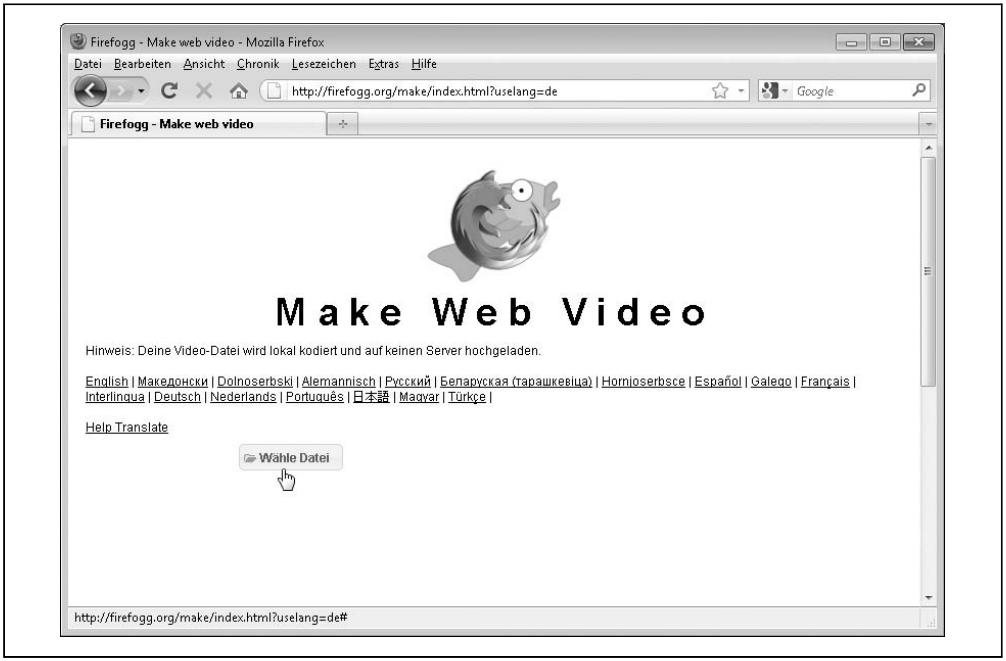


Abbildung 5-7: Wählen Sie Ihre Videodatei

Das Hauptfenster von Firefogg hat sechs Tabs (die Sie in Abbildung 5-8 sehen):

Voreinstellungen
: Die Standardvoreinstellung ist *Internetvideo*, für unsere Zwecke also hervorragend geeignet.

Codierungsbereich
: Das Kodieren von Videos kann sehr lange dauern. Bei den ersten Versuchen sollten Sie vielleicht nur den ersten Teil Ihres Videos kodieren (vielleicht die ersten 30 Sekunden), bis Sie die Einstellungskombination gefunden haben, die Ihnen gefällt.

Qualitäts- und Auflösungs-Basiseinstellungen
: Hier befinden sich die wichtigsten Optionen.

Metadaten
: Das werde ich hier nicht behandeln, aber Sie können Ihrem kodierten Video Metadaten wie Titel und Autor hinzufügen. Wahrscheinlich haben Sie so etwas schon in iTunes oder einer anderen Musikverwaltung gemacht, um Ihre Musiksammlung mit Metadaten auszustatten.

Erweiterte Videokodierungseinstellungen
: Drehen Sie an diesen Einstellungen nicht herum, es sei denn, Sie wissen, was Sie tun. (Firefogg bietet eine interaktive Hilfe zu den meisten Optionen, über die Sie mehr zu ihnen in Erfahrung bringen können.)

Erweitere Audiocodierungseinstellungen
: Auch von diesen Einstellungen sollten Sie die Finger lassen, wenn Sie nicht ganz genau wissen, was Sie tun.

Abbildung 5-8: Kodieren wir ein Video

Wir werden uns nur die Einstellungsmöglichkeiten unter *Qualitäts- und Auflösungsbasiseinstellungen* ansehen (Abbildung 5-9), dort finden Sie alle wichtigen Optionen:

Videoqualität
: Diese wird auf einer Skala von 0 (geringste Qualität) bis 10 (höchste Qualität) angezeigt. Größere Zahlen führen zu größeren Dateien. Sie müssen etwas experimentieren, um das beste Verhältnis von Größe? zu Qualität für Ihre Bedürfnisse zu finden.

Audioqualität
: Die Audioqualität wird auf einer Skala von –1 (geringste Qualität) bis 10 (höchste Qualität) angezeigt. Größere Zahlen stehen für größere Dateien, genau wie bei der Einstellung für die Videoqualität.

Videocodec
: Das sollte immer *theora* sein.

Audiocodec
: Das sollte immer *vorbis* sein.

Videobreite und Videohöhe
: Der Standardwert hier ist die tatsächliche Breite und Höhe des Ausgangsvideos. Soll die Größe Ihres Videos bei der Kodierung angepasst werden, können Sie hier die Breite oder die Höhe ändern. Firefogg passt die andere Dimension automatisch an, um die Proportionen Ihres Videos zu bewahren, damit Ihr Video nicht gequetscht oder gestreckt wird.

In Abbildung 5-10 verkleinere ich das Video auf gut die Hälfte der ursprünglichen Breite. Beachten Sie, wie Firefogg automatisch die Höhe anpasst.

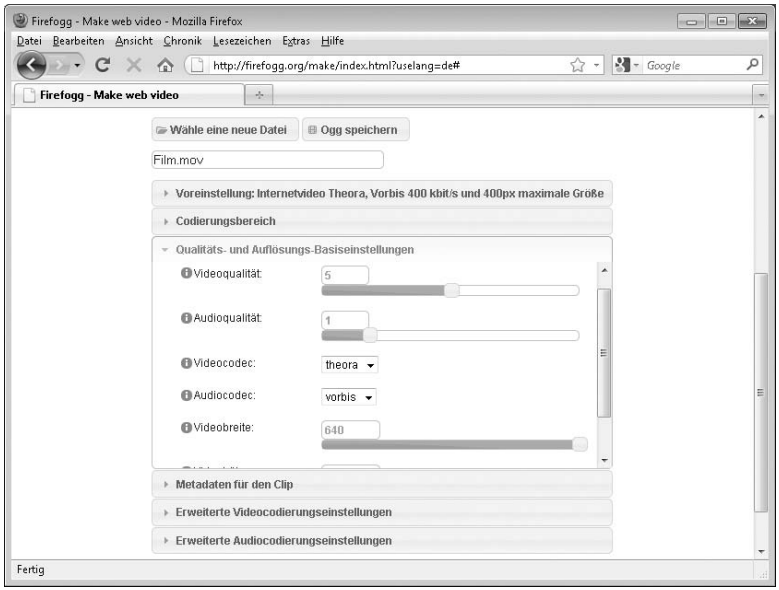


Abbildung 5-9: Qualitäts- und Auflösungsbasiseinstellungen

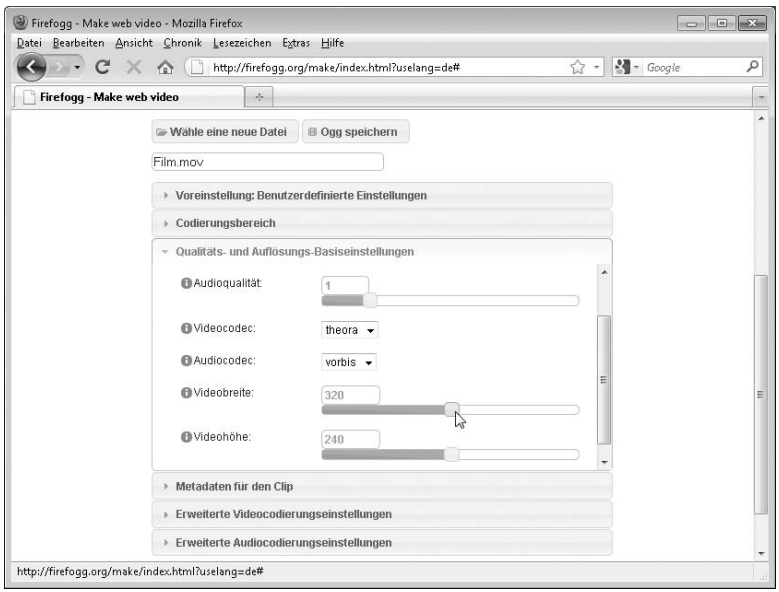


Abbildung 5-10: Breite und Höhe des Videos anpassen

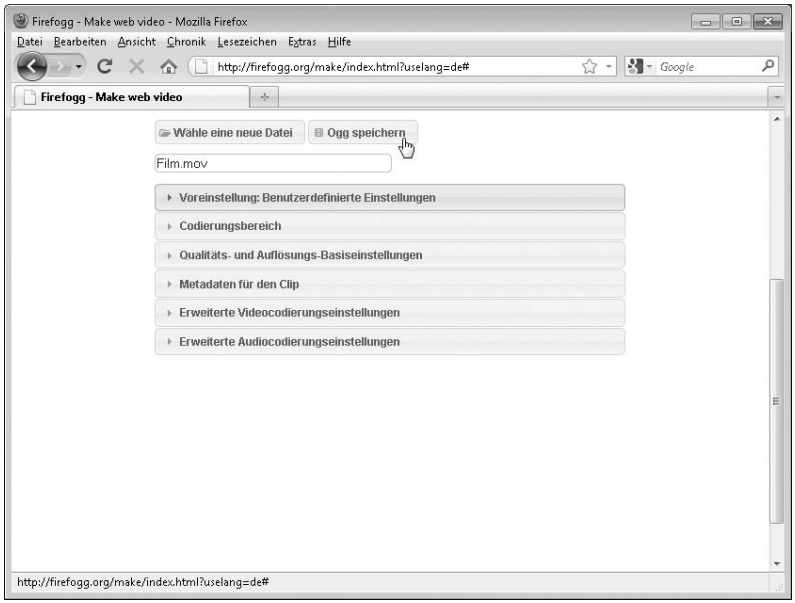


Abbildung 5-11: Mit einem Klick auf Ogg speichern den Kodierungsprozess starten

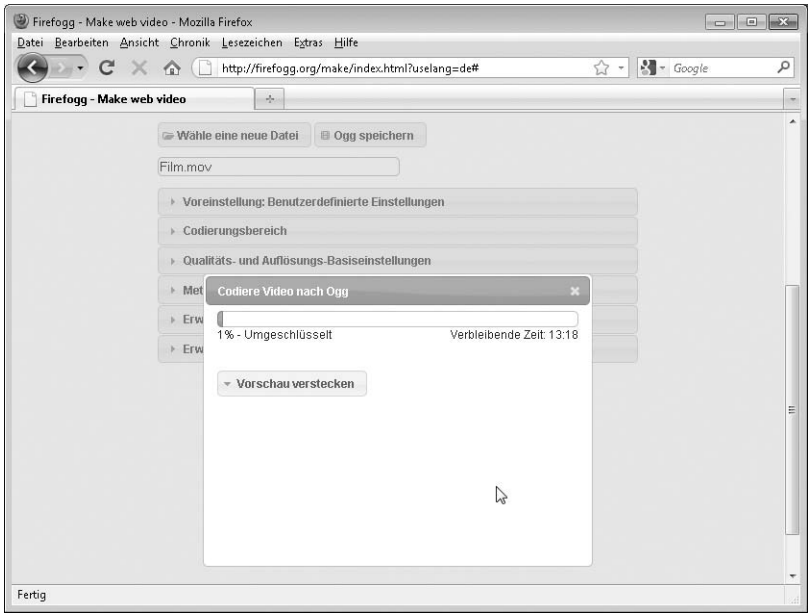
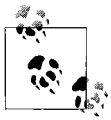


Abbildung 5-12: Die Kodierung läuft

Nachdem Sie etwas an den Schrauben gedreht haben, klicken Sie auf *Ogg speichern*, um den eigentlichen Kodierungsprozess zu starten (Abbildung 5-11). Firefogg wird Sie nach einem Dateinamen für das kodierte Video fragen.

Firefogg blendet eine freundliche Fortschrittsleiste ein, während Ihr Video kodiert wird (Abbildung 5-12). Sie müssen nur noch warten (und warten und warten)!

Batch-Kodierung von Ogg-Video mit ffmpeg2theora



Wie im letzten Abschnitt werde ich »Ogg-Video« als Kurzform für »Theora-Video und Vorbis-Audio in einem Ogg-Container« nutzen. Das ist die Kombination aus Codec und Container, die nativ in Mozilla Firefox und Google Chrome funktioniert.

Es gibt eine Reihe von Offlinekodierern für Ogg-Video. Wenn Sie viele Videodateien kodieren müssen und den Prozess automatisieren wollen, sollten Sie sich unbedingt *ffmpeg2theora* (*http://v2v.cc/~j/ffmpeg2theora/*) ansehen.

ffmpeg2theora ist eine Open Source-Anwendung unter der GPL zur Kodierung von Ogg-Video. Es werden fertige Programmdateien für Mac OS X, Windows und moderne Linux-Distributionen angeboten. Das Programm kann mit fast allen Videodateien als Eingabe arbeiten, einschließlich der DV-Videos, die die meisten einfachen Camcorder liefern.

Wenn Sie *ffmpeg2theora* nutzen wollen, müssen Sie es von der Kommandozeile aus aufrufen. (Gehen Sie unter Windows zu *Start→Alle Programme→Zubehör→Eingabeaufforderung*. Unter Mac OS X wählen Sie *Programme→Dienstprogramme→Terminal*.)

ffmpeg2theora kennt eine große Anzahl von Kommandozeilenschaltern. (Geben Sie `ffmpeg2theora --help` ein, wenn Sie alle kennenlernen wollen.) Ich werde mich hier auf nur drei konzentrieren:

- `--video-quality` *`Q`*, wobei *`Q`* eine Zahl zwischen 0 und 10 ist.
- `--audio-quality` *`Q`*, wobei *`Q`* eine Zahl zwischen –2 und 10 ist.
- `--max_size=`*`B`*`x`*`H`*, wobei *`B`* und *`H`* die gewünschte maximale Breite und Höhe für das Video sind. (Das `x` dazwischen ist tatsächlich nur der Buchstabe »x«.) *ffmpeg2theora* passt die Größe des Videos so an, dass die Proportionen erhalten bleiben. Es kann also sein, dass das kodierte Video kleiner ist als *B*×*H*. Kodieren Sie ein Video mit 720 × 480 Pixeln mit `--max_size 320x240`, erhalten Sie ein kodiertes Video, das 320 Pixel breit und 213 Pixel hoch ist.

Folgendermaßen könnten Sie also ein Video mit den gleichen Einstellungen wie im letzten Abschnitt kodieren (siehe dazu den Abschnitt »Ogg-Video mit Firefogg kodieren« auf Seite 94):

```
you@localhost$ ffmpeg2theora --videoquality 5
                             --audioquality 1
                             --max_size 320x240
                             pr6.dv
```

Das kodierte Video wird im gleichen Verzeichnis gespeichert wie das ursprüngliche Video. An den Dateinamen wird die Dateinamenserweiterung *.ogv* angehängt. Einen anderen Ort und/oder Dateinamen können Sie angeben, indem Sie auf der Kommandozeile das Argument `--output=`*`/Pfad/zu/kodiertem/Video`* an *ffmpeg2theora* übergeben.

H.264-Video mit HandBrake kodieren

In diesem Abschnitt werde ich »H.264-Video« als Kurzform für »H.264 Baseline-Profil-Video und AAC-Low-Complexity-Profil-Audio in einem MPEG-4-Container« nutzen. Das ist die Kombination aus Codec und Container, die nativ in Safari, in Adobe Flash, auf dem iPhone und auf Google Android-Geräten funktioniert.

Ungeachtet der Lizenzprobleme (siehe dazu den Abschnitt »Lizenzprobleme bei H.264 Video« auf Seite 93), ist der einfachste Weg, H.264-Videos zu kodieren, der Einsatz von HandBrake (*http://handbrake.fr*). HandBrake ist eine Open Source-Anwendung unter der GPL zur Kodierung von H.264-Videos. (Früher unterstützte es auch andere Videoformate, aber in der jüngsten Version haben die Entwickler die Unterstützung für die meisten anderen Formate gestrichen und ihre ganzen Anstrengungen auf H.264-Videos gelegt.) Es sind fertige Binärpakete für Windows, Mac OS X und moderne Linux-Distributionen (*http://handbrake.fr/downloads.php*) verfügbar.

HandBrake gibt es in zwei Versionen: mit GUI und für die Kommandozeile. Ich werde Ihnen erst die grafische Schnittstelle vorführen, später werden wir uns dann ansehen, wie Sie die von mir empfohlenen Einstellungen auf die Kommandozeilenversion übertragen.

Nachdem Sie die HandBrake-Anwendung geöffnet haben, wählen Sie zunächst das Ausgangsvideo (Abbildung 5-13). Klicken Sie auf *Source* und wählen Sie im Drop-down-Menü *Video-File*, um eine Datei auszuwählen. HandBrake kann fast alle Video-Dateien als Eingabe nehmen, einschließlich der DV-Videos, die von den meisten einfachen Camcordern erzeugt werden.

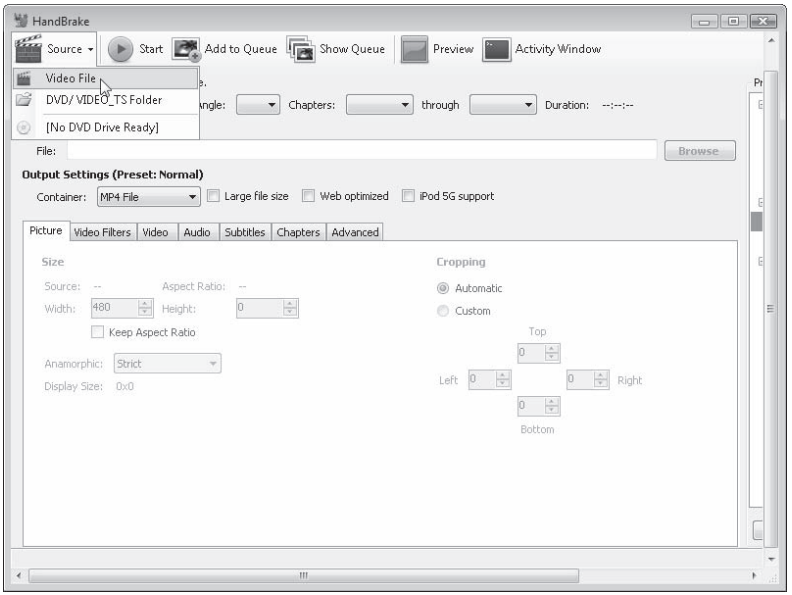


Abbildung 5-13: Das Quellvideo auswählen

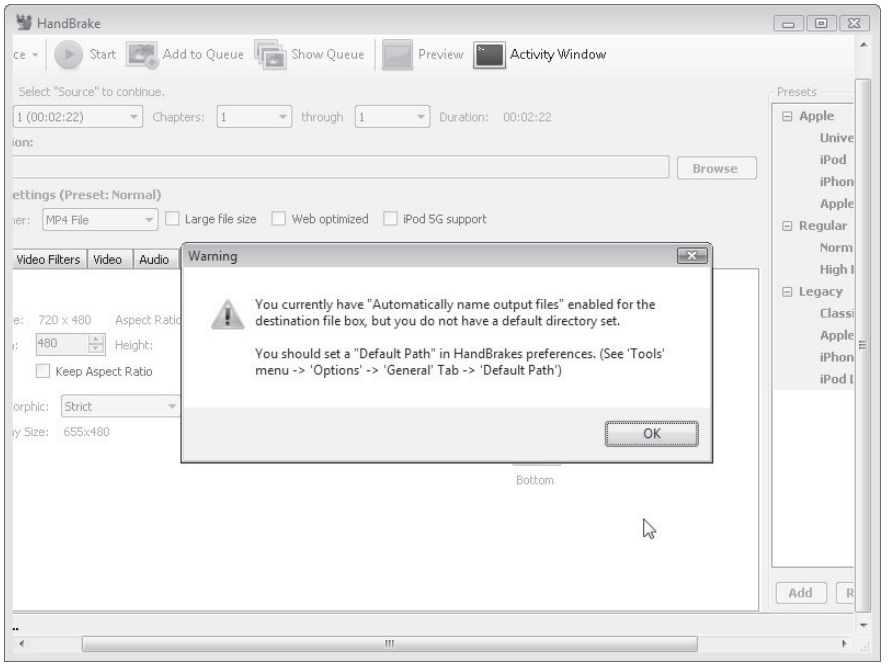


Abbildung 5-14: Ignorieren Sie diese Warnung

HandBrake beschwert sich, wenn Sie kein Standardverzeichnis für die Speicherung der kodierten Videos gewählt haben (Abbildung 5-14). Diese Warnung können Sie einfach ignorieren. Sie können aber auch das Optionsfenster öffnen (das finden Sie im Menü *Tools*) und ein Standardausgabeverzeichnis auswählen.

Auf der rechten Seite sehen Sie eine Liste mit Voreinstellungen. Wählen Sie *iPhone & iPod Touch* wie in Abbildung 5-15, werden die meisten Optionen gesetzt, die Sie benötigen.

Eine wichtige Option, die standardmäßig nicht eingeschaltet ist, ist *Web optimized*. Wählen Sie diese Option, wie in Abbildung 5-16 gezeigt, werden einige der Metadaten im kodierten Video umgeordnet, damit man den Anfang des Videos schon schauen kann, während der Rest im Hintergrund noch heruntergeladen wird. Ich empfehle Ihnen wärmstens, diese Option immer anzuwählen. Sie wirkt sich weder auf die Qualität noch auf die Dateigröße des kodierten Videos aus. Es gibt also keinen Grund, das nicht zu tun.

Im Tab *Picture* können Sie die maximale Breite und Höhe des kodierten Videos (Abbildung 5-17) einstellen. Sie sollten außerdem das Kontrollkästchen *Keep Aspect Ratio* aktivieren, um zu sichern, dass HandBrake Ihr Video nicht quetscht oder streckt, wenn Sie seine Größe anpassen.

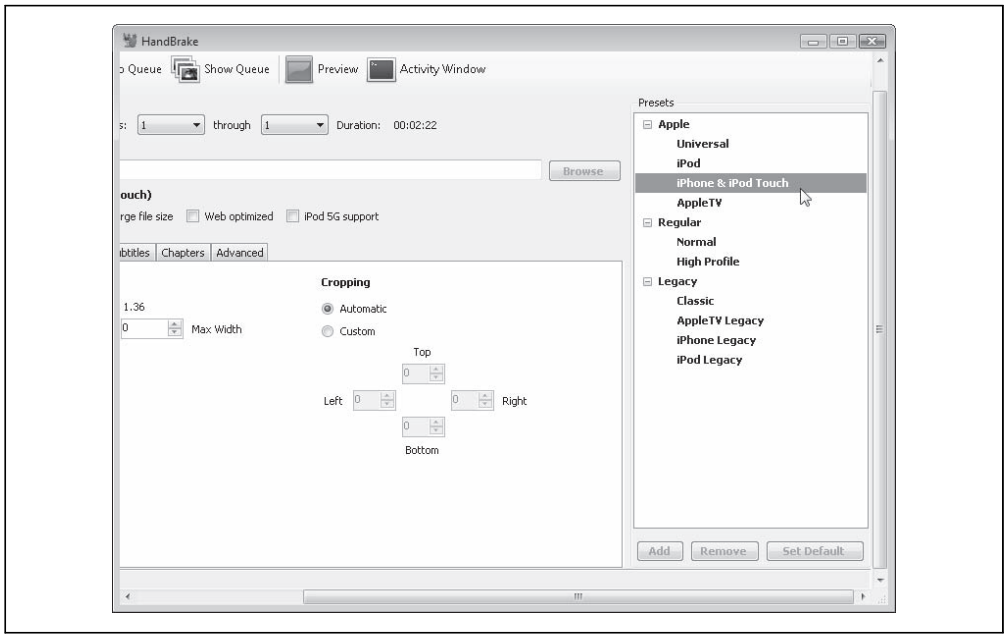


Abbildung 5-15: Die iPhone-Voreinstellung wählen

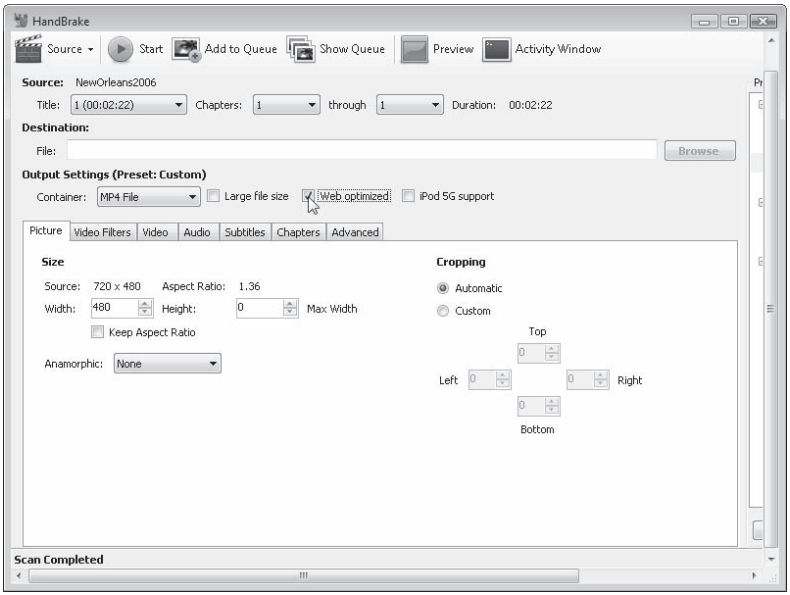


Abbildung 5-16: Optimieren Sie immer für das Web

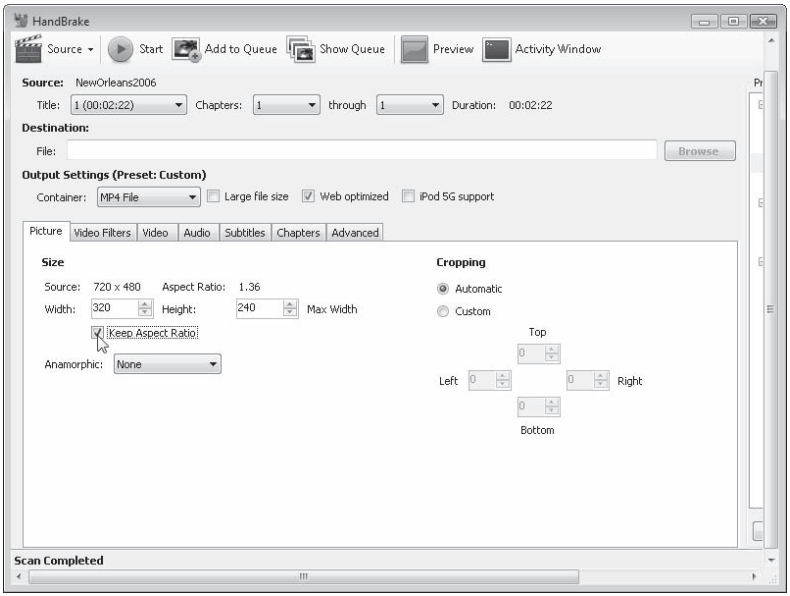


Abbildung 5-17: Breite und Höhe setzen

Im *Video*-Tab, das Sie in Abbildung 5-18 sehen, können Sie verschiedene wichtige Optionen setzen:

Video Codec
: Stellen Sie sicher, dass das *H.264 (x264)* ist.

2-Pass Encoding
: Ist diese Option aktiviert, führt HandBrake den Videodekodierer zwei Mal aus. Beim ersten Mal analysiert es das Video nur und prüft Dinge wie die Farbkomposition, Bewegung und Unterbrechungen. Beim zweiten Mal führt es die eigentliche Kodierung durch und nutzt dabei die im ersten Durchlauf ermittelten Informationen. Wie zu erwarten war, dauert das ungefähr doppelt so lange wie eine Single-Pass-Kodierung, führt aber zu besseren Videos, ohne dass dabei die Dateien größer werden. Ich aktiviere die Two-Pass-Kodierung bei H.264-Video immer. Wenn Sie nicht gerade YouTube neu erfinden und rund um die Uhr Videos kodieren, sollten wahrscheinlich auch Sie das tun.

Turbo First Pass
: Wenn Sie die Two-Pass-Kodierung aktiviert haben, können Sie mit dieser Option etwas Zeit zurückgewinnen. Sie reduziert die Arbeit, die beim ersten Durchlauf (der Analyse) der Videos durchgeführt wird, mindert die Qualität des Ergebnisses aber kaum. Ich aktiviere dieses Kontrollkästchen meist, aber wenn Qualität für Sie alles ist, sollten Sie es deaktiviert lassen.

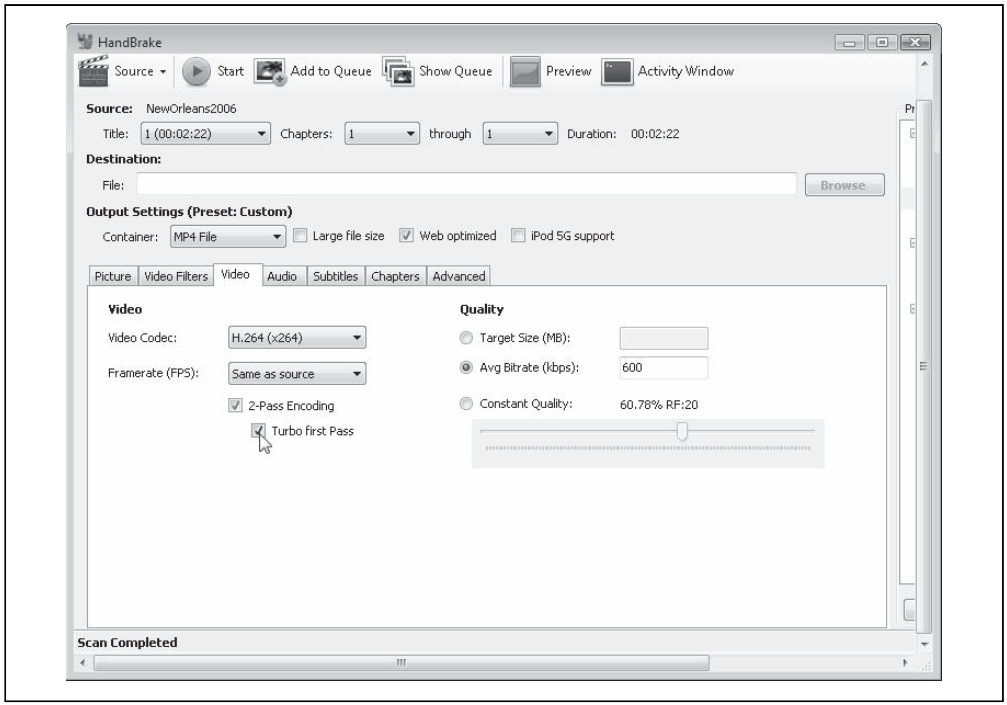


Abbildung 5-18: Videoqualitätsoptionen

Quality

Es gibt unterschiedliche Möglichkeiten, die Qualität des kodierten Videos zu steuern. Sie können eine Zielgröße für die Datei vorgeben, und HandBrake gibt dann sein Bestes, dass das kodierte Video diese Dateigröße nicht überschreitet. Sie können die durchschnittliche Bitrate festlegen, die ziemlich genau der Anzahl von Bits entspricht, die erforderlich sind, um eine Sekunde des kodierten Videos zu speichern. (Sie heißt die »durchschnittliche« Bitrate, weil einige Sekunden mehr Bits erfordern als andere.) Oder Sie können eine konstante Qualität auf einer Skala von 0 bis 100 Prozent angeben. Je größer die Zahl, umso besser ist das Ergebnis und umso größer wird natürlich auch die Datei. Die eine richtige Einstellung für diese Qualitätseinstellungen gibt es natürlich nicht.

Fragen an Professor Markup

A: Kann ich auch bei Ogg-Video Two-Pass-Kodierung wählen?

A: Ja. Aber aufgrund fundamentaler Unterschiede in der Arbeitsweise der Kodierer müssen Sie das wahrscheinlich nicht. Two-Pass-H.264-Kodierung führt fast immer zu Videos höherer Qualität. Two-Pass-Ogg-Kodierung ist nur nützlich, wenn Sie versuchen, das kodierte Video auf eine bestimmte Dateigröße zu bekommen. (Vielleicht ist das ja etwas, das für Sie interessant ist, aber es ist nicht das, was diese Beispiele zeigen. Im Allgemeinen ist es die zusätzliche Zeit bei der Kodierung von Webvideos nicht wert.) Die beste Qualität bei Ogg-Videos erzielen Sie mit den richtigen Qualitätseinstellungen. Eine Two-Pass-Kodierung ist dafür nicht erforderlich.

In diesem Beispiel habe ich eine durchschnittliche Bitrate von 600 kbps gewählt, die für ein kodiertes Video einer Größe von 320 × 240 ziemlich hoch ist. Außerdem habe ich die Two-Pass-Kodierung mit einem ersten »Turbo-Durchlauf« gewählt.

Im *Audio*-Tab, das Sie in Abbildung 5-19 sehen, brauchen Sie wahrscheinlich nichts zu ändern. Wenn Ihr Video mehrere Audiospuren hat, müssen Sie vielleicht die wählen, die in das kodierte Video sollen. Wird in Ihrem Video im Wesentlichen gesprochen (im Gegensatz zu Musik oder Umgebungsgeräuschen), können Sie die Bitrate wahrscheinlich auf etwa 96 kbps reduzieren. Ansonsten sollten die Einstellungen, die Sie von der *iPhone*-Voreinstellung geerbt haben, in Ordnung sein.

Klicken Sie dann auf den Button *Browse* und wählen Sie ein Verzeichnis und einen Dateinamen, um das kodierte Video zu speichern (Abbildung 5-20).

Klicken Sie abschließend auf *Start*, um die Kodierung zu starten (Abbildung 5-21).

HandBrake zeigt einige Fortschrittsmeldungen an, während das Video kodiert wird (Abbildung 5-22).

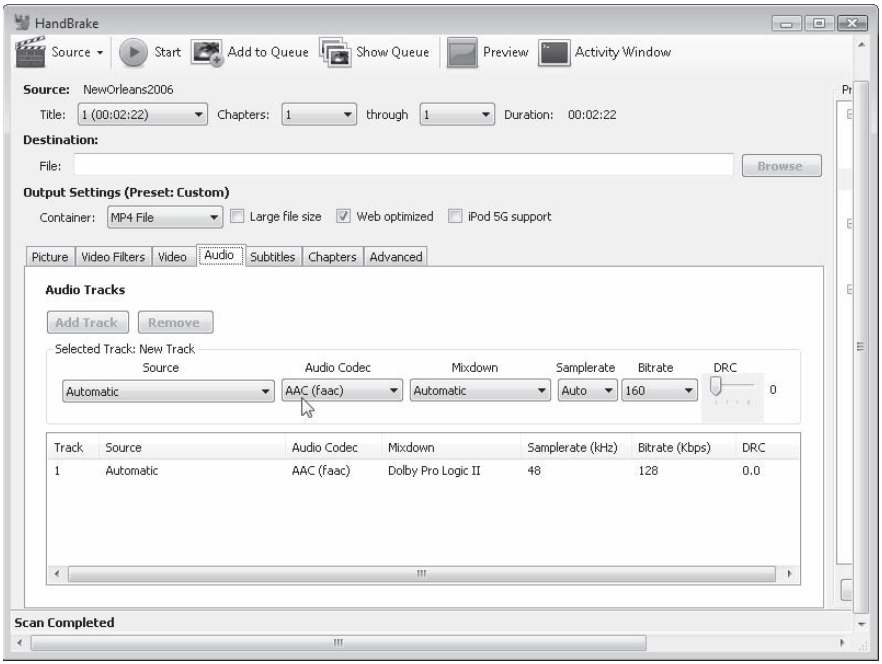


Abbildung 5-19: Audioqualitätsoptionen

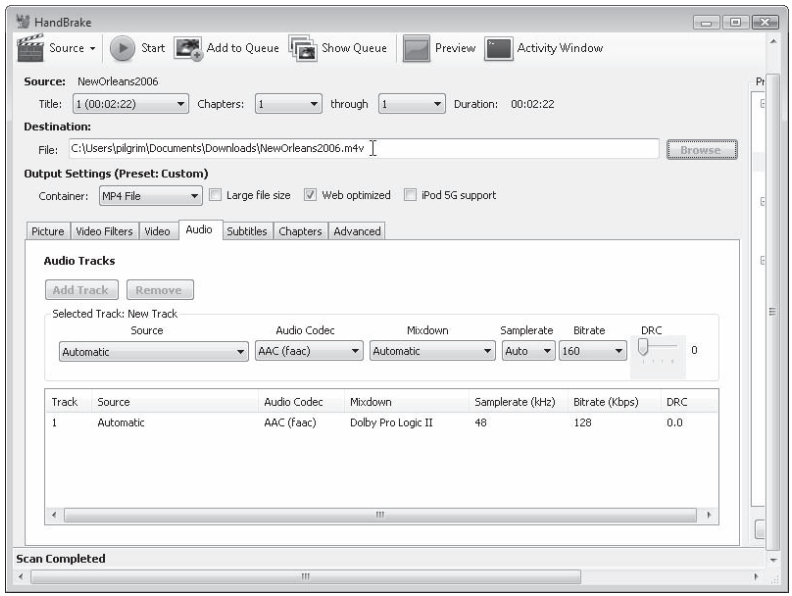


Abbildung 5-20: Den Zieldateinamen setzen

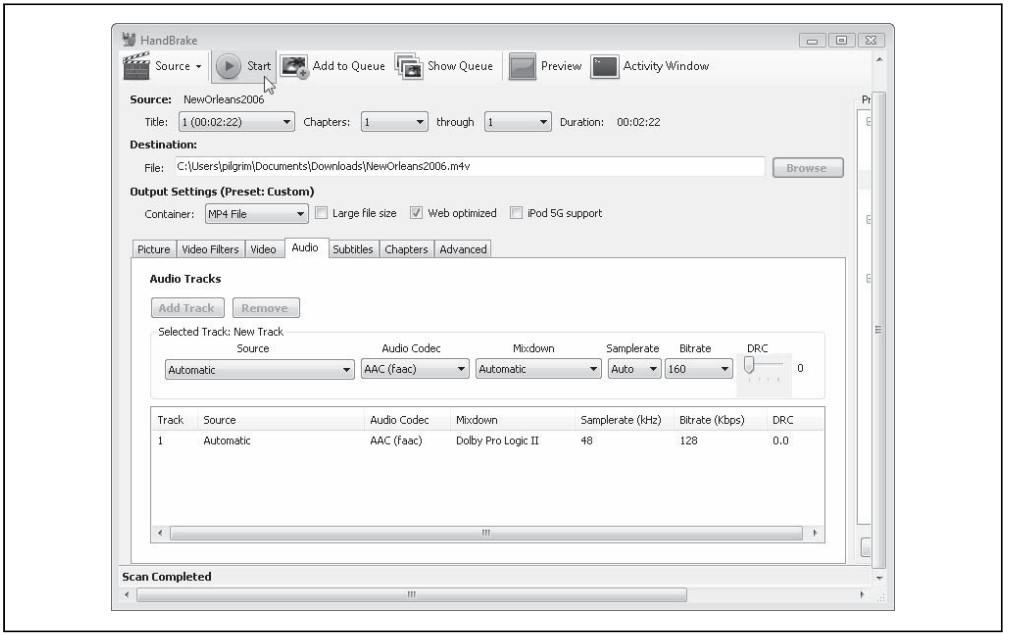


Abbildung 5-21: Das Video kodieren

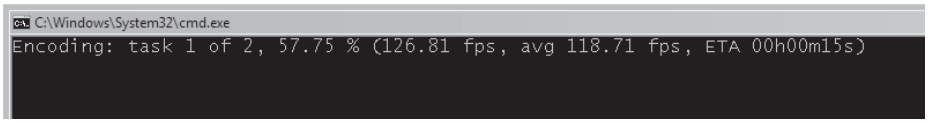


Abbildung 5-22: Geduldsübung

Batch-Kodierung von H.264-Video mit HandBrake

Wie im vorangegangenen Abschnitt werde ich »H.264-Video« als Kurzform für »H.264-Baseline-Profil-Video und AAC-Low-Complexity-Profil-Audio in einem MPEG-4-Container« nutzen. Das ist die Kombination aus Codec und Container, die nativ in Safari, in Adobe Flash, auf dem iPhone und auf Google Android-Geräten funktioniert.

Wie ich im letzten Abschnitt erwähnte, gibt es HandBrake auch als Kommandozeilenversion. Sie sollten, wie bei der GUI-Version, den aktuellen Snapshot von *http://handbrake.fr/downloads2.php* herunterladen.

Vergleichbar mit *ffmpeg2theora* (siehe dazu den Abschnitt »Batch-Kodierung von Ogg-Video mit ffmpeg2theora« auf Seite 102), bietet auch die Kommandozeilenversion von HandBrake eine schwindelerregende Vielzahl von Optionen. (Geben Sie in einem Termi-

nalfenster oder auf der Kommandozeile `HandBrakeCLI --help` ein, wenn Sie alles über diese Optionen erfahren wollen.) Ich werde mich auf nur ein paar beschränken:

- `--preset "X"`, wobei "*X*" der Name einer HandBrake-Voreinstellung ist (es ist wichtig, dass Sie den Namen in Anführungszeichen angeben). Die Voreinstellung, die Sie für H.264-Webvideo nutzen sollten, heißt *iPhone & iPod Touch*.
- `--width B`, wobei *B* die Breite des kodierten Videos ist. HandBrake passt automatisch die Höhe an, damit das Seitenverhältnis des ursprünglichen Videos bewahrt bleibt.
- `--vb Q`, wobei *Q* die durchschnittliche Bitrate (in Kilobits pro Sekunde) ist.
- `--two-pass`, aktiviert die Two-Pass-Kodierung.
- `--turbo` aktiviert den beschleunigten ersten Durchlauf bei einer Two-Pass-Kodierung.
- `--input D`, wobei *D* der Dateiname des Quellvideos ist.
- `--output E`, wobei *E* der Zielname für das kodierte Video ist.

Hier ist ein Beispiel für einen Aufruf von HandBrake auf der Kommandozeile mit den Kommandozeilenschaltern, die den Einstellungen entsprechen, die wir mit der grafischen Version von HandBrake wählten:

```
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch"
                            --width 320
                            --vb 600
                            --two-pass
                            --turbo
                            --input pr6.dv
                            --output pr6.mp4
```

Von oben nach unten: Dieser Befehl führt HandBrake mit der *iPhone & iPod Touch*-Voreinstellung aus, passt die Größe des Videos auf 320 × 240 an, setzt die durchschnittliche Bitrate auf 600 kbps, aktiviert Two-Pass-Kodierung mit schnellem ersten Durchlauf, liest die Datei *pr6.dv* und kodiert sie als *pr6.mp4*. Wow!

WebM-Video mit ffmpeg kodieren

WebM wird von *ffmpeg 0.6* und höher vollständig unterstützt. Führen Sie *ffmpeg* auf der Kommandozeile ohne Parameter aus und prüfen Sie, ob es mit VP8-Unterstützung kompiliert wurde:

```
you@localhost$ ffmpeg
FFmpeg version SVN-r23197, Copyright (c) 2000-2010 the FFmpeg developers
  built on May 19 2010 22:32:20 with gcc 4.4.3
  configuration: --enable-gpl --enable-version3 --enable-nonfree --enable-postproc
  --enable-pthreads --enable-libfaac --enable-libfaad --enable-libmp3lame
  --enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora
  --enable-libx264 --enable-libxvid --enable-x11grab --enable-libvorbis --enable-libvpx
```

Wenn Sie die magischen Worte `--enable-libvorbis` und `--enable-libvpx` nicht sehen, haben Sie nicht die richtige Version von *ffmpeg*. (Haben Sie selbst kompiliert, sollten Sie

prüfen, ob Sie zwei Versionen installiert haben. Das ist in Ordnung und führt nicht zu Versionskonflikten. Sie müssen aber eventuell den vollständigen Pfad zu der Version angeben, die VP8 unterstützt.)

Ich werde Ihnen zeigen, wie Sie eine Two-Pass-Kodierung (siehe dazu den Abschnitt »H.264-Video mit HandBrake kodieren« auf Seite 103 durchführen. Beim ersten Durchlauf wird die zu kodierende Videodatei durchleuchtet (`-i pr6.dv`), und es werden einige Statistiken in eine Logdatei geschrieben (die automatisch den Namen *pr6.dv-0.log* erhält). Den Videocodec geben Sie mit dem Parameter `-vcodec` an:

```
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -keyint_min 0
               -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
               -vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -an -y NUL
```

Der größte Teil der *ffmpeg*-Kommandozeile hat nichts mit VP8 oder WebM zu tun. *libvpx* unterstützt eine Reihe von VP8-spezifischen Optionen, die Sie an *ffmpeg* übergeben können, aber ich habe noch keinen Schimmer, wie auch nur eine einzige davon funktioniert. Sobald ich eine gute Erläuterung dieser Optionen finde, werde ich auf der Webseite zum Buch einen Link bereitstellen.

Beim zweiten Durchlauf liest *ffmpeg* die beim ersten Durchlauf geschriebenen Statistiken und führt die eigentliche Kodierung von Video und Audio durch. Es schreibt folgende WebM-Datei:

```
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -keyint_min 0
               -g 250 -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv
               -vcodec libvpx -b 614400 -s 320x240 -aspect 4:3 -acodec vorbis
               -y pr6.webm
```

Hier sind fünf Parameter wichtig:

`-vcodec libvpx`
: Gibt an, dass wir mit dem VP8-Videocodec kodieren. WebM nutzt immer VP8-Video.

`-b 614400`
: Gibt die Bitrate an. Anders als die anderen Formate erwartet *libvp8* die Bitrate in Bit, nicht in Kilobit. Wenn Sie ein 600-kbps-Video wollen, müssen Sie 600 mit 1024 multiplizieren und erhalten 614400.

`-s 320x240`
: Gibt in Breite mal Höhe die Zielgröße an.

`-aspect 4:3`
: Gibt das Seitenverhältnis des Videos an. Gewöhnliches Video hat üblicherweise 4:3, aber High-Definition-Video hat meist 16:9 oder 16:10. Bei meinen Tests habe ich festgestellt, dass ich das explizit auf der Kommandozeile angeben musste und mich nicht darauf verlassen konnte, dass *ffmpeg* das Seitenverhältnis automatisch ermittelt.

`-acodec vorbis`
: Gibt an, dass wir mit dem Vorbis-Audiocodec kodieren. WebM nutzt immer Vorbis-Audio.

Endlich zum Markup

Sollte das nicht eigentlich ein HTML-Buch sein? Wo bitte ist dann das Markup?

HTML5 gibt Ihnen zwei Möglichkeiten, Video in eine Webseite einzuschließen. Beide basieren auf dem `<video>`-Element. Wenn Sie nur eine Videodatei haben, können Sie diese einfach über ein `src`-Attribut angeben. Das hat erstaunliche Ähnlichkeit mit dem Einbinden eines einfachen Bilds mit einem `<img src="...">`-Tag.

```
<video src="pr6.webm"></video>
```

Technisch ist das alles, was Sie benötigen. Aber wie bei einen `<img>`-Tag sollten Sie immer die Attribute `width` und `height` in Ihre `<video>`-Tags einschließen. Die Attribute `width` und `height` können der maximalen Breite und Höhe entsprechen, die Sie bei der Kodierung des Videos angaben:

```
<video src="pr6.webm" width="320" height="240"></video>
```

Machen Sie keine Gedanken, wenn eine der Seitenlängen des Videos etwas kleiner ist. Ihr Browser zentriert das Video in der vom `<video>`-Tag definierten Box. Es wird nie so gequetscht oder gestreckt, dass es sein Seitenverhältnis verliert.

Standardmäßig sorgt das `<video>`-Element nicht dafür, dass Steuerelemente zum Abspielen angezeigt werden. Aber mit gutem altem HTML, CSS und JavaScript können Sie ganz einfach Ihre eigenen erstellen. Dass `<video>`-Element bietet Methoden wie `play()` und `pause()` sowie die les- und schreibbare Eigenschaft `currentTime`. Außerdem gibt es die les- und schreibbaren Eigenschaften `volume` und `muted`. Sie haben also alles, was Sie benötigen, um eine eigene Schnittstelle aufzubauen.

Wenn Sie keine eigene Schnittstelle aufbauen möchten, können Sie dem Browser sagen, dass er einen Satz eingebauter Steuerelemente anzeigen soll. Das tun Sie, indem Sie einfach das Attribut `controls` in Ihr `<video>`-Tag einschließen:

```
<video src="pr6.webm" width="320" height="240" controls></video>
```

Es gibt zwei weitere optionale Attribute, die ich erwähnen möchte, bevor wir fortfahren: `preload` und `autoplay`. Nein, bitte nicht den Überbringer der Nachricht erschlagen. Lassen Sie mich erklären, warum diese Attribute wichtig sind. Das Attribut `preload` sagt dem Browser, dass er mit dem Download des Videos beginnen soll, sobald die Seite geladen wird. Das ist sinnvoll, wenn das Video der Einstiegspunkt für die Seite ist. Ist es nur eine Ergänzung zu dem, was sonst noch auf der Seite ist, die sich wahrscheinlich nur wenige Besucher ansehen werden, können Sie `preload` auf `none` setzen, damit der Browser den Netzwerkverkehr nicht unnötig erhöht.

Hier ist ein Beispiel für ein Video, das unmittelbar beim Laden der Seite heruntergeladen (aber noch nicht abgespielt) wird:

```
<video src="pr6.webm" width="320" height="240" preload></video>
```

Und hier das Beispiel für ein Video, dass nicht unmittelbar beim Laden der Seite heruntergeladen wird:

```
<video src="pr6.webm" width="320" height="240" preload="none"></video>
```

Das Attribut `autoplay` macht genau, was der Name verspricht: Es sagt dem Browser, dass die Videodatei unmittelbar beim Laden der Seite heruntergeladen und sobald wie möglich abgespielt werden soll. Manche lieben es, manche hassen es. Aber lassen Sie mich erklären, warum es wichtig ist, dass es in HTML5 ein solches Attribut gibt. Es wird auf alle Fälle Leute geben, die wollen, dass ihre Videos automatisch abgespielt werden, selbst wenn es ihre Besucher nervt. Würde HTML5 kein Standardverfahren für das automatische Abspielen von Videos definieren, würden sie dann zu JavaScript-Hacks greifen, um es trotzdem zu erreichen, beispielsweise indem sie die `play()`-Methode von `video` während des `load`-Events des Fensters aufrufen. Dem entgegenzuwirken, wäre für den Besucher erheblich schwerer. Gibt es hingegen die Attribute, muss man den Browser einfach mit einer passenden Erweiterung ausstatten (oder gegebenenfalls selbst schreiben), mit deren Hilfe man ihm sagen kann: »Ignoriere das `autoplay`-Attribut. Bei mir spielt niemand Videos automatisch ab.«

Hier ist ein Video, das unmittelbar beim Laden der Seite heruntergeladen und sobald wie möglich abgespielt wird:

```
<video src="pr6.webm" width="320" height="240" autoplay></video>
```

Und hier ist ein Greasemonkey-Skript (*http://www.greasespot.net*), das Sie in Ihrer lokalen Firefox-Version installieren können: Es verhindert, dass HTML5 Videos automatisch abspielt. Es nutzt das DOM-Attribut `autoplay`, das von HTML5 definiert wird. Das ist das JavaScript-Äquivalent des `autoplay`-Attributs in Ihrem HTML-Markup:

```
// ==UserScript==
// @name           Automatisches Abspielen von Videos deaktivieren
// @namespace      http://diveintomark.org/projects/greasemonkey/
// @description    Sichert, dass HTML5-Videoelemente nicht automatisch abspielen
// @include        *
// ==/UserScript==

var arVideos = document.getElementsByTagName('video');
for (var i = arVideos.length - 1; i >= 0; i--) {
    var elmVideo = arVideos[i];
    elmVideo.autoplay = false;
}
```

Einen Moment noch. Wenn Sie in diesem Kapitel aufgepasst haben, wissen Sie, dass Sie nicht nur eine Videodatei haben, sondern drei. Eine ist eine *.ogv*-Datei, die Sie mit Firefogg (siehe dazu den Abschnitt »Ogg-Video mit Firefogg kodieren« auf Seite 94 oder *ffmpeg2theora* (siehe dazu den Abschnitt »Batch-Kodierung von Ogg-Video mit ffmpeg2theora« auf Seite 102) erstellt haben. Die zweite ist eine *.mp4*-Datei, die Sie mit

HandBrake (siehe dazu den Abschnitt »H.264-Video mit HandBrake kodieren« auf Seite 103) erstellt haben. Die dritte ist eine *.webm*-Datei, erstellt mit *ffmpeg* (siehe dazu den Abschnitt »WebM-Video mit ffmpeg kodieren« auf Seite 111). HTML5 bietet eine Möglichkeit, alle drei einzubinden: das `<source>`-Element. Jedes `<video>`-Element kann so viele `<source>`-Elemente enthalten, wie Sie benötigen. Ihr Browser geht die Liste der Videoquellen der Reihe nach durch und spielt das erste Video ab, das er abspielen kann.

Das führt zu einer weiteren Frage: Woher weiß der Browser, welches Video er abspielen kann? Im schlimmsten Fall lädt er alle Videos nacheinander herunter und versucht, sie abzuspielen. Das ist eine riesige Verschwendung von Bandbreite. Sie sparen eine Menge Netzwerkverkehr, wenn Sie dem Browser im Vorhinein hinreichende Informationen über die einzelnen Videos geben. Das tun Sie mit dem `type`-Attribut des `<source>`-Elements.

Hier ist das Ganze:

```
<video width="320" height="240" controls>
  <source src="pr6.mp4"  type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="pr6.ogv"  type='video/ogg; codecs="theora, vorbis"'>
</video>
```

Gehen wir es schrittweise durch. Das `<video>`-Element gibt die Breite und die Höhe des Videos an, bietet aber noch keinen Verweis auf das Video selbst. Im `<video>`-Element gibt es `<source>`-Elemente. Beide `<source>`-Elemente verweisen auf jeweils eine Videodatei (mit dem Attribut `src`) und liefern Informationen zum Videoformat (im `type`-Attribut).

Das `type`-Attribut sieht kompliziert aus – und es ist kompliziert. Es enthält drei Informationsbestandteile: das Containerformat (siehe dazu den Abschnitt »Videocontainer« auf Seite 83), den Videocodec (siehe dazu den Abschnitt »Videocodecs« auf Seite 85) und den Audiocodec (siehe Abschnitt »Audiocodecs« auf Seite 88). Beginnen wir von unten. Bei der *.ogv*-Videodatei ist das Containerformat Ogg, das hier durch `video/ogg` angegeben wird. (Technisch gesagt, ist das der MIME-Typ für Ogg-Videodateien.) Der Videocodec ist Theora und der Audiocodec Vorbis. Das ist eigentlich nicht weiter kompliziert, nur das Format des Attributwerts ist etwas chaotisch. Der `codecs`-Wert selbst muss Anführungszeichen enthalten, was bedeutet, dass Sie eine andere Art von Anführungszeichen nutzen müssen, um den gesamten `type`-Wert einzuschließen:

```
<source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"'>
```

Das `<source>`-Element für die WebM-Datei sieht ganz ähnlich aus, hat aber einen anderen MIME-Typ (`video/webm` statt `video/ogg`) und einen anderen Videocodec (`vp8` statt `theora`), der im `codecs`-Parameter aufgeführt wird:

```
<source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"'>
```

Das `<source>`-Element für die H.264-Datei ist sogar noch komplizierter. Erinnern Sie sich daran, dass ich sagte, dass es H.264-Video (siehe dazu den Abschnitt »H.264« auf Seite 86) und AAC-Audio (siehe dazu den Abschnitt »Advanced Audio Coding« auf Seite 90) in unterschiedlichen Profilen gibt? Wir haben unsere Datei mit dem H.264-Baseline-Profil

und dem AAC-Low-Complexity-Profil kodiert und dann in einen MPEG-4-Container gepackt. Alle diese Informationen werden in das type-Attribut eingeschlossen:

```
<source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
```

Diese ganzen Umstände machen wir uns, damit der Browser anhand des type-Attributs prüfen kann, ob er eine bestimmte Videodatei abspielen kann. Stellt er fest, dass er ein bestimmtes Video nicht abspielen kann, lädt er die Datei erst gar nicht herunter. Nicht einmal einen winzigen Teil der Datei. Sie sparen Bandbreite, und Ihre Besucher sehen schneller das Video, für das sie zu Ihrer Site kamen.

Wenn Sie die Anleitungen in diesem Kapitel bei der Kodierung Ihrer Videos befolgt haben, können Sie einfach die type-Attributwerte aus diesem Beispiel kopieren. Ansonsten müssen Sie den Wert für die type-Parameter wohl oder übel selbst ausarbeiten.

Professor Markup sagt

Als dies geschrieben wurde (20. Mai 2010), hatte das iPad einen Bug, der bewirkte, dass es nur die erste aufgeführte Videoquelle bemerkte. Leider bedeutet das, dass Sie die MP4-Datei an erster Stelle aufführen müssen und die freien Videoformate hintenan stellen müssen. *Seufz.*

Die MIME-Typen schlagen zurück

Das Videopuzzle hat so viele Teile, dass ich das Folgende am liebsten gar nicht erwähnen würde. Aber es ist wichtig, weil ein falsch konfigurierter Webserver zu unendlicher Frustration führen kann, wenn Sie herauszufinden versuchen, warum die Videos auf Ihrem lokalen Rechner abgespielt werden, aber nicht mehr laufen, wenn Sie sie auf Ihrem Produktionsserver veröffentlichen. Sollten Sie dieses Problem haben, sind wahrscheinlich MIME-Typen die Wurzeln allen Übels.

Ich habe MIME-Typen in Kapitel 1 erwähnt (siehe dazu den Abschnitt »MIME-Typen« auf Seite 1), aber da Ihnen die Bedeutung nicht klar war, haben Sie Ihre Aufmerksamkeit damals vielleicht nicht ganz bei der Sache gelassen. Deswegen hier noch einmal in Großbuchstaben.

Professor Markup schreit

VIDEODATEIEN MÜSSEN MIT DEM KORREKTEN MIME-TYP AUSGELIEFERT WERDEN!

Was der korrekte MIME-Typ ist? Sie haben ihn bereits gesehen. Er ist Teil des Werts des type-Attributs eines <source>-Elements. Aber es reicht nicht aus, das type-Attribut in Ihrem HTML-Markup zu setzen. Sie müssen auch dafür sorgen, dass Ihr Webserver den passenden MIME-Typ in den Content-Type-HTTP-Header einschließt.

Wenn Sie den Apache-Webserver oder einen Apache-Abkömmling nutzen, können Sie in Ihrer Site-weiten *httpd.conf* oder in einer *.htaccess*-Datei in dem Verzeichnis, in dem Sie die Videodateien speichern, eine `AddType`-Direktive verwenden. (Verwenden Sie einen anderen Webserver, werfen Sie einen Blick in die Dokumentation Ihres Servers und lese Sie nach, wie man den `Content-Type` -HTTP-Header für bestimmte Dateitypen setzt.) Die `AddType`-Direktiven, die Sie einfügen müssen, sehen so aus:

```
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

Die erste Zeile ist für Videos in einem Ogg-Container, die zweite für Videos in einem MPEG-4-Container, die dritte für WebM. Machen Sie das einmal und vergessen Sie dann die ganze Sache. Tun Sie es nicht, werden Ihre Videos ein einigen Browsern nicht abgespielt, selbst wenn Sie den korrekten MIME-Typ im `type`-Attribut in Ihrem HTML-Markup angeben.

Wenn Sie wirklich alles über die entsprechende Konfiguration Ihres Webservers erfahren wollen, schauen Sie sich den folgenden ausgezeichneten Artikel im Mozilla Developer Center an: »Configuring servers for Ogg media« (*https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media*). (Die Tipps in diesem Artikel gelten auch für MP4- und WebM-Video.)

Was ist mit dem IE?

Als ich dies schrieb, veröffentlichte Microsoft eine »Entwicklervorschau« des Internet Explorer 9. Noch unterstützt er das HTML5- `<video>`-Element noch nicht. Aber Microsoft hat öffentlich versprochen (*http://blogs.msdn.com/ie/archive/2010/03/16/html5-hardware-accelerated-first-ie9-platform-preview-available-for-developers.aspx*), dass die endgültige Version des IE9 H.264-Video und AAC-Audio in einem MPEG-4-Container unterstützen wird, wie Safari auf Desktop-Macs und Mobile Safari auf iOS.

Aber was ist mit älteren Versionen des Internet Explorer? Alle aktuell gebräuchlichen Versionen des IE8 also? Die meisten, die den Internet Explorer nutzen, haben das Adobe Flash-Plug-in installiert. Moderne Versionen von Adobe Flash (seit 9.0.60.184) unterstützen H.264-Video und AAC-Audio in einem MPEG-4-Container. Haben Sie H.264-Video für Safari kodiert (siehe dazu den Abschnitt »H.264-Video mit HandBrake kodieren« auf Seite 103), können Sie es in einem Flash-basierten Videoplayer abspielen, wenn Sie feststellen, dass Ihr Besucher keinen HTML5-fähigen Browser hat.

FlowPlayer (*http://flowplayer.org*) ist ein Flash-basierter Open Source-Videoplayer unter der GPL (es gibt auch kommerzielle Lizenzen; siehe *http://flowplayer.org/download/*.) FlowPlayer weiß nichts über das `<video>`-Element. Er kann nicht magisch ein `<video>`-Tag in ein Flash-Objekt umwandeln. Aber HTML5 ist so entworfen, dass Sie ein `<object>`-Element in ein `<video>`-Element einbetten können. Browser, die HTML5-Video nicht unterstützen, ignorieren das `<video>`-Element und geben einfach das eingebettete `<object>` wieder, das das Flash-Plug-in aufruft und den Film mit FlowPlayer abspielt. Browser, die

HTML5-Video unterstützen, finden eine Videoquelle, die sie spielen können, und ignorieren das eingebettete <object>-Element.

Der letzte Happen ist der Schlüssel zum ganzen Puzzle: HTML5 legt fest, dass alle Elemente (außer den <source>-Elementen), die Kinder eines <video>-Elements sind, vollständig ignoriert werden müssen. Das ermöglicht Ihnen, HTML5-Video in neueren Browsern zu verwenden und in älteren Browsern auf Flash auszuweichen, ohne dass dazu ausgefallene JavaScript-Hacks erforderlich sind. Mehr zu dieser Technik erfahren Sie auf der Video for Everybody!-Site (*http://camendesign.com/code/video_for_everybody*).

Ein vollständiges Beispiel

Schauen wir uns ein Beispiel an, das diese Technik nutzt. Ich habe den Video for Everybody!-Code so erweitert, dass er ein Video im WebM-Format einschließt. Mit folgenden Befehlen habe ich die gleiche Videoquelle in drei verschiedenen Formaten kodiert.

```
## Theora/Vorbis/Ogg
you@localhost$ ffmpeg2theora --videobitrate 200 --max_size 320x240
               --output pr6.ogv pr6.dv

## H.264/AAC/MP4
you@localhost$ HandBrakeCLI --preset "iPhone & iPod Touch" --vb 200 --width 320
               --two-pass --turbo --optimize --input pr6.dv --output pr6.mp4

## VP8/Vorbis/WebM
you@localhost$ ffmpeg -pass 1 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
               -an -f rawvideo -y NULffmpeg --videobitrate 200
you@localhost$ ffmpeg -pass 2 -passlogfile pr6.dv -threads 16 -token_partitions 4
               -altref 1 -lag 16 -keyint_min 0 -g 250 -mb_static_threshold 0
               -skip_threshold 0 -qmin 1 -qmax 51 -i pr6.dv -vcodec libvpx_vp8
               -b 204800 -s 320x240 -aspect 4:3 -acodec vorbis -ac 2 -y pr6.mkv
you@localhost$ mkclean --doctype 4 pr6.mkv pr6.webm
```

Das endgültige Markup nutzt ein <video>-Element für HTML5-Video mit einem eingebetteten <object>-Element, um auf Flash auszuweichen:

```
<video id="movie" width="320" height="240" preload controls>
  <source src="pr6.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"' />
  <source src="pr6.webm" type='video/webm; codecs="vp8, vorbis"' />
  <source src="pr6.ogv" type='video/ogg; codecs="theora, vorbis"' />
  <object width="320" height="240" type="application/x-shockwave-flash"
    data="flowplayer-3.2.1.swf">
    <param name="movie" value="flowplayer-3.2.1.swf" />
    <param name="allowfullscreen" value="true" />
    <param name="flashvars" value='config={
     "clip": {"url": "http://wearehugh.com/dih5/good/bbb_480p.mp4",
     "autoPlay":false, "autoBuffering":true}}' />
    <p>Video herunterladen als <a href="pr6.mp4">MP4</a>,
```

```
      <a href="pr6.webm">WebM</a>, or
      <a href="pr6.ogv">Ogg</a>.</p>
  </object>
</video>
```

Mit einer Kombination aus HTML5 und Flash sollten Sie dieses Video in fast allen Browsern und Geräten abspielen können.

Weitere Lektüre

- »The `<video>`-Element« in der HTML5-Spezifikation (*http://bit.ly/a3kpiq*)
- Video for Everybody! (*http://camendesign.com/code/video_for_everybody*)
- »A gentle introduction to video encoding« (*http://diveintomark.org/tag/give*)
- »Theora 1.1 is released–what you need to know« (*http://hacks.mozilla.org/2009/09/theora-1-1-released/*), von Christopher Blizzard
- »Configuring servers for Ogg media« (*https://developer.mozilla.org/en/Configuring_servers_for_Ogg_media*)
- »Encoding with the `x264` codec« (*http://www.mplayerhq.hu/DOCS/HTML/en/menc-feat-x264.html*)
- »Video type parameters« (*http://wiki.whatwg.org/wiki/Video_type_parameters*)
- Zencoder Video JS (*http://videojs.com*), angepasste Steuerelemente für HTML5-Video
- »Everything you need to know about HTML5 audio and video« (*http://dev.opera.com/articles/view/everything-you-need-to-know-about-html5-video-and-audio/*), von Simon Pieters

KAPITEL 6

Sie befinden sich hier (alle anderen auch)

Einstieg

Geolocation ist die Kunst, herauszufinden, wo Sie sich auf der Welt befinden, und dies (wenn Sie es möchten) allen anderen mitzuteilen, denen Sie vertrauen. Es gibt viele Möglichkeiten, Ihren Aufenthaltsort herauszufinden – über Ihre IP-Adresse, Ihre Wireless-Netzwerkverbindung, die Zelle, mit der Ihr Handy verbunden ist, oder spezielle GPS-Hardware, die Breiten- und Längengradangaben von Satelliten im All erhält.

Fragen an Professor Markup

F: Geolocation klingt bedrohlich. Kann ich das abschalten?

A: Die Wahrung der Privatssphäre ist natürlich ein Problem, wenn es darum geht, seinen aktuellen Aufenthaltsort einem entfernten Webserver mitzuteilen. Die Geolocation-API sagt explizit (*http://www.w3.org/TR/geolocation-API/#security*): »User-Agents dürfen Ortsinformationen nur mit der ausdrücklichen Genehmigung des Benutzers an Webseiten senden.« Anders gesagt: Wenn Sie Ihren Ort anderen nicht mitteilen wollen, müssen Sie das auch nicht.

Die Geolocation-API

Die Geolocation-API ermöglicht Ihnen, Websites Ihres Vertrauens Ihren Aufenthaltsort mitzuteilen. Breiten- und Längengrad sind für JavaScript auf der Seite verfügbar, das diese Informationen an einen entfernten Webserver senden kann, um damit ausgefallene ortsgebundene Dinge anzustellen – beispielsweise Ihnen die Geschäfte vor Ort zu nennen oder Ihre Position in einer Karte anzuzeigen.

Wie Sie in Tabelle 6-1 sehen können, wird die Geolocation-API von einigen wichtigen Browsern auf dem Desktop oder auf Mobilgeräten unterstützt. Darüber hinaus können manche ältere Browser und Geräte mithilfe von Wrapper-Bibliotheken unterstützt werden, wie wir später in diesem Kapitel sehen werden.

Tabelle 6-1: Geolocation-API-Unterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	3.5+	5.0+	5.0+	·	3.0+	2.0+

Neben der Unterstützung für die Standard-Geolocation-API gibt es eine Vielzahl von gerätspezifischen APIs auf anderen mobilen Plattformen. Diese werde ich später in diesem Kapitel behandeln.

Zeige mir den Code

Die Geolocation-API basiert auf einer neuen Eigenschaft des globalen `navigator`-Objekts: `navigator.geolocation`.

Die einfachste Verwendung der Geolocation-API sieht so aus:

```
function get_location() {
  navigator.geolocation.getCurrentPosition(show_map);
}
```

Hier fehlen Erkennung, Fehlerbehandlung und Alternativen. Ihre Webanwendung sollte wahrscheinlich zumindest die beiden ersten enthalten. Die Unterstützung für die Geolocation-API (siehe dazu den Abschnitt »Geolocation« auf Seite 27) können Sie mit Modernizr prüfen:

```
function get_location() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(show_map);
  } else {
    // Keine native Unterstützung; vielleicht probieren Sie es mit Gears?
  }
}
```

Was Sie tun, wenn es keine Geolocation-Unterstützung gibt, sei Ihnen überlassen. Die Gears-Alternative werde ich Ihnen gleich vorstellen, aber erst möchte ich erläutern, was während dieses Aufrufs von `getCurrentPosition()` passiert. Wie am Anfang dieses Kapitels erwähnt, muss man die Geolocation-Unterstützung zulassen. Das heißt, dass Ihr Browser Sie nie zwingt, einem entfernten Server Ihren aktuellen Aufenthaltsort mitzuteilen. Was der Benutzer sieht, ist von Browser zu Browser unterschiedlich. In Mozilla Firefox veranlasst ein Aufruf der `getCurrentPosition()`-Funktion der Geolocation-API den Browser, oben im Browserfenster eine Informationsleiste anzuzeigen. Diese sieht aus wie die in Abbildung 6-1.

Abbildung 6-1: Geolocation-Informationsleiste

Hier geschieht eine ganze Menge auf Benutzerseite:

- Ihnen wird mitgeteilt, dass eine Webseite Ihren Aufenthaltsort wissen will.
- Ihnen wird mitgeteilt, welche Website Ihren Aufenthaltsort wissen will.
- Sie können auf Mozillas »Location-Aware Browsing«-Hilfeseite gehen, die Ihnen erklärt, was zum Teufel hier geschieht.
- Sie können beschließen, Ihren Ort mitzuteilen.
- Sie können entscheiden, Ihren Ort nicht mitzuteilen.
- Sie können dem Browser sagen, dass er Ihre Entscheidung (den Ort mitzuteilen oder nicht) speichern soll, damit Sie die Informationsleiste für diese Website nie wieder sehen.

Außerdem ist diese Informationsleiste:

- Nicht modal, verhindert also nicht, zu einem anderen Browserfenster oder Browser-Tab zu wechseln.
- Tab-spezifisch, verschwindet also, wenn Sie zu einem anderen Browserfenster oder -Tab wechseln, und erscheint wieder, wenn Sie zum ursprünglichen Tab zurückkehren.
- Bedingungslos, d.h., Websites haben keine Möglichkeit, sie zu umgehen.
- Blockierend, d.h., die Website hat keine Möglichkeit, Ihren Ort zu ermitteln, während auf Ihre Antwort gewartet wird

Gerade haben Sie den JavaScript-Code gesehen, der bewirkt, dass diese Informationsleiste erscheint. Es ist ein einziger Funktionsaufruf, der eine Callback-Funktion erwartet (die hier `show_map()` heißt). Der Aufruf von `getCurrentPosition()` kehrt unmittelbar zurück. Das aber heißt nicht, dass Sie Zugriff auf die Position des Benutzers haben. Dieser Zugriff ist Ihnen erst in der Callback-Funktion garantiert. Die Callback-Funktion sieht so aus:

```
function show_map(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  // Zeigen wir eine Karte oder tun wir etwas anderes Interessantes!
}
```

Die Callback-Funktion wird mit einem einzigen Parameter aufgerufen, einem Objekt mit zwei Eigenschaften: `coords` und `timestamp`. `timestamp` ist einfach das Datum und die Uhrzeit, zu der der Ort berechnet wurde. (Da das alles asynchron verläuft, können Sie im Voraus nicht wissen, wann das passiert. Es kann eine Weile dauern, bis der Benutzer die Informationsleiste liest und zustimmt, dass sein Ort mitgeteilt werden kann, Geräte mit spezieller GPS könnten etwas länger brauchen, bis sie sich mit einem GPS-Satelliten

verbunden haben und so weiter.) Das coords-Objekt hat Eigenschaften wie latitude und longitude, die genau das repräsentieren, was der Name andeutet. Die Eigenschaften des position-Objekts sehen Sie in Tabelle 6-2.

Tabelle 6-2: Die Eigenschaften des Positionsobjekts

Eigenschaft	Typ	Anmerkungen
coords.latitude	double	Dezimalgrad
coords.longitude	double	Dezimalgrad
coords.altitude	double oder null	Meter über dem Referenzellipsoid
coords.accuracy	double	Meter
coords.altitudeAccuracy	double oder null	Meter
coords.heading	double oder null	Grad im Uhrzeigersinn vom echten Norden
coords.speed	double oder null	Meter/Sekunde
timestamp	DOMTimeStamp	wie ein Date()-Objekt

Nur drei der Eigenschaften sind garantiert (coords.latitude, coords.longitude und coords.accuracy). Die anderen können null sein, je nach den Fähigkeiten Ihres Geräts und dem Positionsserver im Hintergrund, mit dem es funktioniert. Die Eigenschaften heading und speed werden auf Basis der letzten Position des Benutzers berechnet, wenn möglich.

Fehlerbehandlung

Geolocation ist kompliziert. So viele Dinge können schieflaufen. Den Aspekt der »Benutzerzustimmung« haben wir bereits erwähnt. Wenn Ihre Webanwendung den Ort des Benutzers wissen will, der Sie Ihnen aber nicht mitteilen will, sind Sie aufgeschmissen. Der Benutzer gewinnt immer. Aber wie sieht das im Code aus? Es sieht aus wie das zweite Argument der Funktion getCurrentPosition() – eine Fehlerbehandlungs-Callback-Funktion:

```
navigator.geolocation.getCurrentPosition(
  show_map, handle_error)
```

Geht etwas schief, wird Ihre Fehler-Callback-Funktion mit einem PositionError-Objekt aufgerufen. Es hat die in Tabelle 6-3 aufgeführten Eigenschaften.

Tabelle 6-3: Die Eigenschaften des PositionError-Objekts

Eigenschaft	Typ	Anmerkungen
code	short	ein Enumerationswert
message	DOMString	nicht für Endbenutzer gedacht

Die Eigenschaft `code` hat einen der folgenden Werte:

- `PERMISSION_DENIED` (`1`), wenn der Benutzer auf den Button *Don't Share* klickt oder Ihnen den Zugriff auf seinen Ort auf andere Weise verweigert.
- `POSITION_UNAVAILABLE` (`2`), wenn das Netzwerk nicht verfügbar ist oder die Positionssatelliten nicht angesprochen werden können.
- `TIMEOUT` (`3`), wenn das Netzwerk verfügbar ist, es aber zu lange dauert, die Position des Benutzers zu berechnen. Wie lange »zu lange« ist? Wie Sie das definieren, werde ich Ihnen im nächsten Abschnitt zeigen.
- `UNKNOWN_ERROR` (`0`), wenn etwas anderes schiefgeht.

Zum Beispiel:

```
function handle_error(err) {
  if (err.code == 1) {
    // Der Benutzer hat Nein gesagt!
  }
}
```

Fragen an Professor Markup

F: Funktioniert die Geolocation-API auch auf der International Space Station, dem Mond oder anderen Planeten?

A: Die Geolocation-Spezifikation (*http://www.w3.org/TR/geolocation-API/#coordinates_interface*) sagt: »Das geografische Koordinatenreferenzsystem, das von den Attributen in dieser Schnittstelle genutzt wird, ist das World Geodetic System (2d) [WGS84]. Es wird kein anderes Referenzsystem unterstützt.« Die International Space Station umkreist die Erde, die Astronauten auf der Station (*http://twitter.com/Astro_TJ*) können ihren Ort also mit Breitengrad, Längengrad und Höhe beschreiben. Aber das World Geodetic System ist erdzentriert, kann also nicht genutzt werden, um Orte auf dem Mond oder anderen Planeten zu beschreiben.

Optionen! Ich verlange Optionen!

Einige beliebte Mobilgeräte – wie das iPhone und Android-Handys – unterstützen zwei Methoden, den Ort zu bestimmen. Die erste Methode trianguliert Ihre Position auf Basis der relativen Nähe zu verschiedenen Empfangsstationen Ihres Netzbetreibers. Diese Methode ist schnell und erfordert keine spezielle GPS-Hardware, gibt Ihnen aber nur eine ungefähre Vorstellung des Orts. Je nachdem, wie viele Antennen in Ihrem Bereich sind, kann diese »grobe Vorstellung« auf einen Häuserblock oder auch auf einen Kilometer in jede Richtung genau sein.

Die zweite Methode nutzt spezielle GPS -Hardware auf dem Gerät, um mit dedizierten GPS-Positionssatelliten zu kommunizieren, die die Erde umkreisen. GPS kann Ihre Position üblicherweise auf eine Genauigkeit von wenigen Metern berechnen. Der Nachteil

ist, dass der dedizierte GPS-Chip auf dem Gerät eine Menge Strom verbraucht. Deswegen ist er bei Handys und anderen mobilen Geräten ausgeschaltet, bis er benötigt wird. Das heißt, dass es eine Startverzögerung gibt, während der Chip seine Verbindung mit den GPS-Satelliten im Himmel initialisiert. Wenn Sie je Google Maps auf einem iPhone oder einem anderen Smartphone genutzt haben, haben Sie beide Methoden im Einsatz gesehen. Erst sehen Sie einen großen Kreis, der Ihre Position grob umreißt (die nächste Antenne findet), dann einen kleineren Kreis (Triangulation mit anderen Antennen), dann einen einzigen Punkt mit einer genauen Position (die von GPS-Satelliten geliefert wird).

Das erwähne ich aus folgendem Grund: Je nach Webanwendung brauchen Sie eventuell keine vollkommene Genauigkeit. Wenn Sie beispielsweise nach einer Aufstellung der in der Nähe laufenden Filme suchen, reicht eine ungefähre Position wahrscheinlich aus. Selbst in dicht besiedelten Städten wird es nicht so viele Kinos geben, und Sie werden sich wahrscheinlich ohnehin die Programme mehrerer Kinos anzeigen lassen. Aber wenn Sie Navigationshilfen in Echtzeit bieten wollen, müssen Sie genau wissen, wo sich der Benutzer befindet, damit Sie Dinge wie »Biegen Sie in 20 Metern rechts ab!« sagen können.

Die Funktion `getCurrentPosition()` erwartet ein optionales drittes Argument, ein `PositionOptions`-Objekt. Es gibt drei Eigenschaften, die Sie auf einem `PositionOptions`-Objekt setzen können (siehe Tabelle 6-4). Alle Eigenschaften sind optional; Sie können alle setzen, einige oder keine.

Tabelle 6-4: PositionOptions-Objekteigenschaften

Eigenschaft	Typ	Standard	Anmerkungen
`enableHighAccuracy`	`boolean`	`false`	`true` kann langsamer sein
`timeout`	`long`	(kein Standardwert)	in Millisekunden
`maximumAge`	`long`	`0`	in Millisekunden

Die Eigenschaft `enableHighAccuracy` aktiviert eine hohe Genauigkeit. Ist diese Eigenschaft auf `true` gesetzt, wird das Gerät versuchen, eine genaue Position zu liefern, falls das vom Gerät unterstützt wird und der Benutzer die Übermittlung seiner Position gestattet. iPhones und Android-Handys haben beide unterschiedliche Berechtigungen für die ungefähre und die genaue Positionsermittlung. Es kann also sein, dass `getCurrentPosition()` mit `enableHighAccuracy:true` scheitert, während ein Aufruf mit `enableHighAccuracy:false` erfolgreich ist.

Die Eigenschaft `timeout` gibt die Anzahl der Millisekunden an, die Ihre Webanwendung auf eine Position wartet. Der Timer wird erst in Gang gesetzt, wenn der Benutzer die Berechtigung zur Positionsermittlung erteilt hat. Die Zeitbeschränkung wird nicht für den Benutzer aufgestellt, sondern für das Netzwerk.

Die Eigenschaft `maximumAge` ermöglicht dem Gerät, sofort mit einer gespeicherten Position zu antworten. Nehmen wir beispielsweise an, dass Sie `getCurrentPosition()` das erste Mal aufrufen, der Benutzer zustimmt und Ihr Callback für den Erfolgsfall mit einer Position aufgerufen wird, die genau um 10:00 berechnet wurde. Nehmen Sie dann an, dass Sie

genau eine Minute später um 10:01 die Methode `getCurrentPosition()` erneut aufrufen, mit der Eigenschaft `maximumAge` auf `75000` gesetzt:

```
navigator.geolocation.getCurrentPosition(
  success_callback, error_callback, {maximumAge: 75000});
```

Damit sagen Sie, dass Sie nicht unbedingt die aktuelle Position des Benutzers benötigen. Sie wären damit zufrieden, zu erfahren, wo er vor 75 Sekunden (75000 Millisekunden) war. In diesem Fall weiß das Gerät, wo der Benutzer vor 60 Sekunden (60000 Millisekunden) war, weil die Position beim ersten Aufruf von `getCurrentPosition()` berechnet wurde. Da das im angegebenen Zeitfenster liegt, macht sich das Gerät nicht die Mühe, die aktuelle Position des Benutzers neu zu berechnen. Es liefert genau dieselbe Position wie beim ersten Aufruf: den gleichen Längen- und Breitengrad, die gleiche Genauigkeit und den gleichen Zeitstempel (10:00 AM).

Bevor Sie nach der Position des Benutzers fragen, sollten Sie darüber nachdenken, wie viel Genauigkeit Sie benötigen, und `enableHighAccuracy` entsprechend setzen. Wenn Sie die Position des Benutzers mehrfach bestimmen müssen, müssen Sie sich überlegen, wie alt die Informationen sein dürfen, wenn sie immer noch nützlich sein sollen, und `maximumAge` entsprechend setzen. Müssen Sie die Position des Benutzers kontinuierlich ermitteln, ist `getCurrentPosition()` nicht dafür geeignet. Dann müssen Sie `watchPosition()` einsetzen.

Die Funktion `watchPosition()` hat die gleiche Struktur wie `getCurrentPosition()`. Sie erwartet zwei Callback-Funktionen – eine erforderliche für den Erfolgsfall und eine optionale für die Fehlerbedingungen – und kann außerdem ein optionales `PositionOptions`-Objekt übernehmen, das genau die Eigenschaften hat, die Sie gerade kennengelernt haben. Der Unterschied ist, dass Ihre Callback-Funktion jedes Mal aufgerufen wird, wenn sich die Position des Benutzers ändert. Sie müssen nicht aktiv die Positionen abfragen. Das Gerät ermittelt das optimale Abfrageintervall und ruft Ihre Callback-Funktion jedes Mal auf, wenn sich die Position des Benutzers geändert hat. Das können Sie nutzen, um eine sichtbare Markierung auf einer Karte zu aktualisieren, Weganweisungen anzubieten – was Ihnen eben gefällt. Das ist vollkommen Ihnen überlassen.

Die Funktion `watchPosition()` liefert eine Zahl zurück. Diese Zahl sollten Sie irgendwo speichern. Wollen Sie irgendwann die Überwachung der Position des Benutzers beenden, können Sie die Methode `clearWatch()` aufrufen und ihr diese Zahl übergeben. Das Gerät hört dann auf, Ihre Callback-Funktion aufzurufen. Das funktioniert genau so wie die JavaScript-Funktionen `setInterval()` und `clearInterval()` (falls Sie mit diesen schon einmal gearbeitet haben).

Was ist mit dem IE?

Der Internet Explorer unterstützt die gerade beschriebene Geolocation-API des W3C nicht (siehe dazu den Abschnitt »Die Geolocation-API« auf Seite 121). Aber verzweifeln Sie nicht! Gears (*http://tools.google.com/gears/*) ist ein Open Source-Browser-Plug-in von Google, das auf Windows, Mac, Linux, Windows Mobile und Android funktioniert. Es bietet eine Reihe von Funktionen für ältere Browser, einschließlich einer Geolocation-API.

Das ist nicht ganz das Gleiche wie die W3C-Geolocation-API, erfüllt aber den gleichen Zweck.

Da wir gerade von älteren Plattformen reden, sollte ich darauf hinweisen, dass es eine Reihe von gerätespezifischen Geolocation-APIs auf Handyplattformen gibt. BlackBerry, Nokia, Palm und OMTP BONDI bieten jeweils ihre eigenen Geolocation-APIs. Natürlich arbeiten diese alle anders als Gears, das seinerseits anders funktioniert als die W3C-Geolocation-API. Tja!

Die Rettung: geo.js

geo.js (*http://code.google.com/p/geo-location-javascript/*) ist eine Open Source-JavaScript-Bibliothek unter der MIT-Lizenz, die die Unterschiede zwischen der W3C-Geolocation-API, der Gears-API und den verschiedenen APIs der unterschiedlichen Mobilplattformen glättet. Wollen Sie sie nutzen, müssen Sie am Ende Ihrer Seite zwei `<script>`-Elemente einsetzen. (Eigentlich könnten Sie sie auch an anderer Stelle platzieren, aber wenn Sie sie im `<head>` angeben, wird Ihre Seite langsamer geladen. Tun Sie das also nicht!)

Das erste Skript ist *gears_init.js* (*http://code.google.com/apis/gears/gears_init.js*). Es initialisiert Gears, falls es installiert ist. Das zweite Skript ist *geo.js* (*http://geo-location-javascript.googlecode.com/svn/trunk/js/geo.js*). Sie können sie folgendermaßen in Ihre Seite einschließen:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Einstieg in HTML5</title>
</head>
<body>
  ...
  <script src="gears_init.js"></script>
  <script src="geo.js"></script>
</body>
</html>
```

Jetzt können Sie jede Geolocation-API nutzen, die installiert ist:

```
if (geo_position_js.init()) {
  geo_position_js.getCurrentPosition(geo_success, geo_error);
}
```

Schauen wir uns das schrittweise an. Erst müssen Sie explizit eine `init()`-Funktion aufrufen. Diese `init()`-Funktion liefert `true`, wenn eine unterstützte Geolocation-API verfügbar ist:

```
if (geo_position_js.init()) {
```

Mit dem Aufruf von `init()` wird noch nicht die Position des Benutzers ermittelt. Es wird nur geprüft, ob die Ermittlung der Position möglich ist. Um die Position des Benutzers tatsächlich zu ermitteln, müssen Sie die Funktion `getCurrentPosition()` aufrufen:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Die Funktion `getCurrentPosition()` löst die Nachfrage beim Benutzer um Erlaubnis aus, die Position des Benutzers zu ermitteln und weiterzugeben. Wird die Geolocation durch Gears gestellt, wird ein Dialog eingeblendet, der den Benutzer fragt, ob er der Webseite vertraut und sie Gears nutzen darf. Bietet der Browser native Unterstützung der Geolocation-API, sieht der Dialog anders aus. Firefox 3.5 zeigt beispielsweise nur eine Informationsleiste über der Seite an, die fragt, ob der Benutzer der Website seine Position mitteilen will.

Die Funktion `getCurrentPosition()` erwartet zwei Callback-Funktionen als Argumente. War der `getCurrentPosition()`-Aufruf bei der Ermittlung der Position erfolgreich, d.h., hat der Benutzer seine Genehmigung gegeben und konnte die Geolocation-API ihre Arbeit erfolgreich ausführen, wird die Funktion aufgerufen, die als erstes Argument übergeben wurde. In diesem Beispiel ist das Callback für den Erfolgsfall die Funktion `geo_success`:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Die Callback-Funktion für den Erfolgsfall erhält ein Argument, das die Positionsinformationen beinhaltet:

```
function geo_success(p) {
  alert("Sie befinden sich bei den Koordinaten: Breite: " + p.coords.latitude +
        ", Länge: " + p.coords.longitude);
}
```

Konnte die Funktion `getCurrentPosition()` die Position des Benutzers nicht ermitteln – entweder weil er die Erlaubnis verweigerte oder weil die Geolocation-API aus einem anderen Grund scheiterte –, wird die Funktion aufgerufen, die als zweites Argument übergeben wurde. In diesem Beispiel heißt die Callback-Funktion für den Fehlerfall `geo_error`:

```
geo_position_js.getCurrentPosition(geo_success, geo_error);
```

Die Callback-Funktion für den Fehlerfall erhält keine Argumente:

```
function geo_error() {
  alert("Konnte Sie nicht finden!");
}
```

geo.js unterstützt die Funktion `watchPosition()` aktuell noch nicht. Wenn Sie kontinuierliche Positionsinformationen benötigen, müssen Sie selbst dafür sorgen, dass `getCurrentPosition()` regelmäßig aufgerufen wird.

Ein vollständiges Beispiel

Schauen wir uns ein vollständiges Beispiel auf Basis von *geo.js* an, das versucht, die Position des Benutzers zu erfragen, und gegebenenfalls eine Karte der unmittelbaren Umgebung anzeigt.

Wenn die Seite geladen wird, muss sie `geo_position_js.init()` aufrufen, um zu prüfen, ob Geolocation über eine der von *geo.js* unterstützten Schnittstellen verfügbar ist. Ist das der Fall, können Sie einen Link einrichten, auf den der Benutzer klicken kann, um seinen Ort zu erfahren. Ein Klick auf diesen Link ruft die Funktion `lookup_location()` auf, die Sie hier sehen:

```
function lookup_location() {
  geo_position_js.getCurrentPosition(show_map, show_map_error);
}
```

Wenn der Benutzer dem Nachhalten seiner Position zustimmt und der Hintergrunddienst zur Ermittlung der Position tatsächlich verfügbar ist, ruft *geo.js* die erste Callback-Funktion, `show_map()`, mit einem einzigen Argument, `loc`, auf. Das `loc`-Objekt hat eine `coords`-Eigenschaft, die Längen-, Breiten- und Genauigkeitsangaben enthält. (Dieses Beispiel nutzt die Genauigkeitsinformationen nicht.) Der Rest der Funktion `show_map()` verwendet die Google Maps-API, um eine eingebettete Karte anzuzeigen:

```
function show_map(loc) {
  $("#geo-wrapper").css({'width':'320px','height':'350px'});
  var map = new GMap2(document.getElementById("geo-wrapper"));
  var center = new GLatLng(loc.coords.latitude, loc.coords.longitude);
  map.setCenter(center, 14);
  map.addControl(new GSmallMapControl());
  map.addControl(new GMapTypeControl());
  map.addOverlay(new GMarker(center, {draggable: false, title: "Sie sind (ungefähr)
hier"}));
}
```

Kann *geo.js* die Position nicht bestimmen, wird die zweite Callback-Funktion aufgerufen, `show_map_error()`:

```
function show_map_error() {
  $("#live-geolocation").html('Konnte Ihre Position nicht ermitteln.');
}
```

Weitere Lektüre

- W3C-Geolocation-API (*http://www.w3.org/TR/geolocation-API/*)
- Gears (*http://tools.google.com/gears/*)
- BlackBerry-Geolocation-API (*http://www.tonybunce.com/2008/05/08/Blackberry-Browser-Amp-GPS.aspx*)

- Nokia-Geolocation-API (*http://www.forum.nokia.com/infocenter/index.jsp?topic=/Web_Developers_Library/GUID-4DDE31C7-EC0D-4EEC-BC3A-A0B0351154F8.html*)
- Palm-Geolocation-API (*http://developer.palm.com/index.php?option=com_content&view=article&id=1673#GPS-getCurrentPosition*)
- OMTP BONDI-Geolocation-API (*http://bondi.omtp.org/1.0/apis/geolocation.html*)
- *geo.js* (*http://code.google.com/p/geo-location-javascript/*), das Geolocation-API-Wrapper-Skript

KAPITEL 7

Lokaler Speicher für Webanwendungen – gestern, heute und morgen

Einstieg

Persistente lokale Speicherung ist einer der Bereiche, in denen native Clientanwendungen Webanwendungen traditionell überlegen waren. Bei nativen Anwendungen bietet das Betriebssystem üblicherweise eine Abstraktionsschicht zum Speichern und Abrufen anwendungsspezifischer Daten wie Einstellungen oder Laufzeitzuständen. Diese Werte können in einer Registrierung, in INI-Dateien, XML-Dateien oder an einem anderen Ort gespeichert werden, je nach Konvention der Plattform. Benötigt Ihre native Clientanwendung eine lokale Speicherung über Schlüssel/Wert-Paare hinaus, können Sie eine eigene Datenbank einbetten, ein eigenes Dateiformat erfinden oder eine beliebige andere Lösung implementieren.

Historisch hatten Webanwendungen diesen Luxus nicht. Cookies wurden früh in der Geschichte des Webs erfunden und können tatsächlich genutzt werden, um kleine Datenmengen dauerhaft lokal zu speichern. Aber sie haben ernste Nachteile, die ihren Einsatz für bestimmte Dinge verhindern:

- Cookies werden in jede HTTP-Anfrage eingeschlossen und bremsen Ihre Webanwendung dadurch aus, dass sie unnötig immer wieder die gleichen Daten übermitteln.
- Cookies werden in jede HTTP-Anfrage eingeschlossen und senden dabei unverschlüsselt Daten über das Internet (es sei denn, die gesamte Webanwendung wird über SSL bereitgestellt).
- Cookies sind auf rund 4 KB Daten beschränkt – genug, um Ihre Anwendung auszubremsen (siehe oben), aber nicht genug, um wirklich von Nutzen zu sein.

Was wir wirklich wollen, ist:

- Massenhaft Speicherplatz ...
- auf dem Client ...

- der über eine Seitenaktualisierung erhalten bleibt ...
- und nicht an den Server übertragen wird.

Dafür gab es bereits eine Menge Versuche, die alle letztendlich auf unterschiedliche Weise unbefriedigend waren.

Eine kurze Geschichte des lokalen Speichers vor HTML5

Am Anfang gab es nur den Internet Explorer. Oder zumindest war es das, was Microsoft die Welt gern glauben machen wollte. Zu diesem Zweck erfand Microsoft gegen Ende des ersten großen Browserkriegs viele tolle Dinge und schloss sie in den Browser ein, der alle Kriege ein für alle Male beenden sollte, den Internet Explorer. Eins dieser Dinge hieß DHTML Behaviors (DHTML-Verhalten), und eins dieser Verhalten heißt `userData` (Benutzerdaten).

`userData` ermöglicht Webseiten, bis zu 64 KB Daten pro Domain in einer hierarchischen, XML-basierten Struktur zu speichern. (Domains, denen vertraut wird, wie Intranetseiten, können die zehnfache Menge an Daten speichern. Na, meinen Sie nicht auch, 640 KB sollten für jede Art Anwendung reichen?) Der IE bietet keinerlei Erlaubnisdialog, und es gibt keine Möglichkeit, die Menge des verfügbaren Speicherplatzes zu erhöhen.

In 2002 führte Adobe in Flash 6 eine Funktion ein, die den unglücklichen und irreführenden Namen »Flash-Cookies« erhielt. In der Flash-Umgebung selbst heißt die Funktion passender Local Shared Objects oder LSOs. Kurz gesagt, sie ermöglicht Flash-Objekten, bis zu 100 KB Daten pro Domain zu speichern. 2005 entwickelte Brad Neuberg einen frühen Prototyp einer Flash-to-JavaScript-Bridge, die als AJAX Massive Storage System oder AMASS bezeichnet wird, aber ihr Nutzen hielt sich aufgrund einiger Mängel von Flash in Grenzen. Ab 2006, mit dem Aufkommen des ExternalInterface in Flash 8, wurde der Zugriff auf LSOs von JavaScript erheblich einfacher und schneller. Brad schrieb AMASS um und integrierte es unter dem Titel `dojox.storage` in das beliebte Dojo-Toolkit. Mit dieser Lösung erhält jede Domain die üblichen 100 KB Speicher »kostenlos«. Darüber hinaus wird der Benutzer bei jeder Größenerweiterung des Speicherplatzes um Bestätigung gefragt (1 MB, 10 MB usw.).

2007 startete Google Gears, ein Open Source-Browser-Plug-in, das Browsern zusätzliche Fähigkeiten geben sollte. (Wir haben Gears bereits im Kontext der Geolocation-API-Unterstützung im Internet Explorer – siehe dazu den Abschnitt »Was ist mit dem IE?« auf Seite 127 – behandelt). Gears bietet eine API zur Einbettung einer SQL-Datenbank auf Basis von SQLite. Nachdem einmal die Berechtigung des Benutzers eingeholt ist, kann Gears eine unbegrenzte Menge an Daten pro Domain in SQL-Datenbanktabellen speichern.

In der Zwischenzeit arbeiteten Brad Neuberg und andere an `dojox.storage` weiter, um eine einheitliche Schnittstelle für all diese Plug-ins und APIs zu schaffen. Ab 2009 konnte `dojox.storage` automatisch Adobe Flash, Gears, Adobe AIR und einen frühen Prototyp

von HTML5 Storage erkennen, der nur in älteren Firefox-Versionen implementiert war, und eine einheitliche Schnittstelle auf ihnen bieten.

Wenn Sie diese Lösungen einmal überblicken, werden Sie ein Muster erkennen: Alle sind entweder auf einen einzigen Browser bezogen oder von einem externen Plug-in abhängig. Trotz heroischer Anstrengungen, die Unterschiede zu glätten (in `dojox.storage`), bieten sie alle radikal unterschiedliche Schnittstellen, haben unterschiedliche Speicherbegrenzungen und bieten andere Benutzererfahrungen. Das also ist das Problem, das sich HTML5 zu lösen aufmachte: eine standardisierte API zu bieten, die nativ und konsistent in mehreren Browsern implementiert ist, ohne von externen Plug-ins abhängig zu sein.

Der erste Auftritt von HTML5 Storage

Was ich hier als »HTML5 Storage« bezeichne, ist eigentlich eine Spezifikation namens Web Storage. Diese war zunächst Teil der eigentlichen HTML5-Spezifikation, wurde aber aus uninteressanten politischen Gründen in eine eigene Spezifikation ausgelagert. Einige Browserhersteller nennen sie auch »Local Storage« oder »DOM Storage«. Die Namenssituation wird durch einige verwandte, ähnlich benannte aufkommende Standards weiter verkompliziert, mit denen wir uns später in diesem Kapitel befassen werden.

Was also ist HTML5 Storage? Einfach gesagt: Es ist eine Möglichkeit für Browser, benannte Schlüssel/Wert-Paare lokal im Client-Webbrowser zu speichern. Wie die in Cookies gespeicherten Daten bleiben diese Daten erhalten, wenn Sie die Website verlassen, Ihren Browser-Tab schließen, den Browser beenden oder etwas anderes machen. Aber anders als Cookies werden diese Daten nie an entfernte Webserver übertragen (es sei denn, Sie machen sich die Mühe, sie manuell zu versenden). Und im Unterschied zu allen vorangehenden Versuchen, persistente lokale Speicherung (wie wir sie im letzten Abschnitt behandelt haben) zu ermöglichen, ist sie nativ in Webbrowsern implementiert und so auch dann verfügbar, wenn keine externen Plug-ins verfügbar sind.

Welche Browser? Wie Sie in Tabelle 7-1 sehen, wird HTML5 von den letzten Versionen so gut wie aller Browser unterstützt ... sogar vom Internet Explorer!

Tabelle 7-1: HTML5 Storage-Unterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
8.0+	3.5+	4.0+	4.0+	10.5+	2.0+	2.0+

Aus Ihrem JavaScript-Code greifen Sie auf HTML5 Storage über das `localStorage`-Objekt auf dem globalen `window`-Objekt zu. Bevor Sie es verwenden können, sollten Sie prüfen, ob der Browser es unterstützt (siehe dazu den Abschnitt »Local Storage« auf Seite 24):

```
function supports_html5_storage() {
  return ('localStorage' in window) && window['localStorage'] !== null;
}
```

Statt eigenhändig diese Funktion zu schreiben, können Sie auch Modernizr (siehe dazu den Abschnitt »Modernizr: Eine Bibliothek zur HTML5-Erkennung« auf Seite 18) nutzen, um die Unterstützung für HTML5 Storage zu prüfen:

```
if (Modernizr.localstorage) {
  // window.localStorage ist verfügbar!
} else {
  // Keine native Unterstützung für HTML5 Storage :(
  // Vielleicht versuchen Sie es mit dojox.storage oder einen externen Lösung!
}
```

HTML5 Storage verwenden

HTML5 Storage basiert auf benannten Schlüssel/Wert-Paaren. Sie speichern Daten auf Basis eines benannten Schlüssels, über den Sie die Daten dann wieder abrufen können:

```
interface Storage {
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
};
```

Der benannte Schlüssel ist ein String. Die Daten können von jedem beliebigen Typ sein, der von JavaScript unterstützt wird, wie Strings, Booleans, Integer und Floats. Aber tatsächlich werden derartige Daten als String gespeichert. Wenn Sie etwas anderes als Strings speichern und abrufen, müssen Sie Funktionen wie `parseInt()` oder `parseFloat()` nutzen, um die Daten in den erwarteten JavaScript-Datentyp zu zwingen.

Wird `setItem()` mit einem benannten Schlüssel aufgerufen, der bereits besteht, wird der vorhandene Wert ohne Meldung überschrieben. Sollte `getItem()` auf einem nicht existenten Schlüssel aufgerufen werden, wird `null` geliefert und keine Exception ausgelöst.

Wie andere JavaScript-Objekte können Sie das `localStorage`-Objekt als assoziatives Array behandeln. Anstelle der Methoden `getItem()` und `setItem()` können Sie auch einfach eckige Klammern verwenden. Schauen Sie sich zum Beispiel folgendes Codefragment an:

```
var foo = localStorage.getItem("bar");
// ...
localStorage.setItem("bar", foo);
```

Es könnte mit eckigen Klammern folgendermaßen umgeschrieben werden:

```
var foo = localStorage["bar"];
// ...
localStorage["bar"] = foo;
```

Es gibt auch Methoden, um den Wert eines angegebenen benannten Schlüssels zu entfernen und den gesamten Speicherbereich zu löschen (also alle Schlüssel und Werte auf einmal zu löschen):

```
interface Storage {
  deleter void removeItem(in DOMString key);
  void clear();
};
```

Wird `removeItem()` mit einem Schlüssel aufgerufen, den es nicht gibt, passiert nichts.

Schließlich gibt es noch eine Eigenschaft zur Ermittlung der Gesamtzahl von Werten im Speicherbereich und zum Durchlaufen aller Schlüssel über den Index (um die Namen aller Schlüssel abzurufen):

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
};
```

Rufen Sie `key()` mit einem Index auf, der nicht zwischen 0 und (`length`–1) liegt, liefert die Funktion `null`.

Änderungen im HTML5-Speicherbereich nachhalten

Wenn Sie mit Code nachhalten möchten, wann sich der Speicherbereich ändert, können Sie das `storage`-Event abfangen. Dieses wird auf dem `window`-Objekt abgesetzt, wenn `setItem()`, `removeItem()` oder `clear()` aufgerufen wird und sich tatsächlich etwas ändert. Setzen Sie ein Element auf den Wert, den es bereits hat, oder rufen Sie `clear()` auf, wenn es keine benannten Schlüssel gibt, wird das `storage`-Event nicht ausgelöst, da sich tatsächlich nichts am Speicherbereich geändert hat.

Das `storage`-Event wird überall dort unterstützt, wo das `localStorage`-Objekt unterstützt wird, Internet Explorer 8 eingeschlossen. Der IE8 unterstützt den W3C-Standard `addEventListener` nicht (während das im IE9 endlich eingeführt werden wird). Deswegen müssen Sie, wenn Sie sich für das `storage`-Event registrieren, prüfen, welchen Mechanismus Ihr Browser unterstützt. (Haben Sie das bereits für andere Events gemacht, können Sie zum Ende dieses Abschnitts springen. Das `storage`-Event abzufangen, funktioniert genau so wie das Abfangen jedes anderen Events. Ziehen Sie zum Registrieren Ihrer Event-Handler den Einsatz einer JavaScript-Bibliothek wie jQuery vor, können Sie das auch beim `storage`-Event tun.) So geht es:

```
if (window.addEventListener) {
  window.addEventListener("storage", handle_storage, false);
} else {
  window.attachEvent("onstorage", handle_storage);
};
```

Die Callback-Funktion `handle_storage()` wird mit einem `StorageEvent`-Objekt aufgerufen – außer im Internet Explorer, bei dem das Event-Objekt in `window.event` gespeichert wird:

```
function handle_storage(e) {
  if (!e) { e = window.event; }
}
```

An diesem Punkt ist die Variable `e` ein `StorageEvent`-Objekt, das die nützlichen Eigenschaften hat, die in Tabelle 7-2 aufgeführt werden.

Tabelle 7-2: Die Eigenschaften des StorageEvent-Objekts

Eigenschaft	Typ	Beschreibung
`key`	String	Der benannte Schlüssel, der eingefügt, entfernt oder verändert wurde.
`oldValue`	Beliebig	Der vorangegangene (jetzt überschriebene) Wert oder `null`, falls ein neues Element eingefügt wurde.
`newValue`	Beliebig	Der neue Wert oder `null`, falls ein Element entfernt wurde.
`url`[1]	String	Die Seite, die die Methode aufrief, die diese Veränderung auslöste.

Das `storage`-Event kann nicht abgebrochen werden. In der Callback-Funktion `handle_storage()` gibt es keine Möglichkeit, die Durchführung der Änderung zu verhindern. Der Browser sagt Ihnen: »Eh, das ist passiert. Kannst nichts dran machen, mein Freund, aber ich dachte, ich halte dich mal auf dem Laufenden.«

Einschränkungen in aktuellen Browsern

Als ich die Geschichte der Vorläufer von HTML5 Storage mit externen Plug-ins erzählte (siehe dazu den Abschnitt »Eine kurze Geschichte des lokalen Speichers vor HTML5« auf Seite 134), habe ich viel Gewicht darauf gelegt, die Grenzen der einzelnen Techniken, Speichergrenzen beispielsweise, zu erwähnen. Zu den Beschränkungen des jetzt standardisierten HTML5 Storage habe ich noch nichts gesagt.

Standardmäßig erhält jede Quelle 5 MB Speicherplatz. Das ist bei den unterschiedlichen Browsern erstaunlich konsistent, obwohl es in der HTML5 Storage-Spezifikation eher als Empfehlung ausgedrückt wird. Sie müssen bedenken, dass Ihre Daten als String gespeichert werden, nicht im ursprünglichen Format. Wenn Sie eine Menge Integer- oder Float-Werte speichern, kann sich der Unterschied in der Repräsentation bemerkbar machen: Jede Ziffer der Zahl wird als Zeichen gespeichert, nicht in der üblichen Repräsentation einer Gleitkommazahl.

Überschreiten Sie Ihre Speicherquote, wird eine `QUOTA_EXCEEDED_ERR` ausgelöst. Die Antwort auf die wahrscheinlich folgende Frage, »Kann ich den Benutzer um mehr Speicherplatz bitten?«, ist Nein. Als die geschrieben wurde, bot kein Browser einen Mechanismus, der es Webentwicklern ermöglichen würde, mehr Speicherplatz anzufordern. Einige Browser wie Opera erlauben dem Benutzer, die Speicherquote für jede Site festzulegen. Das aber ist eine Aktion, die rein vom Benutzer ausgeht, nicht etwas, das man als Webentwickler in eine Anwendung einbauen kann.

HTML5 Storage im Einsatz

Schauen wir uns anhand eines Beispiels HTML5 Storage im Einsatz an. Denken Sie zurück an das Halma-Spiel, das wir in Kapitel 4 aufgebaut haben (siehe dazu den Abschnitt »Ein vollständiges Beispiel« auf Seite 78). Das Spiel hat ein kleines Problem:

1 Die Eigenschaft `url` hieß ursprünglich `uri`. Einige Browser wurden mit dieser Eigenschaft ausgeliefert, bevor sich die Spezifikation änderte. Zur maximalen Kompatibilität sollten Sie also prüfen, ob die Eigenschaft `url` existiert; wenn nicht, sollten Sie prüfen, ob es dann vielleicht `uri` gibt.

Wenn Sie mitten im Spiel das Browserfenster schließen, geht der aktuelle Spielstand verloren. Aber mit HTML5 Storage können wir diesen lokal im Browser selbst speichern. Eine Live-Demonstration können Sie unter *http://diveintohtml5.org/examples/localstorage-halma.html* sehen. Machen Sie ein paar Züge, schließen Sie das Browserfenster und öffnen Sie es dann wieder. Wenn Ihr Browser HTML5 Storage unterstützt, sollte sich die Beispielseite an die genaue Spielsituation erinnern, einschließlich der Anzahl gemachter Züge, der Position aller Steine auf dem Spielbrett und des gerade ausgewählten Spielsteins.

Wie das funktioniert? Jedes Mal, wenn sich etwas im Spiel ändert, rufen wir die folgende Funkion auf:

```
function saveGameState() {
    if (!supportsLocalStorage()) { return false; }
    localStorage["halma.game.in.progress"] = gGameInProgress;
    for (var i = 0; i < kNumPieces; i++) {
      localStorage["halma.piece." + i + ".row"] = gPieces[i].row;
      localStorage["halma.piece." + i + ".column"] = gPieces[i].column;
    }
    localStorage["halma.selectedpiece"] = gSelectedPieceIndex;
    localStorage["halma.selectedpiecehasmoved"] gSelectedPieceHasMoved;
    localStorage["halma.movecount"] = gMoveCount;
    return true;
}
```

Wie Sie sehen können, nutzt die Funktion das `localStorage`-Objekt, um zu speichern, ob gerade ein Spiel läuft (`gGameInProgress`, einen Booleschen Wert). Ist das der Fall, durchläuft sie alle Steine (`gPieces`, ein JavaScript-`Array`) und speichert die Zeilen- und Spaltennummer jedes Steins. Dann speichert sie einige zusätzliche Spieldaten, einschließlich des aktuell ausgewählten Spielsteins (`gSelectedPieceIndex`, ein Integer), ob der Spielstein in der Mitte einer möglicherweise langen Serie von Sprüngen steht (`gSelectedPieceHasMoved`, ein Boolescher Wert) und die Gesamtzahl der bisher gemachten Züge (`gMoveCount`, ein Integer).

Wird die Seite geladen, wird nicht automatisch die Funktion `newGame()` aufgerufen, die diese Variablen auf festgeschriebene Werte setzen würde, sondern die Funktion `resumeGame()`. Diese nutzt HTML5 Storage, um zu prüfen, ob lokal der Zustand eines laufenden Spiels gespeichert ist. Ist das der Fall, werden die Werte über das `localStorage`-Objekt wiederhergestellt:

```
function resumeGame() {
    if (!supportsLocalStorage()) { return false; }
    gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
    if (!gGameInProgress) { return false; }
    gPieces = new Array(kNumPieces);
    for (var i = 0; i < kNumPieces; i++) {
      var row = parseInt(localStorage["halma.piece." + i + ".row"]);
      var column = parseInt(localStorage["halma.piece." + i + ".column"]);
      gPieces[i] = new Cell(row, column);
    }
    gNumPieces = kNumPieces;
    gSelectedPieceIndex = parseInt(localStorage["halma.selectedpiece"]);
```

```
        gSelectedPieceHasMoved = localStorage["halma.selectedpiecehasmoved"] == "true";
        gMoveCount = parseInt(localStorage["halma.movecount"]);
        drawBoard();
        return true;
    }
```

Der wichtigste Teil dieser Funktion ist der Haken, den ich weiter oben in diesem Kapitel erwähnte und hier noch einmal wiederholen will: *Daten werden als Strings gespeichert. Wenn Sie etwas anderes als Strings speichern, müssen Sie es selbst wieder in die passende Form zwingen, wenn Sie es abrufen.* Der Schalter, der anzeigt, ob ein Spiel läuft (`gGameInProgress`), ist beispielsweise ein Boolescher Wert. In der Funktion `saveGameState()` haben wir diesen einfach gespeichert und uns keine Gedanken über den Datentyp gemacht:

```
localStorage["halma.game.in.progress"] = gGameInProgress;
```

Aber in der Funktion `resumeGame()` müssen wir den Wert, den wir aus dem lokalen Speicherbereich abgerufen haben, als String behandeln und den richtigen Booleschen Wert manuell wiederherstellen:

```
gGameInProgress = (localStorage["halma.game.in.progress"] == "true");
```

Auf ähnliche Weise verfahren wir mit dem ursprünglichen Integer für `gMoveCount`. In `saveGameState()` haben wir ihn einfach gespeichert:

```
localStorage["halma.movecount"] = gMoveCount;
```

Aber in `resumeGame()` müssen wir den gelesenen Wert mit der eingebauten JavaScript-Funktion `parseInt()` in einen Integer umwandeln:

```
gMoveCount = parseInt(localStorage["halma.movecount"]);
```

Über benannte Schlüssel/Wert-Paare hinaus: Konkurrierende Vorstellungen

Während die Vergangenheit voller Hacks und Workarounds ist (siehe dazu den Abschnitt »Eine kurze Geschichte des lokalen Speichers vor HTML5« auf Seite 134), erscheint die aktuelle Lage von HTML5 Storage überraschend rosig. Eine neue API wurde standardisiert und in den wichtigsten Browsern, Plattformen und Geräten implementiert. Als Webentwickler sieht man so etwas nicht alle Tage, oder? Aber man erwartet mehr vom Leben als 5 MB benannter Schlüssel/Wert-Paare, und die Zukunft persistenter lokaler Speicherung ist ... wie soll ich es sagen? Na gut, es gibt da einige konkurrierende Vorstellungen.

Eine Vorstellung basiert auf einem Akronym, das Ihnen wahrscheinlich vertraut ist: SQL. 2007 startete Google Gears, ein browserübergreifendes Open Source-Plug-in, das eine eingebettete Datenbank auf Basis von SQLite beinhaltete. Dieser frühe Prototyp beeinflusste später die Entwicklung der Web SQL Database-Spezifikation. Web SQL Database (zuvor unter dem Namen »WebDB« bekannt) stellt einen schlanken Wrapper um eine SQL-Datenbank zur Verfügung, der es Ihnen ermöglicht, Dinge wie dieses aus JavaScript auszuführen:

```
openDatabase('documents', '1.0', 'Local document storage', 5*1024*1024,
function (db) {
  db.changeVersion('', '1.0', function (t) {
    t.executeSql('CREATE TABLE docids (id, name)');
  }, error);
});
```

Wie Sie sehen, steckt der größte Teil der Aktion in dem String, den Sie der Methode `executeSql()` übergeben. Dieser String kann jede unterstützte SQL-Anweisung sein, einschließlich `SELECT`, `UPDATE`, `INSERT` und `DELETE`. Das ist genau wie bei der Programmierung eines Datenbank-Backends, nur dass Sie es hier aus JavaScript tun! Welch eine Freude!

Tabelle 7-3 zeigt, welche Browser die Web SQL Database-Spezifikation implementiert haben.

Tabelle 7-3: Web SQL Database-Unterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	·	4.0+	4.0+	10.5+	3.0+	·

Wenn Sie im Leben bereits das Vergnügen hatten, mit unterschiedlichen Datenbanksystemen zu arbeiten, wissen Sie natürlich, dass »SQL« eher ein Marketingbegriff als ein stabiler Standard ist. (Es gibt Leute, die das Gleiche von »HTML5« sagen ... aber wir können uns ja nicht mit allem und jedem auseinandersetzen.) Sicher, eigentlich gibt es eine SQL-Spezifikation – der Name des Standards ist SQL-92. Aber es gibt auf der ganzen Welt nicht einen Datenbankserver, der dieser Spezifikation und nur dieser Spezifikation genügt. Es gibt Oracles SQL, Microsofts SQL, MySQLs SQL, PostgreSQLs SQL und SQLites SQL. Und jedes dieser Produkte hat mit der Zeit eigene SQL-Funktionen eingeführt. Es reicht also nicht einmal, von »SQLites SQL« zu sprechen, um genau zu bezeichnen, wovon man redet. Sie müssen »die Version von SQL, die mit SQLite Version X.Y.Z ausgeliefert wurde« sagen.

All das führt uns zu folgendem Haftungsausschluss, der groß über der Web SQL Database-Spezifikation steht:

> Diese Spezifikation ist in eine Sackgasse geraten: Alle betroffenen Implementierer haben das gleiche SQL-Backend (SQlite) genutzt, aber wir benötigen mehrere unabhängige Implementierungen, um den Standardisierungspfad fortzusetzen. Bis sich ein anderer Implementierer dafür interessiert, diese Spezifikation zu implementieren, bleibt die Beschreibung des SQL-Dialekts einfach eine Referenz für SQlite. Das aber ist für einen Standard nicht akzeptabel.

Vor diesem Hintergrund möchte ich Ihnen eine andere, konkurrierende Vorstellung für persistente, lokale Speicherung für Webanwendungen vorstellen: die Indexed Database-API, die früher als »WebSimpleDB« bezeichnet wurde und jetzt nur noch zärtlich »IndexedDB« genannt wird.

Die Indexed Database-API bietet einen *Objektspeicher*. Ein Objektspeicher teilt viele Aspekte mit einer SQL-Datenbank. Es gibt »Datenbanken« mit »Datensätzen«, und jeder Datensatz hat eine Anzahl von »Feldern«. Jedes Feld hat einen bestimmten Datentyp, der

definiert wird, wenn die Datenbank erstellt wird. Sie können eine Untermenge von Datensätzen auswählen und dann mit einem »Cursor« durchlaufen. Änderungen am Objektspeicher werden über »Transaktionen« abgewickelt.

Wenn Sie bereits SQL-Datenbanken programmiert haben, werden Ihnen diese Begriffe wahrscheinlich vertraut vorkommen. Der Hauptunterschied ist, dass dieser Objektspeicher keine strukturierte Abfragesprache bietet. Sie bauen keine Anweisungen wie `"SELECT * from USERS where ACTIVE = 'Y'"` auf. Stattdessen nutzen Sie die Methoden, die der Objektspeicher bereitstellt, um einen Cursor auf die Datenbank namens `USERS` zu öffnen, die Datensätze zu durchlaufen, die Datensätze für inaktive Benutzer herauszufiltern und Akzessormethoden einzusetzen, um die Werte der Felder der verbleibenden Datensätze abzurufen. Eine der ersten Einführungen in IndexedDB (*http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/*) ist ein guter Einstieg in die Funktionsweise von IndexedDB und liefert eine Gegenüberstellung von IndexedDB und Web SQL Database.

Als ich dies schrieb, war IndexedDB noch in keinem wichtigeren Browser implementiert. Anfang Juni 2010 hat Mozilla erklärt, »in den nächsten Wochen einige Test-Builds zu erstellen«, aber es gibt noch keine Garantie, dass die Implementierung in Firefox 4 ausgeliefert wird. (Im Gegensatz dazu hat Mozilla bekannt gegeben, dass es nie Web SQL Database implementieren wird.) Google hat gesagt, man erwäge IndexedDB-Unterstützung, und selbst Microsoft gab bereits zu, dass IndexedDB »eine wunderbare Lösung für das Web« sei.

Was können Sie, als Webentwickler, also mit IndexedDB tun? Zurzeit absolut nichts. In einem Jahr? Vielleicht etwas. Werfen Sie einen Blick in den Abschnitt »Weitere Lektüre«. Dort finden Sie einige Links zu guten Einführungen, die Ihnen auf die Sprünge helfen.

Weitere Lektüre

HTML5 Storage:

- HTML5 Storage-Spezifikation (*http://dev.w3.org/html5/webstorage/*)
- »Einführung in DOM Storage« (*http://msdn.microsoft.com/de-de/library/cc197062(VS.85).aspx*) auf MSDN
- »Web Storage: easier, more powerful client-side data storage« (*http://dev.opera.com/articles/view/web-storage/*) von Shwetank Dixit
- »Introduction to DOM Storage« (*https://developer.mozilla.org/en/dom/storage*) im Mozilla Developer Center (Hinweis: Diese Seite widmet sich vorwiegend Firefox' Prototypimplementierung eines `globalStorage`-Objekts, eines nicht standardgemäßen Vorläufers von `localStorage`. Unterstützung für die `localStorage`-Schnittstelle hat Mozilla in Firefox 3.5 eingeführt.)
- »Unlock local storage for mobile Web applications with HTML 5« (*http://www.ibm.com/developerworks/xml/library/x-html5mobile2/*), eine Einführung auf IBM DeveloperWorks

Frühe Arbeiten von Brad Neuberg und anderen (vor HTML5):

- »Internet Explorer Has Native Support for Persistence?!?!« (*http://codinginparadise.org/weblog/2005/08/ajax-internet-explorer-has-native.html*), über das `userData`-Objekt im IE
- Dojo Storage (*http://docs.google.com/View?docid=dhkhksk4_8gdp9gr#dojo_storage*), Teil eines umfangreicheren Tutorials zur (mittlerweile veralteten) Dojo-Offlinebibliothek
- `dojox.storage.manager`-API-Referenz (*http://api.dojotoolkit.org/jsdoc/HEAD/dojox.storage.manager*)
- `dojox.storage`-Subversion-Repository (*http://svn.dojotoolkit.org/src/dojox/trunk/storage/*)

Web SQL Database:

- Web SQL Database-Spezifikation (*http://dev.w3.org/html5/webdatabase/*)
- »Introducing Web SQL Databases« (*http://html5doctor.com/introducing-web-sql-databases/*), von Remy Sharp
- Web Database-Demonstration (*http://html5demos.com/database*)
- *persistence.js* (*http://zef.me/2774/persistence-js-an-asynchronous-javascript-orm-for-html5gears*), ein »asynchroner JavaScript ORM«, der auf Web SQL Database und Gears aufsetzt

IndexedDB:

- Indexed Database API-Spezifikation (*http://dev.w3.org/2006/webapi/IndexedDB/*)
- »Beyond HTML5: Database APIs and the Road to IndexedDB« (*http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/*), von Arun Ranganathan und Shawn Wilsher
- »Firefox 4: An early walk-through of IndexedDB« (*http://hacks.mozilla.org/2010/06/comparing-indexeddb-and-webdatabase/*), von Arun Ranganathan

KAPITEL 8

Gehen wir offline

Einstieg

Was ist eine Offline-Webanwendung? Auf den ersten Blick klingt das wie ein Widerspruch in sich. Normalerweise laden Webseiten Dinge herunter und stellen sie dar. Herunterladen aber impliziert eine Netzwerkverbindung. Wie bitte kann man etwas herunterladen, wenn man offline ist? Das geht natürlich nicht. Aber Sie können herunterladen, wenn Sie online sind. Und so funktionieren HTML5-Offlineanwendungen.

Auf der einfachsten Ebene ist eine Offline-Webanwendung einfach eine Liste von URLs, die auf HTML-, CSS- oder JavaScript-Dateien, Bilder oder andere vorhandene Ressourcen verweisen. Die Homepage der Offline-Webanwendung zeigt auf diese Liste, die als *Manifest-Datei* bezeichnet wird. Das ist eine gewöhnliche Textdatei, die sich an anderer Stelle auf dem Webserver befindet. Ein Webbrowser, der HTML5-Offlineanwendungen implementiert, liest die Liste der URLs aus der Manifest-Datei, lädt die Ressourcen herunter, speichert sie lokal zwischen und hält automatisch die lokalen Kopien aktuell, wenn sie sich ändern. Wenn Sie versuchen, auf die Webanwendung ohne Netzwerkverbindung zuzugreifen, nutzt Ihr Webbrowser automatisch die lokalen Kopien.

Von da ab bleibt die meiste Arbeit Ihnen überlassen, dem Webentwickler. Im DOM gibt es einen Schalter, der Ihnen sagt, ob Sie online oder offline sind, und es gibt Events, die ausgelöst werden, wenn sich Ihr Status ändert (eine Minute sind Sie offline und die nächste Minute online oder umgekehrt). Aber das war es auch schon. Wenn Ihre Anwendung Daten erstellt oder einen Zustand speichert, müssen Sie sich darum kümmern, dass die Daten lokal gespeichert werden (siehe Kapitel 7), solange Sie offline sind, und mit dem entfernten Server synchronisiert werden, wenn Sie wieder online sind. Anders gesagt, HTML5 kann Ihre Webanwendung offline nehmen, aber was Sie dann tun, bleibt Ihnen überlassen. Tabelle 8-1 zeigt, welche Browser offline Webanwendungen unterstützen.

Tabelle 8-1: Offlineunterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
·	✓	✓	✓	·	✓	✓

Das Cache-Manifest

Eine Offline-Webanwendung basiert auf einer Cache-Manifest-Datei. Wie ich bereits erwähnte, ist die Manifest-Datei eine Liste aller Ressourcen, auf die Ihre Webanwendung Zugriff braucht, wenn sie genutzt werden soll, während keine Netzwerkverbindung besteht. Der Prozess des Herunterladens und Speicherns der Ressourcen wird durch ein `manifest`-Attribut angestoßen, mit dem Sie aus Ihrem `<html>`-Element auf die Manifest-Datei verweisen:

```
<!DOCTYPE HTML>
<html manifest="/cache.manifest">
<body>
...
</body>
</html>
```

Ihre Cache-Manifest-Datei kann sich an beliebiger Stelle Ihres Webservers befinden, muss aber mit dem richtigen Content-Type, `text/cache-manifest`, geliefert werden. Wenn Sie einen Apache-basierten Webserver haben, reicht es wahrscheinlich, wenn Sie einfach der *.htaccess*-Datei in Ihrem Web-Wurzelverzeichnis eine `AddType`-Direktive hinzufügen:

```
AddType text/cache-manifest .manifest
```

Stellen Sie dann sicher, dass der Name Ihrer Cache-Manifest-Datei mit *.manifest* endet. Nutzen Sie einen anderen Webserver oder eine andere Apache-Konfiguration, schauen Sie in der Dokumentation Ihres Servers nach, wie man den `Content-Type`-Header steuert.

Fragen an Professor Markup

F: Meine Webanwendung umfasst mehrere Seiten. Benötige ich ein `manifest`*-Attribut in jeder Seite, oder reicht es, wenn ich es auf der Portalseite angebe?*

A: Jede Seite Ihrer Webanwendung braucht ein `manifest`-Attribut, das auf das Cache-Manifest für die gesamte Anwendung verweist.

Weiter im Text: Ihre HTML-Seiten verweisen auf eine Cache-Manifest-Datei, und Ihre Cache-Manifest-Datei wird mit dem richtigen `Content-Type`-Header geliefert. Aber was kommt in die Manifest-Datei? Das ist der Punkt, an dem es interessant wird.

Die erste Zeile in jeder Cache-Manifest-Datei ist:

```
CACHE MANIFEST
```

Danach sind alle Manifest-Dateien in drei Teile aufgespalten: einen »expliziten« Abschnitt, einen Ausweichabschnitt und die sogenannte »Online Whitelist«. Jeder dieser Abschnitte hat eine Überschrift, die auf einer eigenen Zeile steht. Hat die Manifest-Datei keine Abschnittsüberschriften, stehen alle Ressourcen implizit im expliziten Abschnitt. Versuchen Sie, sich nicht zu fest an die Terminologie klammern, damit Ihnen nicht der Kopf explodiert.

Hier ist eine gültige Manifest-Datei. Sie führt drei Ressourcen auf – eine CSS-Datei, eine JavaScript-Datei und ein JPEG-Bild:

```
CACHE MANIFEST
/clock.css
/clock.js
/clock-face.jpg
```

Dieses Cache-Manifest hat keine Abschnittsüberschriften, also sind alle aufgeführten Ressourcen explizit. Explizite Ressourcen werden heruntergeladen, lokal gecacht und anstelle des jeweilige Onlinegegenstücks verwendet, wenn keine Netzwerkverbindung besteht. Beim Laden dieses Cache-Manifests lädt Ihr Browser *clock.css*, *clock.js* und *clock-face.jpg* aus dem Wurzelverzeichnis Ihres Webservers herunter. Dann können Sie Ihr Netzwerkkabel lösen, und die Seite neu laden, und alle Ressourcen sind offline verfügbar.

Fragen an Professor Markup

F: Muss ich meine HTML-Seiten in einem Cache-Manifest aufführen?

A: Ja und nein. Wenn Ihre vollständige Webanwendung aus einer einzigen Seite besteht, müssen Sie nur sicherstellen, dass diese Seite mit einem `manifest`-Attribut auf das Cache-Manifest verweist. Gehen Sie zu einer HTML-Seite mit einem `manifest`-Attribut, wird davon ausgegangen, dass sie selbst Teil der Webanwendung ist. Sie müssen sie selbst also nicht in die Manifest-Datei aufnehmen. Aber wenn Ihre Webanwendung umfasst, dann sollten Sie alle HTML-Seiten in die Manifest-Datei aufnehmen. Andernfalls weiß der Browser nicht, dass es noch andere HTML-Seiten gibt, die heruntergeladen und gecacht werden müssen.

Netzwerkabschnitte

Hier ist ein etwas komplizierteres Beispiel. Angenommen, Sie wollen Ihre Uhr-Anwendung Besucher nachhalten lassen, indem Sie ein *tracking.cgi*-Skript nutzen, das dynamisch über ein `<img src>`-Attribut geladen wird. Das Cachen dieser Ressource würde den Zweck des Nachhaltens vernichten. Diese Ressource sollte also nie gecacht werden und nie offline verfügbar sein. Das machen Sie folgendermaßen:

```
CACHE MANIFEST
NETWORK:
/tracking.cgi
CACHE:
/clock.css
```

```
/clock.js
/clock-face.jpg
```

Diese Cache-Manifest-Datei schließt *Abschnittsüberschriften* ein. Die mit `NETWORK:` markierte Zeile ist der Anfang der »Online Whitelist«. Ressourcen in diesem Abschnitt werden nie gecacht und sind offline nicht verfügbar. (Versuche, sie zu laden, während man offline ist, führen zu einem Fehler.) Die mit `CACHE:` markierte Zeile ist der Anfang des expliziten Abschnitts. Der Rest des Cache-Manifests ist mit dem des letzten Beispiels identisch. Alle drei aufgeführten Ressourcen werden gecacht und sind offline verfügbar.

Der fallback-Abschnitt

Es gibt noch eine weitere Art von Abschnitt in einem Cache-Manifest: einen *Ausweichabschnitt*. In einem Ausweichabschnitt können Sie Ersatzressourcen für Onlineressourcen definieren, die aus irgendeinem Grund nicht gecacht werden können oder nicht erfolgreich gecacht wurden. Die HTML5-Spezifikation zeigt dieses clevere Beispiel der Verwendung eines Ausweichabschnitts:

```
CACHE MANIFEST
FALLBACK:
/ /offline.html
NETWORK:
*
```

Was das bewirkt? Denken Sie zunächst an eine Site, die Millionen von Seiten enthält, wie beispielsweise Wikipedia. Sie können unmöglich und wollen wohl auch kaum die vollständige Site herunterladen. Aber angenommen, Sie könnten Teile der Seite offline verfügbar machen. Wie könnten Sie dann entscheiden, welche Seiten gecacht werden sollen? Wie wäre es damit: Alle Seiten, die Sie je auf einer offlinefähigen Wikipedia betrachtet haben, werden heruntergeladen und gecacht. Das würde alle Enzyklopädie-Einträge, die Sie besucht haben, alle Diskussionsseiten (auf denen man Spiegelgefechte über Enzyklopädie-Einträge führen kann) und auch alle Bearbeiten-Seiten (auf denen Sie tatsächlich Änderungen an einem bestimmten Eintrag vornehmen können) einschließen.

Das ist das, was dieses Cache-Manifest macht. Angenommen, alle HTML-Seiten auf Wikipedia (Eintrag, Diskussion, Bearbeiten, Versionsgeschichte) zeigen auf dieses Cache-Manifest. Wenn Sie eine Seite besuchen, die auf ein Cache-Manifest zeigt, sagt Ihr Browser: »Hey, diese Seite ist Teil einer Offline-Webanwendung. Ist das eine, die ich kenne?« Wenn Ihr Browser diese spezielle Cache-Manifest-Datei noch nie heruntergeladen hat, richtet er einen Offline-»Appcache« (kurz für »Anwendungscache«) ein, lädt alle im Manifest aufgeführten Seiten herunter und fügt dann die aktuelle Seite in den Appcache ein. Kennt der Browser dieses Cache-Manifest bereits, fügt er einfach die aktuelle Seite dem bestehenden Appcache hinzu. In beiden Fällen landet die gerade besuchte Seite im Appcache. Das ist wichtig. Es bedeutet, dass Sie eine Offline-Webanwendung haben können, der »nach Bedarf« Seiten hinzufügt werden, wenn Sie diese besuchen. Sie müssen nicht sämtliche HTML-Seiten Ihrer Anwendung in Ihrem Cache-Manifest anführen.

Schauen wir uns jetzt den Ausweichabschnitt an. Der Ausweichabschnitt in diesem Cache-Manifest enthält nur eine Zeile. Der erste Teil der Zeile (vor dem Leerzeichen) ist keine URL. Es ist ein URL-*Muster*. Der einzelne Slash (/) findet jede Seite auf Ihrer Site, nicht nur die Homepage. Wenn Sie versuchen, eine Seite zu besuchen, während Sie offline sind, sucht Ihr Browser im Appcache danach. Findet Ihr Browser dort eine Seite (weil Sie sie besucht haben, während Sie online waren und sie damals dem Appcache hinzugefügt wurde), zeigt er die gecachte Kopie an. Findet er sie nicht, zeigt er stattdessen eine Fehlermeldung an. In diesem Fall ist das die Seite */offline.html*, die in der zweiten Hälfte der Zeile des Ausweichabschnitts angegeben ist.

Schauen wir uns zum Abschluss den Netzwerkabschnitt an. Der Netzwerkabschnitt in diesem Cache-Manifest enthält lediglich eine einzige Zeile, die ebenfalls nur ein einziges Zeichen enthält (*). Dieses Zeichen hat im Netzwerkabschnitt eine besondere Bedeutung. Man bezeichnet es als »Online-Whitelist-Jokerzeichen«. Das ist ein etwas blumiger Ausdruck dafür, dass alles, was nicht im Appcache ist, von der ursprünglichen Webadresse heruntergeladen werden kann, wenn eine Internetverbindung besteht. Das ist für eine »offene« Offline-Webanwendung wichtig. Bleiben wir bei unserem Beispiel, heißt das, dass Ihr Browser beim aktiven Besuch dieser hypothetischen offlinefähigen Wikipedia alle Bilder, Videos und andere eingebettete Ressourcen ganz normal herunterlädt, selbst wenn sich diese in einer anderen Domain befinden. (Bei großen Websites ist das sehr verbreitet, selbst wenn sie nicht Teil einer Offline-Webanwendung sind: HTML-Seiten werden lokal generiert und ausgeliefert, während Bilder und Videos von einem CDN (Netzwerk zur Ressourcenauslieferung) in einer anderen Domain kommen.) Ohne dieses Jokerzeichen würde sich unsere hypothetische offlinefähige Wikipedia seltsam verhalten, wenn Sie online sind – genauer gesagt, sie würde keine extern gespeicherten Bilder oder Videos abrufen!

Ist dieses Beispiel vollständig? Nein. Wikipedia besteht nicht nur aus HTML-Dateien. Es nutzt CSS, JavaScript und Bilder auf jeder Seite. Jede dieser Ressourcen müsste explizit im `CACHE:`-Abschnitt der Manifest-Datei aufgeführt werden, damit Seiten offline ordentlich dargestellt werden und sich vernünftig verhalten. Aber der entscheidende Aspekt des Ausweichabschnitts ist, dass er Ihnen »offene« Offline-Webanwendung ermöglicht, die sich über die Seiten hinaus erstreckt, die in der Manifest-Datei aufgeführt werden.

Der Strom der Ereignisse

Bislang haben wir über Offline-Webanwendungen, das Cache-Manifest und den Offline-Anwendungscache (Appcache) nur mit ungenauen halbmagischen Begriffen gesprochen. Dinge werden heruntergeladen, Browser treffen Entscheidungen, und alles funktioniert einfach so. Aber das werden Sie so sicher kaum glauben, oder? Schließlich reden wir hier von der Webentwicklung. Nichts funktioniert einfach so.

Schauen wir uns zunächst den Fluss der Ereignisse an, genauer der DOM-Events. Besucht Ihr Browser das erste Mal eine Seite, die auf ein Cache-Manifest zeigt, löst er eine Reihe von Events auf dem `window.applicationCache`-Objekt aus, wie ich weiter unten beschrei-

ben werde. Ich weiß, dass das kompliziert klingt, aber glauben Sie mir, dass ist die einfachste Version, die mir eingefallen ist, die keine wichtige Information vernachlässigt. So sieht der Prozess aus:

1. Sobald der Browser ein `manifest`-Attribut auf dem `<html>`-Element entdeckt, löst er ein `checking`-Event aus. (Alle hier aufgeführten Events werden auf dem `window.applicationCache`-Objekt ausgelöst.) Das `checking`-Event wird immer ausgelöst, unabhängig davon, ob Sie die Seite oder eine andere Seite, die auf das gleiche Cache-Manifest verweist, bereits besucht haben.
2. Wenn Ihr Browser die Seite dieses Cache-Manifests noch nicht gesehen hat:
   - Er löst ein `downloading`-Event aus und beginnt dann, die Ressourcen herunterzuladen, die im Cache-Manifest aufgeführt werden.
   - Während er herunterlädt, löst der Browser periodisch `progress`-Events aus, die Informationen dazu enthalten, wie viele Dateien bereits heruntergeladen worden sind und wie viele Dateien noch zum Download anstehen.
   - Nachdem alle Ressourcen, die im Cache-Manifest aufgeführt werden, erfolgreich heruntergeladen worden sind, löst der Browser ein letztes Event aus: `cached`. Das ist das Signal dafür, dass die Offline-Webanwendung vollständig gecacht ist und offline genutzt werden kann. Das ist es. Sie sind fertig.
3. Haben Sie diese Seite oder eine andere Seite, die auf das gleiche Cache-Manifest zeigt, schon besucht, kennt der Browser das Cache-Manifest bereits und hat die entsprechenden Ressourcen eventuell schon im Appcache. Vielleicht befindet sich bereits die vollständige und funktionsfähige Webanwendung im Appcache. Für den Browser heißt das, dass er prüfen muss, ob sich das Cache-Manifest seit der letzten Prüfung geändert hat.
   - Ist die Antwort Nein, hat sich das Cache-Manifest noch nicht geändert. In diesem Fall löst der Browser sofort ein `noupdate`-Event aus. Und das war es dann. Sie sind fertig.
   - Ist die Antwort Ja, hat sich das Cache-Manifest geändert. In diesem Fall löst Ihr Browser ein `downloading`-Event aus und beginnt, jede einzelne Ressource herunterzuladen, die im Cache-Manifest aufgeführt wird.

   Während er die Ressourcen herunterlädt, löst der Browser periodisch `progress`-Events aus, die Informationen dazu enthalten, wie viele Dateien bereits heruntergeladen worden sind und wie viele Dateien noch zum Download anstehen.

   Sind erfolgreich alle Ressourcen heruntergeladen worden, die im Cache-Manifest aufgeführt sind, löst der Browser ein letzte Event aus: `updateready`. Das ist Ihr Signal, dass die neue Version der Offline-Webanwendung vollständig gecacht und offline einsatzbereit ist. Die neue Version ist noch nicht in Gebrauch! Um die neue Version im Betrieb zu aktivieren, ohne den Benutzer zu zwingen, die Seite neu zu laden, können Sie manuell die Funktion `window.applicationCache.swapCache()` aufrufen.

Geht bei diesem Vorgang irgendetwas schrecklich schief, löst Ihr Browser ein `error`-Event aus und bricht ab. Hier ist eine hoffnungslos verkürzte Liste der Dinge, die schieflaufen können:

- Das Cache-Manifest lieferte ein HTTP 404-Fehler (Seite nicht gefunden) oder einen 410-Fehler (für immer verschwunden) aus.
- Das Cache-Manifest wurde gefunden und hatte sich nicht geändert, aber die HTML-Seite, die auf das Manifest zeigte, wurde nicht korrekt heruntergeladen.
- Das Cache-Manifest wurde gefunden und hatte sich geändert, aber der Browser konnte die Ressourcen nicht herunterladen, die im Cache-Manifest aufgeführt wurden.

Die Kunst des Debuggens

Ich möchte hier auf zwei wichtige Dinge hinweisen. Das erste ist etwas, das Sie gerade gelesen, aber wahrscheinlich nicht verinnerlicht haben. Also ist es hier erneut: *Wenn auch nur eine einzige der Ressourcen, die im Cache-Manifest aufgeführt werden, nicht ordentlich heruntergeladen werden kann, schlägt der gesamte Prozess des Cachens der Offline-Webanwendung fehl*. Ihr Browser löst das `error`-Event aus, aber es gibt keinen Hinweis darauf, was das eigentliche Problem war. Das kann das Debuggen von Offline-Webanwendungen äußerst frustrierend machen.

Der zweite wichtige Punkt ist etwas, das technisch betrachtet, kein Fehler ist, aber wie ein ernsthafter Browser-Bug erscheinen kann, bis Sie realisieren, was geschieht. Das hat etwas damit zu tun, wie genau Ihr Browser prüft, ob sich die Cache-Manifest-Datei geändert hat. Das ist ein dreiphasiger Prozess – langweilig, aber wichtig. Passen Sie also auf. Hier ist der Vorgang:

1. Der Browser prüft gemäß den üblichen HTTP-Regeln, ob das Cache-Manifest verfallen ist. Genau wie jede andere Datei, die über HTTP geliefert wird, schließt Ihr Webserver üblicherweise Metainformationen zu der Datei in die HTTP-Response-Header ein. Einige dieser HTTP-Header (`Expires` und `Cache-Control`) sagen Ihrem Browser, ob es erlaubt ist, die Datei zu cachen, ohne den Server später zu fragen, ob sie sich geändert hat. Diese Art von Caching hat nichts mit Offline-Webanwendungen zu tun. Das passiert bei so gut wie jeder HTML-Seite, jedem Stylesheet, Skript, Bild und jeder anderen Ressource im Web.
2. Ist das Cache-Manifest (gemäß den HTTP-Headern) verfallen, fragt Ihr Browser den Server, ob es eine neue Version gibt, und lädt diese gegebenenfalls herunter. Dazu sendet der Browser einen HTTP-Request, der das letzte Veränderungsdatum der Cache-Manifest-Datei einschließt, das der Webserver in die HTTP-Response-Header einschloss, als der Browser die Datei das letzte Mal heruntergeladen hatte. Wenn der Webserver feststellt, dass sich die Manifest-Datei seit diesem Datum nicht geändert hat, sendet er einfach einen 304-Statuscode (Nicht verändert). Wieder ist auch das

nichts, was für Offline-Webanwendungen spezifisch ist. Es passiert bei so gut wie allen Ressourcen im Web.

3. Meint der Webserver, dass sich die Manifest-Datei seit jenem Datum geändert hat, liefert er den HTTP-Statuscode 200 (OK) und dann den Inhalt der neuen Datei mit neuen `Cache-Control`-Headern und einem neuen letzten Veränderungsdatum. Das sichert, dass die Schritte 1 und 2 beim nächsten Mal ordentlich funktionieren. (HTTP ist cool: Webserver sorgen immer für die Zukunft vor. Wenn Ihr Webserver Ihnen unbedingt eine Datei senden muss, tut er alles, um zu sichern, dass er sie nicht aus irgendeinem Grund zwei Mal senden muss.) Ist das neue Cache-Manifest einmal heruntergeladen, vergleicht Ihr Browser den Inhalt mit der Kopie, die Sie das letzte Mal heruntergeladen haben. Ist der Inhalt des neuen Cache-Manifests mit dem alten identisch, lädt Ihr Browser keine der Ressourcen herunter, die im Manifest aufgeführt werden.

Bei jedem dieser Schritte kann etwas schiefgehen, während Sie Offline-Webanwendungen entwickeln und testen. Nehmen wir beispielsweise an, Sie veröffentlichen eine Version Ihres Cache-Manifests und stellen zehn Minuten später fest, dass Sie noch eine weitere Ressource hinzufügen müssen. Kein Problem, oder? Sie fügen dem Manifest eine weitere Zeile hinzu und veröffentlichen es dann erneut. Tja. Dann passiert Folgendes: Sie laden die Seite neu, Ihr Browser bemerkt das `manifest`-Attribut, löst das `checking`-Event aus, und es passiert ... nichts. Ihr Browser beharrt stur darauf, dass das Cache-Manifest sich nicht geändert hat. Warum? Weil Ihr Webserver wahrscheinlich so konfiguriert ist, dass er Browsern sagt, dass er statische Dateien für ein paar Stunden cachen soll (gemäß den HTTP-Regeln mit `Cache-Control`-Headern). Das bedeutet, dass Ihr Browser nie über Schritt 1 des dreistufigen Prozesses hinauskommt. Klar, der Webserver weiß, dass sich die Datei geändert hat, aber Ihr Browser macht sich überhaupt nie die Mühe, den Webserver zu fragten. Warum? Weil der Webserver dem Browser beim letzten Herunterladen des Cache-Manifests (gemäß HTTP-Regeln mit `Cache-Control`-Headern) gesagt hatte, dass er die Ressource einige Stunden cachen soll. Und jetzt, nur zehn Minuten später, tut Ihr Browser genau das.

Um es noch einmal klar zu sagen: Das ist kein Fehler, sondern ein Feature. Alles funktioniert genau so, wie es funktionieren soll. Hätten Webserver keine Möglichkeit, Browsern (und zwischengeschalteten Proxys) zu sagen, dass sie Dinge cachen sollen, würde das Web über Nacht zusammenbrechen. Aber das ist kein Trost, wenn Sie ein paar Stunden damit verbracht haben, herauszufinden, warum Ihr Browser das aktualisierte Cache-Manifest nicht bemerkt hat. (Und noch besser: Wenn Sie lange genug warten, funktioniert die Sache auf mysteriöse Weise wieder! Weil der HTTP-Cache verfällt! Genau wie es sein soll!)

Hier ist eine Sache, die Sie auf alle Fälle tun sollten: Rekonfigurieren Sie Ihren Webserver so, dass Ihre Cache-Manifest-Datei nicht gemäß HTTP-Konventionen gecacht werden darf. Wenn Sie einen Apache-basierten Webserver haben, müssen Sie nur diese beiden Dateien in Ihre *.htaccess*-Datei einfügen:

```
ExpiresActive On
ExpiresDefault "access"
```

Das schaltet das Caching sogar für alle Dateien in diesem Verzeichnis und seinen Unterverzeichnissen ab. Bei einer laufenden Website werden Sie das wahrscheinlich nicht wollen und sollten diese Direktiven entweder mit einer `<Files>`-Direktive so einschränken, dass sie sich nur auf Ihre Cache-Manifest-Datei auswirken oder ein Unterverzeichnis erstellen, das nur diese *.htaccess*-Datei und Ihre Cache-Manifest-Datei enthält. Wie üblich sind derartige Konfigurationsdetails vom Webserver abhängig. Schlagen Sie gegebenenfalls also in der Konfiguration Ihres Webservers nach, um zu sehen, wie Sie die HTTP-Caching-Header steuern.

Die Deaktivierung des HTTP-Cachings für die Cache-Manifest-Datei allein löst noch nicht alle Probleme. Es passiert immer noch, dass Sie eine der Ressourcen für den Appcache geändert haben, ohne dass sich ihre Adresse auf dem Webserver geändert hat. Hier schlägt Schritt 2 des dreistufigen Prozesses fehl. Ihre Cache-Manifest-Datei hat sich nicht geändert, und der Browser wird nie bemerken, dass sich eine zuvor gecachte Ressource geändert hat. Betrachten Sie folgendes Beispiel:

```
CACHE MANIFEST
# rev 42
clock.js
clock.css
```

Wenn Sie *clock.css* ändern und erneut veröffentlichen, sehen Sie die Änderungen nicht, weil sich die Cache-Manifest-Datei selbst nicht geändert hat. Jedes Mal, wenn Sie eine Änderung an einer der Ressourcen in Ihrer Offline-Webanwendung vornehmen, sollten Sie also auch die Cache-Manifest-Datei selbst ändern. Dazu reicht es, ein einziges Zeichen zu verändern. Das einfachste Verfahren, das mir dafür eingefallen ist, ist, in die Manifest-Datei eine Kommentarzeile mit einer Versionsnummer einzuschließen. Jedes Mal, wenn Sie eine der Ressourcen ändern, ändern Sie die Versionsnummer in diesem Kommentar. Der Webserver liefert die geänderte Cache-Manifest-Datei, Ihr Browser bemerkt, dass sich der Inhalt der Datei geändert hat, und löst den Prozess des Herunterladens aller im Manifest aufgeführten Ressourcen aus:

```
CACHE MANIFEST
# rev 43
clock.js
clock.css
```

Bauen wir die Sache auf!

Erinnern Sie sich noch an das Halma-Spiel, das wir in Kapitel 4 eingeführt (siehe dazu den Abschnitt »Ein vollständiges Beispiel« auf Seite 78) und später verbessert haben, indem wir den Spielstand lokal gespeichert haben (siehe Abschnitt »HTML5 Storage im Einsatz« auf Seite 138)? Machen wir unser Halma-Spiel jetzt offline.

Dazu benötigen wir ein Manifest, das die Ressourcen aufführt, die das Spiel benötigt. Da ist die Haupt-HTML-Seite, eine JavaScript-Datei, die den gesamten Spielcode enthält – und das war es auch schon. Es gibt keine Bilder, weil alle Zeichnungen mit JavaScript über die Canvas-API durchgeführt werden (siehe Kapitel 4) und alle erforderlichen CSS-Styles

im <style>-Element zu Anfang der HTML-Seite stehen. Unser Cache-Manifest sieht also so aus:

```
CACHE MANIFEST
halma.html
../halma-localstorage.js
```

Ein Wort zu Pfaden. Ich habe ein *offline/*-Unterverzeichnis im *examples/*-Verzeichnis erstellt, und die Cache-Manifest-Datei befindet sich in diesem Unterverzeichnis. Weil die HTML-Seite eine kleine Ergänzung benötigt, damit Sie offline funktioniert (mehr dazu in einer Minute), habe ich eine separate Kopie der HTML-Datei erstellt, die sich ebenfalls im Unterverzeichnis *offline/* befindet. Aber weil es keine Änderungen im JavaScript-Code selbst gibt, seit wir ihn um die Unterstützung für lokale Speicherung erweitert haben (siehe dazu den Abschnitt »HTML5 Storage im Einsatz« auf Seite 138), nutze ich die gleiche *.js*-Datei, die sich im Elternverzeichnis (*examples/*) befindet. Insgesamt sehen die Dateien so aus:

```
/examples/localstorage-halma.html
/examples/halma-localstorage.js
/examples/offline/halma.manifest
/examples/offline/halma.html
```

In der Cache-Manifest-Datei (*/examples/offline/halma.manifest*) wollen wir zwei Dateien referenzieren. Die erste ist die Offlineversion der HTML-Datei (*/examples/offline/halma.html*), die in der Manifest-Datei ohne Präfix steht, weil sich beide Dateien im gleichen Verzeichnis befinden. Das zweite ist die JavaScript-Datei, die sich im Elternverzeichnis (*/examples/halma-localstorage.js*) befindet und in der Manifest-Datei mit einer relativen URL-Notation angegeben wird: *../halma-localstorage.js*. Genau so könnte eine relative URL in einem <img src>-Attribut aussehen. Wie Sie im nächsten Beispiel sehen werden, können Sie auch absolute Pfade (die von der Wurzel der aktuellen Domain ausgehen) oder sogar absolute URLs (die auf Ressourcen in anderen Domains verweisen) nutzen.

Jetzt müssen wir in die HTML-Datei nur noch das manifest-Attribut einfügen, das auf die Cache-Manifest-Datei zeigt:

```
<!DOCTYPE html>
<html lang="en" manifest="halma.manifest">
```

Und das war es! Wenn ein offlinefähiger Browser die offlinefähige HTML-Seite zum ersten Mal lädt, lädt er die Cache-Manifest-Datei herunter, auf die verwiesen wird, beginnt dann, alle referenzierten Ressourcen herunterzuladen, und speichert sie im Offline-Appcache. Von da ab übernimmt der Offline-Anwendungsalgorithmus, wenn Sie die Seite besuchen. Sie können das Spiel offline spielen, und da es den Spielstand lokal speichert, können Sie die Seite verlassen und, so oft Sie wollen, zu ihr zurückkehren.

Weitere Lektüre

Standards:

- Offline-Webanwendungen in der HTML5-Spezifikation (*http://bit.ly/cCkWZa*)

Browserhersteller-Dokumentation:

- »Offline resources in Firefox« (*https://developer.mozilla.org/En/Offline_resources_in_Firefox*), im Mozilla Developer Center
- »HTML5 Offline Application Cache« (*http://developer.apple.com/safari/library/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/OfflineApplicationCache/OfflineApplicationCache.html*), ein Teil des »Safari Client-Side Storage and Offline Applications Programming Guide« (*http://developer.apple.com/safari/library/documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Introduction/Introduction.html*)

Einführungen und Demos:

- »Gmail for Mobile HTML5 Series: Using Appcache to Launch Offline – part 1« (*http://googlecode.blogspot.com/2009/04/gmail-for-mobile-html5-series-using.html*), von Andrew Grieve
- »Gmail for Mobile HTML5 Series: Using Appcache to Launch Offline – part 2« (*http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-2.html*), von Andrew Grieve
- »Gmail for Mobile HTML5 Series: Using Appcache to Launch Offline – part 3« (*http://googlecode.blogspot.com/2009/05/gmail-for-mobile-html5-series-part-3.html*), von Andrew Grieve
- »Debugging HTML 5 Offline Application Cache« (*http://jonathanstark.com/blog/2009/09/27/debugging-html-5-offline-application-cache/*), von Jonathan Stark
- »An HTML5 offline image editor and uploader application« (*http://hacks.mozilla.org/2010/02/an-html5-offline-image-editor-and-uploader-application/*), von Paul Rouget

KAPITEL 9

Formularwahn

Einstieg

Jeder kennt Webformulare, nicht wahr? Man schlägt ein `<form>`-Element auf, gibt ein paar `<input type="text">`-Elemente hinzu, rührt vielleicht noch ein kleines `<input type="password">` ein und rundet das Ganze mit einem `<input type="submit">`-Button ab. Fertig ist der Salat.

Sie kennen noch nicht einmal die Hälfte. HTML5 definiert mehr als ein Dutzend neuer Eingabetypen, die Sie in Ihren Formularen nutzen können. Und wenn ich »nutzen« sage, meine ich, dass Sie sie hier und jetzt nutzen können, ohne irgendwelche Tricks, Hacks oder Workarounds. Was aber nun wieder nicht heißt, dass Sie gleich aus dem Häuschen sein dürfen: Ich will damit nicht sagen, dass alle diese neuen Funktionen tatsächlich in jedem Browser unterstützt werden. Nein, das will ich damit keineswegs sagen. In modernen Browsern, ja, da können Ihre Formulare richtig loslegen. In älteren Browsern funktionieren sie noch, verlieren aber etwas an Pepp. Das bedeutet, dass ältere Browsern bei allen diesen Funktionen verlässliche Ausweichmöglichkeiten haben. Sogar der IE6.

Platzhaltertext

Die erste Verbesserung, die HTML5 Webformularen bringt, ist die Möglichkeit, einen Platzhaltertext für ein Eingabefeld zu setzen. Der Platzhaltertext wird im Formular angezeigt, solange das Feld leer und noch nicht fokussiert ist. Sobald Sie in das Feld hineinklicken (oder per Tabulator hineinspringen), verschwindet der Platzhaltertext.

Wahrscheinlich ist Ihnen Platzhaltertext bereits begegnet. Beispielsweise zeigt Mozilla Firefox 3.5 jetzt Platzhaltertext in der Adressleiste an, wie Sie in Abbildung 9-1 sehen.

Abbildung 9-1: Platzhaltertext im Firefox-Suchfeld

Klicken Sie auf (oder springen Sie per Tabulator in) die Adressleiste, verschwindet der Platzhaltertext (Abbildung 9-2).

Abbildung 9-2: Der Platzhaltertext verschwindet bei Aktivierung des Felds

Ironischerweise unterstützt Firefox 3.5 keinen Platzhaltertext in Webformularen. C'est la vie. Die Browserunterstützung für Platzhalter sehen Sie in Tabelle 9-1.

Tabelle 9-1: Platzhalterunterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
•	3.7+	4.0+	4.0+	•	•	•

So können Sie Platzhaltertext in Ihre Webformulare einbauen:

```
<form>
  <input name="q" placeholder="Lesezeichen und Verlauf durchsuchen">
  <input type="submit" value="Search">
</form>
```

Browser, die das `placeholder`-Attribut nicht unterstützen, ignorieren es einfach. Kein Schaden, kein Foul.

Fragen an Professor Markup

F: Kann ich im `placeholder`*-Attribut HTML-Markup nutzen? Ich möchte ein Bild einfügen oder vielleicht die Farbe ändern.*

A: Das `placeholder`-Attribut kann nur Text enthalten, kein HTML-Markup. Aber es gibt einige herstellerspezifische CSS-Erweiterungen (*http://trac.webkit.org/export/37527/trunk/LayoutTests/fast/forms/placeholder-pseudo-style.html*), die Ihnen gestatten, den Platzhaltertext in einigen Browsern zu stylen.

Autofokusfelder

Viele Websites nutzen JavaScript, um dem ersten Eingabefeld automatisch den Fokus eines Webformulars zu geben. Beispielsweise erhält auf der Google.de-Homepage automatisch das Suchfeld den Fokus, damit Sie Suchbegriffe eingeben können, ohne dass Sie zuvor manuell den Cursor in das Suchfeld bewegen müssen. Für die meisten Anwender ist das bequem, für Poweruser oder Menschen mit speziellen Bedürfnissen kann es nervig sein. Wenn Sie in Erwartung dessen, dass die Seite nach unten gescrollt wird, die Leertaste drücken, passiert nichts, weil das Suchfeld bereits den Fokus hat. Stattdessen geben Sie lediglich ein Leerzeichen in das Suchfeld ein. Setzen Sie manuell den Fokus in ein anderes Eingabefeld, während die Seite noch geladen wird, kann das »hilfsbereite« Autofokus-

skript den Fokus wieder in das ursprüngliche Eingabefeld zurückbefördern, wenn die Seite fertig geladen ist, und damit Ihren Arbeitsablauf unterbrechen und bewirken, dass Sie Ihre Eingaben im falschen Feld landen.

Weil die automatische Verleihung des Fokus mit JavaScript durchgeführt wird, kann es kompliziert sein, all diese Randfälle angemessen zu berücksichtigen. Es wird wenig Rücksicht auf Menschen genommen, denen es nicht passt, dass eine Webseite den Fokus »kapert«.

Zur Lösung dieses Problems führt HTML5 ein `autofocus`-Attribut auf allen Steuerelementen für Webformulare ein. Das `autofocus`-Attribut macht genau, was der Name verspricht: Sobald die Seite geladen wird, verschiebt es den Eingabefokus auf ein bestimmtes Eingabefeld. Aber weil das jetzt nur noch Markup ist, ist das Verhalten bei allen Websites gleich. Außerdem können Browserhersteller (oder Erweiterungsautoren) Benutzern eine Möglichkeit bieten, das Autofokusverhalten abzuschalten. Tabelle 9-2 zeigt, welche Browser Autofokus unterstützen.

Tabelle 9-2: Autofokusunterstützung

IE	Firefox	Safari	Chrome	Opera	iPhone	Android
•	•	4.0+	3.0+	10.0+	✓	✓

Folgendermaßen können Sie einem Formularfeld automatisch den Fokus geben:

```
<form>
  <input name="q" autofocus>
  <input type="submit" value="Suche">
</form>
```

Browser, die das `autofocus`-Attribut nicht unterstützen, ignorieren es einfach.

Wie bitte? Sie sagen, dass Ihre Autofokusfelder gefälligst in allen Browsern funktionieren sollen, nicht nur in diesen Schnickschnack-HTML5-Browsern? Sie können Ihr aktuelles Autofokusskript behalten. Nehmen Sie einfach zwei kleine Änderungen vor:

1. Fügen Sie Ihrem HTML-Markup das `autofocus`-Attribut hinzu.
2. Prüfen Sie, ob der Browser das `autofocus`-Attribut unterstützt (siehe dazu den Abschnitt »Formular-Autofokus« auf Seite 30), und führen Sie Ihr eigenes Autofokusskript aus, wenn der Browser keine native Unterstützung für Autofokus bietet:

```
<form name="f">
  <input id="q" autofocus>
  <script>
    if (!("autofocus" in document.createElement("input"))) {
      document.getElementById("q").focus();
    }
  </script>
  <input type="submit" value="Go">
</form>
...
```

Gehen Sie zu *http://diveintohtml5.org/examples/input-autofocus-with-fallback.html*, wenn Sie sich ein Beispiel für `autofokus` mit Ausweichmöglichkeit ansehen wollen.

Professor Markup sagt

Viele Webseiten warten mit dem Setzen des Fokus, bis `window.onload` ausgelöst wird. Aber das `window.onload`-Event wird erst ausgelöst, wenn alle Bilder geladen sind. Enthalten Ihre Seiten viele Bilder, könnte ein derart naives Skript dem Feld den Fokus geben, nachdem der Benutzer bereits begonnen hat, mit einem anderen Teil Ihrer Seite zu interagieren. (Deswegen hassen Poweruser Autofokusskripten.) Das vorangehende Beispiel platziert das Autofokusskript unmittelbar hinter dem Formularfeld, das es referenziert, aber Ihr Backend-System ist eventuell nicht so flexibel. Wenn Sie ein Skript nicht in die Mitte Ihrer Seite einfügen können, sollten Sie den Fokus während eines selbst definierten Events wie jQuerys `$(document).ready()` statt `window.onload` starten.

E-Mail-Adressen

Seit mehr als einem Jahrzehnt bestehen Webformulare aus nur einigen wenigen Feldern – die gebräuchlichsten davon sind in Tabelle 9-3 aufgeführt.

Tabelle 9-3: Eingabetypen in HTML 4

Feldtyp	HTML-Code	Anmerkungen
Checkbox	`<input type="checkbox">`	Kann angewählt oder abgewählt werden.
Radio-Button	`<input type="radio">`	Kann mit anderen Eingabeelementen gruppiert werden.
Passwortfeld	`<input type="password">`	Gibt bei der Eingabe Punkte statt der vom Benutzer eingegebenen Zeichen aus.
Aufklapplisten	`<select><option>...`	
Dateiwähler	`<input type="file">`	Öffnet einen *Datei öffnen*-Dialog.
Absenden-Button	`<input type="submit">`	
Klartext	`<input type="text">`	Das `type`-Attribut kann weggelassen werden.

Alle diese Eingabetypen funktionieren auch in HTML5 noch. Wenn Sie auf »HTML5 aufrüsten« (vielleicht indem Sie Ihren Doctype ändern, siehe dazu den Abschnitt »Die Doctype-Deklaration« auf Seite 33), müssen Sie keine einzige Änderung an Ihren Webformularen vornehmen. Ein Hoch auf die Rückwärtskompatibilität!

Aber HTML5 definiert mehrere neue Feldtypen, und aus Gründen, die gleich klar werden, spricht absolut nichts dagegen, sie sofort einzusetzen.

Der erste dieser neuen Eingabetypen dient E-Mail-Adressen. Er sieht so aus:

```
<form>
  <input type="email">
  <input type="submit" value="Go">
</form>
```

Ich wollte gerade einen Satz schreiben, der mit folgender Wendung beginnt: »In Browsern, die `type="email"` nicht unterstützen ...« Aber dann habe ich mich besonnen. Warum? Weil mir nicht klar ist, was es heißen soll, wenn man sagt, dass ein Browser `type="email"` nicht unterstützt. Alle Browser »unterstützen« `type="email"`. Vielleicht machen sie nichts Besonderes damit (Sie werden gleich einige Beispiele für eine spezielle Behandlung sehen), aber Browser, die `type="email"` nicht verstehen, werden es einfach als `type="text"` behandeln und als gewöhnliches Textfeld darstellen.

Ich kann nicht nachdrücklich genug betonen, wie wichtig das ist. Das Web bietet Millionen von Formularen, in denen Sie aufgefordert werden, Ihre E-Mail-Adresse einzugeben, die alle `<input type="text">` nutzen. Sie sehen ein Textfeld, geben Ihre E-Mail-Adresse darin es ein und das war es. Dann tritt HTML5 auf die Bühne und definiert `type="email"`. Und? Fahren Browser jetzt aus der Haut? Nein. Alle Browser behandeln unbekannte `type`-Attribute als `type="text"` – sogar der IE6. Sie können Ihre Webformulare also jetzt sofort auf `type="email"` »aufrüsten«.

Was würde es heißen, wenn man sagte, dass ein Browser `type="email"` unterstützt? Es könnte verschiedene Dinge bedeuten. Die HTML5-Spezifikation verlangt keine bestimmte Benutzerschnittstelle für die neuen Eingabetypen. Opera versieht das Formularfeld mit einem kleinen E-Mail-Symbol. Andere HTML5-Browser wie Safari und Chrome stellen es als Textfeld dar – genau wie `type="text"` –, Ihre Benutzer sehen den Unterschied also gar nicht (es sei denn, sie schauen sich den Quelltext der Seite an).

Und dann ist da noch das iPhone.

Das iPhone besitzt kein Hardwaretastatur. Alles »Tippen« erfolgt, indem man auf eine Tastatur auf dem Bildschirm tippt, die zu geeigneter Zeit aufspringt – beispielsweise wenn Sie den Fokus in ein Formularfeld auf einer Webseite setzen. Apple hat im Webbrowser des iPhones etwas sehr Schlaues gemacht: Er erkennt mehrere der neuen HTML5-Eingabetypen und ändert dynamisch die Tastatur auf dem Bildschirm, um sie für die erforderlichen Eingaben zu optimieren.

E-Mail-Adressen beispielsweise sind Text, stimmt's? Allerdings eine bestimmte Art von Text. Das heißt: Fast alle E-Mail-Adressen enthalten das `@`-Zeichen und mindestens einen Punkt (`.`), hingegen nie Leerzeichen. Setzt ein iPhone-Nutzer den Fokus in ein `<input type="email">`-Element, erhält er eine Bildschirmtastatur, in der die Leertaste kleiner als üblich ist, die zusätzlich aber spezielle Tasten für die Zeichen `@` und `.` bietet, wie Sie in Abbildung 9-3 sehen.

Zusammenfassung: Es hat keine Nachteile, wenn Sie sämtliche Formularfelder für E-Mail-Adressen in `type="email"` umwandeln. Es wird so gut wie keiner bemerken, höchstens iPhone-Nutzer, denen es vermutlich aber auch nicht auffällt. Aber die, denen es auffällt, werden leise schmunzeln und Ihnen dafür danken, dass Sie Ihnen den Ausflug ins Web noch etwas angenehmer gestaltet haben.

Webadressen

Webadressen – die viele auch unter dem Namen URL kennen, einige wenige Pedanten bezeichnen sie als URIs – sind eine andere Art von spezialisiertem Text. Die Syntax einer Webadresse wird durch die entsprechenden Internetstandards eingeschränkt. Fordert Sie jemand auf, eine Webadresse in ein Formular einzugeben, wird er etwas wie `http://www.google.de` und nicht »Weitweitwegweg 125« erwarten. Schrägstriche und Punkte tauchen häufig auf, aber Leerzeichen sind verboten. Und jede Webadresse hat ein Domainsuffix wie *.com* oder *.org*.

Aufgepasst ... ein Tusch bitte ... `<input type="url">`. Auf dem iPhone sieht das aus, wie in Abbildung 9-4 gezeigt.

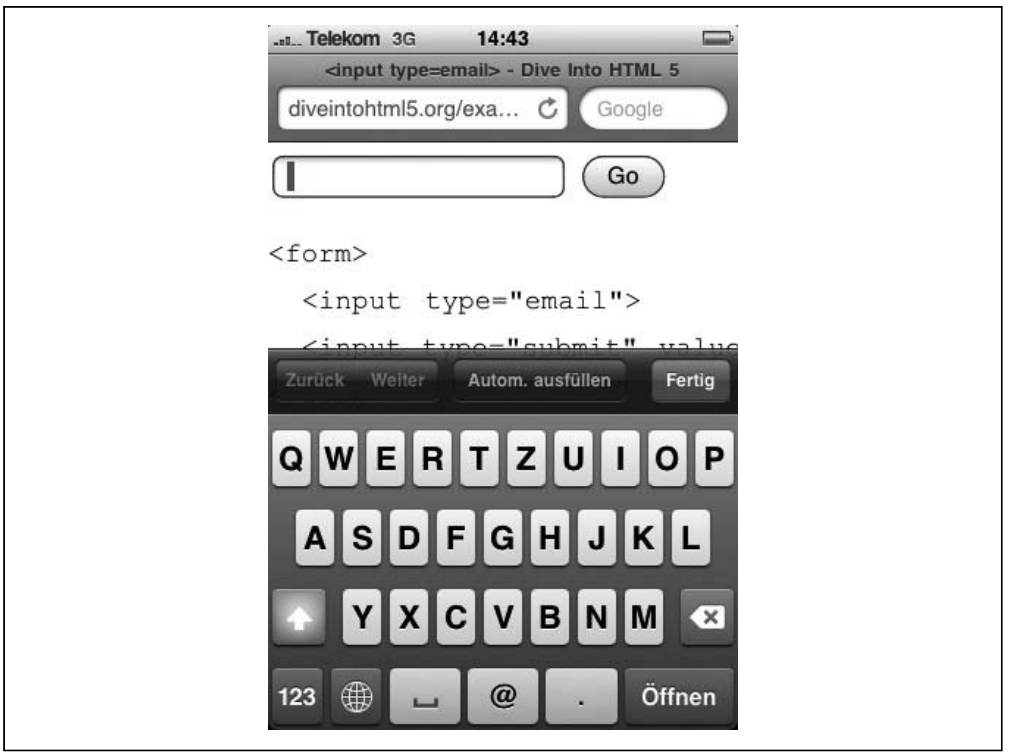


Abbildung 9-3: Zur Eingabe einer E-Mail-Adresse optimierte Tastatur

Abbildung 9-4: Zur Eingabe einer URL optimierte Tastatur

Genau wie bei Feldern für E-Mail-Adressen bietet das iPhone eine spezielle virtuelle Tastatur, die für Webadressen optimiert ist. Die Leerzeichentaste wird vollständig durch drei virtuelle Tasten ersetzt: einen Punkt, einen Schrägstrich und einen *.com*-Button. Drücken Sie länger auf den *.com*-Button, können Sie andere verbreitete Suffixe wie *.org* oder *.net* wählen.

Browser, die HTML5 nicht unterstützen, behandeln `type="url"` genau wie `type="text"`. Auch hier gibt es also nichts, was einer sofortigen Verwendung dieses Elements für die Eingabe von Webadressen im Wege steht.

Zahlen als Spinboxen

Der Nächste bitte: Zahlen. Nach einer Zahl zu fragen, ist in vielerlei Hinsicht verzwickter, als nach einer E-Mail- oder Webadresse zu fragen. Zunächst einmal sind Zahlen erheblich komplizierter, als es auf den ersten Blick den Anschein hat. Schnell. Denken Sie sich eine Zahl aus. –1? Nein. Ich meinte eine Zahl zwischen 1 und 10. 7½? Bitte, doch kein Bruch. π? Das ist aber wirklich nicht mehr rational!

Was ich damit sagen will, ist, dass Sie nur selten nach »irgendeiner Zahl« fragen. Wahrscheinlicher ist es, dass Sie nach einer Zahl in einem bestimmten Bereich fragen und nur bestimmte Arten von Zahlen in diesem Bereich haben wollen – beispielsweise nur ganze Zahlen, aber kein Brüche oder Dezimalzahlen, oder gar etwas Exotisches wie nur Zahlen,

die durch 10 teilbar sind. HTML5 bietet in dieser Hinsicht alles, was Sie sich nur wünschen können. Schauen wir uns ein Beispiel an:

```
<input type="number"
       min="0"
       max="10"
       step="2"
       value="6">
```

Nehmen wir es uns attributweise vor (Sie können das an einem Onlinebeispiel nachverfolgen, wenn Sie wollen):

- `type="number"` heißt, dass das ein Zahlenfeld ist.
- `min="0"` gibt den Minimumwert für dieses Feld an.
- `max="10"` gibt den Maximumwert an.
- `step="2"` definiert in Kombination mit `min` die zulässigen Werte in diesem Bereich: `0`, `2`, `4` und so weiter bis `max`.
- `value="6"` ist der Anfangswert. Das sollte Ihnen vertraut sein. Genau diesen Attributnamen haben Sie schon immer verwendet, um den Wert von Formularfeldern anzugeben. (Ich erwähne das hier, um noch einmal zu belegen, dass HTML5 auf den vorangegangenen Versionen von HTML aufbaut. Sie müssen nicht neu lernen, wie Sie etwas machen, das Sie schon seit Ewigkeiten tun.)

Das ist die Markup-Seite von Zahlenfeldern. Beachten Sie, dass alle diese Attribute optional sind. Wenn Sie ein Minimum, aber kein Maximum haben, können Sie ein `min`-Attribut ohne ein `max`-Attribut angeben. Der Standardschrittwert ist 1. Sie können das Attribut `step` also weglassen, es sei denn, Sie benötigen einen anderen Schrittwert. Es gibt keinen Standardwert für `value`. Dieses Attribut kann ein leerer String sein oder ganz weggelassen werden.

HTML5 macht hier nicht Halt. Für den gleichen äußerst geringen Preis erhalten Sie auch noch diese praktischen JavaScript-Methoden:

`input.stepUp(`*n*`)`
: Erhöht den Wert des Felds um *n*.

`input.stepDown(`*n*`)`
: Vermindert den Wert des Felds um *n*.

`input.valueAsNumber`
: Liefert den aktuellen Wert als Gleitkommazahl (die Eigenschaft `input.value` ist immer ein String).

Sie haben Probleme, sich das vorzustellen? Wie die Schnittstelle eines Zahlenfelds aussieht, ist Ihrem Browser überlassen, und die verschiedenen Browserhersteller haben die Unterstützung dafür auf unterschiedliche Weise implementiert. Auf dem iPhone, auf dem die Eingabe ohnehin nicht so einfach ist, optimiert der Browser auch hier die virtuelle Tastatur für die Eingabe von Zahlen, wie Sie in Abbildung 9-5 sehen.

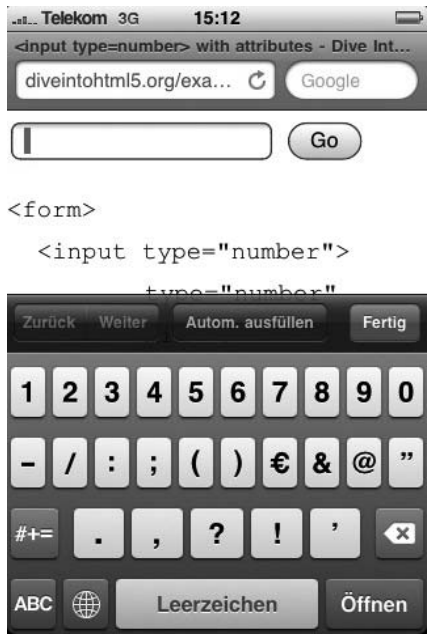


Abbildung 9-5: Für die Eingabe von Zahlen optimierte Tastatur

In der Desktop-Version von Opera wird das gleiche `type="number"`-Feld als »Spinbox«-Steuerelement mit kleinen Pfeilen nach oben und unten dargestellt, auf die Sie klicken können, um den Wert zu ändern (Abbildung 9-6).

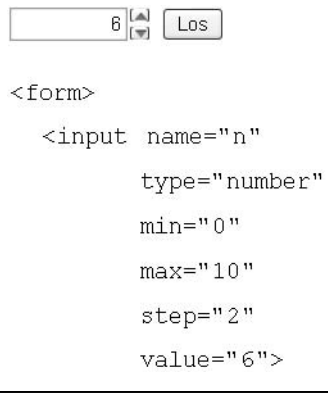


Abbildung 9-6: Eine Spinbox

Opera berücksichtigt `min`, `max` und `step`. Sie können also nur vorgesehene numerische Werte erreichen. Setzen Sie den Wert auf das Maximum oder Minimum, wird der entsprechende Pfeil in der Spinbox deaktiviert (Abbildung 9-7).

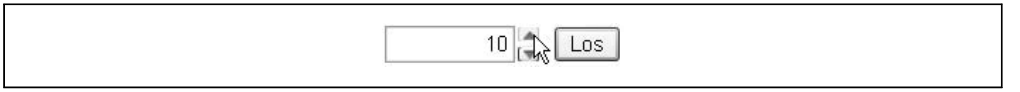


Abbildung 9-7: Eine Spinbox bei ihrem Maximumwert

Wie bei allen anderen Eingabetypen, die ich in diesem Kapitel betrachtet habe, behandeln Browser, die `type="number"` nicht unterstützen, ein entsprechendes Element als `type="text"`. Der Wert von `value` erscheint im Feld (da jenes Attribut bereits bekannt ist), aber andere Attribute wie `min` und `max` werden ignoriert. Sie können sie natürlich selbst implementieren oder eins der vielen JavaScript-Frameworks nutzen, die bereits Spinbox-Steuerelemente implementiert haben. Aber erst sollten Sie folgendermaßen die native HTML5-Unterstützung prüfen (siehe dazu den Abschnitt »input-Typen« auf Seite 28):

```
if (!Modernizr.inputtypes.number) {
  // Keine native Unterstützung für type="number"-Felder!
  // Vielleicht nehmen Sie Dojo oder ein anderes JavaScript-Framework.
}
```

Zahlen als Schieberegler

Spinboxen, die wir uns im letzten Abschnitt angesehen haben, sind nicht die einzige Möglichkeit zur Darstellung numerischer Eingaben. Wahrscheinlich sind Ihnen bereits »Schieberegler«-Steuerelemente begegnet, wie sie in Abbildung 9-8 gezeigt werden.

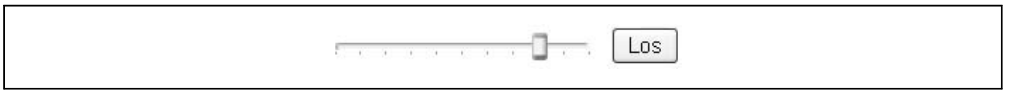


Abbildung 9-8: Ein Schieberegler

Auch in Ihre Webformulare können Sie jetzt Schieberegler integrieren. Das Markup hat erstaunliche Ähnlichkeit mit dem für Spinboxen:

```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6">
```

Die verfügbaren Attribute entsprechen denen für `type="number"` – `min`, `max`, `step`, `value` – und bedeuten auch das Gleiche. Der einzige Unterschied besteht in der Benutzerschnittstelle. Statt als Feld, in das man tippen kann, sollen Browser `type="range"` als Schieberegler darstellen. Aktuell machen das die jüngsten Versionen von Safari, Chrome und Opera. (Bedauerlicherweise stellt das iPhone es als einfaches Textfeld dar und bietet noch nicht einmal die für die numerische Eingabe optimierte virtuelle Tastatur an.) Alle anderen Browser behandeln das Feld einfach als `type="text"`. Es gibt also auch hier keinen Grund, der gegen eine sofortige Verwendung von `type="range"` spricht.

Datumswähler

HTML 4 bot kein Steuerelement zum Auswählen eines Datums. Verschiedene JavaScript-Frameworks stießen in diese Lücke – beispielsweise Dojo, jQuery UI, YUI und die Closure Library –, aber natürlich verlangen alle diese Lösungen, dass man sich für das Framework entscheidet, in das der Datumswähler eingebaut ist.

HTML5 definiert endlich eine Möglichkeit, ein natives Datumswähler-Steuerelement zu verwenden, ohne dass man es selbst per Skript schaffen muss. Genauer gesagt, definiert es gleich sechs: Datum, Monat, Woche, Uhrzeit, Datum und Uhrzeit sowie Datum und Uhrzeit plus Zeitzone.

Wie Tabelle 9-4 zeigt, ist die Unterstützung aktuell noch mager.

Tabelle 9-4: Die Datumswähler-Unterstützung

Eingabetyp	Opera	Alle anderen Browser
`type="date"`	9.0+	•
`type="month"`	9.0+	•
`type="week"`	9.0+	•
`type="time"`	9.0+	•
`type="datetime"`	9.0+	•
`type="datetime-local"`	9.0+	•

Abbildung 9-9 zeigt, wie Opera ein `<input type="date">` darstellt.

Abbildung 9-9: Ein Datumswähler

Wenn Sie zusätzlich zum Datum auch eine Uhrzeit benötigen, unterstützt Opera auch `<input type="datetime">`, wie Sie in Abbildung 9-10 sehen.

Abbildung 9-10: Ein Datums- und Uhrzeitwähler

Brauchen Sie nur Monat und Jahr (vielleicht für das Verfallsdatum einer Kreditkarte), kann Opera, wie Sie in Abbildung 9-11 sehen, `<input type="month">` anbieten.

Abbildung 9-11: Ein Monatswähler

Weniger verbreitet, aber auch verfügbar, ist die Möglichkeit, mit `<input type="week">` eine bestimmte Woche des Jahres zu wählen; siehe Abbildung 9-12.

Abbildung 9-12: Ein Wochenwähler

Und abschließend können Sie mit `<input type="time">` auch eine Zeit wählen, wie Sie in Abbildung 9-13 sehen.

Abbildung 9-13: Ein Zeitwähler

Es ist wahrscheinlich, dass irgendwann auch andere Browser diese Eingabeelemente unterstützen werden. Bis dahin werden Formularfelder wie `type="email"` (siehe dazu den Abschnitt »E-Mail-Adressen« auf Seite 160) und die anderen Eingabeelemente als einfache Textfelder dargestellt, falls ein Browser `type="date"` oder eine der Varianten nicht unterstützt. Wenn Sie wollen, können Sie also `<input type="date">` und Kollegen einfach nutzen, Opera-Nutzer glücklich machen und warten, bis die anderen Browser aufschließen. Alternativ können Sie `<input type="date">` nutzen, prüfen, ob der Browser native Unterstützung für Datumswähler bietet (siehe dazu den Abschnitt »input-Typen« auf Seite 28), und gegebenenfalls auf eine skriptbasierte Lösung ausweichen:

```
<form>
  <input type="date">
</form>
...
<script>
  var i = document.createElement("input");
  i.setAttribute("type", "date");
  if (i.type == "text") {
    // Keine native Datumswählerunterstützung :(
    // Dojo/jQueryUI/YUI/Closure oder eine andere Lösung einsetzen,
    // um einen zu erstellen, und dann dynamisch das <input>-Element ersetzen.
  }
</script>
```

Suchfelder

Gut, aber jetzt wird es komplizierter. Die Idee selbst ist eigentlich einfach, aber die Implementierungen könnten einige Erklärungen erfordern. Auf geht's ...

Suchen. Nicht nur die Google-Suche oder die Yahoo!-Suche. (Ja, die auch.) Denken Sie an Suchfelder auf beliebigen Seiten, auf beliebigen Sites. Amazon hat ein Suchfeld. Newegg hat ein Suchfeld. Die meisten Blogs haben ein Suchfeld. Und mit was für Markup werden die aufgebaut? Mit einem `<input type="text">` wie jedes andere Textfeld im Web auch. Mit HTML5 können wir das reparieren:

```
<form>
  <input name="q" type="search">
  <input type="submit" value="Suchen">
</form>
```

In einigen Browsern werden Sie keinen Unterschied zu einem gewöhnlichen Textfeld feststellen, aber wenn Sie Safari unter Mac OS X nutzen, sieht das wie in Abbildung 9-14.

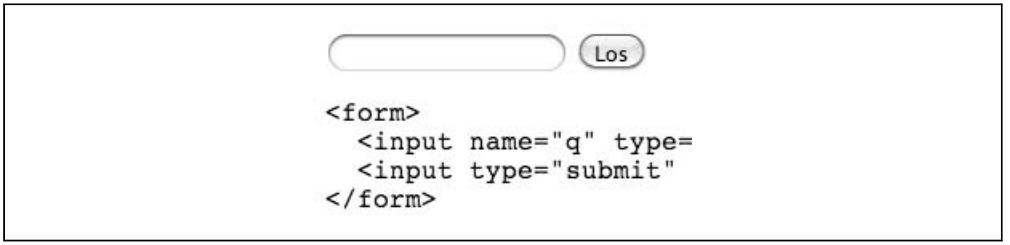


Abbildung 9-14: Ein Suchfeld

Sehen Sie den Unterschied? Das Eingabefeld hat abgerundete Ecken! Ich weiß: Sie können Ihre Aufregung kaum zügeln. Aber Moment, es kommt noch etwas! Wenn Sie beginnen, im `type="search"`-Feld etwas einzutippen, fügt Safari ganz rechts im Feld ein kleines »x« ein. Klicken Sie auf das »x«, wird das Feld geleert. (Google Chrome, der unter der Motorhaube große Teile der Technologie mit Safari teilt, zeigt das gleiche Verhalten.) Diese beiden kleinen Änderungen sollen das Erscheinungsbild nativer Suchfelder in iTunes und anderen Mac OS X-Anwendungen widerspiegeln (Abbildung 9-15).

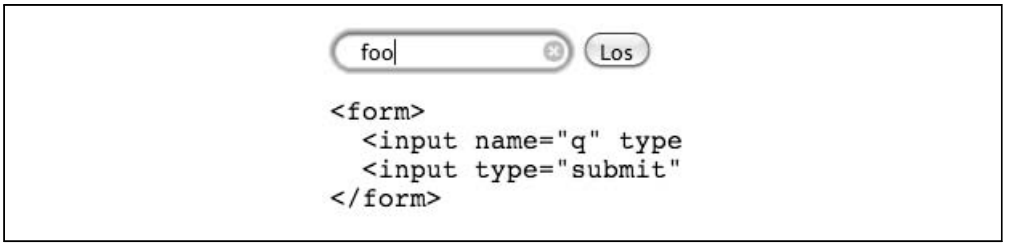


Abbildung 9-15: Ein Suchfeld mit Fokus

Die Apple-Website nutzt `<input type="search">` für alle Site-spezifischen Suchfelder, um der Site einen »Mac-artigen« Anstrich zu geben. Aber eigentlich hat das Ganze nichts Mac-Spezifisches an sich. Es ist einfach nur Markup. Jeder Browser auf jeder Plattform kann Suchfelder also gemäß den plattformspezifischen Konventionen darstellen. Wie bei allen neuen Eingabetypen auch, behandeln Browser, die `type="search"` nicht kennen, dieses Element als `type="text"`. Und auch hier heißt das, dass nichts gegen eine sofortige Verwendung von `type="search"` für alle Suchfelder spricht.

Professor Markup sagt

Eigentlich gibt es einen Grund, der gegen die Verwendung von `<input type="search">` sprechen könnte. Safari wendet auf Suchfelder nicht die üblichen CSS-Styles an. (Und mit »üblichen Styles« meine ich grundlegende Dinge, wie Rahmen, Hintergrundfarbe, Padding und so weiter.) Aber vielleicht können Sie sich ja mit den abgerundeten Ecken trösten!

Farbwähler

HTML5 definiert auch `<input type="color">`, über das Sie eine Farbe wählen können, deren hexadezimale Darstellung dann zurückgeliefert wird. Noch wird das von keinem Browser unterstützt. Wirklich schade. Der Mac OS-Farbwähler hat mir schon immer gefallen. Vielleicht irgendwann ...

Eine Sache noch ...

In diesem Kapitel habe ich von neuen Eingabetypen und neuen Funktionen wie dem Autofokus für Formularfelder gesprochen, aber ich habe noch nichts über den vielleicht aufregendsten Teil von HTML5-Formularen gesagt: automatische Eingabevalidierung. Betrachten Sie das verbreitete Problem der Eingabe von E-Mail-Adressen in Webformularen. Wahrscheinlich haben Sie eine clientseitige Validierung in JavaScript, auf die eine serverseitige Validierung in PHP, Python oder einer anderen serverseitigen Skriptsprache folgt. Es gibt zwei Probleme mit der Validierung von E-Mail-Adressen in JavaScript:

- Eine überraschend große Menge Ihrer Besucher (wahrscheinlich so um die 10 Prozent) haben JavaScript deaktiviert.
- Sie werden dabei wahrscheinlich Fehler machen.

Im Ernst. Sie werden es falsch machen. Das Prüfen, ob eine Zeichenfolge eine gültige E-Mail-Adresse darstellt, ist unglaublich kompliziert (*http://www.regular-expressions.info/email.html*). Je genauer Sie hinsehen, umso komplizierter wird es (*http://www.ex-parrot.com/pdw/Mail-RFC822-Address.html*). Sagte ich schon, dass es wahnwitzig kompliziert ist (*http://haacked.com/archive/2007/08/21/i-knew-how-to-validate-an-email-address-until-i.aspx*)? Wäre es nicht einfacher, Sie könnten es dem Browser überlassen, sich darüber den Kopf zu zerbrechen?

Der Screenshot in Abbildung 9-16 ist aus Opera 10, aber die Funktion gibt es bereits seit Opera 9. Das einzige Markup, das daran beteiligt ist, ist das Setzen des `type`-Attributs auf `"email"` (siehe dazu den Abschnitt »E-Mail-Adressen« auf Seite 160). Wenn ein Opera-Benutzer versucht, ein Formular mit einem `<input type="email">`-Feld abzusenden, bietet Opera automatisch eine RFC-konforme E-Mail-Validierung, selbst wenn Scripting abgeschaltet ist.

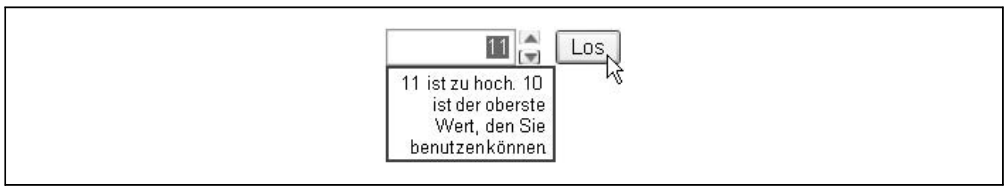


Abbildung 9-16: Opera validiert `type="email"`

Opera bietet ebenfalls eine Validierung von Webadressen, die in `<input type="url">`-Felder eingegeben werden, und von Zahlen in `<input type="number">`-Feldern. Die Validierung von Zahlen berücksichtigt sogar die Attribute `min` und `max`. Opera lässt also nicht zu,

dass Benutzer das Formular absenden, wenn sie eine Zahl eingeben, die zu groß ist (Abbildung 9-17).

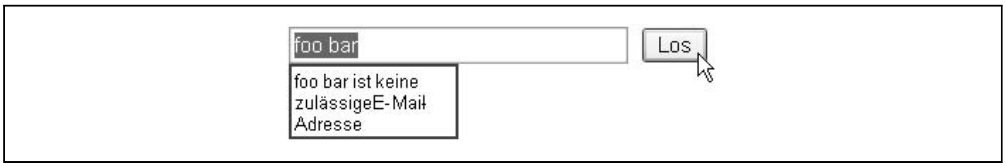


Abbildung 9-17: Opera validiert `type="number"`

Unglücklicherweise unterstützt kein weiterer Browser die automatische HTML5-Formularvalidierung. Eine Weile werden Sie also noch auf skriptbasierte Ausweichlösungen angewiesen sein.

Weitere Lektüre

Spezifikationen und Standards:

- `<input>`-Typen (*http://bit.ly/akweH4*)
- Das `<input placeholder>`-Attribut (*http://bit.ly/caGl8N*)
- Das `<input autofocus>`-Attribut (*http://bit.ly/db1Fj4*)

JavaScript-Bibliotheken:

- Modernizr (*http://www.modernizr.com*), eine HTML5-Erkennungsbibliothek

KAPITEL 10

Mehr Semantik fürs Web

Einstieg

Es gibt in HTML5 über 100 Elemente (*http://simon.html5.org/html5-elements*). Einige sind rein semantisch (siehe Kapitel 3), und andere sind lediglich Container für Skript-APIs (siehe Kapitel 4). Während der gesamten HTML-Entwicklungshistorie (siehe Kapitel 1) haben Standardfetischisten darüber gestritten, welche Elemente in die Sprache eingeschlossen werden sollten. Soll HTML ein `<figure>`-Element haben? Ein `<person>`-Element? Was wäre mit einem `<geschwafel>`-Element? Entscheidungen werden getroffen, Spezifikationen geschrieben, Verfasser verfassen, Implementierer implementieren, und das Web krabbelt gemächlich vorwärts.

Natürlich kann HTML nicht jeden zufriedenstellen. Das kann kein Standard. Einige Ideen setzen sich nicht durch. Beispielsweise gibt es in HTML5 kein `<person>`-Element. (Und leider auch kein `<geschwafel>`-Element!) Nichts hält Sie davon ab, in eine Webseite ein `<person>`-Element einzubauen, aber dann schlägt Ihre Validierung fehl, und außerdem funktioniert das nicht browserübergreifend konsistent (siehe dazu den Abschnitt »Wie Browser mit unbekannten Elementen umgehen« auf Seite 44) und könnte mit zukünftigen HTML-Spezifikationen in Konflikt geraten, sollte es später noch eingeführt werden.

Was also tut ein Autor, der semantische Auszeichnungen liebt, wenn es keine Lösung ist, sich einfach ein eigenes Element auszudenken? Es gab Versuche, die vorangegangenen Versionen von HTML zu erweitern. Die beliebteste Methode sind die sogenannten Mikroformate (*http://microformats.org*), die die `class`- und `rel`-Attribute von HTML 4 nutzen. Eine weitere Option ist RDFa (*http://www.w3.org/TR/rdfa-syntax/*), das ursprünglich für die Verwendung in XHTML (siehe dazu den Abschnitt »Nachtrag« auf Seite 16) gedacht war, jetzt aber auch auf HTML portiert wird.

Mikroformate und RDFa haben jeweils ihre Stärken und Schwächen. Sie wählen radikal verschiedene Ansätze, um das gleiche Ziel zu erreichen: Webseiten mit zusätzlichen semantischen Informationen anzureichern, die nicht Teil des Kerns von HTML sind. Ich will dieses Kapitel nicht für einen Formate-Krieg nutzen. (Zumal ich dazu unbedingt ein `<geschwafel>`-Element bräuchte!) Stattdessen möchte ich mich hier auf eine dritte Option konzentrieren, die ein gut integrierter Teil von HTML5 selbst ist: Mikrodaten.

Was sind Mikrodaten?

Jedes Wort im folgenden Satz ist wichtig, geben Sie also acht.

Professor Markup sagt

Mikrodaten annotieren das DOM mit geltungsbereichsbezogenen Name/Wert-Paaren aus selbst definierten Vokabularen.

Was also heißt das? Beginnen wir von hinten und arbeiten wir uns dann von dort nach vorne vor. Bei Mikrodaten geht er um selbst definierte Vokabulare. Betrachten Sie die »Menge aller HTML5-Elemente« als ein Vokabular. Dieses Vokabular enthält Elemente zur Repräsentation von Abschnitten oder Artikeln (siehe dazu den Abschnitt »Neue semantische Elemente in HTML5« auf Seite 43), aber keine Elemente zur Repräsentation von Personen oder Ereignissen. Wollen Sie auf einer Webseite eine Person darstellen, müssen Sie ein eigenes Vokabular definieren. Genau das können Sie mit Mikrodaten tun. Jeder kann sein eigenes Mikrodatenvokabular definieren und damit beginnen, seine selbst definierten Eigenschaften in seine Webseiten einzubetten.

Als Nächstes müssen Sie über Mikrodaten wissen, dass sie auf Name/Wert-Paaren basieren. Jedes Mikrodatenvokabular definiert eine Menge benannter Eigenschaften. Beispielsweise könnte ein Personvokabular Eigenschaften wie `name` und `photo` definieren. Wollen Sie in Ihre Webseite eine bestimmte Mikrodateneigenschaft einbauen, geben Sie den Namen dieser Eigenschaft an einer bestimmten Stelle an. Mikrodaten haben spezifische Regeln dazu, wie die Eigenschaftswerte abgerufen werden sollen, die davon abhängen, wo der Eigenschaftsname deklariert wurde. (Mehr dazu im nächsten Abschnitt.)

Neben benannten Eigenschaften basieren Mikrodaten stark auf dem Konzept der *Geltung* (Scoping). Am einfachsten stellt man sich den Geltungsbereich von Mikrodaten vor, wenn man an natürliche Eltern-Kind-Beziehungen im DOM denkt. Das `<html>`-Element (siehe dazu den Abschnitt »Das Wurzelelement« auf Seite 35) hat in der Regel zwei Kinder, `<head>` (siehe dazu den Abschnitt »Das <head>-Element« auf Seite 36) und `<body>`. Das `<body>`-Element hat üblicherweise eine ganze Reihe von Kindern, die ihrerseits eigene Kinder haben können. Beispielsweise könnte Ihre Seite ein `<h1>`-Element in einem `<hgroup>`-Element in einem `<header>`-Element (siehe dazu den Abschnitt »Kopfleisten und Überschriften« auf Seite 48) im `<body>`-Element enthalten. Oder eine Datentabelle könnte `<td>`-Elemente in `<tr>`-Elementen in einem `<table>`-Element (im `<body>`) enthalten. Mikrodaten nehmen diese hierarchische Struktur des DOM auf, um ein Mittel zu haben, mit dem sich ausdrücken lässt, dass »alle Eigenschaften in diesem Element aus diesem Vokabular entnommen sind«. Das ermöglicht Ihnen, auf einer Seite mehrere Mikrodatenvokabulare zu verwenden. Sie können Mikrodatenvokabulare sogar in andere Vokabulare einbetten, indem Sie die natürliche Struktur des DOM nutzen. (Ich werde Ihnen in diesem Kapitel mehrere Beispiele für geschachtelte Vokabulare zeigen.)

Ich habe das DOM bereits erwähnt, lassen Sie mich das trotzdem noch weiter ausführen. Bei Mikrodaten geht es darum, den Daten, die bereits auf der Webseite sichtbar sind, zusätzliche semantische Informationen mitzugeben. Mikrodaten sind nicht als eigenständiges Datenformat gedacht. Sie sind eine Ergänzung zu HTML. Wie Sie im nächsten Abschnitt sehen werden, funktionieren Mikrodaten am besten, wenn Sie HTML bereits korrekt einsetzen, das HTML-Vokabular aber nicht hinreichend ausdrucksfähig ist. Die Stärke von Mikrodaten ist die Verfeinerung der Semantik von Daten, die sich bereits im DOM befinden. Wenn die Daten, die Sie mit zusätzlichen semantischen Informationen ausstatten, nicht im DOM sind, sollten Sie einen Schritt zurücktreten und überlegen, ob Mikrodaten die richtige Lösung sind.

Ist die Aussage von Professor Markup jetzt verständlicher? Ich hoffe doch. Schauen wir uns jetzt an, wie das im Einsatz aussieht.

Das Mikrodaten-Datenmodell

Ein eigenes Mikrodatenvokabular zu definieren, ist leicht. Zunächst benötigen Sie einen Namensraum, aber das ist einfach nur eine URL. Die Namensraum-URL kann auf eine laufende Webseite zeigen, aber das ist nicht unbedingt nötig. Nehmen wir an, ich möchte ein Mikrodatenvokabular erstellen, das eine Person beschreibt. Wenn ich die Domain *data-vocabulary.org* habe, nutze ich die URL *http://data-vocabulary.org/Person* als Namensraum für mein Mikrodatenvokabular. Es ist ein ganz einfaches Verfahren, eine allgemeine eindeutige Kennung zu schaffen: Wählen Sie eine URL auf einer Domain, die Sie steuern.

In diesem Vokabular muss ich einige benannte Eigenschaften definieren. Beginnen wir mit drei elementaren Eigenschaften:

- `name` (der vollständige Name der Person)
- `photo` (ein Link auf ein Bild der Person)
- `url` (ein Link auf eine Site, die mit der Person verbunden ist, ein Blog beispielsweise oder ein Google-Profil)

Zwei dieser Eigenschaften sind URLs, die dritte ist gewöhnlicher Text. Für jede dieser Eigenschaft bietet sich bestimmtes Markup an – selbst wenn man überhaupt nicht an Mikrodaten oder Vokabulare denkt. Stellen Sie sich vor, Sie haben eine Profil- oder eine »Über mich«-Seite. Ihre Name ist wahrscheinlich als Überschrift markiert, wie ein `<h1>`-Element. Ihr Foto ist sicherlich ein `<img>`-Element, da die Besucher Ihrer Site es schließlich sehen wollen. Und alle URLs, die mit Ihrem Profil verbunden sind, sind wahrscheinlich bereits als Hyperlinks markiert, weil Ihre Besucher darauf klicken können sollen. Und nehmen wir, unserem Thema entsprechend, an, dass Ihr gesamtes Profil zusätzlich noch in ein `<section>`-Element gehüllt ist, um es vom restlichen Inhalt der Seite zu trennen. Das sähe dann ungefähr so aus:

```
<section>
  <h1>Mark Pilgrim</h1>
```

```
  <p><img src="http://www.example.com/photo.jpg" alt="[me smiling]"></p>
  <p><a href="http://diveintomark.org/">weblog</a></p>
</section>
```

Das Datenmodell für Mikrodaten sind Name/Wert-Paare. Der Name einer Mikrodateneigenschaft (wie `name` oder `photo` oder `url` in diesem Beispiel) wird immer auf einem HTML-Element deklariert. Der entsprechende Eigenschaftswert wird dann aus dem DOM des Elements entnommen. Bei den HTML-Elementen ist der Eigenschaftswert einfach der Textinhalt des Elements. Es gibt allerdings einige Ausnahmen, wie Tabelle 10-1 illustriert.

Tabelle 10-1: Woher kommen die Werte von Mikrodateneigenschaften?

Element	Wert
`<meta>`	`content`-Attribut

`<audio>`
`<embed>`
`<iframe>`
`<img>`
`<source>`
`<video>`
	`src`-Attribut

`<a>`
`<area>`
`<link>`
	`href`-Attribut

`<object>`	`data`-Attribut
`<time>`	`datetime`-Attribut
Alle anderen Elemente	Textinhalt

Wollen Sie Ihrer Seite »Mikrodaten hinzufügen«, müssen Sie den bereits vorhandenen HTML-Elementen nur ein paar Attribute hinzufügen. Der erste Schritt dabei ist immer, dass Sie mit einem `itemtype`-Attribut deklarieren, welches Mikrodatenvokabular Sie nutzen. Im zweiten Schritt deklarieren Sie mit einem `itemscope`-Attribut stets den Geltungsbereich des Vokabulars. In diesem Beispiel befinden sich alle Daten, die wir semantisch anreichern wollen, in einem `<section>`-Element. Wir deklarieren `itemtype` und `itemscope` also auf dem `<section>`-Element:

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```

Ihr Name ist das erste Datenelement im `<section>`-Element. Er steckt in einem `<h1>`-Element. Das `<h1>`-Element hat keine spezielle Verarbeitung, fällt also in die Kategorie »Alle anderen Elemente« in Tabelle 10-1. Der Wert der Mikrodateneigenschaft ist einfach der Textinhalt des Elements (das wäre auch der Fall, wenn sich der Name in einem `<p>`-, `<div>`- oder `<span>`-Element befände):

```
<h1 itemprop="name">Mark Pilgrim</h1>
```

Auf Deutsch sagt das: »Hier ist die `name`-Eigenschaft des Vokabulars `http://data-vocabulary.org/Person`. Der Wert der Eigenschaft ist `Mark Pilgrim`.«

Als Nächstes folgt die Eigenschaft photo. Das soll eine URL sein. Gemäß Tabelle 10-1 ist der Wert eines <img>-Elements sein src-Attribut. Und siehe da, die URL Ihres Profilfotos steckt bereits in einem <img src>-Attribut! Sie müssen also nur noch deklarieren, dass das <img>-Element die photo-Eigenschaft ist:

```
<p><img itemprop="photo"
        src="http://www.example.com/photo.jpg"
        alt="[me smiling]"></p>
```

Auf Deutsch sagt das: »Hier ist die photo-Eigenschaft des Vokabulars http://data-vocabulary.org/Person. Der Wert der Eigenschaft ist http://www.example.com/photo.jpg.«

Auch die url-Eigenschaft ist eine URL. Gemäß Tabelle 10-1 ist der Wert eines <a>-Elements sein href-Attribut. Und wieder einmal passt auch das mit dem bestehenden Markup zusammen. Sie müssen also nur noch sagen, dass das vorhandene <a>-Element die url-Eigenschaft ist:

```
<a itemprop="url" href="http://diveintomark.org/">dive into mark</a>
```

Auf Deutsch sagt das: »Hier ist die url-Eigenschaft des Vokabulars http://data-vocabulary.org/Person. Der Wert der Eigenschaft ist http://diveintomark.org/.«

Natürlich ist es auch kein Problem, wenn Ihr Markup etwas anders aussieht. Sie können Mikrodateneigenschaften und -werte beliebigem HTML-Markup hinzufügen, selbst verrostetem 20.-Jahrhundert-Markup, Tabellenlayout-Markup, Mein-Gott-warum-sagte-ich-dass-ich-das-warten-werde-Markup. Auch wenn ich Ihnen Markup wie das folgende keinesfalls empfehlen will, ist es immer noch sehr verbreitet und kann mit Mikrodaten aufgewertet werden:

```
<TABLE>
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
</TABLE>
```

Um die Eigenschaft name einzufügen, müssen Sie nur der Tabellenzelle mit dem Namen ein itemprop-Attribut hinzufügen. Tabellenzellen haben keine speziellen Regeln in der Tabelle für die Werte von Mikrodateneigenschaften, sie erhalten also den Standardwert, d.h., der Wert der Mikrodateneigenschaften ist der Textinhalt:

```
<TR><TD>Name<TD itemprop="name">Mark Pilgrim
```

Das Einfügen der url-Eigenschaft scheint etwas komplizierter. Dieses Markup nutzt das <a>-Element nicht korrekt. Statt das Linkziel im href-Attribut anzugeben, enthält das href-Attribut hier keinen ordentlichen Wert und nutzt JavaScript im onclick-Attribut, um eine Funktion (die hier nicht gezeigt wird) aufzurufen, die die URL herauszieht und zu ihr navigiert. Nehmen wir für ein paar Bonuspunkte der Kategorie »Was ein Müll, hör sofort damit auf!« an, dass die Funktion den Link in einem kleinen Pop-up-Fenster ohne Scrollleisten öffnet. Was war das Internet im letzten Jahrhundert doch für ein Spaß, nicht wahr?

Aber auch das können Sie immer noch in eine Mikrodateneigenschaft umwandeln. Sie müssen nur etwas kreativ werden. Wir können das <a>-Element nicht unmittelbar verwenden. Das Linkziel befindet sich nicht im href-Attribut, und es gibt keine Möglichkeit, die Regel zu überschreiben, die sagt: »Nimm bei einem <a>-Element den Wert für die Mikrodateneigenschaft aus dem href-Attribut.« Aber Sie können das ganze Chaos in ein Wrapper-Element packen und dieses nutzen, um die url-Mikrodateneigenschaft zu ergänzen:

```
<TABLE itemscope itemtype="http://data-vocabulary.org/Person">
  <TR><TD>Name<TD>Mark Pilgrim
  <TR><TD>Link<TD>
    <span itemprop="url">
      <A href=# onclick=goExternalLink()>http://diveintomark.org/</A>
    </span>
</TABLE>
```

Weil für das -Element keine spezielle Verarbeitung definiert ist, nutzt es die Standardregel: »Die Mikrodateneigenschaft ist der Textinhalt.« »Textinhalt« bedeutet nicht »das gesamte Markup im Element« (was Sie beispielsweise erhalten würden, wenn Sie die DOM-Eigenschaft innerHTML nutzen). Es heißt »nur den Text, bitte«. In diesem Fall ist http://diveintomark.org/ der Textinhalt des <a>-Elements im -Element.

Zusammengefasst: Sie können Mikrodateneigenschaften jedem Markup hinzufügen. Es wird Ihnen leichter fallen, wenn das HTML-Markup in Ordnung ist, als wenn es ein Chaos ist. Aber möglich ist es immer.

Personen auszeichnen

Die Einführungsbeispiele im vorangegangenen Abschnitt waren nicht vollständig an den Haaren herbeigezogen. Es gibt tatsächlich ein Mikrodatenvokabular zur Auszeichnung von Informationen zu Personen, und es ist tatsächlich so einfach. Schauen wir es uns genauer an.

Am einfachsten können Sie Mikrodaten in eine persönliche Website auf Ihrer »Über mich«-Seite einfügen. Sie haben doch eine »Über mich«-Seite, nicht wahr? Wenn nicht, können Sie sich die Sache ansehen, während ich diese Beispiel-»Über mich«-Seite (*http://diveintohtml5.org/examples/person.html*) mit zusätzlichen semantischen Informationen auszustatte. Das endgültige Resultat ist hier: *http://diveintohtml5.org/examples/person-plus-microdata.html*.

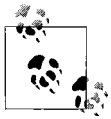
Schauen wir uns zunächst das nackte Markup an, bevor ihr irgendwelche Mikrodateneigenschaften hinzugefügt wurden:

```
<section>
  <img width="204" height="250"
       src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
       alt="[Mark Pilgrim, circa 2000]">

  <h1>Contact Information</h1>
  <dl>
    <dt>Name</dt>
    <dd>Mark Pilgrim</dd>
```

```
    <dt>Position</dt>
    <dd>Developer advocate for Google, Inc.</dd>

    <dt>Mailing address</dt>
    <dd>
      100 Main Street<br>
      Anytown, PA 19999<br>
      USA
    </dd>
  </dl>
  <h1>My Digital Footprints</h1>
  <ul>
    <li><a href="http://diveintomark.org/">weblog</a></li>
    <li><a href="http://www.google.com/profiles/pilgrim">Google profile</a></li>
    <li><a href="http://www.reddit.com/user/MarkPilgrim">Reddit.com profile</a></li>
    <li><a href="http://www.twitter.com/diveintomark">Twitter</a></li>
  </ul>
</section>
```

Wie üblich müssen Sie zunächst das Vokabular deklarieren, das Sie nutzen, sowie seinen Geltungsbereich. Sie tun das, indem Sie dem äußersten Element, das alle anderen Elemente mit den Daten enthält, die Attribute `itemtype` und `itemscope` hinzufügen. Hier ist das ein `<section>`-Element:

```
<section itemscope itemtype="http://data-vocabulary.org/Person">
```



Sie können sich die Änderungen, die wir bislang in diesem Abschnitt gemacht haben, online ansehen. Vorher: *http://diveintohtml5.org/examples/person.html*. Nachher: *http://diveintohtml5.org/examples/person-plus-microdata.html*.

Beginnen Sie jetzt damit, die Mikrodateneigenschaften aus dem `http://data-vocabulary.org/Person`-Vokabular einzusetzen. Aber welche Eigenschaften sind das? Die Liste dieser Eigenschaften können Sie einsehen, indem zu *http://data-vocabulary.org/Person* gehen. Die Mikrodatenspezifikation verlangt das nicht, aber ich würde sagen, dass das auf alle Fälle empfehlenswert ist. Wenn Sie wollen, dass Entwickler Ihr Mikrodatenvokabular nutzen, müssen Sie es irgendwo dokumentieren. Und welchen besseren Ort für eine solche Dokumentation könnte es geben als die Vokabular-URL selbst? Tabelle 10-2 führt die Eigenschaften des Personenvokabulars auf.

Tabelle 10-2: Personenvokabular

Eigenschaft	Beschreibung
`name`	Name.
`nickname`	Spitzname.
`photo`	Ein Link auf ein Bild.
`title`	Der Titel der Person (beispielsweise »Finanzvorstand«).
`role`	Die Rolle der Person (beispielsweise »Buchhalter«).
`url`	Ein Link auf eine Webseite (beispielsweise die Homepage der Person).
`affiliation`	Der Name der Organisation, mit der die Person verbunden ist (beispielsweise ein Arbeitgeber).

Tabelle 10-2: Personenvokabular (Fortsetzung)

Eigenschaft	Beschreibung
`friend`	Identifiziert ein soziales Verhältnis zwischen der beschriebenen Person und einer anderen Person.
`contact`	Identifiziert ein soziales Verhältnis zwischen der beschriebenen Person und einer anderen Person.
`acquaintance`	Identifiziert ein soziales Verhältnis zwischen der beschriebenen Person und einer anderen Person.
`address`	Der Ort der Person (kann die Untereigenschaften `street-address`, `locality`, `region`, `postal-code` und `country-name` enthalten).

Das Erste in dieser Beispiel-»Über mich«-Seite ist ein Bild von mir. Wie zu erwarten, steckt es in einem `<img>`-Element. Wollen wir deklarieren, dass das `<img>`-Element das Profilbild ist, müssen wir ihm nur das Attribut `itemprop="photo"` hinzufügen:

```
<img itemprop="photo" width="204" height="250"
    src="http://diveintohtml5.org/examples/2000_05_mark.jpg"
    alt="[Mark Pilgrim, circa 2000]">
```

Wo ist der Wert der Mikrodateneigenschaft? Er ist bereits da, im `src`-Attribut. Wie Sie in Tabelle 10-1 gesehen haben, ist der »Wert« eines `<img>`-Elements sein `src`-Attribut. Jedes `<img>`-Element hat ein `src`-Attribut – andernfalls wäre es ein defektes Bild – und `src` ist immer eine URL. Verstehen Sie? Wenn Sie HTML korrekt nutzen, sind Mikrodaten leicht.

Außerdem steht das `<img>`-Element nicht allein auf der Seite. Es ist das Kindelement des `<section>`-Elements, des Elements, auf dem wir gerade das `itemscope`-Attribut deklariert haben. Mikrodaten nutzen die Eltern-Kind-Verhältnisse der Elemente auf der Seite, um den Geltungsbereich der Mikrodateneigenschaften zu definieren. Im Klartext heißt das: »Dieses `<section>`-Element repräsentiert eine Person. Alle Mikrodateneigenschaften, die auf Kindern des `<section>`-Elements definiert sind, sind Eigenschaften der Person.« Sollte Ihnen die Vorstellung helfen, können Sie das `<section>`-Element als Subjekt eines Satzes betrachten. Das Attribut `itemprop` repräsentiert das Verb des Satzes – so etwas wie »wird abgebildet bei« –, und der Wert der Mikrodateneigenschaften repräsentiert das Objekt des Satzes:

> Diese Person [explizit, aus `<section itemscope itemtype="...">`]
>
> wird abgebildet bei [explizit, aus `<img itemprop="photo">`]
>
> `http://diveintohtml5.org/examples/2000_05_mark.jpg` [implizit, aus dem `<img src>` -Attribut]

Das Subjekt muss nur einmal definiert werden, indem die Attribute `itemscope` und `itemtype` auf dem äußeren `<section>`-Element definiert werden. Das Verb wird definiert, indem das Attribut `itemprop="photo"` auf dem `<img>`-Element angegeben wird. Das Objekt des Satzes erfordert überhaupt kein besonderes Markup, weil Tabelle 10-1 sagt, dass der Eigenschaftswert eines `<img>`-Elements sein `src`-Attribut ist.

Gehen wir zum nächsten Teil des Markups weiter, sehen wir eine `<h1>`-Überschrift und den Anfang einer `<dl>`-Liste. Weder `<h1>`, noch `<dl>` muss mit Mikrodaten versehen werden. Nicht jeder Teil von HTML muss eine Mikrodateneigenschaft sein. Bei Mikrodaten geht es um die Eigenschaften selbst, nicht um das Markup oder Überschriften, die

die Eigenschaften umgeben. Das <h1> ist keine Eigenschaft, es ist einfach nur eine Überschrift. Ebenso sagt das <dt>, dass »Name« einfach eine Beschriftung ist, keine Eigenschaft:

```
<h1>Contact Information</h1>
  <dl>
    <dt>Name</dt>
    <dd>Mark Pilgrim</dd>
```

Wo also stecken die eigentlichen Informationen? Sie stecken im <dd>-Element, dort also müssen wir das Attribut itemprop einsetzen. Welche Eigenschaft ist es? Die name-Eigenschaft. Wo ist der Eigenschaftswert? Es ist der Text im <dd>-Element. Muss das irgendwie mit Markup versehen werden? Tabelle 10-1 sagt Nein, <dd>-Elemente haben keine spezielle Verarbeitung, der Eigenschaftswert ist also nur der Text im Element:

```
<dd itemprop="name">Mark Pilgrim</dd>
```

Und was haben wir damit gesagt? »Der Name dieser Person ist Mark Pilgrim.« Schön, schön. Also weiter.

Die nächsten beiden Eigenschaften sind etwas komplizierter. So sieht das Markup vor Einführung der Mikrodaten aus:

```
<dt>Position</dt>
<dd>Developer advocate for Google, Inc.</dd>
```

Schauen Sie sich die Definition des Personenvokabulars an, bemerken Sie, dass der Text »Developer advocate for Google, Inc.« eigentlich zwei Eigenschaften umschließt: title (»Developer advocate«) und affiliation (»Google, Inc.«). Wie kann man das mit Mikrodaten ausdrücken? Die kurze Antwort lautet: Das geht nicht. Mikrodaten bieten keine Möglichkeit, einen Text in zwei eigenständige Eigenschaften aufzubrechen. Sie können nicht sagen: »Die ersten 18 Zeichen dieses Texts bilden diese Mikrodateneigenschaft, und die letzten 12 Zeichen des Texts bilden jene Mikrodateneigenschaft.«

Aber es ist noch nicht alles verloren. Stellen Sie sich vor, Sie möchten den Text »Developer advocate« in einer anderen Schrift darstellen als den Text »Google, Inc.«. CSS kann das ebenfalls nicht. Was also wollen wir tun? Normalerweise müssten Sie zunächst die verschiedenen Textteile in Dummy-Elemente wie hüllen. Erst dann können Sie die unterschiedlichen CSS-Regeln auf die einzelnen -Elemente anwenden.

Diese Technik ist auch für Mikrodaten nützlich. Es gibt hier zwei Informationsbestandteile: title und affiliation. Packen Sie jeden dieser Bestandteile in Dummy-s, können Sie deklarieren, dass jedes eine eigene Mikrodateneigenschaft ist:

```
<dt>Position</dt>
<dd><span itemprop="title">Developer advocate</span> for
    <span itemprop="affiliation">Google, Inc.<span></dd>
```

Ta-da! Im Klartext heißt das: »Der Titel dieser Person ist ›Developer advocate‹. Diese Person ist angestellt bei Google, Inc.« Zwei Sätze, zwei Mikrodateneigenschaften. Etwas mehr Markup, aber das ist ein Kompromiss, der die Mühe wert ist.

Die gleiche Technik ist auch bei der Auszeichnung der Postadresse von Nutzen. Das Personenvokabular definiert eine address-Eigenschaft, die selbst ein Mikrodatenelement ist. Das bedeutet, dass es für die Adresse ein eigenes Vokabular gibt (*http://data-vocabulary.org/Address*), das seine eigenen Eigenschaften definiert: street-address, locality, region, postal-code und country-name.

Wenn Sie ein richtiger Programmierer sind, sind Sie wahrscheinlich mit der Punktnotation vertraut, die das Verhältnis von Objekten und ihren Eigenschaften definiert. Stellen Sie sich das Verhältnis wie folgt vor:

- Person
- Person.address
- Person.address.street-address
- Person.address.locality
- Person.address.region
- Person.address.postal-code
- Person.address.country-name

In diesem Beispiel ist die gesamte Straßenadresse in einem einzigen <dd>-Element enthalten. (Wieder ist das <dt>-Element nur eine Beschriftung. Es spielt also keine Rolle bei der Erweiterung der Semantik mit Mikrodaten.) Die address-Eigenschaft zu markieren, ist leicht. Wir fügen einfach dem <dd>-Element ein itemprop-Attribut hinzu:

```
<dt>Mailing address</dt>
<dd itemprop="address">
```

Aber denken Sie daran, dass die address-Eigenschaft ein Mikrodatenelement ist. Das heißt, dass wir ihm ebenfalls die Attribute itemscope und itemtype hinzufügen müssen:

```
<dt>Mailing address</dt>
<dd itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
```

Alles das haben wir bereits gesehen, aber nur auf Elementen der obersten Ebene. Ein <section>-Element definiert itemtype und itemscope, und alle Elemente im <section>-Element, die Mikrodateneigenschaften definieren, fallen unter dieses spezifische Vokabular. Aber das ist das erste Mal, dass wir geschachtelte Geltungsbereiche sehen – neue itemtype- und itemscope-Attribute (auf dem <dd>-Element) in einem bestehenden Geltungsbereich (auf dem <section>-Element). Dieser eingebettete Geltungsbereich funktioniert genau so wie im HTML-DOM. Das <dd>-Element hat eine bestimmte Anzahl von Kindelementen, die alle in den Geltungsbereich des Vokabulars fallen, das auf dem <dd>-Element deklariert wird. Wird das <dd>-Element mit dem entsprechenden </dd>-Tag geschlossen, tritt wieder das Vokabular in Geltung, das vom Elternelement (hier <section>) deklariert wird.

Die Eigenschaften des Address-Vokabulars leiden unter dem gleichen Problem wie die Eigenschaften title und affiliation. Es gibt nur einen langen Text, den wir in mehrere Mikrodateneigenschaften aufspalten wollen. Auch die Lösung ist die gleiche. Wir packen

alle Informationsbestandteile in Dummy-s und deklarieren Mikrodateneigenschaften auf jedem -Element:

```
    <dd itemprop="address" itemscope
        itemtype="http://data-vocabulary.org/Address">
      <span itemprop="street-address">100 Main Street</span><br>
      <span itemprop="locality">Anytown</span>,
      <span itemprop="region">PA</span>
      <span itemprop="postal-code">19999</span>
      <span itemprop="country-name">USA</span>
    </dd>
  </dl>
```

Im Klartext heißt das: »Diese Person hat eine Postanschrift. Der Straße-Teil der Postanschrift lautet ›100 Main Street‹. Der Ort ist ›Anytown‹. Die Region ist ›PA‹. Die Postleitzahl ist ›19999‹. Das Land ist ›USA‹.« Ganz simpel.

Fragen an Professor Markup

F: Ist das Format der Postanschrift US-spezifisch?

A: Nein. Die Eigenschaften des Address-Vokabulars sind so allgemein, dass man Adressen auf der gesamten Welt beschreiben kann. Nicht alle Adressen haben Werte für jede Eigenschaft, aber das ist kein Problem. Einige Adressen könnten erfordern, dass mehrere »Zeilen« in eine Eigenschaft eingehen, aber auch das ist kein Problem. Hätte Ihre Adresse eine Straßenadresse und eine Blocknummer, kämen beide in die `street-address`-Untereigenschaft:

```
<p itemprop="address" itemscope
    itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">
    Hauptstraße 100
    Block A
  </span>
  ...
</p>
```

Es gibt noch eine weitere Sache in dieser »Über mich«-Seite: eine Liste mit URLs. Das Personenvokabular hat eine Eigenschaft dafür namens `url`. Eine `url`-Eigenschaft kann alles sein. (Gut, es muss eine URL sein, aber das war Ihnen sicher klar.) Was ich damit sagen will, ist, dass die Definition der `url`-Eigenschaft sehr vage ist. Die Eigenschaft kann jede Art von URL angeben, die Sie mit der Person verbinden wollen: ein Blog, ein Fotoalbum oder ein Profil auf einer Site wie Facebook oder Twitter.

Die andere bemerkenswerte Sache hier ist, dass eine einzelne Person mehrere `url`-Eigenschaften haben kann. Technisch gesehen, kann jede Eigenschaft mehrfach erscheinen, aber bislang haben wir das noch nicht ausgenutzt. Beispielsweise könnten Sie zwei `photo`-Eigenschaften haben, die jeweils auf eine andere Bild-URL zeigen. In diesem Beispiel möchte ich vier unterschiedliche URLs angeben: mein Weblog, meine Google-

Profilseite, mein Benutzerprofil auf Reddit und mein Twitter-Konto. In HTML ist das eine Liste mit vier Links: vier <a>-Elemente, die jeweils in einem eigenen -Element stehen. In Mikrodaten erhält jedes <a>-Element ein itemprop="url"-Attribut:

```
<h1>My Digital Footprints</h1>
<ul>
  <li><a href="http://diveintomark.org/"
         itemprop="url">weblog</a></li>
  <li><a href="http://www.google.com/profiles/pilgrim"
         itemprop="url">Google profile</a></li>
  <li><a href="http://www.reddit.com/user/MarkPilgrim"
         itemprop="url">Reddit.com profile</a></li>
  <li><a href="http://www.twitter.com/diveintomark"
         itemprop="url">Twitter</a></li>
</ul>
```

Gemäß Tabelle 10-1 werden <a>-Elemente besonders verarbeitet. Der Wert der Mikrodateneigenschaft ist das href-Attribut, nicht der Textinhalt des Kindelements. Der Text der Links wird vom Mikrodatenprozessor ignoriert. Im Klartext sagt das: »Diese Person hat eine URL bei *http://diveintomark.org/*. Diese Person hat eine weitere URL bei *http://www.google.com/profiles/pilgrim*. Diese Person hat eine weitere URL bei *http://www.reddit.com/user/MarkPilgrim*. Diese Person hat eine weitere URL bei *http://www.twitter.com/diveintomark*.«

Google Rich Snippets

Ich möchte für einen Augenblick zurücktreten und fragen: Warum machen wir das? Ist diese zusätzliche semantische Auszeichnung reiner Selbstzweck? Verstehen Sie mich nicht falsch: Mit spitzen Klammern herumzuspielen macht mir genauso viel Spaß wie jedem anderen Webfreak. Aber warum Mikrodaten? Warum sollen wir diesen Aufwand treiben?

Es gibt zwei große Klassen von Anwendungen, die HTML und implizit auch HTML5-Mikrodaten nutzen können:

- Webbrowser
- Suchmaschinen

Für Browser definiert HTML5 einen Satz von DOM-APIs zum Extrahieren von Mikrodatenelementen, -eigenschaften und -eigenschaftswerten aus einer Webseite. Als ich dies schrieb, unterstützte noch kein Browser diese API. Nicht ein einziger. Das ist also eine Sackgasse, zumindest bis Browser mitziehen und die clientseitigen APIs implementieren.

Der andere wichtige Verbraucher von HTML sind Suchmaschinen. Was könnte eine Suchmaschine mit Mikrodateneigenschaften zu einer Person anfangen? Stellen Sie sich Folgendes vor: Statt einfach einen Seitentitel anzuzeigen und einen Textauszug zu präsentieren, könnte eine Suchmaschine einige dieser strukturierten Informationen integrieren und anzeigen – den Namen, die Berufsbeschreibung, den Arbeitgeber, die Adresse, vielleicht sogar eine kleine Vorschau eines Profilbilds. Würde das Ihre Aufmerksamkeit einfangen? Meine bestimmt.

Google unterstützt Mikrodaten als Teil seines Rich Snippets-Programms (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170*). Wenn Googles Webcrawler Ihre Seite parst und Mikrodateneigenschaften findet, die dem Vokabular `http://data-vocabulary.org/Person` entsprechen, parst er diese Eigenschaften und speichert sie neben dem Rest der Seite. Google bietet sogar ein praktisches Werkzeug, das Ihnen zeigt, wie es Ihre Mikrodateneigenschaften »sieht«. Testen wir das mit unserer mikrodatenbereicherten »Über uns«-Seite (*http://diveintohtml5.org/examples/person-plus-microdata.html*), erhalten wir diese Ausgabe:

```
Item
  Type: http://data-vocabulary.org/person
  photo = http://diveintohtml5.org/examples/2000_05_mark.jpg
  name = Mark Pilgrim
  title = Developer advocate
  affiliation = Google, Inc.
  address = Item( 1 )
  url = http://diveintomark.org/
  url = http://www.google.com/profiles/pilgrim
  url = http://www.reddit.com/user/MarkPilgrim
  url = http://www.twitter.com/diveintomark

Item 1
  Type: http://data-vocabulary.org/address
  street-address = 100 Main Street
  locality = Anytown
  region = PA
  postal-code = 19999
  country-name = USA
```

Es ist alles da: die `photo`-Eigenschaft auf dem Attribut `<img src>`, alle vier URLs aus der Liste der `<a href>`-Attribute und sogar das Adressobjekt (das als `Item 1` aufgeführt wird) mit allen fünf Untereigenschaften.

Und wie nutzt Google diese Informationen? Je nachdem. Es gibt keine festen Regeln dafür, wie Mikrodateneigenschaften angezeigt werden sollen oder ob sie überhaupt angezeigt werden sollen. Wenn jemand nach »Mark Pilgrim« sucht und Google feststellt, dass diese »Über mich«-Seite in die Ergebnisse aufgenommen werden sollte, und Google entscheidet, dass die auf der Seite gefundenen Mikrodateneigenschaften das Anzeigen wert sind, könnte die Aufstellung der Suchergebnisse in etwa aussehen, wie in Abbildung 10-1 gezeigt.

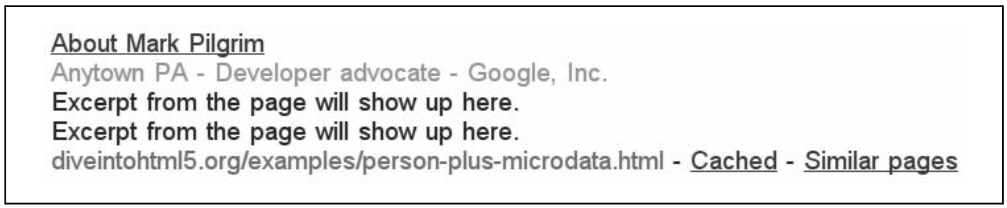


Abbildung 10-1: Beispiel für die Suchergebnisse für eine mikrodatenbereicherte Person-Aufstellung

Die erste Zeile »About Mark Pilgrim« ist tatsächlich der Titel der Seite, der im `<title>`-Element aufgeführt ist. Das ist nicht sonderlich aufregend; Google macht das für jede Seite.

Aber die zweite Zeile enthält die vollständigen Informationen aus den Mikrodatenanmerkungen, mit denen wir die Seite annotiert hatten. »Anytown PA« war Teil der Postanschrift, die mit dem Vokabular `http://data-vocabulary.org/Address` ausgezeichnet wurde. »Developer advocate« und »Google, Inc.« waren zwei Eigenschaften aus dem Vokabular `http://data-vocabulary.org/Person` (`title` respektive `affiliation`).

Das ist wirklich sehr erstaunlich. Sie müssen kein riesiges Unternehmen sein, das einen besonderen Deal mit Suchmaschinenherstellern macht, um die eigenen Suchergebnisse anpassen zu lassen. Nehmen Sie sich zehn Minuten Zeit und fügen Sie Ihrem HTML einige Attribute hinzu, um Daten mit Anmerkungen zu versehen, die Sie ohnehin veröffentlichen wollten.

Fragen an Professor Markup

F: Ich habe genau das gemacht, was Sie gesagt haben, aber meine Google-Suchergebnisse ändern sich nicht. Warum das?

A: »Google garantiert nicht, dass das Markup auf einer Seite in den Suchergebnissen genutzt wird« (http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170 (*http://www.google.com/support/webmas ters/bin/answer.py?hl=en&answer=99170*)). Aber selbst wenn Google sich nicht entschließt, Ihre Mikrodatenanmerkungen zu berücksichtigen – ein anderer tut es vielleicht. Wie der Rest von HTML5 sind Mikrodaten ein offener Standard, den jeder implementieren kann. Es ist Ihre Aufgabe, so viele Daten wie möglich bereitzustellen. Lassen Sie den Rest der Welt entscheiden, was er mit ihnen anstellen will. Sie könnten eine Überraschung erleben!

Organisationen auszeichnen

Mikrodaten sind nicht auf ein einziges Vokabular beschränkt. »Über mich«-Seiten sind nett, aber man hat wahrscheinlich nur eine davon. Ist Ihr Hunger noch nicht gestillt? Schauen wir uns an, wie man Organisationen und Unternehmen auszeichnet.

Ich habe eine Beispielseite mit Daten zu einem Unternehmen erstellt (*http://diveintohtml5.org/examples/organization.html*). Schauen wir uns das ursprüngliche HTML-Markup ohne Mikrodaten an:

```
<article>
  <h1>Google, Inc.</h1>
  <p>
    1600 Amphitheatre Parkway<br>
    Mountain View, CA 94043<br>
    USA
  </p>
  <p>650-253-0000</p>
  <p><a href="http://www.google.com/">Google.com</a></p>
</article>
```

Die Änderungen, die wir in diesem Abschnitt vornehmen werden, können Sie sich ansehen. Vorher: *http://diveintohtml5.org/examples/organization.html*. Nachher: *http://diveintohtml5.org/examples/organization-plus-microdata.html*.

Kurz und knapp. Alle Informationen zu dem Unternehmen befinden sich im `<article>`-Element. Beginnen wir damit:

```
<article itemscope itemtype="http://data-vocabulary.org/Organization">
```

Wie beim Auszeichnen von Menschen müssen Sie die Attribute `itemscope` und `itemtype` auf dem äußersten Element definieren. In diesem Fall ist das äußerste Element ein `<article>`-Element. Das Attribut `itemtype` deklariert das Mikrodatenvokabular, das Sie verwenden (in diesem Fall *http://data-vocabulary.org/Organization*), und das Attribut `itemscope` deklariert, dass alle Eigenschaften, die Sie auf Kindelementen setzen, in diesem Vokabular stehen.

Was also befindet sich im Vokabular für Organisationen? Es ist sehr einfach und gradlinig. Einiges davon sollte Ihnen bereits vertraut sein. Tabelle 10-3 führt die relevanten Eigenschaften auf.

Tabelle 10-3: Vokabular für Organisationen

Eigenschaft	Beschreibung
`name`	Der Name der Organisation (zum Beispiel »Initech«).
`url`	Ein Link auf die Homepage der Organisation.
`address`	Der Ort der Organisation. Kann die Untereigenschaften `street-address`, `locality`, `region`, `postal-code` und `country-name` enthalten.
`tel`	Die Telefonnummer der Organisation.
`geo`	Die geografischen Koordinaten des Orts. Enthält immer zwei Untereigenschaften, `latitude` und `longitude`.

Das erste Markup-Element im äußersten `<article>`-Element ist ein `<h1>`-Element. Dieses enthält den Namen des Unternehmens, wir geben also unmittelbar darauf das Attribut `itemprop="name"` an:

```
<h1 itemprop="name">Google, Inc.</h1>
```

Gemäß Tabelle 10-1 haben `<h1>`-Elemente keine besondere Verarbeitung. Der Wert der Mikrodateneigenschaft ist einfach der Textinhalt des `<h1>`-Elements. Im Klartext sagen wir damit: »Der Name der Organisation ist ›Google, Inc‹.«

Dann folgt die Postanschrift. Die Auszeichnung der Adresse einer Organisation funktioniert genau wie die Auszeichnung der Adresse einer Person. Erst fügen wir dem äußersten Element der Adresse ein `itemprop="address"`-Attribut hinzu (hier einem `<p>`-Element). Das besagt, dass dies die `address`-Eigenschaft der Organisation ist. Aber was ist mit den Eigenschaften der Adresse selbst? Wir müssen auch mit den Attributen `itemtype` und `itemscope` sagen, dass das ein Adresselement mit eigenen Eigenschaften ist:

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
```

Schließlich müssen wir noch die Informationsbestandteile in -Elemente einschließen, damit wir die entsprechenden Mikrodateneigenschaften (`street-address`, `locality`, `region`, `postal-code` und `country-name`) auf den -Elementen deklarieren können:

```
<p itemprop="address" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">1600 Amphitheatre Parkway</span><br>
  <span itemprop="locality">Mountain View</span>,
  <span itemprop="region">CA</span>
  <span itemprop="postal-code">94043</span><br>
  <span itemprop="country-name">USA</span>
</p>
```

Im Klartext heißt das: »Diese Organisation hat eine Adresse. Die Straßenadresse lautet ›1600 Amphitheatre Parkway‹. Der Ort ist ›Mountain View‹. Die Region ist ›CA‹. Die Postleitzahl ist ›94043‹. Der Name des Landes ist ›USA‹.«

Dann folgt die Telefonnummer der Organisation. Telefonnummern sind grundsätzlich kompliziert, und die genaue Syntax ist landesspezifisch. (Und wenn Sie in ein anderes Land telefonieren wollen, ist es noch komplizierter.) In diesem Beispiel haben wir eine Telefonnummer aus den Vereinigten Staaten in einem Format, das für einen Anruf aus einem anderen Teil der Vereinigten Staaten geeignet ist:

```
<p itemprop="tel">650-253-0000</p>
```

(Und sollten Sie es nicht bemerkt haben: Wir haben den Geltungsbereich des Adressvokabulars verlassen, als das <p>-Element geschlossen wurde. Jetzt sind wir wieder bei der Definition von Eigenschaften im Vokabular für Organisationen.)

Wenn Sie mehrere Telefonnummern aufführen wollen, beispielsweise eine für Kunden in den Vereinigten Staaten und eine für Kunden in anderen Ländern, können Sie das tun. Jede Mikrodateneigenschaft kann wiederholt werden. Achten Sie nur darauf, dass jede Telefonnummer in einem eigenen HTML-Element steht und von eventuellen Beschriftungselementen getrennt ist:

```
<p>
  US customers: <span itemprop="tel">650-253-0000</span><br>
  UK customers: <span itemprop="tel">00 + 1* + 6502530000</span>
</p>
```

Gemäß Tabelle 10-1 haben weder <p>- noch -Elemente eine besondere Verarbeitung. Der Wert der Mikrodateneigenschaft `tel` ist einfach der Textinhalt. Das Mikrodatenvokabular für Organisationen versucht nicht, die unterschiedlichen Teile einer Telefonnummer zu trennen. Die gesamte Eigenschaft `tel` ist einfach Klartext. Wenn Sie in Klammern Vorwahlen hinzufügen oder statt Bindestrichen Leerzeichen verwenden wollen, um die Elemente zu trennen, können Sie das tun. Wenn eine Client, der Mikrodaten verbraucht, eine Telefonnummer parsen will, ist er dabei auf sich selbst angewiesen.

Darauf folgt eine weitere vertraute Eigenschaft: `url`. Genau wie Sie eine URL mit einer Person verbinden können – wie wir es im letzten Abschnitt beschrieben haben –, können Sie eine URL mit einer Organisation verbinden. Das könnte die Homepage des Unternehmens sein, eine Kontaktseite, eine Produktseite oder irgendetwas anderes. Wenn es eine URL über, von oder im Eigentum der Organisation ist, versehen Sie sie mit einem `itemprop="url"`-Attribut:

```
<p><a itemprop="url" href="http://www.google.com/">Google.com</a></p>
```

Gemäß Tabelle 10-1 hat das `<a>`-Element eine besondere Verarbeitung. Der Wert der Mikrodateneigenschaft ist der Wert des `href`-Attributs, nicht der Linktext. In Klartext heißt das: »Diese Organisation ist mit der URL *http://www.google.com/* verbunden.« Es sagt nichts Genaueres über die Art der Verbindung und schließt den Linktext »Google.com« nicht ein.

Zum Abschluss möchte ich noch über Geolocation reden. Nein, nicht die W3C-Geolocation-API (siehe Kapitel 6). Ich spreche hier davon, wie man den physischen Ort einer Organisation mit Mikrodaten auszeichnet.

Bislang haben sich alle Beispiele auf die Auszeichnung sichtbarer Daten konzentriert. Das heißt, Sie haben ein `<h1>` mit einem Unternehmensnamen und fügen ihm ein `itemprop`-Attribut hinzu, um zu deklarieren, dass der (sichtbare) Überschriftentext tatsächlich der Name einer Organisation ist. Oder Sie haben ein `<img>`-Element, das auf ein Foto zeigt, und fügen ihm ein `itemprop`-Attribut hinzu, um zu deklarieren, dass das (sichtbare) Bild das Foto einer Person ist.

Geolocation-Informationen sind anderer Natur. Es gibt keinen sichtbaren Text, der den genauen Längen- und Breitengrad (auf vier Stellen genau) der Organisation angibt. Die Beispiel-Unternehmensseite ohne Mikrodaten enthält überhaupt keine Geolocation-Informationen. Sie enthält einen Link auf Google Maps, aber selbst die URL, die dieser Link enthält, enthält keine Längengrad- und Breitengradkoordinaten. (Es enthält ähnliche Informationen in einem Google-spezifischen Format.) Selbst wenn wir einen Link auf einen hypothetischen Onlinekartendienst hätten, der Längen- und Breitengrad als URL-Parameter erwartet, hätten Mikrodaten keine Möglichkeit, die verschiedenen Teile einer URL zu trennen. Sie können nicht deklarieren, dass der erste Abfrageparameter in der URL der Längengrad ist, der zweite der Breitengrad und dass alle weiteren Abfrageparameter irrelevant sind.

Um Randfälle wie diesen zu berücksichtigen, bietet HTML5 eine Möglichkeit, unsichtbare Daten auszuzeichnen. Diese Technik sollte nur als letzter Ausweg genutzt werden. Wenn es eine Möglichkeit gibt, die Daten, die Sie interessieren, anzuzeigen oder darzustellen, sollten Sie das tun. Unsichtbare Daten, die nur Maschinen lesen können, veralten sehr schnell. Das heißt, dass es sehr wahrscheinlich ist, dass später irgendjemand den sichtbaren Text ändert und vergisst, den unsichtbaren entsprechend anzupassen. Das passiert häufiger, als Sie sich vorstellen können, und es wird auch Ihnen passieren, ganz sicher.

Aber es gibt eben Fälle, in denen unsichtbare Daten unvermeidbar sind. Vielleicht will Ihr Boss unbedingt maschinenlesbare Geolocation-Daten, weigert sich aber, die Webseite mit einem Paar unverständlicher sechsstelliger Zahlen zu beflecken. Unsichtbare Daten sind dann die einzige Option. Das Einzige, was Sie hier rettet, ist, dass Sie die unsichtbaren Daten unmittelbar nach dem sichtbaren Text angeben können, der sie beschreibt. Das könnte jemanden, der später den sichtbaren Text anpasst, daran erinnern, dass auch die darauf folgenden unsichtbaren Daten angepasst werden müssen.

In diesem Beispiel können wir in das `<article>`-Element, in dem auch die anderen Eigenschaften für die Organisation stehen, ein `<span>`-Element für die unsichtbaren Geolocation-Daten stecken:

```
    <span itemprop="geo" itemscope
      itemtype="http://data-vocabulary.org/Geo">
      <meta itemprop="latitude" content="37.4149" />
      <meta itemprop="longitude" content="-122.078" />
    </span>
  </article>
```

Geolocation -Informationen werden wie die Adressen von Personen oder Organisationen in einem eigenen Vokabular definiert. Deswegen muss dieses `<span>`-Element drei Attribute haben:

`itemprop="geo"`
: Sagt, dass dieses Element die `geo`-Eigenschaft der umgebenden Organisation darstellt.

`itemtype="http://data-vocabulary.org/Geo"`
: Sagt, welchem Mikrodatenvokabular diese Eigenschaften entsprechen.

`itemscope`
: Sagt, dass dieses Element das einschließende Element für ein Mikrodatenelement mit einem eigenen Vokabular (in einem `itemtype`-Attribut) ist. Alle Eigenschaften in diesem Element sind Eigenschaften des Geo-Vokabulars (`http://data-vocabulary.org/Geo`), nicht des umgebenden Organisationsvokabulars (`http://data-vocabulary.org/Organization`).

Die nächste große Frage, die dieses Beispiel beantwortet, ist: »Wie zeichnet man unsichtbare Daten aus?« Das tun Sie mit einem `<meta>`-Element. In den vorangegangenen Versionen von HTML konnten Sie das `<meta>`-Element nur im `<head>`-Element Ihrer Seite nutzen (siehe dazu den Abschnitt »Das <head>-Element« auf Seite 36). In HTML5 könne Sie das `<meta>`-Element überall verwenden. Und genau das werden wir hier tun:

```
<meta itemprop="latitude" content="37.4149" />
```

Gemäß Tabelle 10-1 hat das `<meta>`-Element eine spezielle Verarbeitung. Der Wert der Mikrodateneigenschaft ist das Attribut `content`. Da dieses Attribut nie sichtbar angezeigt wird, haben wir genau den richtigen Rahmen für unbegrenzte Mengen unsichtbarer Daten. Aber große Macht bringt auch große Verantwortung mit sich. In diesem Fall haben Sie die Verantwortung, zu sichern, dass die unsichtbaren Daten mit dem sichtbaren Text synchron bleiben.

Es gibt keine unmittelbare Unterstützung für das Organisationsvokabular in Google Rich Snippets. Deswegen kann ich Ihnen diesmal keine ansehnlichen Suchergebnisse zeigen. Aber Organisationen spielen in unseren nächsten beiden Fallstudien, Ereignissen und Rezensionen, eine große Rolle, und die werden von Google Rich Snippets unterstützt.

Ereignisse auszeichnen

Dinge geschehen. Manche Dinge geschehen zu vorbestimmten Zeiten. Wäre es nicht nett, man könnte Suchmaschinen mitteilen, wann genau diese Dinge geschehen? Auch dafür gibt es eine Sprache.

Beginnen wir damit, einen Blick auf eine Aufstellung meiner Vorträge (*http://diveintohtml5.org/examples/event.html*) zu werfen:

```
<article>
  <h1>Google Developer Day 2009</h1>
  <img width="300" height="200"
       src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
       alt="[Mark Pilgrim at podium]">
  <p>
    Google Developer Days are a chance to learn about Google
    developer products from the engineers who built them. This
    one-day conference includes seminars and "office hours"
    on web technologies like Google Maps, OpenSocial, Android,
    AJAX APIs, Chrome, and Google Web Toolkit.
  </p>
  <p>
    <time datetime="2009-11-06T08:30+01:00">2009 November 6, 8:30</time>
      &ndash;
    <time datetime="2009-11-06T20:30+01:00">20:30</time>
  </p>
  <p>
    Congress Center<br>
    5th května 65<br>
    140 21 Praha 4<br>
    Czech Republic
  </p>
  <p><a href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
       GDD/Prague home page</a></p>
</article>
```

Die Änderungen, die wir in diesem Abschnitt vornehmen, können Sie sich online ansehen. Vorher: *http://diveintohtml5.org/examples/event.html*. Nachher: *http://diveintohtml5.org/examples/event-plus-microdata.html*.

Alle Informationen über das Ereignis sind in das `<article>`-Element eingeschlossen. Auf ihm müssen wir also die Attribute `itemtype` und `itemscope` angegeben:

```
<article itemscope itemtype="http://data-vocabulary.org/Event">
```

Die URL für das Ereignisvokabular ist *http://data-vocabulary.org/Event*. Sie enthält außerdem eine nette kleine Aufstellung der Eigenschaften des Vokabulars. Und wie lauten diese Eigenschaften? Tabelle 10-4 führt Sie auf.

Tabelle 10-4: Ereignisvokabular

Eigenschaft	Beschreibung
`summary`	Der Name des Ereignisses.
`url`	Ein Link auf die genaue Beschreibung des Ereignisses.
`location`	Der Ort oder Tagungsort des Ereignisses. Kann optional durch eine eingebettete Organisation (siehe dazu den Abschnitt »Organisationen auszeichnen« auf Seite 186) oder Adresse (siehe dazu den Abschnitt »Personen auszeichnen« auf Seite 178) dargestellt werden.
`description`	Eine Beschreibung des Ereignisses.
`startDate`	Der Anfangszeitpunkt des Ereignisses im ISO-Datumsformat.
`endDate`	Der Endzeitpunkt des Ereignisses im ISO-Datumsformat.
`duration`	Die Dauer des Ereignisses im ISO-Zeitdauerformat.
`eventType`	Die Kategorie des Ereignisses (beispielsweise »Konzert« oder »Vortrag«). Ein beliebiger String, kein enumeriertes Attribut.
`geo`	Die geografischen Koordinaten der Orts. Enthält immer zwei Untereigenschaften, `latitude` und `longitude`.
`photo`	Ein Link auf ein Foto oder Bild, das mit dem Ereignis verbunden ist.

Der Name des Ereignisses steht in einem `<h1>`-Element. Gemäß Tabelle 10-1 haben `<h1>`-Elemente keine besondere Verarbeitung. Der Wert der Mikrodateneigenschaft ist einfach der Textinhalt des `<h1>`-Elements. Wir müssen also nur das `itemprop` auf dem `<h1>`-Element deklarieren, das den Namen des Ereignisses enthält:

```
<h1 itemprop="summary">Google Developer Day 2009</h1>
```

Im Klartext sagt das: »Der Name der Ereignisses ist ›Google Developer Day 2009‹.«

Zu diesem Ereignis gibt es ein Foto, das mit der Eigenschaft `photo` ausgezeichnet werden kann. Wie zu erwarten war, steckt das Bild bereits in einem `<img>`-Element. Wie die `photo`-Eigenschaft im Personenvokabular (siehe dazu den Abschnitt »Das Mikrodaten-Datenmodell« auf Seite 175) ist auch die `photo`-Eigenschaft des Ereignisvokabulars eine URL. Da Tabelle 10-1 sagt, dass der Eigenschaftswert eines `<img>`-Element sein `src`-Attribut ist, müssen wir nur noch das entsprechende `itemprop`-Attribut ergänzen:

```
<img itemprop="photo" width="300" height="200"
     src="http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg"
     alt="[Mark Pilgrim at podium]">
```

Im Klartext sagt das: »Das Foto für dieses Ereignis befindet sich bei *http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg*.«

Darauf folgt eine längere Beschreibung des Ereignisses in Form eines gewöhnlichen Absatzes mit beliebigem Text:

```
  <p itemprop="description">Google Developer Days are a chance to
learn about Google developer products from the engineers who built
```

```
them. This one-day conference includes seminars and "office
hours" on web technologies like Google Maps, OpenSocial,
Android, AJAX APIs, Chrome, and Google Web Toolkit.</p>
```

Das folgende Element ist etwas Neues. Ereignisse beginnen und enden üblicherweise zu einem bestimmten Zeitpunkt. In HTML5 sollten Datums- und Zeitangaben mit dem `<time>`-Element ausgezeichnet werden (siehe dazu den Abschnitt »Datum und Uhrzeit« auf Seite 52). Das tun wir hier bereits. Die Frage ist also, wie wir diese `<time>`-Elemente mit Mikrodateneigenschaften versehen können. Werfen wir einen Blick zurück auf Tabelle 10-1, sehen wir, dass das `<time>`-Element eine besondere Verarbeitung hat. Der Wert der Mikrodateneigenschaft auf einem `<time>`-Element ist der Wert des `datetime`-Attributs. Und wunderbarerweise erwarten die `startDate`- und `endDate`-Eigenschaften des Ereignisvokabulars ein Datum im ISO-Stil, genau wie das `datetime`-Attribut eines `<time>`-Elements. Wieder einmal unterstützen die Regeln des Kernvokabulars von HTML sehr gut die Anforderungen unseres Mikrodatenvokabulars. Die Auszeichnung von Anfangs- und Endzeitpunkt mit Mikrodaten ist also eigentlich ganz einfach:

1. HTML sollte korrekt verwendet werden (für Datums- und Zeitwerte sollten `<time>`-Elemente verwendet werden).
2. Ihnen wird ein einziges `itemprop`-Attribut hinzugefügt:

```
<p>
  <time itemprop="startDate" datetime="2009-11-06T08:30+01:00">2009 November 6,
     8:30</time>
    &ndash;
  <time itemprop="endDate" datetime="2009-11-06T20:30+01:00">20:30</time>
</p>
```

In Klartext sagt das: »Das Ereignis beginnt um 8:30 am 6. November 2009 und dauert bis 20:30 am 6. November 2009 (lokale Zeit für Prag, GMT+1).«

Es folgt die Eigenschaft `location`. Die Definition des Vokabulars für Ereignisse sagt, dass das entweder eine Organisation oder eine Adresse ist. In diesem Fall findet das Ereignis an einem Veranstaltungsort statt, dem Kongresszentrum in Prag. Zeichnen wir das als Organisation aus, können wir sowohl den Namen als auch die Adresse des Austragungsorts angeben.

Erklären wir zunächst, dass das `<p>`-Element, das die Adresse enthält, die `location`-Eigenschaft des Ereignisses ist und dass dieses Element ein eigenständiges Mikrodatenelement ist, das dem Vokabular `http://data-vocabulary.org/Organization` angehört:

```
<p itemprop="location" itemscope
   itemtype="http://data-vocabulary.org/Organization">
```

Dann zeichnen wir den Namen der Organisation aus, indem wir ihn in ein `<span>`-Element einhüllen, dem wir das `itemprop`-Attribut hinzufügen.

```
<span itemprop="name">Congress Center</span><br>
```

`"name"` definiert eine Eigenschaft im Organisationsvokabular, nicht im Ereignisvokabular. Das `<p>`-Element definierte den Anfang des Geltungsbereichs der Eigenschaften für Organisationen, und es wurde noch nicht mit einem `</p>`-Tag geschlossen. Alle Mikrodaten-

eigenschaften, die wir hier definieren, sind Eigenschaften des zuletzt geöffneten Vokabulars. Geschachtelte Vokabulare sind wie ein Stack. Noch haben wir das oberste Element nicht von diesem Stack entfernt, reden also immer noch über die Eigenschaften der Organisation.

Und jetzt werden wir sogar noch ein drittes Vokabular auf den Stack packen – eine Adresse für die Organisation für das Ereignis:

```
<span itemprop="address" itemscope
      itemtype="http://data-vocabulary.org/Address">
```

Wieder einmal wollen wir jeden Teil der Adresse als eigene Mikrodateneigenschaft auszeichnen, benötigen also einen Haufen zusätzlicher -Elemente, an die wir unsere itemprop-Attribute hängen können – sollte ich zu schnell vorgehen, kehren Sie noch einmal zu dem Abschnitt zur Auszeichnung der Adresse von Personen (siehe dazu den Abschnitt »Personen auszeichnen« auf Seite 178) und Organisationen (siehe dazu den Abschnitt »Organisationen auszeichnen« auf Seite 186) zurück:

```
<span itemprop="street-address">5th května 65</span><br>
<span itemprop="postal-code">140 21</span>
<span itemprop="locality">Praha 4</span><br>
<span itemprop="country-name">Czech Republic</span>
```

Wir haben keine weiteren Eigenschaften für die Adresse, schließen also das -Element, das den Geltungsbereich für die Adresse öffnete, und holen das Vokabular damit vom Stack:

```
</span>
```

Da es ebenfalls keine weiteren Eigenschaften für die Organisation gibt, schließen wir auch das <p>-Element, das den Geltungsbereich für die Organisation öffnete, und holen damit auch dieses vom Stack:

```
</p>
```

Jetzt kehren wir also zur Definition der Eigenschaften des Ereignisses zurück. Die nächste Eigenschaft ist geo. Sie repräsentiert den physischen Ort des Ereignisses und nutzt das Geo-Vokabular, das wir im letzten Abschnitt zur Auszeichnung des physischen Orts einer Organisation genutzt haben. Wir müssen ein -Element hinzufügen, das als Container dient. Auf ihm geben wir die Attribute itemtype und itemscope an. In diesem -Element benötigen wir zwei <meta>-Elemente, eins für die Eigenschaft latitude und eins für die Eigenschaft longitude:

```
<span itemprop="geo" itemscope itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="50.047893" />
  <meta itemprop="longitude" content="14.4491" />
</span>
```

Weil wir das mit den Geo-Eigenschaften geschlossen haben, definieren wir jetzt wieder Eigenschaften für das Ereignis. Die letzte Eigenschaft ist die Eigenschaft url, die vertraut aussehen wird. Mit einem Ereignis verknüpft man eine URL auf gleiche Weise wie mit einer Person (siehe dazu den Abschnitt »Personen auszeichnen« auf Seite 178) oder Organisation (siehe dazu den Abschnitt »Organisationen auszeichnen« auf Seite 186). Wenn Sie HTML korrekt nutzen (also Hyperlinks mit <a href> markieren), müssen Sie nur

das Attribut `itemprop` auf dem Element angeben, um zu deklarieren, dass der Hyperlink eine Mikrodaten-`url`-Eigenschaft ist:

```
  <p>
    <a itemprop="url"
       href="http://code.google.com/intl/cs/events/developerday/2009/home.html">
      GDD/Prague home page
    </a>
  </p>
</article>
```

Die Beispiel-Ereignisseite (*http://diveintohtml5.org/examples/event.html*) führt noch ein zweites Ereignis auf, meinen Vortrag auf der ConFoo-Konferenz in Montreal. Um die Sache abzukürzen, werde ich das entsprechende Markup nicht zeilenweise durchgehen. Es ist im Wesentlichen mit dem Ereignis in Prag identisch: ein Event-Element mit eingebetteten Geo- und Address-Elementen. Das erwähne ich nur im Vorübergehen, um zu wiederholen, dass eine einzelne Seite mehrere Ereignisse enthalten kann, die jeweils mit Mikrodaten ausgezeichnet sind.

Die Rückkehr von Google Rich Snippets

Gemäß Googles Rich Snippets Testing Tool sind das die Informationen, die Googles Crawler aus unserem Beispiel für eine Ereignisliste (*http://diveintohtml5.org/examples/event-plus-microdata.html*) sammeln würde:

```
Item
    Type: http://data-vocabulary.org/Event
    summary = Google Developer Day 2009
    eventType = conference
    photo = http://diveintohtml5.org/examples/gdd-2009-prague-pilgrim.jpg
    description = Google Developer Days are a chance to learn about Google developer
                  products from the engineers who built them. This one-day
                  conference includes seminars and office hours on web technologies
                  like Goo...
    startDate = 2009-11-06T08:30+01:00
    endDate = 2009-11-06T20:30+01:00
    location = Item(__1)
    geo = Item(__3)
    url = http://code.google.com/intl/cs/events/developerday/2009/home.html

Item
    Id: __1
    Type: http://data-vocabulary.org/Organization
    name = Congress Center
    address = Item(__2)

Item
    Id: __2
    Type: http://data-vocabulary.org/Address
    street-address = 5th května 65
    postal-code = 140 21
    locality = Praha 4
    country-name = Czech Republic
```

```
Item
    Id: __3
    Type: http://data-vocabulary.org/Geo
    latitude = 50.047893
    longitude = 14.4491
```

Wie Sie sehen, sind alle Daten, die wir in Mikrodaten eingefügt haben, da. Eigenschaften, die eigene Mikrodatenelemente sind, erhalten interne IDs (`Item(__1)`, `Item(__2)` und so weiter), aber das ist nicht Teil der Mikrodatenspezifikation. Das ist lediglich eine Konvention, die Googles Test-Werkzeug nutzt, um die Beispielausgabe zu linearisieren und Ihnen die Gruppierung der eingebetteten Elemente und ihre Eigenschaften zu zeigen.

Wie also könnte Google diese Beispielseite in seinen Suchergebnissen repräsentieren? (Auch hier muss ich voranstellen, dass das nur ein Beispiel ist. Google kann das Format seiner Suchergebnisse jederzeit ändern, und es gibt auch keinerlei Garantie, dass Google überhaupt auf Ihre Mikrodaten achtet.) Es könnte aber aussehen wie in Abbildung 10-2.

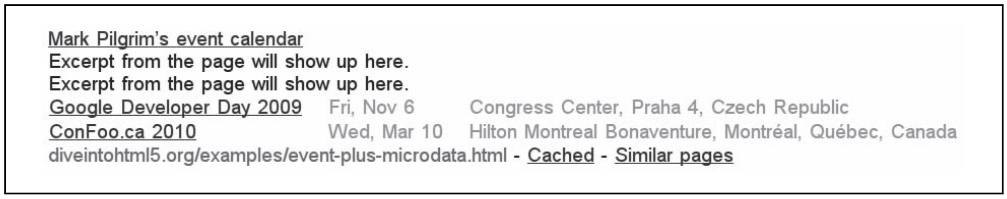


Abbildung 10-2: Suchergebnisse für eine mit Mikrodaten angereicherte Aufstellung von Ereignissen

Nach der Titelseite und einem automatisch generierten Auszug beginnt Google, das von uns der Seite hinzugefügte Markup zu verwenden, um eine kleine Tabelle mit Ereignissen anzuzeigen. Beachten Sie das Datumsformat: »Fri, Nov 6«. Das ist kein String, der an irgendeiner Stelle unseres HTML oder unseres Mikrodaten-Markups erschien. Wir haben zwei vollständige, gemäß ISO formatierte Strings angegeben, `2009-11-06T08:30+01:00` und `2009-11-06T20:30+01:00`. Google nahm diese Daten, ermittelte, dass sie am gleichen Tag liegen, und beschloss, sie als nur ein Datum in einem freundlichen Format anzuzeigen.

Schauen Sie sich die physischen Adressen an. Google entschied, nur Name, Ort und Land des Veranstaltungsorts anzuzeigen, nicht die vollständige Straßenadresse. Das wird dadurch ermöglicht, dass wir die Adresse in fünf Untereigenschaften aufgeteilt haben – `name`, `street-address`, `region`, `locality` und `country-name` – und jeden Teil der Adresse als eine andere Mikrodateneigenschaft identifiziert haben. Google nutzt das, um eine verkürzte Adresse anzuzeigen. Andere Verbraucher des gleichen Mikrodaten-Markups könnten andere Entscheidungen in Bezug auf die Auswahl und die Darstellung der Mikrodateneigenschaften wählen. Richtig oder falsch gibt es hier nicht. Ihre Aufgabe ist, so viele Daten und so genaue Daten wie möglich zu liefern. Wie diese interpretiert werden, bleibt dem Rest der Welt überlassen.

Rezensionen auszeichnen

Hier ist ein weiteres Beispiel dafür, wie man das Web (und wahrscheinlich Suchergebnisse) mit Markup verbessern kann: Unternehmens- und Produktrezensionen.

Dies ist eine kurze Rezension, die ich über meine Lieblingspizzeria in der Nähe meines Hauses geschrieben habe. (Nebenbei: Dieses Restaurant gibt es wirklich. Sollten Sie je in Apex, NC, sein, kann ich es Ihnen nur empfehlen.) Schauen wir uns das ursprüngliche Markup an (*http://diveintohtml5.org/examples/review.html*):

```
<article>
  <h1>Anna's Pizzeria</h1>
  <p>★★★★☆ (4 stars out of 5)</p>
  <p>New York-style pizza right in historic downtown Apex</p>
  <p>
    Food is top-notch. Atmosphere is just right for a "neighborhood
    pizza joint." The restaurant itself is a bit cramped; if you're
    overweight, you may have difficulty getting in and out of your
    seat and navigating between other tables. Used to give free
    garlic knots when you sat down; now they give you plain bread
    and you have to pay for the good stuff. Overall, it's a winner.
  </p>
  <p>
    100 North Salem Street<br>
    Apex, NC 27502<br>
    USA
  </p>
  <p>– reviewed by Mark Pilgrim, last updated March 31, 2010</p>
</article>
```

Die Änderungen, die wir in diesem Abschnitt vornehmen werden, können Sie online nachverfolgen. Vorher: *http://diveintohtml5.org/examples/review.html*. Nachher: *http://diveintohtml5.org/examples/review-plus-microdata.html*.

Diese Rezension steckt in einem `<article>`-Element. Auf diesem werden wir also die Attribute `itemtype` und `itemscope` angeben. Hier ist die Namensraum-URL für dieses Vokabular:

```
<article itemscope itemtype="http://data-vocabulary.org/Review">
```

Welche Eigenschaften stellt dieses Vokabular für Rezensionen bereit? Gut, dass Sie fragen. Sie werden in Tabelle 10-5 aufgeführt.

Tabelle 10-5: Vokabular für Rezensionen

Eigenschaft	Beschreibung
`itemreviewed`	Der Name der Einheit, die rezensiert wird. Kann eine Ware, eine Dienstleistung, ein Unternehmen usw. sein.
`rating`	Eine numerische Qualitätseinstufung für die Einheit auf einer Skala von 1 bis 5. Kann auch eine eingebettete Bewertung unter Verwendung des Vokabulars `http://data-vocabulary.org/Rating` sein, wenn eine andere Skala verwendet werden soll.
`reviewer`	Der Name des Autors der Rezension.

Tabelle 10-5: Vokabular für Rezensionen (Fortsetzung)

Eigenschaft	Beschreibung
dtreviewed	Das Datum, an dem die Einheit rezensiert wurde, im ISO-Datumsformat.
summary	Eine kurze Zusammenfassung der Rezension.
description	Der eigentliche Inhalt der Rezension.

Die erste Eigenschaft ist simpel: itemreviewed ist einfach Text, und hier steht er in einem <h1>-Element. Auf diesem sollten wir also auch das itemprop-Attribut angeben:

```
<h1 itemprop="itemreviewed">Anna's Pizzeria</h1>
```

Die eigentliche Bewertung werde ich jetzt überspringen, um erst später zu ihr zurückzukehren.

Die nächsten beiden Eigenschaften sind ebenfalls kein Problem. Die Eigenschaft summary ist eine kurze Beschreibung dessen, was rezensiert wird, und die Eigenschaft description ist der Inhalt der Rezension:

```
<p itemprop="summary">New York-style pizza right in historic downtown Apex</p>
<p itemprop="description">
  Food is top-notch. Atmosphere is just right for a "neighborhood
  pizza joint." The restaurant itself is a bit cramped; if you're
  overweight, you may have difficulty getting in and out of your
  seat and navigating between other tables. Used to give free
  garlic knots when you sat down; now they give you plain bread
  and you have to pay for the good stuff. Overall, it's a winner.
</p>
```

Auch die Eigenschaften location und geo haben wir bereits behandelt (werfen Sie gegebenenfalls noch einmal einen Blick in die vorangegangenen Abschnitte zum Auszeichnen der Adressen von Personen und Unternehmen sowie zum Auszeichnen von Geolocation-Daten):

```
<p itemprop="location" itemscope
   itemtype="http://data-vocabulary.org/Address">
  <span itemprop="street-address">100 North Salem Street</span><br>
  <span itemprop="locality">Apex</span>,
  <span itemprop="region">NC</span>
  <span itemprop="postal-code">27502</span><br>
  <span itemprop="country-name">USA</span>
</p>
<span itemprop="geo" itemscope
      itemtype="http://data-vocabulary.org/Geo">
  <meta itemprop="latitude" content="35.730796" />
  <meta itemprop="longitude" content="-78.851426" />
</span>
```

Die letzte Zeile präsentiert das übliche Problem: Sie enthält zwei Informationsbestandteile in einem Element. Der Name des Rezensenten ist Mark Pilgrim, und das Datum der Rezension ist March 31, 2010. Wie zeichnen wir diese beiden eigenständigen Eigenschaften aus? Wie üblich können wir sie in eigene Elemente packen (siehe dazu den Abschnitt »Personen auszeichnen« auf Seite 178) und jedes davon mit einem itemprop-Attribut

versehen. Das Datum in diesem Beispiel hätte eigentlich von Anfang an als `<time>`-Element ausgezeichnet werden müssen. Das ist also ein ganz natürlicher Anker für unser `itemprop`-Attribut. Der Name des Rezensenten kann einfach in ein zusätzliches `<span>` gepackt werden:

```
  <p>
     <span itemprop="reviewer">Mark Pilgrim</span>, last updated
     <time itemprop="dtreviewed" datetime="2010-03-31">
       March 31, 2010
     </time>
  </p>
</article>
```

Okay. Wenden wir uns jetzt den Bewertungen zu. Das Komplizierteste bei der Auszeichnung einer Rezension ist die Bewertung. Standardmäßig befinden sich die Bewertungen im Rezensionsvokabular auf einer Skala von 1 bis 5, auf der 1 »entsetzlich« und 5 »umwerfend« heißt. Wenn Sie eine andere Skala nutzen wollen, können Sie das tun. Aber schauen wir uns erst die Standardskala an:

```
<p>★★★★☆ (<span itemprop="rating">4</span> stars out of 5)</p>
```

Nutzen Sie die Standardskala von 1 bis 5, müssen Sie nur eine einzige Eigenschaft auszeichnen: die Bewertung selbst (in diesem Fall 4). Aber was ist, wenn Sie eine andere Skala nutzen wollen? Das ist möglich. Sie müssen einfach die Grenzen der Skala deklarieren, die Sie nutzen. Wollen Sie beispielsweise eine Skala von 1 bis 10 nutzen, würden Sie immer noch die Eigenschaft `itemprop="rating"` deklarieren, den Wert der Bewertung aber nicht mehr direkt angeben, sondern ein eingebettetes Element aus dem Vokabular für Bewertungen unter `http://data-vocabulary.org/Rating` nutzen, um den besten und den schlechtesten Wert für die Skala sowie den eigentlichen Bewertungswert auf dieser Skala anzugeben:

```
<p itemprop="rating" itemscope
   itemtype="http://data-vocabulary.org/Rating">
  ★★★★★★★★★☆
  (<span itemprop="value">9</span> auf einer Skala von
   <span itemprop="worst">0</span> bis
   <span itemprop="best">10</span>)
</p>
```

Im Klartext heißt das: »Das Produkt, das ich hier rezensiere, wird auf einer Skala von 1 bis 10 mit der Note 9 bewertet.«

Hatte ich gesagt, dass sich Rezensionsmikrodaten auf Suchergebnisse auswirken können? Sie können es. Hier sind die »Rohdaten«, die das Google Rich Snippets-Werkzeug aus meiner mit Mikrodaten angereicherten Rezension herauszieht (*http://www.google.com/webmasters/tools/richsnippets?url=//diveintohtml5.org/examples/review-plus-microdata.html*):

```
Item
  Type: http://data-vocabulary.org/Review
  itemreviewed = Anna's Pizzeria
  rating = 4
  summary = New York-style pizza right in historic downtown Apex
  description = Food is top-notch. Atmosphere is just right ...
```

```
  address = Item(__1)
  geo = Item(__2)
  reviewer = Mark Pilgrim
  dtreviewed = 2010-03-31

Item
  Id: __1
  Type: http://data-vocabulary.org/Organization
  street-address = 100 North Salem Street
  locality = Apex
  region = NC
  postal-code = 27502
  country-name = USA

Item
  Id: __2
  Type: http://data-vocabulary.org/Geo
  latitude = 35.730796
  longitude = -78.851426
```

Abbildung 10-3 (ohne die Launen von Google, die Mondphasen und so weiter) zeigt, wie meine Rezension in einer Auflistungen von Suchergebnissen aussehen könnte.

Abbildung 10-3: Beispielsuchergebnisse für eine mikrodatenbereicherte Rezension

Attribute finde ich eigentlich nicht sonderlich prickelnd, aber ich muss zugeben, dass das hier ziemlich cool ist.

Weitere Lektüre

Mikrodatenressourcen:

- Live-Mikrodaten-Spielplatz (*http://foolip.org/microdatajs/live/*)
- HTML5-Mikrodatenspezifikation (*http://bit.ly/ckt9Rj*)

Ressourcen zu Google Rich Snippets:

- »About rich snippets and structured data« (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=99170*)
- »People« (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146646*), Personenvokabular
- »Businesses and organizations«, Vokabular für Unternehmen und Organisationen (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861*)

- »Events« (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=164506*), Ereignisvokabular
- »Reviews« (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146645*), Rezensionsvokabular
- »Review ratings« (*http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=172705*), Rezensionsbewertungsvokabular
- Google Rich Snippets-Testwerkzeug (*http://www.google.com/webmasters/tools/richsnippets*)
- Tipps und Tricks zu Google Rich Snippets (*http://knol.google.com/k/google-rich-snippets-tips-and-tricks*)

Die erschöpfende und fast alphabetische Referenz der Unterstützungserkennung

Verwirrt? In Kapitel 2 finden Sie eine Einführung in das Thema. Sie hätten lieber eine Bibliothek? Schauen Sie sich Modernizr (*http://www.modernizr.com*) an.

Liste der Elemente

<audio>

http://bit.ly/cZxI7K

```
return !!document.createElement('audio').canPlayType;
```

<audio> im MP3-Format

http://en.wikipedia.org/wiki/MP3

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mpeg;').replace(/no/, ''));
```

<audio> im Vorbis-Format

http://en.wikipedia.org/wiki/Vorbis

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/ogg;
          codecs="vorbis"').replace(/no/, ''));
```

<audio> im WAV-Format

http://en.wikipedia.org/wiki/WAV

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/wav;
          codecs="1"').replace(/no/, ''));
```

<audio> im AAC-Format

http://en.wikipedia.org/wiki/Advanced_Audio_Coding

```
var a = document.createElement('audio');
return !!(a.canPlayType && a.canPlayType('audio/mp4;
          codecs="mp4a.40.2"').replace(/no/, ''));
```

<canvas>

Siehe dazu Kapitel 4

```
return !!document.createElement('canvas').getContext;
```

<canvas>-Text-API

Siehe dazu den Abschnitt »Text« auf Seite 66

```
var c = document.createElement('canvas');
return c.getContext && typeof c.getContext('2d').fillText == 'function';
```

<command>

http://bit.ly/aQt2Fn

```
return 'type' in document.createElement('command');
```

<datalist>

http://bit.ly/9WVz5p

```
return 'options' in document.createElement('datalist');
```

<details>

http://bit.ly/c08mQy

```
return 'open' in document.createElement('details');
```

<device>

http://bit.ly/aaBeUy

```
return 'type' in document.createElement('device');
```

<form>-Bedingungsvalidierung

http://bit.ly/cb9Wmj

```
return 'noValidate' in document.createElement('form');
```

<iframe sandbox>

http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox

```
return 'sandbox' in document.createElement('iframe');
```

<iframe srcdoc>

http://blog.whatwg.org/whats-next-in-html-episode-2-sandbox

```
return 'srcdoc' in document.createElement('iframe');
```

<input autofocus>

Siehe dazu den Abschnitt »Autofokusfelder« auf Seite 158

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

<input placeholder>

Siehe dazu den Abschnitt »Platzhaltertext« auf Seite 157

```
return 'placeholder' in document.createElement('input');
```

<input type="color">

http://bit.ly/9HkeNn

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

<input type="email">

Siehe dazu den Abschnitt »E-Mail-Adressen« auf Seite 160

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

<input type="number">

Siehe dazu den Abschnitt »Zahlen als Spinboxen« auf Seite 163

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

<input type="range">

Siehe dazu den Abschnitt »Zahlen als Schieberegler« auf Seite 166

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

<input type="search">

Siehe dazu den Abschnitt »Suchfelder« auf Seite 169

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

<input type="tel">

http://bit.ly/bZm0Q5

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

<input type="url">

Siehe dazu den Abschnitt »Webadressen« auf Seite 162

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

<input type="date">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

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

<input type="time">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

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

<input type="datetime">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

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

<input type="datetime-local">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

```
var i = document.createElement('input');
i.setAttribute('type', 'datetime-local);
return i.type !== 'text';
```

<input type="month">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

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

<input type="week">

Siehe dazu den Abschnitt »Datumswähler« auf Seite 167

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

<meter>

http://bit.ly/c0pX0l

```
return 'value' in document.createElement('meter');
```

<output>

http://bit.ly/asJaqH

```
return 'value' in document.createElement('output');
```

<progress>

http://bit.ly/bjDMy6

```
return 'value' in document.createElement('progress');
```

<time>

http://bit.ly/bI62jp

```
return 'valueAsDate' in document.createElement('time');
```

<video>

Siehe dazu Kapitel 5

```
return !!document.createElement('video').canPlayType;
```

<video>-Überschriften

http://bit.ly/9mLiRr

```
return 'track' in document.createElement('track');
```

<video poster>

http://bit.ly/b6RhzT

```
return 'poster' in document.createElement('video');
```

<video> im WebM-Format

http://www.webmproject.org

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/webm; codecs="vp8,
          vorbis"').replace(/no/, ''));
```

<video> im H.264-Format

Siehe dazu den Abschnitt »H.264« auf Seite 86

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/mp4; codecs="avc1.42E01E,
          mp4a.40.2"').replace(/no/, ''));
```

<video> im Theora-Format

Siehe dazu den Abschnitt »Theora« auf Seite 87

```
var v = document.createElement('video');
return !!(v.canPlayType && v.canPlayType('video/ogg; codecs="theora,
          vorbis"').replace(/no/, ''));
```

contentEditable

http://bit.ly/aLivbS

```
return 'isContentEditable' in document.createElement('span');
```

Dokumentübergreifende Nachrichten

http://bit.ly/cUOqXd

```
return !!window.postMessage;
```

Drag-and-Drop

http://bit.ly/aN0RFQ

```
return 'draggable' in document.createElement('span');
```

File-API

http://dev.w3.org/2006/webapi/FileAPI/

```
return typeof FileReader != 'undefined';
```

Geolocation

Siehe dazu Kapitel 6

```
return !!navigator.geolocation;
```

History

http://bit.ly/9JGAGB

```
return !!(window.history && window.history.pushState &&
window.history.popState);
```

Lokaler Speicher

http://dev.w3.org/html5/webstorage/

```
return ('localStorage' in window) && window['localStorage'] !== null;
```

Mikrodaten

http://bit.ly/dBGnqr

```
return !!document.getItems;
```

Offline-Webanwendungen

Siehe dazu Kapitel 8

```
return !!window.applicationCache;
```

Serverseitige Events

http://dev.w3.org/html5/eventsource/

```
return typeof EventSource !== 'undefined';
```

Sitzungsspeicher

http://dev.w3.org/html5/webstorage/

```
try {
  return ('sessionStorage' in window) && window['sessionStorage'] !== null;
} catch(e) {
  return false;
}
```

SVG

http://www.w3.org/TR/SVG/

```
return !!(document.createElementNS && document.createElementNS
('http://www.w3.org/2000/svg', 'svg').createSVGRect);
```

SVG in text/html

http://hacks.mozilla.org/2010/05/firefox-4-the-html5-parser-inline-svg-speed-and-more/

```
var e = document.createElement('div');
e.innerHTML = '<svg></svg>';
return !!(window.SVGSVGElement && e.firstChild instanceof window.SVGSVGElement);
```

WebSimpleDB

http://dev.w3.org/2006/webapi/WebSimpleDB/

```
return !!window.indexedDB;
```

Web-Sockets

http://dev.w3.org/html5/websockets/

```
return !!window.WebSocket;
```

Web-SQL-Datenbank

http://dev.w3.org/html5/webdatabase/

```
return !!window.openDatabase;
```

Web Workers

http://bit.ly/9jheof

```
return !!window.Worker;
```

Undo

http://bit.ly/bs6JFR

```
return typeof UndoManager !== 'undefined';
```

Weitere Lektüre

Spezifikationen und Standards:

- HTML5 (*http://bit.ly/bYiOQp*)
- Geolocation (*http://www.w3.org/TR/geolocation-API/*)
- Serverseitige Events (*http://dev.w3.org/html5/eventsource/*)
- WebSimpleDB (*http://dev.w3.org/2006/webapi/WebSimpleDB/*)
- Web Sockets (*http://dev.w3.org/html5/websockets/*)
- Web-SQL-Datenbanken (*http://dev.w3.org/html5/webdatabase/*)
- Web-Speicherung (*http://dev.w3.org/html5/webstorage/*)
- Web Workers (*http://bit.ly/9jheof*)

JavaScript-Bibliotheken:

- Modernizr (*http://www.modernizr.com*), eine HTML5-Erkennungsbibliothek

Index

D

E

F

G

H

I

J

K

L

M

N

O

X

Z

Über den Autor

Mark Pilgrim ist Senior Developer Advocate bei Google und hat sich auf Open Source- und andere freie Standards spezialisiert. Er ist Autor verschiedener technischer Bücher, darunter *Dive Into Python* (APress) und *Dive Into Accessability*, ein freies Online-Tutorial über Barrierefreiheit im Web. Mark lebt zusammen mit seiner Frau, zwei Söhnen und einem Hund in North Carolina.

Über den Übersetzer

Lars Schulten ist freier Übersetzer für IT-Fachliteratur und hat für den O'Reilly Verlag schon unzählige Bücher zu ungefähr allem übersetzt, was man mit Computern so anstellen kann. Eigentlich hat er mal Philosophie studiert, aber mit Computern schlägt er sich schon seit den Zeiten herum, da Windows laufen lernte. Die Liste der Dinge, mit denen er sich beschäftigt, ist ungefähr so lang, launenhaft und heterogen wie die seiner Lieblingsessen und Lieblingsbücher.

Kolophon

Das Tier auf dem Cover von *Durchstarten mit HTML5* ist eine Gämse (*Rupicapra rupicapra*), eine ziegen- oder antilopenähnliche Tierart, die in den Bergregionen Europas lebt – unter anderem in den Karparten, im Apennin, in der Tatra, im Balkangebirge, im Kaukasus und in den Alpen. Sogar in Neuseeland lebt eine Gämsenpopulation, nachdem sie 1907 vom österreichischen Kaiser Franz Joseph I. dort angesiedelt wurde.

Gämsen leben in relativ großer Höhe und haben sich ideal an steiles, gerölliges Terrain angepasst. Sie erreichen eine Schulterhöhe von bis zu 75 Zentimetern und wiegen zwischen 20 und 30 Kilogramm (obgleich die Tiere in Neuseeland häufig rund 20 Prozent weniger wiegen als ihre europäischen Artgenossen). Beide Geschlechter tragen kurze Hörner, die sich zur Spitze hin nach hinten krümmen. Im Sommer ist ihr Fell dunkelbraun, im Winter hellgrau. Weitere typische Kennzeichen, die viele Gämsen aufweisen, sind ein weiß gezeichnetes Gesicht mit einem kräftigen schwarzen Streifen unterhalb der Augen, der sich bis zum Rücken zieht.

Ausgewachsene männliche Gämsen leben meist als Einzelgänger und finden sich nur einmal im Jahr zusammen, um gegeneinander zu kämpfen und so die Gunst der paarungsbereiten Weibchen zu erwerben. Die weiblichen Tiere hingegen leben mit den Jungtieren in Herden.

Schon immer wurden Gämsen bejagt: Ihr Fleisch ist besonders schmackhaft, und ihr Leder ist bekannt dafür, dass es weich und saugfähig ist. Aus den Haaren des Widerrists wird der sogenannte Gamsbart gemacht, ein traditioneller Hutschmuck der Alpenländer. Heutzutage ist die Gämsenjagd stark reguliert, da ihr Lebensraum immer mehr vom Menschen beansprucht wird. Dagegen fördert die neuseeländische Regierung die Jagd auf die Gämse, da sie sich in Ermangelung von Fressfeinden stark ausgebreitet hat und eine Gefahr für die heimische Flora und Fauna darstellt.

Das Titelbild stammt aus J. G. Woods *Animate Creation*. Die Schrift auf der Umschlagseite ist Adobe ITC Garamond. Die Textschrift ist Linotype Birka, die Überschriftenschrift Adobe Myriad Condensed und die Codeschrift LucasFont TheSansMonoCondensed.