HTML5

Stand: 29.11.2022

Canvas

Animationen

Anders als SVG- oder SMIL-Animationen sind **Canvas-Animationen** reine Handarbeit. Ingredienzen dafür sind eine Funktion zum Zeichnen plus ein Timer, der diese in regelmäßigen Abständen aufruft. Für den Zeitgeber stellt JavaScript `window.setInterval()` zur Verfügung, der Rest ist dem Einfallsreichtum des Canvas-Programmierers überlassen.

Animation farbiger Kugeln
Unsere Animations-Premiere besteht aus Kugeln in verschiedenen Farben, die an zufälligen Positionen der Canvas-Fläche erscheinen, langsam verblassen und währenddessen durch neue Kugeln überdeckt werden. Das Tempo der Animation soll mit circa 60 Schlägen pro Minute dem Ruhepuls eines Erwachsenen entsprechen, und als Zusatz-Feature soll bei jedem Klick auf den Canvas die Animation gestoppt beziehungsweise neu gestartet werden.

Knapp 50 Zeilen JavaScript-Code reichen dafür aus, doch bevor wir uns näher mit der Analyse des Cods beschäftigen, sehen wir uns gleich einen statischen Screenshot des Ergebnisses an.

```
var canvas = document.querySelector("canvas");
var context = canvas.getContext('2d');
var r,cx,cy,radgrad;
var drawCircles = function() {

// Bestehendes verblassen
context.fillStyle = 'rgba(255,255,255,0.5)';
context.fillRect(0, 0,canvas.width,canvas.height);

// neue Kugeln zeichnen
for (var i=0; i<360; i+=15) {

// Zufallsposition und Größe
cx = Math.random()*canvas.width;
cy = Math.random()*canvas.height;
r = Math.random()*canvas.width/10.0;

// radiale Gradiente mit Lichtpunkt
```

```
radgrad = context.createRadialGradient(
0+(r* 0.15),0-(r* 0.25),r/3.0,
0,0,r );
radgrad.addColorStop(0.0,'hsl('+i+',100%,75%)');
radgrad.addColorStop(0.9,'hsl('+i+',100%,50%)');
radgrad.addColorStop(1.0,'rgba(0,0,0,0)');

// Kugel zeichnen
context.save();
context.translate(cx,cy);
context.beginPath();
context.moveTo(0+r,0);
context.arc(0,0,r,0,Math.PI*2.0,0);
context.fillStyle = radgrad;
context.fill();
context.restore();
}
};
drawCircles();  // erstes Set an Kugeln zeichnen

// Animation mit Puls-Tempo starten/stoppen
var pulse = 60;
var running = null;
canvas.onclick = function() {
if (running) {
window.clearInterval(running);
running = null;
}
else {
running = window.setInterval(
"drawCircles()",60000/pulse
);
}
};
```

Nachdem canvas, context und ein paar weitere **Variablen** vordefiniert sind, beginnt in der Funktion `drawCircles()` die eigentliche Arbeit. Ein semitransparentes, weißes Rechteck verblasst bestehenden Inhalt aus früheren `drawCircles()`-Aufrufen, bevor die `for`-Schleife für das Zeichnen neuer Kugeln sorgt. Die Position jeder Kugel sowie deren Radius wird mithilfe von `Math.random()` wieder zufällig berechnet, wodurch das Zentrum in jedem Fall innerhalb der Canvas-Fläche zu liegen kommt und der Radius ein Zehntel der Canvas-Breite nicht übersteigt.

Den Kugeleffekt erzielen wir durch eine radiale Gradiente, deren Geometrie aus einem Lichtpunkt im oberen rechten Bereich und dem Gesamtkreis besteht. Die Wahl des Inkrements der `for`-Schleife spiegelt den Wunsch nach Farben im HSL-Farbraum für die `colorStops` der Gradiente wider. Bei jedem Durchlauf

erhöht sich der Farbwinkel um 15° und bedingt damit den Farbwechsel von Rot über Grün und Blau zurück nach Rot.

Über die Helligkeit lassen sich dann jeweils zwei zusammenpassende Farben ableiten:

Die erste repräsentiert den Lichtpunkt, die zweite die dunklere Farbe am Kugelrand. Der dritte Aufruf von `addColorStop()` bewirkt, dass direkt am Kugelrand zu transparentem Schwarz ausgeblendet wird. Insgesamt werden auf diese Weise 24 Kreise erzeugt, deren Farbpaare zum besseren Verständnis in Abbildung 2 dargestellt sind.

Daran anschließend wird die Kugel schließlich als Kreis mit dem definierten Farbverlauf gezeichnet. Das Einbetten in `context.save()` und `context.restore()` stellt sicher, dass sich die temporäre Verschiebung mit `translate()` nicht auf nachfolgende Kreise überträgt. Damit ist die Funktion `drawCircles()` komplett, und wir können ein erstes Set an Kugeln zeichnen und uns dann dem Timer widmen.

Knapp fünfzehn Zeilen reichen aus, um das Starten und Stoppen der Animation über einen `onclick`-EventListener zu implementieren. Beim ersten Klick auf den Canvas starten wir die Animation mit `window.setInterval()` und speichern die eindeutige Intervall-ID in der Variablen running. Da Zeitangaben bei `window.setInterval()` in Millisekunden erfolgen, müssen wir natürlich die Schläge pro Minute in der Variablen pulse entsprechend umwandeln.

Läuft die Animation, ist running beim nächsten Klick mit der eindeutigen Intervall-ID belegt, und wir können sie über `window.clearInterval(running)` unterbrechen. Setzen wir running dann wieder auf null, signalisiert der nächste Klick auf den Canvas: Keine Animation läuft. In diesem Fall starten wir wieder, und das Spiel beginnt von Neuem.

Video abspielen mit `drawImage()`

Wie Sie bereits aus Bilder einbetten wissen, kann bei `drawImage()` auch ein **HTMLVideoElement** als Quelle verwendet werden. Wer allerdings hofft, dass sich derart eingebundene Videos automatisch abspielen lassen, wird enttäuscht, denn die Logik dafür muss zur Gänze in JavaScript implementiert werden. Dass das gar nicht so schwer ist, zeigt das letzte Canvas-Animationsbeispiel, eine Erweiterung unserer Yosemite-Nationalpark-Postkarte aus Abbildung 3.

Anstelle des statischen Ausschnitts von El Capitan platzieren wir nun in der rechten oberen Ecke ein dynamisches Video mit einem 360°-Panoramaschwenk, der vom Taft Point aus gefilmt wurde. Zusätzlich kopieren wir während des Abspielens 10 verkleinerte Schnappschüsse des laufenden Videos als Leiste in den unteren Bereich des Canvas. Nach Beendigung des Videos ergibt sich das Erscheinungsbild aus Abbildung 4.

Im Gegensatz zu Bildern, die bisher immer über die JavaScript-Methode `new Image()` ihren Weg in den Canvas gefunden haben, bauen wir den Schwenk als `video`-Element direkt in der HTML-Seite ein. Als Zusatzattribute benötigen wir `preload` zum Vorladen, `oncanplay` als Event-Listener, der uns den Zeitpunkt liefert, zu dem wir die Postkarte layouten und das Starten und Stoppen vorbereiten können, sowie eine `style`-Anweisung zum Verstecken des eingebetteten Originalvideos. Dieses benutzen wir nur dazu, um während des Abspielens in kurzen Intervallen den aktuellen Video-Frame auf den Canvas zu kopieren. Der alternative Text für Browser ohne Videounterstützung weist noch kurz auf den Inhalt des Videos hin.

```
<video src="videos/yosemite_320x240.webm" preload="auto" oncanplay="init(event)"
style="display:none;">
Rundblick vom Taft Point aus ins Yosemite Valley
</video>
```

Um sicherzustellen, dass die Funktion `init(event)` beim Verweis im `oncanplay`-Attribut auch tatsächlich existiert, setzen wir das `script`-Element vor unser `video`-Element. Der schematische Aufbau dieser zentralen Funktion, die sowohl das Layout als auch die Funktionalität der Videopostkarte implementiert, ergibt sich damit folgendermaßen:

```
var init = function(evt) {
// Referenz zum video-Element speichern
// Hintergrundbild erzeugen
image.onload = function() {

// Hintergrundbild zeichnen
// Titel hinzufügen
// ersten Frame zeichnen
canvas.onclick = function() {

// Starten und Stoppen implementieren
// beim Abspielen Video-Frames kopieren
// beim Abspielen regelmäßig Icons erzeugen
};
}
```

```
};
```

Die Referenz zum Video-Objekt des `video`-Elements finden wir in `evt.target` und speichern sie in der Variablen video. Ein neues Hintergrundbild erzeugen wir, wie schon so oft, über `new Image()`, und sobald das Bild vollständig geladen ist, geht es weiter ans Zeichnen von Hintergrund und Titel. Die Schritte bis hierher müssen wir wohl nicht mehr näher erklären, das Zeichnen des ersten Frames vielleicht schon.

```
context.setTransform(1,0,0,1,860,20);
context.drawImage(video,0,0,320,240);
context.strokeRect(0,0,320,240);
```

Zuerst positionieren wir durch `setTransform()` das Koordinatensystem in der rechten oberen Ecke und zeichnen dann über `drawImage()` den ersten Frame mit Randlinie. Diesen Vorgang werden wir später beim Abspielen ständig wiederholen, wobei entscheidend ist, dass das HTMLVideoElement `video` der `drawImage()`-Methode immer das Bild des aktuellen **Frames** bereitstellt.

Das Stoppen, Starten und danach das Kopieren der aktuellen Frames des im Hintergrund laufenden Originalvideos auf den Canvas sowie das Erzeugen von verkleinerten Bildausschnitten implementieren wir beim Klicken auf den Canvas in der Funktion `canvas.onclick()`:

```
var running = null;
canvas.onclick = function() {
if (running) {
video.pause();
window.clearInterval(running);
running = null;
}
else {
var gap = video.duration/10;
video.play();
running = window.setInterval(function () {
if (video.currentTime < video.duration) {

// update video
context.setTransform(1,0,0,1,860,20);
context.drawImage(video,0,0,320,240);
context.strokeRect(0,0,320,240);

// update icons
var x1 = Math.floor(video.currentTime/gap)*107;
var tx = Math.floor(video.currentTime/gap)*5;
context.setTransform(1,0,0,1,10+tx,710);
context.drawImage(video,x1,0,107,80);
```

```
context.strokeRect(x1,0,107,80);
}
else {
window.clearInterval(running);
running = null;
} },35);
}
};
```

Wie im ersten Animationsbeispiel enthält die Variable running die eindeutige **Intervall-ID** von `window.setInterval()` und erlaubt die Steuerung der Animation. Ist running belegt, stoppen wir das versteckte Video mit `video.pause()`, beenden durch Entfernen des Intervalls das Kopieren von Frames und setzen running wieder auf null.

Im Gegenzug starten wir beim ersten oder nächsten Klick das Video mit `video.play()` und kopieren in der Callback-Funktion des Intervalls alle 35 Millisekunden den aktuellen Video-Frame auf den Canvas. Das Ganze setzen wir so lange fort, bis das Video zu Ende ist oder ein weiterer Klick auf den Canvas erfolgt. Bei der Überprüfung, ob die aktuelle Abspielposition noch kleiner als die Gesamtdauer des Videos ist, helfen uns die beiden Attribute `video.currentTime` und `video.duration` des Video-Objekts in der Variablen video.

Das Zeichnen des kopierten Videos rechts oben geschieht analog zum Zeichnen des ersten Frames. Für die Leiste mit verkleinerten Schnappschüssen ermitteln wir hingegen aus der Gesamtlänge des Videos und der gewünschten Anzahl an Ausschnitten jenes Zeitintervall `gap`, nach dem wir den Anfasspunkt x1 mit einer kleinen Lücke tx weiter nach rechts verschieben müssen. Solange x1 denselben Wert besitzt, läuft die Animation auch beim verkleinerten Ausschnitt mit. Wird x1 nach rechts verschoben, bleibt der letzte Frame statisch zurück und die Animation läuft an der neuen Stelle weiter. Nach zirka 40 Sekunden Laufzeit ist das Video beendet, zehn verkleinerte Ausschnitte sind gezeichnet, und wir können die Sequenz durch einen Klick auf den Canvas neu starten.

Bilder einbetten

Zum Einbetten von Bildern stellt **Canvas** die Methode `drawImage()` zur Verfügung, die mit drei unterschiedlichen Parametersätzen aufgerufen werden kann.

```
context.drawImage(image, dx, dy)
context.drawImage(image, dx, dy, dw, dh)
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```

In allen drei Fällen benötigen wir im ersten Parameter ein `image`-, `canvas`– oder `video`-Element, das dynamisch über JavaScript oder statisch im HTML-Code eingebunden sein kann. Animierte Bilder oder Videos werden dabei allerdings nicht bewegt dargestellt, sondern statisch durch ihren ersten Frame beziehungsweise einen Poster-Frame, sofern dieser vorhanden ist.

Alle weiteren Argumente der `drawImage()` Methode beeinflussen Position, Größe oder Ausschnitt des Quellrasters im Ziel-Canvas. Abbildung 1 liefert die grafische Interpretation der möglichen Positionsparameter, wobei die Präfixe s für `source` (Quelle) und d für destination (Ziel) stehen.

Vergleichen wir nun die einzelnen `drawImage()`-Methoden anhand von drei einfachen Beispielen. Das gemeinsame Setup besteht aus einem 1200 x 800 Pixel großen Bild, das dynamisch als **JavaScript-Objekt** erzeugt wird.

```
var image = new Image();
image.src = 'images/yosemite.jpg';
```

Neben Pixelangaben, die uns in den Beispielen begegnen werden, zeigt Abbildung 2 die imposante, 1000 Meter hohe Felswand von El Capitan im Yosemite Nationalpark, vom Taft Point aus fotografiert. Dieses Bild wird nun `onload` in einer der drei Arten auf den 600 x 400 Pixel großen Ziel-Canvas gezeichnet. Die erste und einfachste Möglichkeit bestimmt über dx/dy die linke obere Ecke des Bildes im Ziel-Canvas. In unserem Fall ist dies die Position 0/0.

```
image.onload = function() {
context.drawImage(image,0,0);
};
```

Breite sowie Höhe werden dabei direkt aus dem Originalbild übernommen, und da unser Bild größer als der Ziel-Canvas ist, erscheint wenig überraschend auch nur das linke obere Viertel mit dem Taft Point auf der Canvas-Fläche.

Wollen wir den gesamten Bildausschnitt im Canvas darstellen, müssen wir zusätzlich die gewünschte Breite und Höhe in den Argumenten dw/dh angeben. Die Skalierung des Bildes auf 600 x 400 Pixel übernimmt dann der Browser für uns.

```
image.onload = function() {
context.drawImage(image,0,0,600,400);
};
```

Die dritte Variante von `drawImage()` bietet – im Gegensatz zu den beiden bisherigen, die auch mit CSS

realisierbar gewesen wären – ganz neue Möglichkeiten, mit Bildern zu arbeiten. Beliebige Ausschnitte des Quellbildes (sx, sy, sw, sh) können jetzt in definierte Bereiche des **Ziel-Canvas** (dx, dy, dw, dh) kopiert werden. Der Montage von Bildern steht damit nichts mehr im Wege.

```
image.onload = function() {
context.drawImage(image,0,0);
context.drawImage(
image, 620,300,300,375,390,10,200,250
);
};
```

Der erste `drawImage()`-Aufruf liefert wiederum das linke obere Viertel mit dem Taft Point, der zweite extrahiert den Bereich von El Capitan und zeichnet ihn als Icon in die rechte obere Ecke. Beschriftungen mit Schatten vervollständigen das rudimentäre Layout unserer Postkarte.

Wer lieber El Capitan im Vordergrund und den Taft Point als Briefmarke rechts oben sehen will, der muss nur die `drawImage()`-Aufrufe leicht modifizieren – in unserem Beispiel passiert dies, wenn man auf den Canvas klickt.

```
canvas.onclick = function() {
context.drawImage(
image,600,250,600,400,0,0,600,400
);
context.drawImage(
image,0,0,500,625,390,10,200,250
);
};
```

Neben Vektorgrafiken (Pfaden, Linien usw.) unterstützt die **Canvas-API** auch Bitmap-Bilder. Die Methode `drawImage()` kopiert die Pixel einer Bildquelle (oder eines rechteckigen Ausschnitts der Bildquelle) auf das Canvas und skaliert und rotiert dabei die Pixel des Bilds nach Bedarf.

`drawImage()` kann mit drei, fünf oder neun Argumenten aufgerufen werden. Bei allen Aufrufformen ist das erste Argument die Bildquelle, aus der die Pixel kopiert werden sollen. Dieses Bildargument ist häufig ein `<img>`-Element oder ein Bild, das im Hintergrund mit dem `Image()`-Konstruktor erstellt wurde, kann aber auch ein anderes `<canvas>`-Element oder sogar ein `<video>`-Element sein. Geben Sie ein `<img>`– oder `<video>`-Tag an, dessen Daten noch geladen werden, tut der `drawImage()`-Aufruf nichts.

Beim `drawImage()`-Aufruf mit drei Argumenten geben das zweite und das dritte Argument die x– und y-Koordinaten an, an denen die linke obere Ecke des Bilds gezeichnet werden soll. Bei dieser Aufrufform wird die gesamte Bildquelle auf das Canvas kopiert. Die x– und y-Koordinaten beziehen sich auf das aktuelle Koordinatensyste und das Bild wird entsprechend skaliert und gedreht.

Der `drawImage()`-Aufruf mit fünf Argumenten fügt den gerade beschriebenen x– und y-Argumenten die Argumente `width` und `height` hinzu. Diese vier Argumente definieren das Zielrechteck im Canvas. Die linke obere Ecke befindet sich bei (x,y) und die rechte untere bei (`x+width`, `y+height`). Auch bei dieser Form wird die gesamte Bildquelle kopiert. Das Zielrechteck wird im aktuellen Koordinatensystem gemessen. Bei dieser Form des Methodenaufrufs wird die Bildquelle so skaliert, dass sie in das Zielrechteck passt, auch wenn keine Skalierungstransformation definiert wurde.

Der `drawImage()`-Aufruf mit neun Argumenten gibt sowohl ein Quellrechteck als auch ein Zielrechteck an und kopiert nur die Pixel im Quellrechteck. Die Argumente zwei bis fünf spezifizieren das Quellrechteck und werden in **CSS-Pixeln** angegeben. Ist die Bildquelle ein anderes Canvas, nutzt das Quellrechteck das Standardkoordinatensystem für jenes Canvas und ignoriert sämtliche angegebenen Transformationen. Die Argumente sechs bis neun geben das Zielrechteck an, in das das Bild gezeichnet wird, und beziehen sich auf das aktuelle Koordinatensystem des Canvas, nicht auf das Standardkoordinatensystem.

Beispiel 1 ist eine einfache Demonstration von `drawImage()`. Es nutzt den Aufruf mit neun Argumenten, um Pixel aus einem Teil des Canvas zu kopieren und sie dann, vergrößert und rotiert, wieder in das Canvas zu zeichnen. Wie Sie in Abbildung 1 sehen, wurde das Bild so stark vergrößert, dass es pixelig ist. Sie können sogar die durchscheinenden Pixel sehen, mit denen die Ecken der Linie geglättet wurde.

Beispiel 1: `drawImage()` verwenden

```
// Eine Linie in die linke obere Ecke zeichnen
c.moveTo(5,5);
c.lineTo(45,45);
c.lineWidth = 8;
c.lineCap = "round";
c.stroke();

// Eine Transformation definieren
c.translate(50,100);
c.rotate(-45*Math.PI/180); // Die Linie begradigen
c.scale(10,10);            // Skalieren, damit wir die Pixel
                           // sehen können
// Mit drawImage() die Linie kopieren
c.drawImage(c.canvas, // Aus dem Canvas selbst kopieren
0, 0, 50, 50,         // Nicht transformiertes
                      // Quellrechteck
0, 0, 50, 50);        // Transformiertes Zielrechteck
```

Wir können nicht nur Bilder in ein Canvas zeichnen, wir können auch den Inhalt eines Canvas als Bild

herausziehen. Dazu dient die Methode `toDataURL()`. Im Unterschied zu allen anderen hier beschriebenen Methoden ist `toDataURL()` eine Methode des **Canvas-Objekts** selbst, nicht des CanvasRenderingContext2D-Objekts. Üblicherweise rufen Sie `toDataURL()` ohne Argumente auf und erhalten dann das Canvas als PNG-Bild, das über eine `data:`-URL als String kodiert ist. Die gelieferte URL kann mit einem `<img>`-Tag verwendet werden und ermöglicht Ihnen, folgendermaßen einen statischen Schnappschuss des Canvas zu erstellen:

```
// Ein <img>-Tag erstellen
var img = document.createElement("img");

// Das src-Attribut setzen
img.src = canvas.toDataURL();

// Das Tag ins Dokument einfügen
document.body.appendChild(img);
```

Von Browsern wird verlangt, dass sie das PNG-Format unterstützen. Einige Implementierungen unterstützen eventuell auch noch anderer Formate. Den gewünschten Mime-Typ können Sie über das optionale erste Argument von `toDataURL()` angeben.

Es gibt allerdings eine wichtige Sicherheitseinschränkung, der Sie sich bewusst sein sollten, wenn Sie `toDataURL()` nutzen. Damit es nicht zu Cross-Origin-Informationslöchern kommt, funktioniert `toDataURL()` bei `<canvas>`-Tags nicht, deren Herkunft nicht rein ist. Das ist der Fall, wenn in ein Canvas ein Bild gezeichnet wurde (entweder direkt mit `drawImage()` oder indirekt über ein CanvasPattern), das von einer anderen Quelle stammt als das Dokument, das das Canvas enthält.

Canvas – ein erstes Beispiel

Eines der interessantesten und zugleich der ältesten neuen **HTML5-Elemente** ist Canvas. Bereits im Juli 2004, also nur einen Monat nach Gründung der WHATWG, präsentierte David Hyatt von Apple eine proprietäre Erweiterung für HTML namens Canvas und sorgte damit für Aufregung in der noch jungen HTML5-Bewegung. *The real solution is to bring these proposals to the table* war *Ian Hicksons* erste Reaktion, und nach kurzer Diskussion reichte Apple seinen Vorschlag bei der WHATWG zur Begutachtung ein. Der Weg für die Aufnahme von Canvas in die HTML5-Spezifikation war damit geebnet, ein erster Draft erschien im August 2004.

> Bei Canvas (übersetzt Leinwand) handelt es sich vereinfacht gesagt um ein programmierbares Bild,

auf das mithilfe einer Javascript-API gezeichnet werden kann. Dazu benötigen wir neben einem `canvas`-Element als Leinwand auch ein `script`-Element, in dem die Zeichenbefehle Platz finden. Beginnen wir mit dem `canvas`-Element:

```
<canvas width="1200" height="800">
alternativer Inhalt für Browser ohne Canvas-Unterstützung
</canvas>
```

Die Attribute `width` und `height` bestimmen die Dimension des `canvas`-Elements in Pixel und reservieren entsprechend viel Platz auf der HTML-Seite. Fehlt eines oder fehlen gar beide Attribute, kommen Vorgabewerte zum Tragen: 300 Pixel für die Breite und 150 Pixel für die Höhe. Der Bereich zwischen Start- und End-Tag ist für alternativen Content reserviert, der zur Anzeige gelangt, wenn ein Browser **Canvas** nicht unterstützt. Ähnlich wie das `alt`-Tag bei Bildern sollte dieser alternative Content den Inhalt der Canvas-Applikation beschreiben oder einen entsprechenden Screenshot zeigen. Formulierungen wie Ihr Browser unterstützt Canvas nicht ohne Zusatzinformationen sind wenig hilfreich und deshalb zu vermeiden.

Damit ist unsere Leinwand fertig, und wir können im nächsten Schritt die Zeichenbefehle in einem `script`-Element hinzufügen. Wenige Zeilen Code reichen aus, um unser erstes, zugegebenermaßen recht triviales Canvas-Beispiel zu realisieren.

```
<script>
var canvas = document.querySelector("canvas");
var context = canvas.getContext('2d');
context.fillStyle = 'red';
context.fillRect(0,0,800,600);
context.fillStyle = 'rgba(255,255,0,0.5)';
context.fillRect(400,200,800,600);
</script>
```

Auch wenn wir noch nichts über die Syntax der **Canvas-Zeichenbefehle** wissen, ist bei näherer Betrachtung des Codes das Resultat in Abbildung 1 wenig überraschend. Wir sehen ein rotes und ein hellgelbes Rechteck mit 50 % Opazität, wodurch im Überlappungsbereich ein oranger Farbton entsteht.

Bevor wir auf die Leinwand zeichnen können, benötigen wir eine Referenz zu derselben.

Die erste Zeile im Skript tut genau dies und speichert in der Variablen canvas mithilfe der W3C CSS Selectors API-Methode `document.querySelector()` eine Referenz auf das erste gefundene `canvas`-Element im Dokument:

```
var canvas = document.querySelector("canvas");
```

Neben den Attributen `canvas.width` und `canvas.height` besitzt dieses, auch HTMLCanvasElement genannte Objekt die Methode `getContext()`. Sie erlaubt uns, auf das Herzstück von Canvas zuzugreifen, den CanvasRenderingContext2D, indem wir 2d als Kontext-Parameter übergeben:

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

Damit ist der Zeichenkontext definiert, und wir können mit dem Zeichnen der beiden Rechtecke beginnen. Ohne auf Details des Attributs `fillStyle` oder der Methode `fillRect()` einzugehen, bietet sich zweimal derselbe Ablauf: Füllung definieren und dann das Rechteck hinzufügen:

```
context.fillStyle = 'red';
context.fillRect(0,0,800,600);
context.fillStyle = 'rgba(255,255,0,0.5)';
context.fillRect(400,200,800,600);
```

Die derzeitige **Canvas-Spezifikation** definiert nur einen 2D-Kontext (siehe die HTML Canvas 2D Context-Spezifikation), schließt aber nicht aus, dass weitere, wie zum Beispiel 3D, zu einem späteren Zeitpunkt folgen könnten. Erste Initiativen in diese Richtung laufen bereits bei der Khronos-Gruppe, die in Zusammenarbeit mit Mozilla, Google und Opera an einer JavaScript-Bindung namens WebGL für OpenGL ES 2.0 arbeitet. Erste Implementierungen dieses jungen Standards finden sich bereits in Firefox, WebKit und Chrome.

Wenden wir uns aber wieder dem 2D-Kontext zu, denn die Möglichkeiten, die sich hinter dem CanvasRenderingContext2D-Interface verstecken, sind mannigfaltig und durchaus geeignet, um anspruchsvolle Applikationen zu realisieren. Abbildung 2 zeigt ein einfaches Säulendiagramm, das uns bei der Erklärung der ersten drei Features des Zeichenkontextes begleiten wird: Rechtecke, Farben und Schatten.

Rechtecke

Canvas verfügt über vier Methoden, um Rechtecke zu erstellen. Mit dreien davon wollen wir uns jetzt beschäftigen, die vierte wird uns später beim Thema Pfade begegnen.

```
context.fillRect(x, y, w, h)
context.strokeRect(x, y, w, h)
context.clearRect(x, y, w, h)
```

Die Namen dieser Methoden sind selbsterklärend. So erzeugt `fillRect()` ein gefülltes Rechteck, `strokeRect()` ein Rechteck mit Randlinie ohne Füllung und `clearRect()` ein Rechteck, das bestehenden Inhalt wie ein Radiergummi löscht. Die Dimension des Rechtecks bestimmen vier numerische Parameter: Startpunkt x/y, Breite w und Höhe h.

In Canvas liegt der Koordinatenursprungspunkt übrigens links oben,

wodurch x-Werte nach rechts und y-Werte nach unten größer werden.

Analog zum ersten Beispiel definieren wir beim Balkendiagramm zuerst eine Referenz auf das `canvas`-Element und dann den Zeichenkontext. Für die Hauptarbeit, das Erstellen der horizontalen Balken, ist die Funktion `drawBars()` verantwortlich, der wir zugleich die gewünschte Anzahl zu zeichnender Balken übergeben.

```
<script>
var canvas = document.querySelector("canvas");
var context = canvas.getContext('2d');
var drawBars = function(bars) {
context.clearRect(0,0,canvas.width,canvas.height);
for (var i=0; i<bars; i++) {
var yOff = i*(canvas.height/bars);
var w = Math.random()*canvas.width;
var h = canvas.height/bars*0.8;
context.fillRect(0,yOff,w,h);
context.strokeRect(0,yOff,w,h);
}
};
drawBars(10);
</script>
```

Ein Aufruf dieser Funktion durch `drawBars(10)` löscht somit über `clearRect()` allenfalls bestehenden Inhalt und zeichnet danach in der `for`-Schleife über `fillRect()` und `strokeRect()` die zehn gefüllten Balken mit Randlinie. Die Breite w der Balken variiert von 0 Pixel bis zur Gesamtbreite des `canvas`-Elements und wird mithilfe der JavaScript-Funktion `Math.random()` zufällig gesetzt. `Math.random()` liefert eine Zahl von 0.0 bis 1.0 und ist daher bestens geeignet, um Zufallswerte für Breite, Höhe, aber auch für die Position in Abhängigkeit von der Canvas-Dimension zu ermitteln. Die Multiplikation mit dem entsprechenden Attribut-Wert genügt.

Die gleichabständige, horizontale Aufteilung der Balken folgt der Canvashöhe.

Die Abstände zwischen den Balken resultieren daraus, dass die errechnete, maximale Balkenhöhe h mit

dem Faktor 0.8 multipliziert wird.

Die Breite und Höhe des Canvas lassen sich, wie schon im ersten Beispiel erwähnt, bequem über die Attribute `canvas.width` und `canvas.height` ablesen. Ebenso einfach können wir auch vom Zeichenkontext über dessen Attribut `context.canvas` auf das **HTMLCanvasElement** zugreifen, das wir auch gleich dazu verwenden, um bei jedem Klick auf den Canvas neue Balken zu generieren. Drei Zeilen Code im Anschluss an den `drawBars(10)`-Aufruf reichen dazu aus.

```
context.canvas.onclick = function() {
drawBars(10);
};
```

Nachdem nun geklärt ist, wie die zehn Balken gezeichnet werden, stellt sich die nächste Frage: Wie kommt die hellgraue Farbe der Balken mit schwarzer Randlinie zustande? Ein Blick auf die Möglichkeiten, in Canvas Farben zu vergeben, liefert die Antwort.

Farben und Schatten

Die Attribute `fillStyle` und `strokeStyle` dienen dazu, Farben für Füllungen und Linien festzulegen. Farbangaben folgen dabei den Regeln für CSS-Farbwerte und dürfen aus diesem Grund in einer Reihe verschiedener Formatierungen angegeben werden. Tabelle zeigt die Möglichkeiten am Beispiel der Farbe Rot.

Methode	Farbwert
Hexadezimal	`#FF0000`
Hexadezimal (kurz)	`#F00`
RGB	`rgb(255,0,0)`
RGB (Prozent)	`rgb(100%,0%,0%)`
RGBA	`rgba(255,0,0,1.0)`
RGBA (Prozent)	`rgba(100%,0%,0%,1.0)`
HSL	`hsl(0,100%,50%)`

Methode	Farbwert
HSLA	`hsla(0,100%,50%,1.0)`
SVG-Farbnamen	`red`

Um die aktuell gültige Füll- und Strichfarbe in Canvas festzulegen, genügt es, entsprechende Farbwerte als Zeichenketten für `fillStyle` und `strokeStyle` anzugeben. Im Balkendiagramm-Beispiel sind dies die benannte SVG-Farbe `silver` als Füllung sowie eine halbtransparente schwarze Randlinie in RGBA-Notation. Da alle Balken gleich aussehen sollen, definieren wir die dazugehörigen Stile vor der `drawBars()`-Funktion.

```
context.fillStyle = 'silver';
context.strokeStyle = 'rgba(0,0,0,0.5)';
var drawBars = function(bars) {
// Code zum Balkenzeichnen
};
```

Gültige Werte für die Opazität reichen von 0.0 (transparent) bis 1.0 (opak) und können als vierte Komponente sowohl im RGB-Farbraum als auch im HSL-Farbraum verwendet werden. Letzterer definiert Farben nicht über ihre Rot-, Grün- und Blauanteile, sondern durch eine Kombination aus Farbton, Sättigung und Helligkeit.
Info

Weitere Informationen zum Thema **CSS-Farben** mit **HSL-Farbpaletten** und eine Liste aller gültigen SVG-Farbnamen finden Sie in der CSS Color Module Level 3-Spezifikation.

Bei genauerem Hinsehen erkennt man Schatten hinter den Balken, die durch vier weitere Attribute des Zeichen-Kontextes entstehen:

```
context.shadowOffsetX = 2.0;
context.shadowOffsetY = 2.0;
context.shadowColor = "rgba(50%,50%,50%,0.75)";
context.shadowBlur = 2.0;
```

Die ersten beiden Zeilen legen über `shadowOffsetX` und `shadowOffsetY` den Schattenoffset fest, `shadowColor` bestimmt dessen Farbe und Transparenz, und `shadowBlur` bewirkt schließlich das Weichzeichnen des Schattens, wobei als Faustregel Folgendes gilt: Je höher der Wert von `shadowBlur` ist, desto stärker ist der Weichzeichnungseffekt.

Bevor wir uns nun im nächsten Abschnitt mit dem Thema Farbverläufe beschäftigen, bleibt noch zu klären,

wie der gestrichelte Rahmen im Balkendiagramm-Beispiel und allen folgenden Grafiken zustande kommt. Die Antwort ist einfach: durch CSS. Jedes `canvas`-Element darf natürlich auch mit CSS formatiert werden. Abstände, Position oder `z-index` lassen sich ebenso festlegen wie Hintergrundfarbe oder Rahmen. Im unserem Beispiel sorgt folgendes Stilattribut für den gestrichelten Rahmen:

```
<canvas style="border: 1px dotted black;">
```

Farbverläufe

Als Ergänzung zu Vollfarben für Füllungen und Linien hält Canvas zwei Arten von Farbverläufen bereit: `lineare` und `radiale` **Gradienten**. Am Beispiel eines einfachen Farbverlaufes von Rot über Gelb und Orange nach Violett lässt sich das Grundprinzip beim Erstellen von Gradienten in Canvas leicht demonstrieren.

Zuerst erzeugt `context.createLinearGradient`(x0, y0, x1, y1) ein CanvasGradient-Objekt und legt dabei über die Parameter x0, y0, x1, y1 die Richtung des Farbverlaufs fest. Da wir in einem weiteren Schritt noch Farboffsets bestimmen müssen, speichern wir dieses Objekt in der Variablen linGrad.

```
var linGrad = context.createLinearGradient(
0,450,1000,300
);
```

Die Methode `addColorStop(offset, color)` des CanvasGradient-Objekts dient in einem zweiten Schritt zum Wählen der gewünschten Farben und deren Offsets auf unserer imaginären Farbverlaufslinie. Offset 0.0 steht für die Farbe am Punkt x0/y0 und Offset 1.0 für die Farbe am Endpunkt x1/y1. Alle dazwischen liegenden Farben werden ihrem Offset entsprechend aufgeteilt, und Übergänge zwischen den einzelnen Stopps werden vom Browser im RGBA-Farbraum interpoliert.

```
linGrad.addColorStop(0.0, 'red');
linGrad.addColorStop(0.5, 'yellow');
linGrad.addColorStop(0.7, 'orange');
linGrad.addColorStop(1.0, 'purple');
```

Farbangaben folgen wieder den Regeln für CSS-Farbwerte und sind im Beispiel zur besseren Lesbarkeit als SVG-Farbnamen ausgewiesen. Damit ist unsere lineare Gradiente fertig und kann über `fillStyle` oder `strokeStyle` zugewiesen werden.

```
context.fillStyle = linGrad;
context.fillRect(0,450,1000,300);
```

Im Gegensatz zu linearen Farbverläufen liegen Start und Endpunkt bei radialen Gradienten nicht als Punkte, sondern Kreise vor, weshalb wir nun zur Definition der Gradiente die Methode `context.createRadialGradient`(x0, y0, r0, x1, y1, r1) verwenden müssen.

Im linken Teil der Grafik erkennen wir Start- und Endkreis, in der Mitte die drei Farbstopps mit Offset-Werten und rechts das Endresultat: eine Kugel mit Lichteffekt. So ansprechend das Ergebnis ist, so einfach und übersichtlich ist auch der Quellcode:

```
var radGrad = context.createRadialGradient(
260,320,40,200,400,200
);
radGrad.addColorStop(0.0,'yellow');
radGrad.addColorStop(0.9,'orange');
radGrad.addColorStop(1.0,'rgba(0,0,0,0)');
context.fillStyle = radGrad;
context.fillRect(0,200,400,400);
```

Der Schatteneffekt am Kugelrand entsteht übrigens durch die beiden letzten Farbstopps, bei denen auf kürzestem Weg von Orange zu transparentem Schwarz interpoliert wird, wodurch der sichtbare Teil der Gradiente direkt am Außenkreis endet.

Canvas-Tutorial

Dieser Artikel erläutert, wie man mit JavaScript und dem HTML-`<canvas>`-Tag Grafiken in Webseiten zeichnet. Die Möglichkeit, komplexe Grafiken dynamisch im Webbrowser zu generieren, statt sie von einem Server herunterzuladen, stellt eine Revolution dar:

- Der Code, der auf Clientseite zur Erstellung der Grafiken genutzt wird, ist normalerweise erheblich kleiner als die Bilder selbst und spart damit eine Menge Bandbreite.
- Dass Aufgaben vom Server auf den Client verlagert werden, reduziert die Last auf dem Server und kann die Ausgaben für Hardware um einiges mindern.
- Die clientseitige Grafikgenerierung fügt sich gut in die Architektur von Ajax-Anwendungen ein, in denen der Server die Daten stellt und sich der Client um die Darstellung dieser Daten kümmert.
- Der Client kann Grafiken schnell und dynamisch neu zeichnen. Das ermöglicht grafikintensive Anwendungen (wie Spiele und Simulationen), die schlicht nicht machbar sind, wenn jeder Frame einzeln von einem Server heruntergeladen werden muss.
- Das Schreiben von Grafikprogrammen ist ein Vergnügen, und das `<canvas>`-Tag lindert für den Webentwickler die Pein, die die Arbeit mit dem DOM mit sich bringen kann.

Das <canvas>-Tag tritt nicht an sich in Erscheinung, sondern es erstellt eine Zeichenfläche im Dokument und bietet clientseitigem JavaScript eine mächtige API zum Zeichnen. Das <canvas>-Tag wird von **HTML5** standardisiert, treibt sich aber schon deutlich länger herum. Es wurde von Apple in Safari 1.3 eingeführt und wird von Firefox seit Version 1.5 und Opera seit Version 9 unterstützt. Außerdem wird es von allen Versionen von Chrome unterstützt. Vom Internet Explorer wird es erst ab Version 9 unterstützt, kann in IE 6, 7 und 8 aber hinreichend gut emuliert werden.

Canvas im IE

Wenn Sie das <canvas>-Tag im Internet Explorer 6, 7 oder 8 einsetzen wollen, können Sie das Open Source-Projekt ExplorerCanvas herunterladen. Entpacken Sie das Archiv und fügen Sie das excanvas-Skript im <head> Ihrer Webseiten mit einem bedingten Kommentar (Conditional Comment) für den Internet Explorer in folgender Form ein:

```
<!--[if lte IE 8]>
<script src="excanvas.compiled.js"></script>
<![endif]-->
```

Stehen diese Zeilen am Anfang Ihrer Webseiten, funktionieren <canvas>-Tag und elementare **Canvas-Zeichenbefehle** im IE. Radiale Verläufe und Clipping werden nicht unterstützt. Die Linienbreite wird nicht korrekt skaliert, wenn die x- und y-Dimensionen um unterschiedliche Beträge skaliert werden. Zusätzlich müssen Sie damit rechnen, beim IE weitere Rendering-Unterschiede zu sehen.

Große Teile der Canvas-Zeichen-API werden nicht im <canvas>-Element selbst definiert, sondern auf einem „Zeichenkontext-Objekt“, das Sie sich mit der getContext()-Methode des Canvas beschaffen. Rufen Sie getContext() mit dem Argument 2d auf, erhalten Sie ein CanvasRenderingContext2D-Objekt, über das Sie zweidimensionale Grafiken in das Canvas zeichnen können. Es ist wichtig, sich darüber bewusst zu sein, dass das Canvas-Element und sein Kontext-Objekt zwei sehr verschiedene Objekte sind. Da der Klassenname so lang ist, verweise ich auf das CanvasRenderingContext2D-Objekt nur selten mit diesem Namen, sondern nenne es einfach das „Kontext-Objekt“. Und wenn ich von der „Canvas-API“ spreche, meine ich damit in der Regel „die Methoden des CanvasRenderingContext2D-Objekts„. Da der lange Klassenname CanvasRenderingContext2D nicht gut auf diese schmalen Seiten passt, wird er außerdem im auf diese Einführung folgenden Referenzabschnitt mit CRC abgekürzt.

3-D-Grafiken in einem Canvas

Als dies geschrieben wurde, begannen Browserhersteller gerade damit, eine 3-D-Grafik-API für das <canvas>-Tag zu implementieren. Die entsprechende API wird als WebGL bezeichnet und ist eine JavaScript-Schnittstelle zur OpenGL-Standard-API. Ein Kontext-Objekt für 3-D-Grafiken erhalten Sie, wenn Sie der getContext()-Methode des Canvas den String webgl übergeben. WebGL ist eine umfangreiche und komplizierte Low-Level-API, die in diesem Artikel nicht dokumentiert wird: Webentwickler werden wahrscheinlich eher auf WebGL aufgesetzte Hilfsbibliotheken nutzen als die WebGL-API selbst.

Ein einfaches Beispiel für die **Canvas-API** sehen Sie im folgenden Code, der ein rotes Rechteck und einen blauen Kreis in <canvas>-Tags schreibt und eine Ausgabe wie die generiert, die Sie in Abbildung 1 sehen.

```
<body>
```

```
Das ist ein rotes Rechteck:
<canvas id="square" width=10 height=10></canvas>.
Das ist ein blauer Kreis:
<canvas id="circle" width=10 height=10></canvas>.
<script>

// Das erste Canvas-Element und seinen Kontext abrufen
var canvas = document.getElementById("square");
var context = canvas.getContext("2d");

// Etwas in das Canvas zeichnen
context.fillStyle = "#f00";  // Die Farbe auf Rot setzen
context.fillRect(0,0,10,10); // Ein kleines Rechteck füllen

// Das zweite Canvas und seinen Kontext abrufen
canvas = document.getElementById("circle");
context = canvas.getContext("2d");

// Einen Pfad beginnen und ihm einen Kreis hinzufügen
context.beginPath();
context.arc(5, 5, 5, 0, 2*Math.PI, true);
context.fillStyle = "#00f"; // Die Füllfarbe auf Blau setzen
context.fill();             // Den Pfad füllen
</script>
</body>
```

Die Canvas-API beschreibt komplexe Figuren als „Pfade“ von Linien und Kurven, die gezeichnet oder gefüllt werden können. Ein Pfad wird durch eine Folge von Methodenaufrufen definiert, wie beispielsweise die `beginPath()`- und `arc()`-Aufrufe aus dem vorangegangenen Code. Nachdem ein Pfad definiert ist, operieren andere Methoden wie `fill()` auf diesem Pfad. Verschiedene Eigenschaften des Kontext-Objekts wie `fillStyle` legen fest, wie diese Operationen ausgeführt werden. Die nächsten Unterabschnitte erläutern die folgenden Dinge:

- Wie man Pfade definiert und die Linie des Pfads zeichnet oder das Innere eines Pfads füllt.
- Wie man die Grafikattribute des Canvas-Kontext-Objekts setzt und abfragt und wie man den aktuellen Status dieser Attribute speichert und wiederherstellt.
- Canvas-Maße, das Standard-Canvas-Koordinatensystem und wie man dieses Koordinatensystem transformiert.
- Die verschiedenen Methoden zum Zeichnen von Kurven, die von der Canvas-API definiert werden.
- Einige spezielle Hilfsmethoden zum Zeichnen von Rechtecken.
- Wie man Farben angibt, mit Transparenz arbeitet, Farbverläufe zeichnet und Bildmuster wiederholt.
- Die Attribute, die die Strichbreite und das Erscheinungsbild der Linienendpunkte und -eckpunkte

steuert.
- Wie man Text in ein <canvas> schreibt.
- Wie man Grafiken so beschneidet, dass nichts außerhalb der von Ihnen angegebenen Region gezeichnet wird.
- Wie man Grafiken Schlagschatten hinzufügt.
- Wie man Bilder in ein Canvas zeichnet (und optional skaliert) und wie man den Inhalt eines Canvas herauszieht und als Bild speichert.
- Wie man den Compositing-Prozess steuert, über den neu gezeichnete (durchscheinende) Pixel mit den im Canvas vorhandenen Pixeln kombiniert werden.
- Wie man die rohen Rot-, Grün-, Blau- und Alphawerte (Transparenz) von Pixeln im Canvas abfragt und setzt.
- Wie man ermittelt, ob über etwas, das Sie auf das Canvas gezeichnet haben, ein Mausereignis eingetreten ist.

Dieses Kapitel endet mit einem praktischen Beispiel, das das <canvas>-Tag nutzt, um kleine Inline-Diagramme zu zeichnen, die als Sparklines bezeichnet werden. Auf dieses Einführungskapitel folgt eine Referenz, die die Canvas-API in allen Details dokumentiert.

Viele der folgenden <canvas>-Codebeispiele operieren auf einer Variablen mit dem Namen c. Diese Variable hält das CanvasRenderingContext2D-Objekt des Canvas fest. Der Code, der diese Variable initialisiert, wird üblicherweise jedoch nicht gezeigt. Diese Beispiele funktionieren nur, wenn Sie das **HTML-Markup** mit einem Canvas mit den entsprechenden `width`– und `height`-Attributen haben und dann Code wie folgenden ergänzen, um die Variable c zu initialisieren:

```
var canvas = document.getElementById("my_canvas_id");
var c = canvas.getContext('2d');
```

Alle nachfolgenden Figuren werden mit JavaScript-Code generiert, der in ein <canvas>-Tag zeichnet, üblicherweise in ein großes Canvas, das nicht auf dem Bildschirm angezeigt wird, um druckfähige Grafiken mit hoher Auflösung zu erzeugen.

Clipping

Wenn Sie einen Pfad definiert haben, rufen Sie normalerweise `stroke()` oder `fill()` (oder beides) auf. Sie können aber auch die Methode `clip()` aufrufen, um einen **Clipping**-Bereich oder Ausschnitt zu definieren. Haben Sie einen Ausschnitt definiert, wird nichts außerhalb des entsprechenden Bereichs gezeichnet. Abbildung 1 zeigt eine komplizierte Zeichnung, die mit Ausschnitten erzeugt wurde. Die Umrisse des vertikalen Streifens in der Mitte und des Texts unten an der Figur wurden ohne Ausschnitt gezeichnet. Ihr Inhalt wurde gefüllt, nachdem der dreieckige Ausschnitt definiert wurde.

Abbildung 1 wurde mit der `polygon()`-Methode in Linien und Polygone Beispiel 1 und dem Code von Beispiel 1 unten erzeugt.

Beispiel 1: Einen Clipping-Bereich definieren

```
// Einige Zeichenattribute definieren
c.font = "bold 60pt sans-serif";   // Große Schrift
c.lineWidth = 2;                   // Schmale Linien
c.strokeStyle = "#000";            // Schwarze Linien

// Den Umriss eines Rechtecks und von etwas Text zeichnen
c.strokeRect(175, 25, 50, 325);          // Vertikale Streifen
c.strokeText("<canvas>", 15, 330); // Textumriss

// Einen komplexen Pfad mit einem Inneren definieren,
// das außen ist
polygon(c,3,200,225,200);         // Großes Dreieck
polygon(c,3,200,225,100,0,true);  // Kleines umgekehrtes Dreieck

// Diesen Pfad zum Clipping-Bereich machen
c.clip();

// Den Pfad mit einer 5 Pixel breiten Linie umgeben,
// die vollständig innerhalb des Clipping-Bereichs ist.
c.lineWidth = 10;

// Die Hälfte der Linie wird abgeschnitten
c.stroke();

// Die Teile des Rechtecks und des Texts füllen,
// die sich innerhalb des Clipping-Bereichs befinden.
c.fillStyle = "#aaa"              // Hellgrau
c.fillRect(175, 25, 50, 325);     // Den vertikalen Streifen füllen
c.fillStyle = "#888"              // Dunkelgrau
c.fillText("<canvas>", 15, 330);  // Den Text füllen
```

Es ist wichtig, sich zu merken, dass beim Aufruf von `clip()` der aktuelle Pfad selbst auf den aktuellen Ausschnitt beschnitten wird und dieser Pfad zum neuen Ausschnitt wird. Das bedeutet, dass die `clip()`-Methode den Ausschnitt verkleinern, aber nie vergrößern kann. Es gibt keine Methode, mit der man den Ausschnitt zurücksetzen kann. Bevor Sie `clip()` aufrufen, sollten Sie deswegen stets `save()` aufrufen, damit Sie später mit `restore()` den unbeschnittenen Bereich wiederherstellen können.

Dimensionen, Koordinaten

Die `width`– und `height`-Attribute des `<canvas>`-Tags und die entsprechenden `width`– und `height`-Eigenschaften des Canvas-Objekts geben die Ausmaße des **Canvas** an. Das Standard-Canvas-Koordinatensystem hat den Ursprung (0,0) in der linken oberen Ecke des Canvas. Die x-Koordinate wächst,

wenn Sie auf dem Bildschirm nach rechts gehen, die y-Koordinate, wenn Sie auf dem Bildschirm nach unten gehen. Punkte auf dem Canvas können mit Fließkommawerten angegeben werden, und diese werden nicht automatisch zülntegern aufgerundet – das Canvas nutzt Anti-Aliasing-Techniken, um partiell gefüllte Pixel züsimulieren.

Die Maße eines Canvas sind so grundlegend, dass sie nicht geändert werden können, ohne das Canvas vollständig zurückzusetzen. Das Setzen der `width`– oder `height`-Eigenschaft des Canvas (sogar ein Setzen auf den aktuellen Wert) leert das Canvas, löscht den aktuellen Pfad und setzt alle Grafikattribute (einschließlich der aktuellen Transformation und des Clippings) auf den ursprünglichen Zustand zurück.

Trotz der elementaren Bedeutung müssen die Maße des Canvas

nicht der tatsächlichen Größe des Canvas auf dem Bildschirm oder der Anzahl von Pixeln auf der **Canvas-Zeichenfläche** entsprechen. Die Canvas-Maße (und das Standardkoordinatensystem) werden in CSS-Pixeln gemessen. CSS-Pixel entsprechen üblicherweise gewöhnlichen Pixeln. Auf Bildschirmen mit hoher Auflösung ist es Implementierungen gestattet, mehrere Gerätepixel auf ein CSS-Pixel abzubilden. Das bedeutet, dass das Pixelrechteck, das das Canvas zeichnet, größer sein kann als die nominellen Maße des Canvas. Dessen müssen Sie sich bewusst sein, wenn Sie mit den Pixelmanipulationsmethoden des Canvas arbeiten, aber die Unterschiede zwischen virtuellen CSS-Pixeln und tatsächlichen Hardware-Pixeln wirkt sich ansonsten in keiner Weise auf den Canvas-Code aus, den Sie schreiben.

Standardmäßig wird ein `<canvas>`-Tag auf dem Bildschirm in der Größe (in CSS-Pixeln) gezeichnet, die von den HTML-Attributen `width` und `height` vorgegeben wird. Wie jedes HTML-Element kann die Bildschirmgröße eines `<canvas>`-Tags auch durch die CSS-Style-Attribute `width` und `height` angegeben werden. Wenn Sie eine Größe angeben, die sich von den tatsächlichen Maßen des Canvas unterscheidet, werden die Pixel auf dem Canvas automatisch so skaliert, dass sie den Bildschirmmaßen entsprechen, die von den **CSS-Attributen** vorgegeben werden. Die Bildschirmgröße auf dem Canvas wirkt sich nicht auf die Anzahl der CSS-Pixel oder Hardware-Pixel aus, die in der Canvas-Bitmap reserviert sind, und die Skalierung, die angewandt wird, erfolgt gemäß einer Skalierungsoperation für Bilder. Wenn sich die Bildschirmmaße erheblich von den tatsächlichen Ausmaßen des Canvas unterscheiden, führt das züverpixelten Grafiken. Aber das ist ein Problem, mit dem sich Grafiker befassen müssen, es wirkt sich nicht auf die Canvas-Programmierung aus.

Koordinaten-Systemtransformationen

Wie oben gesagt, befindet sich der Ursprung des Standardkoordinatensystems eines Canvas in der oberen linken Ecke, die x-Koordinaten verlaufen nach rechts, die y-Koordinaten nach unten. In diesem Standardsystem entsprechen die Koordinaten eines Punkts direkt einem CSS-Pixel (das dann unmittelbar auf ein oder mehrere Gerätepixel abgebildet wird). Bestimmte Canvas-Operationen und -Attribute (wie das Ermitteln der rohen Pixelwerte und das Setzen von Schattenverschiebungen) nutzen immer dieses Standardkoordinatensystem. Zusätzlich züdiesem Standardkoordinatensystem besitzt jedes Canvas eine „aktuelle Transformationsmatrix“ als Teil des Grafikzustands. Diese Matrix definiert das aktuelle Koordinatensystem des Canvas. Bei den meisten Canvas-Operationen werden die Punktkoordinaten, die Sie angeben, als Punkte im aktuellen Koordinatensystem betrachtet, nicht im Standardkoordinatensystem. Die aktuelle Transformationsmatrix wird genutzt, um die angegebenen Punkte in die äquivalenten

Koordinaten im Standardkoordinatensystem umzuwandeln.

Über die Methode `setTransform()` können Sie die Transformations-Matrix eines Canvas direkt setzen, aber Koordinaten-Systemtransformationen lassen sich in der Regel leichter als Folge von Translations-, Rotations- und Skalierungs-Operationen angeben. Abbildung 1 illustriert diese Operationen und ihre Auswirkungen auf das **Canvas-Koordinatensystem**. Das Programm, das diese Abbildung erzeugte, zeichnete sieben Mal die gleiche Gruppierung von Achsen. Das Einzige, was dabei geändert wurde, war die aktuelle Transformations-Matrix. Beachten Sie, dass sich die Transformation nicht nur auf die Linien, sondern auch auf den Text auswirkt.

Die `translate()`-Methode verschiebt den Ursprung des Koordinatensystems nach links, rechts, oben oder unten. Die `rotate()`-Methode rotiert die Achsen im Uhrzeigersinn um den angegebenen Winkel. (Die Canvas-API gibt Winkel immer in Radiant an. Grad wandeln Sie in Radiant um, indem Sie durch 180 teilen und mit Math.PI multiplizieren.) Die `scale()`-Methode streckt oder kontrahiert Abstände auf der x– oder y-Achse.

Wird `scale()` ein negativer Skalierungsfaktor angegeben, wird die entsprechende Achse über den Ursprung umgekehrt, als wäre sie in einem Spiegel reflektiert worden. Das wurde in der linken unteren Ecke von Abbildung 1 gemacht: Zunächst wurde der Ursprung mit `translate()` in die linke untere Ecke des Canvas verschoben; dann wurde die y-Achse gespiegelt mit `scale()`, sodass die Koordinaten nach oben hin ansteigen. Ein umgekehrtes Koordinatensystem wie dieses sollte Ihnen aus dem Geometrieunterricht bekannt sein und kann geeignet sein, wenn Sie Datenpunkte für Diagramme zeichnen müssen. Beachten Sie, dass das die Lesbarkeit des Texts erheblich verschlechtert!

Transformationen mathematisch verstehen

Mir fällt es am leichtesten, **Transformationen** geometrisch züverstehen und `translate()`, `rotate()` und `scale()` als Transformation der Achsen des Koordinatensystems zübetrachten, wie es in Abbildung 1 illustriert wird. Man kann Transformationen auch algebraisch als Gleichungen betrachten, die die Koordinaten des Punkts (x,y) im transformierten Koordinatensystem wieder in die Koordinaten des gleichen Punkts (x‘,y‘) im vorangegangenen Koordinatensystem überführen.

Der Methodenaufruf `c.translate(dx,dy)` kann mit folgenden Gleichungen beschrieben werden:

```
// (0,0) im neuen System entspricht (dx,dy) im alten
x' = x + dx;
y' = y + dy;
```

Die Gleichungen für Skalierungsoperationen sind ähnlich einfach. Der Aufruf `c.scale(sx,sy)` kann folgendermaßen beschrieben werden:

```
x' = sx * x;
y' = sy * y;
```

Rotationen sind etwas komplizierter. Der Aufruf `c.rotate(a)` wird durch folgende trigonometrische Gleichungen beschrieben:

```
x' = x * cos(a) - y * sin(a);
y' = y * cos(a) + x * sin(a);
```

Beachten Sie, dass die Reihenfolge der Transformationen wichtig ist. Angenommen, wir beginnen mit dem Standardkoordinatensystem eines Canvas und verschieben und skalieren es dann. Wollen wir den Punkt (x,y) im aktuellen Koordinatensystem wieder in den Punkt (x‘,,y‘) im Standardkoordinatensystem überführen, müssen wir erst die Skalierungsgleichungen anwenden, um den Punkt auf einen Zwischenpunkt (x‘, y‘) im verschobenen, aber nicht skalierten **Koordinatensystem** züüberführen. Dann wenden wir die Translationsgleichungen an, um diesen Zwischenpunkt in (x‘,y‘) züüberführen. Das Ergebnis sieht so aus:

```
x'' = sx*x + dx;
y'' = sy*y + dy;
```

Hätten wir hingegen erst `scale()` und dann `translate()` aufgerufen, hätten die resultierenden Gleichungen anders ausgesehen:

```
x'' = sx*(x + dx);
y'' = sy*(y + dy);
```

Der entscheidende Punkt, den Sie sich bei der algebraischen Betrachtung von Transformationen merken müssen, ist, dass Sie sich rückwärts von der letzten Transformation zur ersten hocharbeiten müssen. Wenn Sie transformierte Achsen geometrisch betrachten, arbeiten Sie sich von der ersten zur letzten Transformation hoch.

Die vom Canvas unterstützten Transformationen werden als affine Transformationen bezeichnet. Affine Transformationen können den Abstand zwischen zwei Punkten und die Winkel zwischen Linien ändern, aber parallele Linien bleiben nach einer affinen Transformation parallel – mit einer affinen Transformation ist es beispielsweise nicht möglich, eine Fischaugenverzerrung anzugeben. Jede affine Transformation kann mit den sechs Parametern a bis f in den folgenden Gleichungen beschrieben werden:

```
x' = ax + cy + e
y' = bx + dy + f
```

Sie können beliebige Transformationen auf das aktuelle Koordinatensystem anwenden, indem Sie diese sechs Parameter an die Methode `transform()` übergeben. Abbildung 1 illustriert zwei Typen von Transformationen – Scherungen und Drehungen um einen angegebenen Punkt – die Sie mit der `transform()`-Methode folgendermaßen erreichen können:

```
// Scherung:
// x' = x + kx*y;
// y' = y + ky*x;
function shear(c, kx, ky) {
c.transform(1, ky, kx, 1, 0, 0);
}

// Im Uhrzeigersinn theta-Radians um (x,y) drehen.
// Das kann auch mit der Transformationsfolge
// Translation -> Rotation -> Translation
// erreicht werden.
function rotateAbout(c, theta, x, y) {
var ct = Math.cos(theta), st = Math.sin(theta);
c.transform(ct, -st, st, ct,
-x*ct-y*st+x, x*st-y*ct+y);
}
```

Die Methode `setTransform()` erwartet die gleichen Argumente wie `transform()`, aber anstelle des aktuellen Koordinatensystems, das ignoriert wird, wird das Standardkoordinatensystem transformiert und das Ergebnis zum neuen aktuellen Koordinatensystem gemacht. Mit `setTransform()` kann man das Canvas vorübergehend auf das Standardkoordinatensystem zurücksetzen:

```
c.save(); // Aktuelles Koordinatensystem speichern

// Zum Standardkoordinatensystem zurückkehren
c.setTransform(1,0,0,1,0,0);

// Jetzt mit den Standard-CSS-Pixel-Koordinaten zeichnen
c.restore(); // Das gespeicherte Koordinatensystem
             // wiederherstellen
```

Farben, Transparenz, Verläufe und Muster

Die Attribute `strokeStyle` und `fillStyle` legen fest, wie Striche gezogen und Flächen gefüllt werden. Am häufigsten werden diese Attribute genutzt, um blickdichte oder durchscheinende Farben anzugeben, aber Sie können sie auch auf **CanvasPattern**– oder **CanvasGradient**-Objekte setzen, um Strich oder Füllung mit einem sich wiederholenden Hintergrundbild oder mit einem linearen oder radialen Farbverlauf zu versehen. Zusätzlich können Sie die Eigenschaft `globalAlpha` nutzen, um alles Gezeichnete

durchscheinend zu machen.

Wollen Sie eine blickdichte Farbe angeben, können Sie einen der Farbnamen nutzen, die vom HTML4-Standard definiert werden, oder einen CSS-Farbstring:

```
context.strokeStyle = "blue"; // Blaue Striche
context.fillStyle = "#aaa"; // Hellgrau füllen
```

Der Standardwert für `strokeStyle` und `fillStyle` ist `#000000`: blickdichtes Schwarz.

Aktuelle Browser unterstützen CSS3-Farben und ermöglichen neben den gewöhnlichen hexadezimalen RGB-Farbangaben die Verwendung von RGB-, RGBA-, HSL- und HSLA-Farbräumen. Hier sind einige mögliche Farbstrings:

```
"#f44"                      // Hexadezimaler RGB-Wert: red
"#44ff44"                   // RRGGBB-Wert: green
"rgb(60, 60, 255)"          // RGB als Ganzzahlen: blue
"rgb(100%, 25%, 100%)"      // RGB als Prozentzahlen: purple
"rgba(100%,25%,100%,0.5)"   // Plus Alphä0-1:
                            // durchscheinend
"rgba(0,0,0,0)"             // Transparentes Schwarz
"transparent"               // Synonym für vorangehende
                            // Zeile
"hsl(60, 100%, 50%)"        // Vollständig gesättigtes Gelb
"hsl(60, 75%, 50%)"         // Weniger gesättigtes Gelb
"hsl(60, 100%, 75%)"        // Vollständig gesättigt, heller
"hsl(60, 100%, 25%)"        // Vollständig gesättigt,
                            // dunkler
"hsla(60,100%,50%,0.5)"     // 50% durchscheinend
```

Der **HSL-Farbraum** definiert eine Farbe mit drei Zahlen, der Farbton (`Hue`), Sättigung (`Saturation`) und Helligkeit (`Lightness`) angeben. Der Farbton ist ein Winkel in Grad auf einem Farbkreis. Der Farbton 0 ist Rot, 60 ist Gelb, 120 ist Grün, 180 ist Cyan, 240 ist Blau, 300 ist Magenta, und 360 ist wieder Rot. Die Sättigung beschreibt die Intensität der Farbe über einen Prozentwert. Farben mit einer Sättigung von 0% sind Grautöne. Die Helligkeit beschreibt, wie hell oder dunkel eine Farbe ist, und wird ebenfalls über einen Prozentwert angegeben. Eine HSL-Farbe mit 100% Helligkeit ist reines Weiß, und jede Farbe mit der Helligkeit 0% ist reines Schwarz. Der HSLA-Farbraum entspricht HSL, ergänzt aber einen Alphawert zwischen 0.0 (durchsichtig) und 1.0 (undurchsichtig).

Wenn Sie mit durchscheinenden Farben arbeiten, aber nicht explizit einen Alphakanal für jede Farbe angeben oder blickdichten Bildern oder Mustern Transparenz hinzufügen wollen, können Sie die Eigenschaft `globalAlpha` nutzen. Der Alphawert jedes Pixels, das Sie zeichnen, wird mit `globalAlpha` multipliziert. Der Standardwert ist 1 und fügt keine Transparenz hinzu. Setzen Sie `globalAlpha` auf 0, ist

alles, was Sie zeichnen, vollständig transparent. Es erscheint alsönichts auf dem Canvas. Setzen Sie diese Eigenschaft auf 0.5, werden Pixel, die sonst blickdicht wären, zu 50% blickdicht. Pixel, die zu 50% blickdicht wären, sind nun zu 25% blickdicht. Setzen Sie `globalAlpha` auf einen Wert kleiner 1, sind alle Pixel durchscheinend. Gegebenenfalls werden Sie überlegen müssen, wie die Pixel mit den Pixeln kombiniert werden, die darübergezeichnet werden.

> Aber statt einfarbig mit einer (gegebenenfalls durchscheinenden) Farbe zu zeichnen,

können Sie auch Farbverläufe und sich wiederholende Bilder nutzen, wenn Sie Pfade füllen und ziehen. Abbildung 1 zeigt ein Rechteck, das mit breiten Linien und einem Strichmuster über einer Füllung mit einem linearen Verlauf und unter einer Füllung mit einem durchscheinenden radialen Verlauf gezeichnet wurde. Die **Codefragmente** unten zeigen, wie die Muster und Verläufe erstellt wurden.

Setzen Sie `fillStyle` oder `strokeStyle` auf das CanvasPattern-Objekt, das von der `createPattern()`-Methode des Kontext-Objekts geliefert wird, wenn Sie zum Zeichnen oder Füllen ein Hintergrundbildmuster statt einer Farbe nutzen wollen:

```
var image = document.getElementById("myimage");
c.fillStyle = c.createPattern(image, "repeat");
```

Das erste Argument für `createPattern()` gibt das Bild an, das als Muster verwendet wird. Es muss ein `<img>`-, `<canvas>`– oder `<video>`-Element aus dem Dokument sein (oder ein Bildobjekt, das mit dem `Image()`-Konstruktor erzeugt wurde). Das zweite Argument ist üblicherweise `repeat`, damit ein Bild unabhängig von der Größe des Bilds wiederholt eingefügt wird, aber Sie können auch `repeat-x`, `repeat-y` oder `no-repeat` angeben.

Beachten Sie, dass Sie ein `<canvas>`-Tag (selbst eins, das nie dem Dokument hinzugefügt wurde und nicht sichtbar ist) als Musterquelle für ein anderes `<canvas>` nutzen können:

```
// Ein nicht angezeigtes Canvas erstellen und seine Größe
// setzen
var offscreen = document.createElement("canvas");
offscreen.width = offscreen.height = 10;

// Kontext abrufen und in ihn zeichnen
offscreen.getContext("2d").strokeRect(0,0,6,6);
```

```
// Als Muster nutzen
var pattern = c.createPattern(offscreen,"repeat");
```

Wollen Sie mit einem **Farbverlauf** füllen (oder zeichnen), setzen Sie `fillStyle` (oder `strokeStyle`) auf ein `CanvasGradient`-Objekt, das von den `createLinearGradient()`– oder `createRadialGradient()`-Methoden des Kontexts geliefert wird. Die Erstellung von Verläufen erfordert mehrere Schritte, und ihr Einsatz ist komplizierter als der von Mustern.

Der erste Schritt ist, dass Sie ein `CanvasGradient`-Objekt erstellen. Die Argumente für `createLinearGradient()` sind die Koordinaten zweier Punkte, die eine Linie definieren (die nicht horizontal oder vertikal sein muss), an der entlang sich die Farbe ändert. Die Argumente für `createRadialGradient()` geben den Mittelpunkt und die Radien von zwei Kreisen an. (Sie müssen nicht konzentrisch sein, aber der erste Kreis liegt üblicherweise vollständig im zweiten.) Bereiche im kleineren Kreis oder außerhalb des größeren werden einfarbig gefüllt, Bereiche zwischen den beiden Kreisen werden mit einem Farbverlauf gefüllt.

Nachdem wir das `CanvasGradient`-Objekt erstellt und die Bereiche des **Canvas** definiert haben, die gefüllt werden, definieren wir die Verlaufsfarben, indem wir die `addColorStop()`-Methode des `CanvasGradient`-Objekts nutzen. Das erste Argument für diese Methode ist eine Zahl zwischen 0.0 und 1.0, das zweite eine CSS-Farbangabe. Sie müssen diese Methode mindestens zwei Mal aufrufen; um einen Farbverlauf zu definieren, können Sie sie allerdings auch öfter aufrufen. Die Farbe bei 0.0 erscheint am Anfang des Verlaufs und die Farbe bei 1.0 am Ende. Geben Sie zusätzliche Farben an, erscheinen diese an den entsprechenden Positionen. An den anderen Punkten werden die Farben stetig interpoliert. Hier sind einige Beispiele:

```
// Ein linearer Verlauf, quer über das Canvas
// (keine Transformationen vorausgesetzt).
var bgfade = c.createLinearGradient(0,0,canvas.width, canvas.height);

// Mit Hellblau oben links anfangen und dann nach
// unten rechts Weiß verblassen.
bgfade.addColorStop(0.0, "#88f");
bgfade.addColorStop(1.0, "#fff");

// Ein Verlauf zwischen zwei konzentrischen Kreisen.
// Transparent in der Mitte, zu durchscheinendem Grau
// werden und dann wieder transparent.
var peekhole = c.createRadialGradient(300,300,100, 300,300,300);
peekhole.addColorStop(0.0, "transparent");
peekhole.addColorStop(0.7, "rgba(100,100,100,.9)");
peekhole.addColorStop(1.0, "rgba(0,0,0,0)");
```

Ein wichtiger Punkt bei Verläufen ist, dass sie nicht positionsunabhängig sind. Wenn Sie einen Verlauf

erstellen, geben Sie Grenzen für den Verlauf vor. Versuchen Sie später, einen Bereich außerhalb dieser Grenzen zu füllen, erhalten Sie eine einfarbige Fläche in der Farbe, die am entsprechenden Ende des Verlaufs definiert ist. Haben Sie einen Verlauf über eine Gerade zwischen (0,0) und (100,100) definiert, sollten Sie diesen Verlauf beispielsweise nur nutzen, um Objekte zu füllen, die sich im Rechteck (0,0,100,100) befinden.

Die Grafik in Abbildung 1 wurde mit dem nachfolgenden **Code** erstellt (unter Verwendung des oben definierten Musters `pattern` und der Verläufe `bgfade` und `peekhole`):

```
c.fillStyle = bgfade;              // Den linearen Verläufe nutzen.
c.fillRect(0,0,600,600);           // um das ganze Canvas zu füllen.
c.strokeStyle = pattern;           // Das Muster und
c.lineWidth = 100;                 // breite Striche nutzen,
c.strokeRect(100,100, 400,400);    // um ein großes Quadrat zu zeichnen.
c.fillStyle = peekhole;            // Mit dem durchscheinenden radialen Verlauf
das
c.fillRect(0,0,600,600);           // Canvas füllen.
```

Grafikattribute

Linien zeichnen und Polygone füllen setzt die Eigenschaften `fillStyle`, `strokeStyle` und `lineWidth` auf dem Kontext-Objekt des **Canvas**. Diese Eigenschaften sind Grafikattribute, die die Farben angeben, die von `fill()` bzw. `stroke()` genutzt werden, sowie die Breite der Striche, die `stroke()` zeichnet. Beachten Sie, dass diese Parameter nicht an `fill()` und `stroke()` übergeben werden, sondern Teile eines allgemeinen Grafikzustands des Canvas sind.

Wenn Sie eine Methode definieren, die eine Figur zeichnet und diese Eigenschaften nicht selbst setzt, kann der Aufrufer Ihrer Methode die Farbe der Figur setzen, indem er vor dem Aufruf der Methode die Eigenschaften `strokeStyle` und `fillStyle` setzt. Diese Trennung von Grafikzustand und Zeichenbefehlen ist grundlegend für die Canvas-API und mit der Trennung von Darstellung und Inhalt verwandt, die man durch die Anwendung von Cascading Style Sheets (CSS) auf HTML-Dokumente erreicht.

Die **Canvas-API** definiert auf dem CanvasRenderingContext2D-Objekt 15 Grafikattribut-Eigenschaften. Diese Eigenschaften werden in Tabelle 1 aufgeführt und in den nachfolgenden Abschnitten ausführlich erläutert.

Eigenschaft	Bedeutung
`fillStyle`	Die Farbe, der Verlauf oder das Muster zum Füllen.

Eigenschaft	Bedeutung
`font`	Die CSS-Schriftart für Befehle zum Zeichnen von Text.
`globalAlpha`	Die Transparenz, die allen gezeichneten Pixeln hinzugefügt wird.
`globalCompositeOperation`	Wie die Pixelfarben kombiniert werden.
`lineCap`	Wie die Linienenden dargestellt werden.
`lineJoin`	Wie Eckpunkte dargestellt werden.
`lineWidth`	Die Breite der gezogenen Striche.
`miterLimit`	Die maximale Länge spitzwinkliger Eckpunkte.
`textAlign`	Horizontale Textausrichtung.
`textBaseline`	Vertikale Textausrichtung.
`shadowBlur`	Wie scharf oder unscharf Schatten sind.
`shadowColor`	Die Farbe des Schlagschattens.
`shadowOffsetX`	Die horizontale Verschiebung von Schatten.
`shadowOffsetY`	Die vertikale Verschiebung von Schatten.
`strokeStyle`	Die Farbe, der Verlauf oder das Muster für Striche.

Da die Canvas-API die Grafikattribute auf dem Kontext-Objekt definiert, könnten Sie versucht sein, mehrfach `getContext()` aufzurufen, um mehrere Kontext-Objekte zu erhalten. Wäre das möglich, könnten Sie auf diesen Kontexten unterschiedliche Attribute definieren, und jeder Kontext wäre dann wie

ein anderer Pinsel, der mit anderer Farbe oder anderer Breite zeichnet. Unglücklicherweise können Sie das Canvas auf diese Weise nicht nutzen. Jedes <canvas>-Tag hat nur ein einziges Kontext-Objekt, und jeder Aufruf von `getContext()` liefert das gleiche CanvasRenderingContext2D-Objekt.

Obwohl die Canvas-API Ihnen nicht ermöglicht, eigene Sätze von **Grafikattributen** zu definieren, ermöglicht sie Ihnen doch, den aktuellen Grafikzustand zu speichern, damit Sie ihn bearbeiten und später leicht wiederherstellen können. Die Methode `save()` schiebt den aktuellen Grafikzustand auf einen Stapel gespeicherter Zustände. Die Methode `restore()` nimmt das oberste Element dieses Stapels und stellt den entsprechenden Zustand wieder her. Alle in Tabelle 1 aufgeführten Eigenschaften sind Teil des gespeicherten Zustands, außerdem auch die aktuelle Transformation und der Clipping-Bereich (die beide später erläutert werden). Wichtig ist, dass der aktuell definierte Pfad und der aktuelle Punkt nicht Teil des Grafikzustands sind und nicht gespeichert und wiederhergestellt werden können.

Wenn Sie mehr Flexibilität benötigen, als Ihnen ein einfacher Stapel bietet, kann eine Hilfsmethode wie die in Beispiel vorgestellte nützlich sein.

```
// Zum letzten gespeicherten Grafikzustand zurückkehren,
// ohne den Stapel der gespeicherten Zustände zu verändern.
CanvasRenderingContext2D.prototype.revert = function() {
this.restore(); // Alten Grafikzustand wiederherstellen
this.save();    // Wieder speichern, damit wir zu ihm
                // zurückkehren können
return this;    // Methodenverkettung ermöglichen
};

// Die Grafikattribute auf die durch das Objekt o definierten
// Eigenschaften setzen. Oder, wenn kein Argument übergeben
// wird, die aktuellen Attribute als Objekt liefern. Beachten
// Sie, dass diese Methode Transformation und Clipping nicht
// berücksichtigt.
CanvasRenderingContext2D.prototype.attrs = function(o) {
if (o) {
for(var a in o) // Für jede Eigenschaft in o
this[a] = o[a]; // Ein Attribut auf dem Kontext
                // setzen
return this;    // Methodenverkettung ermöglichen
}
else return {
fillStyle: this.fillStyle,
font: this.font,
globalAlpha: this.globalAlpha,
globalCompositeOperation:
this.globalCompositeOperation,
lineCap: this.lineCap,
lineJoin: this.lineJoin,
```

```
lineWidth: this.lineWidth,
miterLimit: this.miterLimit,
textAlign: this.textAlign,
textBaseline: this.textBaseline,
shadowBlur: this.shadowBlur,
shadowColor: this.shadowColor,
shadowOffsetX: this.shadowOffsetX,
shadowOffsetY: this.shadowOffsetY,
strokeStyle: this.strokeStyle
};
};
```

Kurven zeichnen und füllen

Ein Pfad ist eine Folge von Teilpfaden, und ein Teilpfad ist eine Folge verbundener Punkte. In den Pfaden, die wir in Linien zeichnen und Polygone füllen und Transformations-Beispiel definiert haben, wurden diese Punkte durch gerade Liniensegmente verbunden, aber das muss nicht immer der Fall sein. Das CanvasRenderingContext2D-Objekt definiert eine Reihe von Methoden, die einem Teilpfad einen neuen Punkt hinzufügen und diesen über eine Kurve mit dem aktuellen Punkt verbinden:

- `arc()`

Die Methode fügt dem aktuellen Teilpfad einen **Kreisbogen** hinzu. Sie verbindet den aktuellen Punkt über eine gerade Linie mit dem Anfang des Bogens, dann den Anfang des Bogens über einen Kreisbogen mit dem Ende des Bogens und macht das Ende des Bogens zum neuen aktuellen Punkt. Der zu zeichnende Bogen wird durch sechs Parameter bestimmt: die x– und y-Koordinaten des Kreismittelpunkts, den Radius des Kreises, den Start- und den Endwinkel des Bogens und die Richtung (im Uhrzeigersinn oder entgegen dem Uhrzeigersinn) des Bogens zwischen diesen beiden Winkeln.

- `arcTo()`

Die Methode zeichnet wie `arc()` eine gerade Linie und einen Kreisbogen, gibt den zu zeichnenden Bogen aber mit anderen Parametern an. Die Argumente für `arcTo()` geben die Punkte P1 und P2 und einen Radius an. Der Bogen, der dem Pfad hinzugefügt wird, hat den angegebenen Radius und tangentiert die Linie zwischen dem aktuellen Punkt und P1 sowie der Linie zwischen P1 und P2. Diese ungewöhnlich scheinende Methode zur Angabe von Bogen ist eigentlich ziemlich nützlich, wenn Sie Figuren mit abgerundeten Ecken zeichnen wollen. Geben Sie für den Radius 0 an, zeichnet diese Methode eine gerade Linie vom aktuellen Punkt zum Punkt P1. Ist der Radius größer als null, wird eine gerade Linie vom aktuellen Punkt in Richtung von P1 gezeichnet, die sich dann in einem Kreis beugt, bis sie in Richtung P2 läuft.

- `bezierCurveTo()`

Die Methode fügt dem Teilpfad einen neuen Punkt P hinzu und verbindet diesen über eine kubische **Bezierkurve** mit dem aktuellen Punkt. Die Gestalt der Kurve wird durch zwei „Kontrollpunkte“ C1 und C2 bestimmt. Am Anfang der Kurve (am aktuellen Punkt) läuft die Kurve in Richtung von C1. Das Ende der Kurve (der Punkt P) wird aus Richtung von C2 erreicht. Zwischen diesen Punkten ändert sich die Richtung der Kurve stetig. Der Punkt P wird der neue aktuelle Punkt für den Teilpfad.

- `quadraticCurveTo()`

Diese Methode ähnelt `bezierCurveTo()`, nutzt aber eine quadratische Bezierkurve statt einer kubischen Bezierkurve und hat nur einen einzigen Kontrollpunkt.

Mit diesen Methoden können Sie Pfade wie die in Abbildung 1 zeichnen.

Beispiel zeigt den Code, mit dem Abbildung 1 erstellt wurde. Die in diesem Code illustrierten Methoden zählen zu den kompliziertesten Methoden in der Canvas-API.
Einem Pfad Kurven hinzufügen

```
// Eine Hilfsfunktion, die Grad in Radiant umwandelt

function rads(x) { return Math.PI*x/180; }

// Einen Kreis zeichnen. Eine Ellipse erhalten Sie durch
// Skalierung und Drehung. Es gibt keinen aktuellen Punkt.
// Es wird also nur ein Kreis ohne gerade Linie vom aktuellen
// Punkt zum Anfang des Kreises gezeichnet.
c.beginPath();
c.arc(75,100,50,              // Mittelpunkt bei (75,100),
                              // Radius 50
0,rads(360),false);           // Mit der Uhr von 0 zu 360° gehen

// Ein Kreissegment zeichnen. Winkel werden im Uhrzeigersinn
// von der positiven x-Achse gemessen. Beachten Sie, dass
// arc() eine Linie vom aktuellen Punkt zum Anfang des Bogens
// zeichnet.
c.moveTo(200, 100);           // Im Kreismittelpunkt beginnen
c.arc(200, 100, 50,           // Kreismittelpunkt und Radius,
rads(-60), rads(0),           // bei -60° beginnen und nach 0°
                              // gehen
false);                       // false bedeutet entgegen dem
                              // Uhrzeigersinn
c.closePath();                // Zum Kreismittelpunkt zurückkehren
```

```
// Gleiches Kreissegment, andere Richtung
c.moveTo(325, 100);
c.arc(325, 100, 50, rads(-60), rads(0), true);
c.closePath();

// Mit arcTo() gerundete Ecken erzeugen. Hier zeichnen wir ein
// Rechteck mit oberer linker Ecke bei (400,50) und
// verschiedenen Eckradien.
c.moveTo(450, 50);            // In der Mitte der Oberseite
                              // beginnen
c.arcTo(500,50,500,150,30);   // Oberseite und Ecke oben rechts
c.arcTo(500,150,400,150,20);  // Rechte Seite und Ecke unten
                              // rechts
c.arcTo(400,150,400,50,10);   // Unterseite und Ecke unten
                              // links
c.arcTo(400,50,500,50,0);     // Linke Seite und Ecke oben
                              // links
c.closePath();                // Der Rest der Oberseite

// Quadratische Bezierkurve: ein Kontrollpunkt
c.moveTo(75, 250);                        // Anfang bei (75,250)
c.quadraticCurveTo(100,200, 175, 250);    // Kurve nach (175,250)
c.fillRect(100-3,200-3,6,6);              // Den Kontrollpunkt markieren

// Kubische Bezierkurve: zwei Kontrollpunkte
c.moveTo(200, 250);                       // Startpunkt
c.bezierCurveTo(220,220,280,280,300,250); // Nach (300,250)
c.fillRect(220-3,220-3,6,6);              // Kontrollpunkte markieren
c.fillRect(280-3,280-3,6,6);

// Einige Grafikattribute setzen und die Kurven zeichnen
c.fillStyle = "#aaa"; // Graue Füllung
c.lineWidth = 5;      // 5 Pixel breite, (standardmäßig)
schwarze Striche
c.fill();             // Die Kurven füllen
c.stroke();           // Die Umrisse nachziehen
```

Rechtecke

CanvasRenderingContext2D definiert vier Methoden zum Zeichnen von **Rechtecken**. Beispiel nutzt eine davon, `fillRect()`, um die Kontrollpunkte der Bezierkurven zu markieren. Alle vier Rechteckmethoden erwarten zwei Argumente, die eine Ecke des Rechtecks angeben, sowie zwei, die die Breite und die Höhe des Rechtecks angeben. Üblicherweise geben Sie die linke obere Ecke an und dann eine positive Breite und Höhe, aber Sie können auch andere Ecken und negative Maße bestimmen.

`fillRect()` füllt das angegebene Rechteck mit dem aktuellen `fillStyle`. `strokeRect()` zieht den

Umriss des angegebenen Rechtecks mit dem aktuellen `strokeStyle` und anderen Strichattributen. `clearRect()` ist wie `fillRect()`, ignoriert aber den aktuellen Füllstil und füllt das Rechteck mit transparenten schwarzen Pixeln (der Standardfarbe eines vollständig leeren Canvas). Das Wichtige an diesen drei Methoden ist, dass sie sich nicht auf den aktuellen Pfad oder den aktuellen Punkt auf diesem Pfad auswirken.

Die letzte Rechteckmethode heißt `rect()` und wirkt sich auf den aktuellen Pfad aus: Sie fügt das definierte Rechteck in einem eigenen Teilpfad dem Pfad hinzu. Wie andere Methoden zur Pfaddefinition zeichnet oder füllt auch diese von allein nichts.

Linien zeichnen und Polygone füllen

Wenn Sie Linien in ein **Canvas** zeichnen und die von ihnen eingeschlossenen Flächen füllen wollen, definieren Sie zunächst einen Pfad. Ein Pfad ist eine Folge von einem oder mehreren Teilpfaden. Ein Teilpfad ist eine Folge von zwei oder mehr Punkten, die durch Liniensegmente (oder, wie wir später sehen werden, Kurvensegmente) verbunden sind. Einen neuen Pfad beginnen Sie mit der Methode `beginPath()`. Einen neuen Teilpfad beginnen Sie mit der Methode `moveTo()`. Wenn Sie den Startpunkt eines Teilpfads mit `moveTo()` eingerichtet haben, können Sie diesen Punkt über eine Gerade mit einem neuen Punkt verbinden, indem Sie die Methode `lineTo()` aufrufen. Der folgende Code definiert einen Pfad mit zwei Liniensegmenten:

```
c.beginPath();      // Einen neuen Pfad beginnen
c.moveTo(20, 20);   // Einen Teilpfad bei (20,20) beginnen
c.lineTo(120, 120); // Eine Linie nach (120,120) ziehen
c.lineTo(20, 120);  // Eine weitere nach (20,120)ziehen
```

Der vorangehende Code definiert nur einen Pfad. Er zeichnet noch nichts auf das Canvas. Rufen Sie die Methode `stroke()` auf, um die beiden Liniensegmente im Pfad zu zeichnen (oder „zu ziehen“). Rufen Sie die Methode `fill()` auf, um die von diesen Liniensegmenten definierte Fläche zu füllen:

```
c.fill();   // Einen dreieckigen Bereich füllen
c.stroke(); // Die beiden Seiten des Dreiecks zeichnen
```

Der vorangehende Code (sowie etwas zusätzlicher Code, der die Strichbreite und die Füllfarbe setzt) erzeugt die in Abbildung 1 gezeigte Zeichnung.

Beachten Sie, dass der oben definierte Teilpfad „offen“ ist. Er besteht aus zwei Liniensegmenten, deren Endpunkt nicht wieder mit dem Startpunkt verbunden ist. Das bedeutet, dass der Pfad keine Fläche einschließt. Die Methode `fill()` füllt offene Teilpfade, indem sie so tut, als wäre der Endpunkt über eine gerade Linie mit dem Startpunkt verbunden. Deswegen füllt der vorangehende Code ein Dreieck, zeichnet

aber nur zwei Seiten dieses Dreiecks.

Wenn alle drei Seiten des zuvor gezeigten Dreiecks gezeichnet werden sollen, müssen Sie die Methode `closePath()` aufrufen, um den Endpunkt des Teilpfads mit dem Startpunkt zu verbinden. (Sie könnten auch `lineTo(20,20)` aufrufen, hätten damit aber drei Liniensegmente, die dann einen Start- und Endpunkt teilen, aber nicht wirklich geschlossen sind. Zeichnen Sie breite Linien, ist das sichtbare Ergebnis besser, wenn Sie `closePath()` nutzen.)

Es gibt zwei weitere wichtige Punkte, die Sie sich in Bezug auf `stroke()` und `fill()` merken sollten. Zunächst operieren beide Methoden auf allen Teilpfaden des aktuellen Pfads. Angenommen, wir hätten einen weiteren Teilpfad im Code:

```
c.moveTo(300,100); // Einen neuen Teilpfad bei (300,100) beginnen
c.lineTo(300,200); // Eine vertikale Linie nach (300,200) ziehen
```

Würden wir jetzt `stroke()` aufrufen, würden wir zwei verbundene Schenkel eines Dreiecks zeichnen und eine nicht verbundene vertikale Linie.

Der zweite Punkt ist, dass weder `stroke()` noch `fill()` den aktuellen Pfad ändern: Sie können `fill()` aufrufen, und der Pfad ist immer noch da, wenn Sie `stroke()` aufrufen. Wenn Sie die Arbeit mit dem Pfad abgeschlossen haben und einen anderen Pfad eröffnen wollen, dürfen Sie nicht vergessen, `beginPath()` aufzurufen. Tun Sie das nicht, fügen Sie dem bestehenden Pfad neue Teilpfade hinzu und zeichnen womöglich immer wieder diese Teilpfade.

Beispiel 1 definiert eine Funktion zum Zeichnen gleichseitiger **Polygone** und illustriert die Verwendung von `moveTo()`, `lineTo()` und `closePath()` zur Definition von Teilpfaden sowie `fill()` und `stroke()` zum Zeichnen dieser Pfade. Es erzeugt die in Abbildung 1 gezeigte Zeichnung.

Beispiel 1: Gleichseitige Polygone mit `moveTo()`, `lineTo()` und `closePath()`

```
// Definiert ein gleichseitiges Polygon mit n Seiten, beim
// Mittelpunkt (x,y) mit dem Radius r. Die Ecken werden
// gleichmäßig auf dem Rand eines Kreises verteilt. Die erste
// Ecke wird über dem Mittelpunkt oder beim angegebenen
// Winkel gezeichnet. Die Linien werden im Uhrzeigersinn
// gezogen, es sei denn, das letzte Argument ist true.
function polygon(c,n,x,y,r,angle,counterclockwise) {
angle = angle || 0;
counterclockwise = counterclockwise || false;

// Die Position der Ecke berechnen und dort einen Teilpfad beginnen
c.moveTo(x + r*Math.sin(angle),
y - r*Math.cos(angle));
```

```
var delta = 2*Math.PI/n;     // Der Winkel zwischen den Seiten
for(var i = 1; i < n; i++) { // Für die verbleibenden Ecken
// Winkel dieser Seite berechnen
angle += counterclockwise?-delta:delta;

// Die Position einer Ecke berechnen und eine Linie dorthin ziehen
c.lineTo(x + r*Math.sin(angle),
y - r*Math.cos(angle));
}
c.closePath(); // Die letzte Ecke mit der ersten verbinden
}

// Einen neuen Pfad beginnen und ihm Polygon-Teilpfade hinzufügen
c.beginPath();
polygon(c, 3, 50, 70, 50);               // Dreieck
polygon(c, 4, 150, 60, 50, Math.PI/4);   // Quadrat
polygon(c, 5, 255, 55, 50);              // Fünfeck
polygon(c, 6, 365, 53, 50, Math.PI/6);   // Sechseck

// Ein kleines Rechteck gegen die Uhr in das Sechseck zeichnen
polygon(c, 4, 365, 53, 20, Math.PI/4, true);

// Eigenschaften setzen, die steuern, wie die Grafik aussieht
c.fillStyle = "#ccc";   // Hellgraues Inneres,
c.strokeStyle = "#008"; // umrahmt von dunkelblauen Linien
c.lineWidth = 5;        // mit fünf Pixeln Breite.

// Jetzt alle Polygone zeichnen (jedes im eigenen Teilpfad)
c.fill();   // Die Figuren füllen
c.stroke(); // Die Ränder zeichnen
```

Beachten Sie, dass dieses Beispiel ein Sechseck zeichnet, das ein Quadrat einschließt. Das Quadrat und das Sechseck werden durch separate Teilpfade gebildet, die sich überschneiden. Wenn das passiert (oder wenn es innerhalb eines Teilpfads Überschneidungen gibt), muss das Canvas ermitteln können, welche Bereiche im Pfad sind und welche außerhalb des Pfads.

Das Canvas nutzt einen Test, der als **Nonzero Winding-Regel** bezeichnet wird. In diesem Fall wird das Innere des Quadrats nicht gefüllt, da Quadrat und Sechseck in umgekehrter Richtung gezeichnet wurden: Die Ecken des Sechsecks wurden im Uhrzeigersinn durch Liniensegmente verbunden, die Ecken des Quadrats entgegen dem Uhrzeigersinn. Wären die Ecken des Quadrats ebenfalls im Uhrzeigersinn verbunden worden, hätte der `fill()`-Aufruf auch das Innere des Quadrats gefüllt.
Die Nonzero Winding-Regel

Ob sich ein Punkt P innerhalb eines Pfads befindet, testen Sie mit der Nonzero Winding-Regel, indem Sie sich einen Strahl vorstellen, der von P aus in beliebiger Richtung bis ins Unendliche geht (oder, was praktikabler ist, bis zu einem Punkt außerhalb des Rahmenrechtecks des Pfads). Initialisieren Sie jetzt einen Zähler auf null und zählen Sie alle Punkte, an denen der Pfad den Strahl schneidet. Erhöhen Sie den Zähler um eins, wenn der Pfad den Strahl im Uhrzeigersinn schneidet. Verringern Sie die Zähler um eins, wenn der Pfad den Strahl entgegen dem Uhrzeigersinn schneidet. Ist der Zähler nach Auszählung aller Schnittpunkte ungleich null, befindet sich der Punkt innerhalb des Pfads. Ist der Zähler null, befindet sich der Punkt außerhalb des Pfads.

Logos zeichnen

Das `canvas`-Element ist genauso wie das `script`-Element ein Container, sozusagen eine leere Tafel, auf der wir zeichnen können. So definieren Sie ein `canvas`-Element mit einer bestimmten Breite und Höhe:

```
<canvas id="my_canvas" width="150" height="150">
Ausweichlösung hierhin
</canvas>
```

Leider können Sie die Breite und Höhe eines `canvas`-Elements nicht mit CSS anpassen, ohne die Inhalte zu verzerren. Insofern müssen Sie sich also schon bei der Deklaration für die Maße Ihres `canvas`-Elements entscheiden.

> Wir verwenden JavaScript, um Formen darauf zu zeichnen.

Selbst wenn Sie für Browser ohne `canvas`-Element alternativen Inhalt bereitstellen, müssen Sie dennoch verhindern, dass dieser durch den **JavaScript-Code** verändert wird. Suchen Sie das `canvas`-Element anhand seiner ID, und prüfen Sie, ob der Browser die `getContext`-Methode des `canvas`-Elements unterstützt.

```
var canvas = document.getElementById("my_canvas");
if (canvas.getContext){
var context = canvas.getContext("2d");
}else{
// Versteckten Inhalt des <canvas>-Elements anzeigen oder
// den Browser den darin enthaltenen Text anzeigen lassen.
}
```

Wenn die `getContext`-Methode etwas zurückliefert, rufen wir den 2D-Kontext des `canvas`-Elements auf, damit wir Objekte hinzufügen können. Wenn wir keinen Kontext erhalten, müssen wir eine Möglichkeit finden, den Alternativinhalt anzuzeigen. Da wir wissen, dass das `canvas`-Element nur mit JavaScript funktioniert, bauen wir von Anfang an ein Framework für die Ausweichlösung auf.

Sobald Sie über den Kontext des canvas-Elements verfügen, können Sie Elemente dazu hinzufügen. Ein rotes Rechteck fügen Sie zum Beispiel ein, indem Sie die Füllfarbe festlegen und dann ein Rechteck erstellen:

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
```

Der **2D-Kontext** des canvas-Elements ist ein Raster, dessen standardmäßiger Ursprung die linke obere Ecke ist. Zum Zeichnen einer Form geben Sie die x– und y-Koordinaten sowie Breite und Höhe an.

Jede Form wird in einer eigenen Ebene gezeichnet,

sodass Sie beispielsweise auch drei Rechtecke mit einem Versatz von je 10 Pixel erstellen können:

```
context.fillStyle = "rgb(200,0,0)";
context.fillRect (10, 10, 100, 100);
context.fillStyle = "rgb(0,200,0)";
context.fillRect (20, 20, 100, 100);
context.fillStyle = "rgb(0,0,200)";
context.fillRect (30, 30, 100, 100);
```

Die Dreiecke werden dann auch entsprechend übereinander geschichtet:

Da Sie nun die Grundlagen bereits verstehen, stellen wir gleich das Logo zusammen. Wie Sie in Abbildung sehen, ist es relativ simpel.

Das Logo zeichnen

Das Logo besteht aus einem Schriftzug, einem gewinkelten Pfad, einem Quadrat und einem Dreieck. Erstellen wir ein neues **HTML5-Dokument** mit einem canvas-Element und einer JavaScript-Funktion zum Zeichnen des Logos, die ermittelt, ob wir das 2D-Canvas verwenden können.

```
var drawLogo = function(){
var canvas = document.getElementById("logo");
var context = canvas.getContext("2d");
```

```
};
```

Diese Methode rufen wir aber erst auf, nachdem wir geprüft haben, ob das `canvas`-Element existiert:

```
$(function(){
var canvas = document.getElementById("logo");
if (canvas.getContext){
drawLogo();
}
});
```

Beachten Sie, dass wir auch hier wieder die **jQuery-Funktion** verwenden, um sicherzustellen, dass das Event ausgelöst wird, wenn das Dokument bereit ist. Wir suchen ein Element mit der ID `logo` auf der Seite, also sollten wir dafür sorgen, dass unser Dokument auch ein `canvas`-Element enthält. Denn nur so können wir es finden und unsere Prüfung durchführen.

```
<canvas id="logo" width="900" height="80">
<h1>AwesomeCo</h1>
</canvas>
```

Als Nächstes fügen wir unser Text zum `canvas`-Element hinzu.

Text einfügen

Um Text zum `canvas`-Element hinzuzufügen, müssen wir eine Schrift, eine Schriftgröße sowie die Ausrichtung wählen und anschließend den Text auf die entsprechenden Koordinaten im Raster anwenden. So fügen wir den Text in unser `canvas`-Element ein:

```
context.font = "italic 40px sans-serif";
context.textBaseline = "top";
context.fillText("AwesomeCo", 60, 0);
```

Wir definieren den Schrifttyp und legen die Grundlinie bzw. die vertikale Ausrichtung fest, bevor wir den Text in das `canvas`-Element einfügen. Wir verwenden die `fillText`-Methode und erhalten Text, der mit der Füllfarbe gefüllt ist und den wir 60 Pixel nach rechts platzieren, damit wir Platz für den großen dreieckigen Pfad haben, den wir als Nächstes zeichnen.

Linien zeichnen

Linien können wir auf dem `canvas`-Element ganz einfach mit einer Runde „Malen nach Zahlen“ zeichnen: Wir geben einen Anfangspunkt im Raster an und legen dann zusätzliche Punkte im Raster fest, die mit Punkten verbunden werden:

Wir verwenden die Methode `beginPath()`, um mit dem Zeichnen einer Linie zu beginnen, und erstellen dann unseren Pfad:

```
context.lineWidth = 2;
context.beginPath();
context.moveTo(0, 40);
context.lineTo(30, 0);
context.lineTo(60, 40);
context.lineTo(285, 40);
context.stroke();
context.closePath();
```

Wenn wir damit fertig sind, uns über das `canvas`-Element zu bewegen, rufen wir die Methode `stroke` auf, um die Linie zu zeichnen. Anschließend rufen wir die Methode `closePath` auf, um den Zeichenvorgang zu beenden.

> Bleibt also nur noch die Kombination aus Rechteck und Dreieck innerhalb des größeren Dreiecks.

Den Ursprung verschieben

Innerhalb des größeren Dreiecks müssen wir ein kleines Quadrat und ein kleines Dreieck zeichnen. Wenn wir Formen und Pfade zeichnen, geben wir die x– und y-Koordinaten relativ zum Ursprung in der linken oberen Ecke des `canvas`-Elements an. Wir können den Ursprung aber auch an eine neue Position verschieben.

Zum Zeichnen des inneren kleinen Quadrats verschieben wir den Ursprung:

```
context.save();
context.translate(20,20);
context.fillRect(0,0,20,20);
```

Beachten Sie, dass wir vor dem Verschieben des Ursprungs die Methode `save()` aufrufen. Hierdurch wird der bisherige Zustand des `canvas`-Elements gespeichert, sodass wir jederzeit Änderungen rückgängig machen können. Das funktioniert wie ein Wiederherstellungspunkt – Sie können sich das wie einen Stapel vorstellen. Jedes Mal, wenn Sie `save()` aufrufen, erhalten Sie einen neuen Eintrag. Wenn wir fertig sind, rufen wir `restore()` auf, wodurch die jeweils letzte Speicherung auf dem Stapel wiederhergestellt wird.

Nun zeichnen wir mit Pfaden das innere Dreieck. Aber statt Linien verwenden wir eine Füllung, um den Eindruck zu erwecken, dass das Dreieck aus dem Quadrat „herausgeschnitten“ wird.

```
context.fillStyle = "#fff";
context.strokeStyle = "#fff";
context.lineWidth = 2;
context.beginPath();
context.moveTo(0, 20);
context.lineTo(10, 0);
context.lineTo(20, 20);
context.lineTo(0, 20);
context.fill();
context.closePath();
context.restore();
```

In diesem Fall legen wir für die Striche und die Füllung die Farbe Weiß (`#fff`) fest, bevor wir mit dem Zeichnen beginnen. Anschließend zeichnen wir unsere Linien. Nachdem wir zuvor den Ursprung verschoben haben, bewegen wir uns nun relativ zur oberen linken Ecke des zuvor gezeichneten Quadrats.

Wir sind beinahe fertig, aber es fehlt noch ein bisschen Farbe.

Farben

Im Abschnitt Den Ursprung verschieben haben Sie kurz gesehen, wie Sie die Striche und die Füllfarbe für die Zeichenwerkzeuge festlegen. Wir könnten beispielsweise für alles die Farbe Rot wählen, indem wir vor dem Zeichnen den folgenden Code einfügen:

```
context.fillStyle = "#f00";
context.strokeStyle = "#f00";
```

Aber das wäre ein bisschen langweilig. Stattdessen erzeugen wir Verläufe und weisen diese den Strichen und Füllungen zu:

```
var gradient = context.createLinearGradient(0, 0, 0, 40);
gradient.addColorStop(0, "#a00"); // Rot
gradient.addColorStop(1, "#f00"); // Rot
context.fillStyle = gradient;
context.strokeStyle = gradient;
```

Wir erstellen einfach ein **Gradient-Objekt** und legen die Farbwerte fest. In diesem Beispiel bewegen wir uns nur zwischen zwei Rotschattierungen. Wir könnten aber auch den gesamten Regenbogen zeichnen, wenn wir das wollten.

Beachten Sie, dass wir die Farbe festlegen müssen, bevor wir etwas zeichnen.

Damit ist unser Logo fertig und wir haben ein besseres Verständnis davon, wie wir einfache Formen im `canvas`-Element zeichnen können. Allerdings bietet der Internet Explorer vor Version 9 keine Unterstützung für das `canvas`-Element. Darum müssen wir uns kümmern.

Ausweichlösung

Google hat eine Bibliothek mit dem Namen `ExplorerCanvas` veröffentlicht, die den größten Teil der **Canvas-API** auch Benutzern des Internet Explorer zur Verfügung stellt. Wir müssen lediglich diese Bibliothek in unsere Seite einbinden:

```
<!--[if lte IE 8]>
<script src="javascripts/excanvas.js"></script>
<![endif]-->
```

Und schon sollte alles auch im Internet Explorer funktionieren – tut es aber nicht. Als dieser Artikel geschrieben wurde, hat die neueste stabile Version das Einfügen von Text noch gar nicht unterstützt. Und die Versionen aus dem Subversion-Repository verwenden falsche Schriftarten. Außerdem bietet diese Bibliothek noch keine Unterstützung für Farbverläufe von Strichen.

Also müssen wir auf andere Lösungen zurückgreifen, wie etwa ein PNG des Logos innerhalb des `canvas`-Elements. Oder wir verwenden das `canvas`-Element überhaupt nicht. Dieser Teil sollte nur eine Übung sein, um Ihnen zu zeigen, wie Sie zeichnen können. Es ist also kein Weltuntergang, wenn wir dieses bestimmte Beispiel noch nicht in einem plattformübergreifenden Produktionssystem einsetzen können.

Muster

Zur Festlegung eigener Muster für Füllungen und Linien stellt die Spezifikation die Methode `createPattern()` zur Verfügung, die ähnlich wie `drawImage()` sowohl `image`– als auch `canvas`– oder `video`-Elemente als Input akzeptiert und im Parameter `repetition` die Art der **Musterwiederholung** definiert.

```
context.createPattern(image, repetition)
```

Gültige Werte für das `repetition`-Argument sind, wie schon vom `background-color`-Attribut der **CSS-Spezifikation** her bekannt, `repeat`, `repeat-x`, `repeat-y`, und `no-repeat`. Verwenden wir wieder die 16 benannten Grundfarben, können wir über ein paar Zeilen Code Schachbrettmuster mit jeweils zwei zusammenpassenden Farbpaaren erzeugen.

Das Pattern selbst erzeugen wir als `in-memory`-Canvas mit 20 x 20 Pixeln Breite und vier 10×10 Pixel großen Quadraten. Am Beispiel des grünen Musters sieht dieser Schritt folgendermaßen aus:

```
var cvs = document.createElement("CANVAS");
cvs.width = 20;
```

```
cvs.height = 20;
var ctx = cvs.getContext('2d');
ctx.fillStyle = 'lime';
ctx.fillRect(0,0,10,10);
ctx.fillRect(10,10,10,10);
ctx.fillStyle = 'green';
ctx.fillRect(10,0,10,10);
ctx.fillRect(0,10,10,10);
```

Mit `createPattern()` definieren wir dann den **Canvas** cvs als sich wiederholendes Muster, weisen ihn dem Attribut `fillStyle` zu und füllen damit das Quadrat.

```
context.fillStyle = context.createPattern(cvs,'repeat');
context.fillRect(0,0,220,220);
```

Patterns sind am Koordinatenursprung verankert und werden ab dort aufgetragen. Würden wir im obigen Beispiel `fillRect()` nicht bei 0/0, sondern um zehn Pixel nach rechts versetzt bei 10/0 beginnen, wäre dementsprechend Dunkelgrün (`green`) und nicht Hellgrün (`lime`) die erste Farbe links oben.

Neben selbst gestalteten `canvas`-Elementen können auch Bilder als Quelle für **Patterns** verwendet werden. Abbildung 1 zeigt ein kleines Beispiel dafür und verwendet `createPattern()` zur Füllung des Hintergrundes, als Muster für die Titelschrift und zum Ausstechen einzelner Ausschnitte aus dem bereits bestens bekannten Yosemite-Bild. Die beiden anderen Bilder, pattern_107.png und pattern_125.png, sind Teil der Squidfinger-Pattern-Bibliothek, in der 160 weitere ansprechende Muster zum Download bereitstehen.

Sehen wir uns zuerst an, wie der Hintergrund zustande kommt:

```
var bg = new Image();
bg.src = 'icons/pattern_125.png';
bg.onload = function() {
context.globalAlpha = 0.5;
context.fillStyle = context.createPattern(bg,'repeat');
context.fillRect(0,0,canvas.width,canvas.height);
};
```

Die ersten beiden Zeilen erzeugen wieder ein neues **Image-Objekt** und setzen dessen `src`-Attribut auf das Bild pattern_125.png im Verzeichnis icons. Genau wie bei `drawImage()` müssen wir vor dem Definieren des Patterns sicherstellen, dass das gewünschte Bild auch wirklich geladen ist. Die Funktion

`bg.onload()` enthält dabei den eigentlichen Code zum Generieren des sich wiederholenden Musters, das wir mit 50 % Opazität auf der gesamten Canvas-Fläche auftragen. Über das gleiche Prozedere füllen wir den Titeltext Yosemite! mit pattern_107.png.

Für die überlappenden Bildausschnitte setzen wir kurzerhand das ganze Yosemite-Bild yosemite.jpg als Muster ein und arbeiten dann in einer `for`-Schleife das **Input-Array** `extents` ab, in dem sich x-, y-, `width`– und `height`-Werte der gewünschten Ausschnitte befinden. Über den Aufruf von `fillRect()` wird der entsprechende Bildbereich als Füllmuster gezeigt und mittels `strokeText()` mit einem zusätzlichen Rahmen versehen.

```
var extents = [
{ x:20,y:50,width:120,height:550 } // und 7 weitere ...
];
var image = new Image();
image.src = 'images/yosemite.jpg';
image.onload = function() {
context.fillStyle = context.createPattern(
image,'no-repeat'
);
for (var i=0; i<extents.length; i++) {
var d = extents ; // short-cut
context.fillRect(d.x,d.y,d.width,d.height);
context.strokeRect(d.x,d.y,d.width,d.height);
}
};
```

Da in Abbildung 1 drei verschiedene Bilder zum Zuge kommen und alle drei vor ihrer Verwendung vollständig geladen sein müssen, bleibt uns nichts anderes übrig, als die drei `onload`-Funktionen ineinander zu verschachteln. Nur so behalten wir die Kontrolle über die Reihenfolge beim Zeichnen. Der Pseudo-Code für eine mögliche Schachtelung sieht so aus:

```
// alle Bilder erzeugen
bg.onload = function() {
// Hintergrund zeichnen
image.onload = function() {
// Bildausschnitte hinzufügen
pat.onload = function() {
// Titel mit Muster füllen
};
};
};
```

Die einzige Möglichkeit, dieses Verschachteln zu vermeiden, wäre, alle beteiligten Bilder im **HTML-Code**

der Seite als über `visibility:hidden` versteckte `img`-Elemente zu verlinken und mit `getElementById()` oder `getElementsByTag-Name()` nach dem Laden der Seite in `window.onload()` zu referenzieren.

Bleibt abschließend noch festzuhalten, dass bei der Verwendung eines `video`-Elements als Quelle für `createPattern()` analog zur `drawImage()`-Methode der erste Frame des Videos beziehungsweise der Poster-Frame, sofern vorhanden, als Muster zum Einsatz kommt.

Pfade

Die Abläufe beim Erstellen von Pfaden in **Canvas** sind vergleichbar mit dem Zeichnen auf einem Blatt Papier: Stift an einer Position am Blatt absetzen, Zeichnen, Stift wieder anheben und nach Belieben an anderer Stelle weiterzeichnen. Zeicheninhalte können dabei von einfachen Linien über komplexe Kurven bis hin zu daraus gebildeten Polygonen reichen. Ein erstes Beispiel verdeutlicht dieses Konzept und übersetzt die Schritte beim Schreiben des Buchstabens A in Canvas-Pfadbefehle:

```
context.beginPath();
context.moveTo(300,700);
context.lineTo(600,100);
context.lineTo(900,700);
context.moveTo(350,400);
context.lineTo(850,400);
context.stroke();
```

Sehen wir uns den Quellcode für dieses Beispiel näher an, so erkennen wir drei Phasen der Pfaderstellung:

- 1. Initialisieren eines neuen Pfades mit `beginPath()`
- 2. Definieren der Pfadgeometrie durch `moveTo()` und `lineTo()`-Aufrufe
- 3. Zeichnen der Linien mit `stroke()`

Jeder Pfad muss mit `beginPath()` initialisiert werden und kann dann beliebig viele Segmente enthalten. In unserem Beispiel sind es zwei, die die Bewegungen der Hand beim Schreiben über Kombinationen von `moveTo()` und `lineTo()` nachbilden. Somit entsteht zuerst die Dachform und dann die horizontale Linie des Buchstabens A. Mit `stroke()` wird schließlich der zuvor definierte Pfad auf die Canvas-Fläche gezeichnet.

> Die Entscheidung, ob und wann Segmente eines Pfades in mehrere einzelne Pfade aufgetrennt werden,

hängt einzig und allein vom Layout ab, denn jeder Pfad kann nur in seiner Gesamtheit formatiert werden. Sollte die horizontale Linie des Buchstabens also eine eigene Farbe bekommen, müssten auch zwei **Pfade** definiert werden.

Wenden wir uns nun den wichtigsten Pfad-Zeichenmethoden im Detail zu.

Linien

Zur Konstruktion von Linienzügen wie im Buchstaben-Beispiel stellt Canvas die Methode `lineTo()` zur Verfügung:

```
context.lineTo(x, y)
```

Übersetzt bedeutet das so viel wie „Linie zum Punkt x/y„, womit klar wird, dass der Ausgangspunkt schon vorher über `moveTo()` oder als Endpunkt der letzten Zeichenoperation existieren muss. Nach dem Zeichnen wird die Koordinate x/y zum neuen aktuellen Punkt.

Info

Bei allen Grafiken zur Erklärung der Pfad-Zeichenmethoden sind der **Ausgangspunkt** x0/y0 in Hellgrau und der neue aktuelle Punkt in fetter Schrift ausgewiesen.

Bezierkurven

Canvas kennt zwei Arten von **Bezierkurven**: quadratische und kubische, die fälschlicherweise nur als `bezierCurveTo()` bezeichnet werden.

```
context.quadraticCurveTo(cpx, cpy, x, y)
context.bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)
```

Zur Konstruktion der Bezierkurven benötigen wir neben dem aktuellen Punkt als Ausgangskoordinate die Zielkoordinate und je nach Kurvenart einen oder zwei Kontrollpunkte. Neuer aktueller Punkt nach dem Zeichnen ist in beiden Fällen die Koordinate x/y.

Kreisbögen

Etwas schwieriger zu verstehen sind mit Sicherheit die Methoden zum Konstruieren von Kreisbögen, den sogenannten `arcs`. Die erste davon definiert sich über zwei Koordinaten und einen Radius:

```
context.arcTo(x1, y1, x2, y2, radius)
```

Wie aus Abbildung 1 erkennbar ist, konstruiert `arcTo()` den neuen Pfad auf folgende Weise: An den Linienzug von x0/y0 über x1/y1 nach x2/y2 wird ein Kreis mit gegebenem Radius so angelegt, dass er die Linien in genau zwei Punkten, der Starttangente t1 und Endtangente t2, schneidet. Der Bogen zwischen diesen beiden Punkten wird Teil des Pfades, und die Endtangente t2 wird zum neuen aktuellen Punkt.

In der Praxis gut einsetzbar ist diese Methode für Rechtecke mit abgerundeten Ecken – eine wiederverwendbare Funktion, die das erledigt, ist da nicht fehl am Platz:

```
var roundedRect = function(x,y,w,h,r) {
```

```
context.beginPath();
context.moveTo(x,y+r);
context.arcTo(x,y,x+w,y,r);
context.arcTo(x+w,y,x+w,y+h,r);
context.arcTo(x+w,y+h,x,y+h,r);
context.arcTo(x,y+h,x,y,r);
context.closePath();
context.stroke();
};
roundedRect(100,100,700,500,60);
roundedRect(900,150,160,160,80);
roundedRect(700,400,400,300,40);
roundedRect(150,650,400,80,10);
```

Die Funktion `roundedRect()` erwartet neben den Basiswerten für das Rechteck selbst den Radius zum Abrunden und zeichnet dann mit einer `moveTo()`-, vier `arcTo()`– und einer `closePath()`-Methode das gewünschte Rechteck. Die Methode `closePath()` kennen Sie noch nicht: Sie verbindet den letzten Punkt wieder mit dem Anfangspunkt und bewirkt damit das Schließen des Rechtecks.

Auf den ersten Blick noch komplizierter erscheint die zweite Variante, Kreisbögen zu erstellen: die Methode `arc()`. Neben Zentrum und Radius müssen jetzt auch noch zwei Winkel und die Drehrichtung übergeben werden.

```
context.arc(x, y, radius, startAngle, endAngle, anticlockwise)
```

Der Ausgangspunkt für den `arc` in Abbildung 2 ist das Zentrum eines Kreises mit gegebenem Radius. Von dort aus werden über die Winkel `startAngle` und `endAngle` zwei Zeiger konstruiert, die den Kreis in zwei Punkten schneiden. Die Richtung des Kreisbogens zwischen diesen beiden Koordinaten bestimmt der Parameter anticklockwise, wobei 0 für im Uhrzeigersinn und 1 für gegen den Uhrzeigersinn steht.

Der resultierende `arc` beginnt somit im Zentrum des Kreises bei x0/y0,

verbindet dieses in einer gerade Linie mit dem ersten Schnittpunkt spx/spy und zeichnet von dort aus den Kreisbogen zum Endpunkt epx/epy, der dann zum neuen aktuellen Punkt wird.

Der größte Wermutstropfen bei der Konstruktion von `arcs` ist, dass alle Winkel in Radiant statt Grad angegeben werden müssen. Ein kurzes Helferlein als Gedächtnisstütze bei der Umrechnung kann also nicht schaden:

```
var deg2rad = function(deg) {
return deg*(Math.PI/180.0);
};
```

Wenn wir schon bei Hilfsfunktionen sind, können wir gleich noch zwei weitere hinzufügen, die uns das Zeichnen von Kreisen und Kreissektoren erleichtern. Für Kreise zum Beispiel benötigen wir ja eigentlich nur Zentrum und Radius, den Rest soll unsere Funktion `circle()` erledigen:

```
var circle = function(cx,cy,r) {
context.moveTo(cx+r,cy);
context.arc(cx,cy,r,0,Math.PI*2.0,0);
};
```

Bei Kreisdiagrammen, auch Kuchen- oder Tortendiagramme genannt, sind vor allem die Winkelangaben in Radiant wenig intuitiv. Unsere Funktion `sector()` erledigt die lästige Arbeit der Umrechnung für uns und erlaubt, Start- und Endwinkel in Grad zu übergeben:

```
var sector = function(cx,cy,r,
startAngle,endAngle, anticlockwise
){ context.moveTo(cx,cy); context.arc(
cx,cy,r,
startAngle*(Math.PI/180.0),
endAngle*(Math.PI/180.0),
anticlockwise
);
context.closePath();
};
```

Wenige Zeilen Code genügen nun schon, um Kreise und Kreisdiagramme zu zeichnen und dabei nicht den Überblick zu verlieren. Abbildung 3 zeigt das Ergebnis.

```
context.beginPath();
circle(300,400,250);
circle(300,400,160);
circle(300,400,60);
sector(905,400,250,-90,30,0);
sector(900,410,280,30,150,0);
sector(895,400,230,150,270,0);
context.stroke();
```

Rechtecke
In der Handhabung unseren beiden Helferlein schon ähnlicher ist die Methode `rect()`, die damit auch etwas aus der Reihe tanzt.

```
context.rect(x, y, w, h)
```

Im Gegensatz zu allen bisherigen Pfadzeichenmethoden wird nämlich bei `rect()` der aktuelle Punkt x0/y0 beim Zeichnen gänzlich ignoriert und das Rechteck nur über die Parameter x, y, Breite w und Höhe h definiert. Neuer aktueller Punkt nach dem Zeichnen wird der Ursprungspunkt x/y.

Umrisse, Füllungen und Clipmasken
Wenn wir uns an die drei Phasen der Pfaderstellung mit Initialisierung, Festlegen der Pfadgeometrie und Zeichnen erinnern, dann sind wir jetzt bei der dritten und letzten Phase angekommen: dem Zeichnen. Hier fällt die Entscheidung, wie der fertige Pfad aussehen soll. Alle bisherigen Beispiele entschieden sich an dieser Stelle für eine einfache Umrisslinie, die durch folgende Methode erzeugt wird:

```
context.stroke()
```

Die Farbe der Linie bestimmt dabei das Attribut `strokeStyle`. Daneben lassen sich die Linienstärke (`lineWidth`), das Aussehen der Linienenden (`lineCap`) sowie das Erscheinungsbild der Verbindung zwischen Linien (`lineJoin`) über drei weitere **Canvas-Attribute** festlegen (der Stern kennzeichnet Defaultwerte und wird uns ab jetzt noch öfter begegnen).

```
context.lineWidth = [ Pixel ]
context.lineCap = [ *butt, round, square ]
context.lineJoin = [ bevel, round, *miter ]
```

Die Linienbreite `lineWidth` wird in Pixel angegeben, ihr Defaultwert liegt bei 1.0. Sie gilt, wie die beiden anderen Linienattribute, nicht nur für Linien und Polygone, sondern auch für Rechtecke, die mit `strokeRect()` erstellt wurden.

Das Linienende `lineCap` kann entweder gekappt (`butt`), rund (`round`) oder quadratisch (`square`) mit `butt` als Standardwert sein. Wird `round` verwendet, erfolgt die Abrundung der Linie durch Hinzufügen eines Halbkreises am Linienende mit halber `lineWidth` als Radius. Bei der Methode `square` wird der Halbkreis durch ein Rechteck mit halber Linienbreite als Höhe ersetzt.

Abgeschrägte Linienverbindungen erzeugt das Attribut `lineJoin` über `bevel;` für die Abrundung der Ecken steht `round` zur Verfügung, und spitze Eckverbindungen, vergleichbar einer Gehrung, erhält man durch `miter`, das gleichzeitig Defaultwert ist. Um zu verhindern, dass über `miter` verbundene Linien zu spitz werden, hält die Spezifikation das **Attribut** `miterLimit` mit Standardwert 10.0 bereit. Dabei handelt

es sich um das Verhältnis von Länge der Spitze (das ist der Abstand zwischen dem Schnittpunkt der Linien und der Spitze) zur halben Linienbreite. Wird `miterlimit` überschritten, erfolgt das Kappen der Spitze, wodurch der gleiche Effekt wie bei `bevel` entsteht.

Wollen wir Pfade mit einer Farbe oder Gradiente füllen, müssen wir zuvor das entsprechende Stilattribut über `fillStyle` setzen und im Anschluss daran die folgende Pfadmethode aufrufen:

```
context.fill()
```

So einfach das klingt, so kompliziert kann es werden, wenn sich Pfade selbst schneiden oder ineinander verschachtelt sind. Dann kommt die sogenannte Nonzero-Füllregel zum Tragen, die anhand der Laufrichtung der beteiligten Pfadbestandteile entscheidet, ob gefüllt wird oder nicht.

Abbildung 4 zeigt die **Nonzero-Regel** in Aktion. Links sind beide Kreise im Uhrzeigersinn gezeichnet, und rechts läuft der innere Kreis gegen den Uhrzeigersinn und bewirkt damit das Loch in der Mitte.

Beim Zeichnen der gerichteten Kreise hilft uns übrigens wieder das Helferlein aus dem `arc()`-Abschnitt, diesmal mit einer kleinen Modifikation: Die gewünschte Richtung wird nun als Argument übergeben. Gültige Werte für `anticlockwise` sind 0 und 1.

```
var circle = function(cx,cy,r,anticlockwise) {
context.moveTo(cx+r,cy);
context.arc(cx,cy,r,0,Math.PI*2.0,anticlockwise);
};
```

Der Code für den rechten Kreis mit Loch ergibt sich demnach folgendermaßen:

```
context.beginPath();
context.fillStyle = 'yellow';
circle(900,400,240,0);
circle(900,400,120,1);
context.fill();
context.stroke();
```

Nach `stroke()` und `fill()` fehlt uns damit nur noch eine einzige Methode beim Zeichnen von Pfaden, nämlich die Methode:

```
context.clip()
```

So kurz wie ihr Name fällt auch die Erklärung aus: `clip()` sorgt dafür, dass der definierte Pfad nicht gezeichnet, sondern als Ausstechform für alle weiteren Zeichenelemente verwendet wird. Alles, was innerhalb der Maske liegt, ist sichtbar, der Rest bleibt verborgen. Zurücksetzen kann man diese Maske, indem man eine weitere Clipmaske mit der gesamten **Canvas-Fläche** als Geometrie erstellt.

Pixelmanipulation

Die Methoden unserer Wahl, um **Pixelwerte** zu lesen und zu manipulieren, lauten `getImageData()`, `putImageData()` und `createImageData()`. Nachdem in allen dreien der Begriff ImageData heraussticht, gilt es diesen als ersten zu definieren.

Arbeiten mit dem ImageData-Objekt

Nähern wir uns dem ImageData-Objekt mit einem 2 x 2 Pixel großen Canvas, auf den wir vier 1 x 1 Pixel große, gefüllte Rechtecke in den benannten Farben `navy`, `teal`, `lime` und `yellow` zeichnen.

```
context.fillStyle = "navy";
context.fillRect(0,0,1,1);
context.fillStyle = "teal";
context.fillRect(1,0,1,1);
context.fillStyle = "lime";
context.fillRect(0,1,1,1);
context.fillStyle = "yellow";
context.fillRect(1,1,1,1);
```

Über die Methode `getImageData(sx, sy, sw, sh)` greifen wir im nächsten Schritt auf das **ImageData-Objekt** zu, wobei die vier Argumente den gewünschten Canvas-Ausschnitt als Rechteck festlegen.

```
ImageData = context.getImageData(
0,0,canvas.width,canvas.height
);
```

Das ImageData-Objekt selbst besitzt die Attribute `ImageData.width`, `ImageData.height` und `ImageData.data`, wobei sich hinter Letzterem die tatsächlichen Pixelwerte im sogenannten CanvasPixelArray verstecken. Dabei handelt es sich um ein flaches Array mit Rot-, Grün-, Blau- und Alpha-Werten für jedes Pixel im gewählten Ausschnitt, beginnend links oben, von links nach rechts und von oben nach unten. Die Anzahl aller Werte ist im Attribut `ImageData.data.length` gespeichert.

Mithilfe einer einfachen `for`-Schleife können wir nun die einzelnen Werte des **CanvasPixelArray** auslesen und mit `alert()` sichtbar machen. Beginnend bei 0, arbeiten wir uns von Pixel zu Pixel vor, indem wir nach jedem Schleifendurchgang den Zähler um 4 erhöhen. Die RGBA-Werte ergeben sich dann über Offsets von der aktuellen Position aus, wobei Rot beim Zähler i, Grün bei i+1, Blau bei i+2 und die Alpha-Komponente bei i+3 zu finden ist.

```
for (var i=0; i<ImageData.data.length; i+=4) {
var r = ImageData.data ;
var g = ImageData.data ;
var b = ImageData.data ;
var a = ImageData.data ;
alert(r+" "+g+" "+b+" "+a);
}
```

Genau demselben Prinzip folgt das Modifizieren von Pixelwerten, indem wir jetzt das CanvasPixelArray `in-place` durch Zuweisung neuer Werte verändern. In unserem Beispiel werden die RGB-Werte mit `Math.random()` auf Zufallszahlen zwischen 0 und 255 gesetzt; die Alpha-Komponente bleibt unberührt.

```
for (var i=0; i<ImageData.data.length; i+=4) {
ImageData.data  = parseInt(Math.random()*255);
ImageData.data  = parseInt(Math.random()*255);
ImageData.data  = parseInt(Math.random()*255);
}
```

Der Canvas erscheint nach diesem Schritt allerdings noch unverändert. Erst durch Zurückschreiben des modifizierten CanvasPixelArray über die Methode `putImageData()` werden die neuen Farben sichtbar. Beim Aufruf von `putImageData()` sind maximal sieben Parameter erlaubt.

```
context.putImageData(
ImageData, dx, dy, [ dirtyX, dirtY, dirtyWidth, dirtyHeight ]
)
```

Die ersten drei Argumente sind verpflichtend anzugeben und beinhalten neben dem ImageData-Objekt die Koordinate des Ursprungspunktes dx/dy, von dem aus das CanvasPixelArray über seine `width`– und `height`-Attribute aufgetragen wird. Die optionalen `dirty`-Parameter dienen dazu, einen bestimmten Bereich des CanvasPixelArray auszuschneiden und nur diesen mit reduzierter Breite und Höhe zurückzuschreiben. Abbildung 1 zeigt unseren 4-Pixel-Canvas vor und nach der Modifikation und listet die jeweiligen Werte des CanvasPixelArray auf.

Auf direktem Weg lässt sich ein leeres ImageData-Objekt über die Methode `createImageData()` initialisieren. Breite und Höhe entsprechen dabei den Argumenten sw/sh oder den Dimensionen eines beim Aufruf übergebenen ImageDataObjekts. In beiden Fällen werden alle Pixel des CanvasPixelArray auf transparent/schwarz, also `rgba(0,0,0,0)`, gesetzt.

```
context.createImageData(sw, sh)
context.createImageData(imagedata)
```

Den 2 x 2 Pixel großen, modifizierten Canvas in Abbildung 1 könnten wir mithilfe von `createImageData()` demnach auch direkt erzeugen und über `putImageData()` zeichnen:

```
var imagedata = context.createImageData(2,2);
for (var i=0; i<ImageData.data.length; i+=4) {
imagedata.data  = parseInt(Math.random()*255);
imagedata.data  = parseInt(Math.random()*255);
imagedata.data  = parseInt(Math.random()*255);
}
context.putImageData(imagedata,0,0);
```

So viel zur nüchternen CanvasPixelArray-Theorie, die Praxis ist viel spannender, denn mit `getImageData()`, `putImageData()`, `createImageData()` und etwas Mathematik lassen sich sogar eigene Farbfilter zum Manipulieren von Bildern schreiben. Wie das geht, zeigt der folgende Abschnitt.

Farbmanipulation mit `getImageData()`, `createImageData()` und `putImageData()`

Das Musterbild für alle Beispiele ist wieder die Aufnahme aus dem Yosemite-Nationalpark, die `onload` mit `drawImage()` auf den Canvas gezeichnet wird. In einem zweiten Schritt definieren wir über `getImageData()` das originale CanvasPixelArray, das wir dann im dritten Schritt modifizieren. Dabei werden in einer `for`-Schleife die RGBA-Werte jedes Pixels nach einer mathematischen Formel neu berechnet und in ein zuvor über `createImageData()` erzeugtes CanvasPixelArray eingetragen, das wir am Ende mit `putImageData()` wieder auf den Canvas zurückschreiben.

Der Code liefert das **JavaScript-Grundgerüst** für alle Filter, die in Abbildung 2 Verwendung finden. Die Funktion `grayLuminosity()` ist nicht Teil des Code-Beispiels, sondern wird, wie alle anderen Filter, im Anschluss behandelt.

```
var image = new Image();
image.src = "images/yosemite.jpg";
image.onload = function() {
context.drawImage(image,0,0,360,240);
var modified = context.createImageData(360,240);
var imagedata = context.getImageData(0,0,360,240);
for (var i=0; i<imagedata.data.length; i+=4) {
var rgba = grayLuminosity(
imagedata.data ,
imagedata.data ,
imagedata.data ,
```

```
imagedata.data
);
modified.data  = rgba[0];
modified.data  = rgba[1];
modified.data  = rgba[2];
modified.data  = rgba[3];
}
context.putImageData(modified,0,0);
};
```

Info
Das Server-Icon in der rechten unteren Ecke von Abbildung 2 signalisiert, dass dieses Beispiel bei Verwendung von Firefox als Browser nur über einen Server mit dem *http://*-Protokoll aufgerufen werden kann.

Zum Umwandeln der Farbe in Graustufen liefert die Dokumentation des freien Bildbearbeitungsprogramms GIMP im Kapitel Entsättigen drei Formeln, um den Grauwert über Helligkeit (`Lightness`), Leuchtkraft (`Luminosity`) oder durchschnittliche Helligkeit (`Average`) zu berechnen. Setzen wir diese Formeln in JavaScript um, erhalten wir unsere ersten drei Farbfilter:

```
var grayLightness = function(r,g,b,a) {
var val = parseInt(
(Math.max(r,g,b)+Math.min(r,g,b))*0.5
);
return [val,val,val,a];
};
var grayLuminosity = function(r,g,b,a) {
var val = parseInt(
(r*0.21)+(g*0.71)+(b*0.07)
);
return [val,val,val,a];
};
var grayAverage = function(r,g,b,a) {
var val = parseInt(
(r+g+b)/3.0
);
return [val,val,val,a];
};
```

Mit `grayLuminosity()` verwenden wir in Abbildung 2 die zweite Formel und ersetzen die RGB-Komponenten jedes Pixels durch den neu berechneten Wert. Nicht vergessen dürfen wir in dieser und allen folgenden Berechnungen, dass RGBA-Werte nur Integerzahlen sein dürfen – die JavaScript-Methode `parseInt()` stellt dies sicher.

Der Algorithmus für `sepiaTone()` entstammt einem Artikel von Zach Smith mit dem Titel *How do I … convert images to greyscale and sepia tone using C#?*.

```
var sepiaTone = function(r,g,b,a) {
var rS = (r*0.393)+(g*0.769)+(b*0.189);
var gS = (r*0.349)+(g*0.686)+(b*0.168);
var bS = (r*0.272)+(g*0.534)+(b*0.131);
return [
(rS>255) ? 255 : parseInt(rS),
(gS>255) ? 255 : parseInt(gS),
(bS>255) ? 255 : parseInt(bS),
a
];
};
```

Durch Aufsummieren der multiplizierten Komponenten können in jeder der drei Berechnungen natürlich auch Werte größer als 255 entstehen – in diesen Fällen wird 255 als neuer Wert eingesetzt.

Sehr einfach ist das Invertieren von Farben im Filter `invertColor()`, denn jede **RGB-Komponente** muss nur von 255 abgezogen werden.

```
var invertColor = function(r,g,b,a) {
return [
(255-r),
(255-g),
(255-b),
a
];
};
```

Der Filter `swapChannels()` modifiziert die Reihenfolge der Farbkanäle. Dazu müssen wir als vierten Parameter die gewünschte Neuanordnung in einem Array definieren, wobei 0 für Rot, 1 für Grün, 2 für Blau und 3 für den AlphaKanal anzugeben ist. Beim Vertauschen der Kanäle hilft uns das Array `rgba` mit den entsprechenden Eingangswerten, das wir in neuer Reihung zurückliefern. Ein Wechsel von RGBA nach BRGA, wie in unserem Beispiel, lässt sich mit `order=[2, 0, 1, 3]` realisieren.

```
var swapChannels = function(r,g,b,a,order) {
var rgba = [r,g,b,a];
return [
rgba[order[0]],
rgba[order[1]],
rgba[order[2]],
rgba[order[3]]
```

```
];
};
```

Die letzte Methode, `monoColor()`, setzt die RGB-Komponente jedes Pixels auf eine bestimmte Farbe und verwendet den Grauwert des Ausgangspixels als Alpha-Komponente. Der vierte Parameter beim Aufruf definiert die gewünschte Farbe als Array von RGB-Werten – in unserem Fall ist dies Blau mit `color= [0, 0, 255]`.

```
var monoColor = function(r,g,b,a,color) {
return [
color[0],
color[1],
color[2],
255-(parseInt((r+g+b)/3.0))
];
};
```

Die vorgestellten Filter sind noch sehr einfach gestrickt, da sie Farbwerte einzelner Pixel immer ohne Berücksichtigung der Nachbarpixel verändern. Bezieht man diese in die Berechnung ein, sind komplexere Methoden wie Schärfen, Unschärfemasken oder Kantenerkennung möglich.

Die Methode `getImageData()` liefert ein ImageData-Objekt, das rohe (nicht vorab mit dem Alphawert multiplizierte) Pixel (als R-, G-, B- und A-Komponenten) aus einem rechteckigen Bereich des **Canvas** liefert. Leere ImageData-Objekte können Sie mit `createImageData()` erstellen. Die Pixel in einem ImageData-Objekt sind schreibbar, können also beliebig gesetzt werden. Dann können Sie diese Pixel mit `putImageData()` wieder in das Canvas kopieren.

Diese Methoden zur **Pixelmanipulation** bieten einen sehr elementaren Zugriff auf das Canvas. Das Rechteck, das Sie an `getImageData()` übergeben, bezieht sich auf das Standardkoordinatensystem: Seine Ausmaße werden in **CSS-Pixeln** gemessen, und die aktuelle Transformation wirkt sich darauf nicht aus. Rufen Sie `putImageData()` auf, wird ebenfalls das Standardkoordinatensystem genutzt. Außerdem ignoriert `putImageData()` alle Grafikattribute. Die Methode führt kein Compositing durch, multipliziert keine Pixel mit `globalAlpha` und zeichnet keine Schatten.

Die Methoden zur Pixelmanipulation sind gut zur Implementierung einer Bildverarbeitung geeignet. Beispiel 1 zeigt, wie man eine einfache Bewegungsunschärfe oder einen „Verwischeffekt“ auf den Zeichnungen auf einem Canvas umsetzt. Das Beispiel illustriert die Verwendung von `getImageData()` und `putImageData()` und zeigt, wie man die Pixelwerte in einem ImageData-Objekt durchläuft und bearbeitet, erklärt die jeweiligen Aspekte allerdings nicht im Detail.

Beispiel 1: Bewegungsunschärfe mit ImageData

```
// Die Pixel des Rechtecks nach rechts verwischen und damit
```

```
// eine Art Bewegungsunschärfe erzeugen, als würden sich
// Objekte von rechts nach links bewegen. n muss 2 oder größer
// sein. Größere Werte führen zu stärkerer Verzerrung. Das
// Rechteck wird im Standard-Koordinatensystem angegeben.
function smear(c, n, x, y, w, h) {

// Das ImageData-Objekt abrufen, das das Rechteck
// der zu verwischenden Pixel repräsentiert.
var pixels = c.getImageData(x,y,w,h);

// Das Verwischen wird vor Ort ausgeführt und erfordert
// nur das ImageData-Objekt. Einige Bildverarbeitungs-
// algorithmen erfordern zusätzliche ImageData-Objekte,
// um transformierte Pixelwerte zu speichern. Würden wir
// einen Ausgabepuffer benötigen, könnten wir so ein neues
// ImageData-Objekt mit den gleichen Maßen erstellen:
var output_pixels = c.createImageData(pixels);

// Diese Maße können sich von den Argumenten für Breite
// und Höhe unterscheiden: Einem CSS-Pixel können mehrere
// Gerätepixel entsprechen.
var width = pixels.width, height = pixels.height;

// Das ist das byte-Array, das die rohen Pixeldaten von
// links oben nach rechts unten enthält. Jedes Pixel nimmt
// vier aufeinanderfolgende Bytes in der Reihenfolge R, G, B, A ein.
var data = pixels.data;

// Ab dem zweiten Pixel werden in jeder Zeile die Pixel
// verschmiert, indem sie durch 1/n des eigenen
// Werts plus m/n des Werts des vorangehenden Pixels
// ersetzt werden.
var m = n-1;

for(var row = 0; row < height; row++) { // Für jede Zeile
// die Position des zweiten Pixels der Zeile berechnen
var i = row*width*4 + 4;

// Für alle Pixel der Zeile ab dem zweiten Pixel
for(var col 1; col < width; col++, i += 4) {
data  =   (data +data[i-4]*m)/n;    // Rot
data  = (data +data[i-3]*m)/n;  // Grün
data  = (data +data[i-2]*m)/n;  // Blau
data  = (data +data[i-1]*m)/n;  // Alpha
}
}
```

```
// Jetzt die verschmierten Pixeldaten wieder in das Canvas
// kopieren
c.putImageData(pixels, x, y);
}
```

Beachten Sie, dass `getImageData()` den gleichen Cross-Origin-Sicherheitsbeschränkungen unterliegt wie `toDataURL()`: Die Methode funktioniert nicht, wenn in das Canvas (direkt mit `drawImage()` oder indirekt über ein **CanvasPattern**) ein Bild gezeichnet wurde, das eine andere Herkunft hat als das Dokument, das das Canvas enthält.

Vier Grafikattribut eigenschaften des CanvasRenderingContext2D-Objekts steuern das Zeichnen von Schlagschatten. Wenn Sie diese Eigenschaften passend setzen, erhält jede Linie, jede Fläche, jeder Text und jedes Bild einen Schlagschatten, der es so aussehen lässt, als würde das Element über der **Canvas**-Oberfläche schweben. Abbildung 1 zeigt einen Schatten unter einem gefüllten Rechteck, einem nicht gefüllten Rechteck und einem gefüllten Text.

Die Eigenschaft `shadowColor` gibt die Farbe des Schattens an. Der Standard ist vollständig transparentes Schwarz, und Schatten erscheinen nur, wenn Sie diese Eigenschaft auf eine durchscheinende oder blickdichte Farbe setzen. Diese Eigenschaft kann nur auf einen Farbstring gesetzt werden: Muster und Verläufe sind für Schatten nicht gestattet. Durchscheinende Schatten bewirken die realistischsten Schatteneffekte, weil sie den Hintergrund durchscheinen lassen.

Die Eigenschaften `shadowOffsetX` und `shadowOffsetY` geben die x– und y-Verschiebung des Schattens an. Der Standard für beide Eigenschaften ist 0; dies lässt den Schatten unmittelbar unter der Zeichnung erscheinen, wo er nicht sichtbar ist. Wenn Sie beide Eigenschaften auf einen positiven Wert setzen, erscheinen die Schatten rechts unter dem gezeichneten Objekt, so, als schiene die Lichtquelle von links oben und außerhalb des Bildschirms auf das Canvas. Größere Verschiebungen führen zu größeren Schatten und erwecken den Eindruck, als schwebten die Objekte „höher“ über dem Canvas.

Die Eigenschaft `shadowBlur` gibt an, wie weich die Kanten der **Schatten** sind. Der Standardwert ist 0 und bewirkt scharfe und nicht verschwommene Schatten. Größere Werte führen zu größerer Unschärfe bis zu einer Implementierungs-Definierten Obergrenze. Diese Eigenschaft ist ein Parameter für die Funktion Gaußscher Weichzeichner und keine Größe oder Länge in Pixeln.

Beispiel 1 zeigt den Code, mit dem Abbildung 1 erzeugt wurde, und führt alle vier Schatteneigenschaften vor.

Beispiel 1: Schattenattribute setzen

```
// Einen weichen Schatten definieren
c.shadowColor = "rgba(100,100,100,.4)";   // Durchscheinendes
                                          // Grau
c.shadowOffsetX = c.shadowOffsetY = 3;    // Leichte
                                          // Verschiebung
c.shadowBlur = 5;                         // Weiche Kanten

// Etwas Text und einen blauen Kasten mit diesen Schatten
// zeichnen
c.lineWidth = 10;
c.strokeStyle = "blue";
c.strokeRect(100, 100, 300, 200);        // Einen Kasten
                                         // zeichnen
c.font = "Bold 36pt Helvetica";
c.fillText("Hello World", 115, 225);     // Etwas Text zeichnen

// Einen etwas auffälligeren Schatten definieren. Größere
// Verschiebung lässt Gegenstände höher "schweben". Beachten
// Sie, wie sich Schatten und Kasten überschneiden.
c.shadowOffsetX = c.shadowOffsetY = 20; // Große Verschiebung
c.shadowBlur = 10;          // Weichere Kanten
c.fillStyle = "red";        // Ein undurchsichtiges rotes
                            // Rechteck,
c.fillRect(50,25,200,65);   // das über einem blauen Kasten
                            // schwebt.
```

Die Eigenschaften `shadowOffsetX` und `shadowOffsetY` werden immer im Standardkoordinatensystem gemessen und von den Methoden `rotate()` oder `scale()` nicht beeinflusst. Nehmen Sie zum Beispiel an, Sie drehen das Koordinatensystem um 90 Grad, um Text vertikal zu zeichnen, und stellen dann das ursprüngliche **Koordinatensystem** wieder her, um einen horizontalen Text zu zeichnen. Die Schatten des vertikalen Texts und des horizontalen Texts sind in die gleiche Richtung ausgerichtet, und genau so sollte es wohl auch sein. Die Schatten von Figuren, die mit der gleichen Skalierung gezeichnet wurden, haben ebenfalls Schatten der gleichen „Höhe."

Sparklines sind kleine Diagramme, die unmittelbar in den Textfluss eingebettet werden. Der Begriff „Sparkline" wurde von *Edward Tufte* geprägt, der sie so beschreibt: *Sparklines sind kleine, in einen Wort-, Zahl- oder Bildkontext eingebettete Grafiken mit hoher Auflösung. Sie sind datenintensive, einfach gestaltete und wortgroße Diagramme.* Mehr zu Sparklines erfahren Sie in Tuftes Buch *Beautiful Evidence*.

Beispiel 1 ist ein recht einfaches Modul mit nicht interaktivem JavaScript-Code zur Einbettung von Sparklines in Webseiten. Die Kommentare erklären, wie er funktioniert.

Beispiel 1: Sparklines mit dem `<canvas>`-Tag

Alle Elemente der CSS-Klasse „`sparkline`“ finden, ihren Inhalt als Zahlenfolge parsen und durch eine grafische Darstellung ersetzen.

Definieren Sie Sparklines mit Markup wie diesem:

```
<span class="sparkline">3 5 7 6 6 9 11 15</span>
```

Stylen Sie Sparklines mit CSS wie diesem:

```
.sparkline {
background-color: #ddd;
color: red;
}
```

Die Sparkline-Farbe ist der berechnete Stil der CSS-Eigenschaft `color`.

Sparklines sind transparent, es schimmert also die Hintergrundfarbe durch.

Die Höhe der Sparklines wird durch das Attribut `data-height` festgelegt, falls dieses definiert ist, andernfalls mithilfe des berechneten Werts für `font-size`.

Die Breite der Sparklines wird durch das Attribut `data-width` festgelegt, wenn es definiert ist, oder durch die Anzahl von Datenpunkten mal `data-dx`, wenn das definiert ist, oder die Anzahl von Datenpunkten mal der Höhe durch 6.

Die Minimum- und Maximumwerte der y-Achse werden aus den Attributen `data-ymin` und `data-ymax` entnommen, wenn diese definiert sind, andernfalls werden sie aus den größten und kleinsten Datenwerten ermittelt.

```
// Diesen Code ausführen, wenn das Dokument geladen wird:
window.addEventListener("load", function() {

// Alle Elemente der Klasse "sparkline" finden
var elts = document.getElementsByClassName("sparkline");

// Diese Elemente durchlaufen
main: for(var e = 0; e < elts.length; e++) {
var elt = elts[e];

// Den Inhalt des Elements abrufen und in ein Array
// mit Zahlen umwandeln. Falls die Umwandlung
// fehlschlägt, dieses Element überspringen.
```

```
var content = elt.textContent || elt.innerText;

// Vorangehenden und nachfolgenden Leerraum
// abschneiden
var content = content.replace(/^\s+|\s+$/g, "");

// Kommentare entfernen
var text = content.replace(/#.*$/gm, "");

// Zeilenumbrüche usw. in Leerzeichen umwandeln
text = text.replace(/[\n\r\t\v\f]/g, " ");

// Zahlen an Kommata oder Leerzeichen trennen
var data = text.split(/\s+|\s*,\s*/);

// Für jeden Teil des Strings
for(var i = 0; i < data.length; i++) {
data  = Number(data );   // In eine Zahl
                            // umwandeln.
if (isNaN(data ))          // Bei Fehlschlag
continue main;               // das Element
                             // überspringen.
}

// Jetzt Farbe, Breite, Höhe und Grenzen der y-Achse
// der Sparkline aus den Daten oder den data-
// Attributen des Elements und dem berechneten Stil
// des Elements ermitteln.
var style = getComputedStyle(elt, null);
var color = style.color;
var height =
parseInt(elt.getAttribute("data-height")) ||
parseInt(style.fontSize) || 20;
var datadx = parseInt(elt.getAttribute("data-dx"));
var width =
parseInt(elt.getAttribute("data-width")) ||
data.length*(datadx || height/6);
var ymin =
parseInt(elt.getAttribute("data-ymin")) ||
Math.min.apply(Math, data);
var ymax =
parseInt(elt.getAttribute("data-ymax")) ||
Math.max.apply(Math, data);
if (ymin >= ymax) ymax = ymin + 1;

// Das Canvas-Element erstellen
```

```
var canvas = document.createElement("canvas");
canvas.width = width;       // Die Canvas-Maße bestimmen
canvas.height = height;

// Den Elementinhalt als Tooltipp nutzen
canvas.title = content;
elt.innerHTML = "";         // Den bestehenden Inhalt
                            // löschen
elt.appendChild(canvas);    // Canvas in Element einfügen

// Jetzt die Punkte in das Canvas zeichnen
var context = canvas.getContext('2d');
for(var i = 0; i < data.length; i++) {

// (i,data ) in Canvas-Koordinaten umwandeln
var x = width*i/data.length;
var y = (ymax-data )*height/(ymax-ymin);

// Eine Linie nach (x,y) zeichnen. Beachten Sie,
// dass der erste Aufruf von lineTo() stattdessen
// ein moveTo() bewirkt.
context.lineTo(x,y);
}
context.strokeStyle = color;  // Eine Farbe angeben
context.stroke();             // und zeichnen.
}
}, false); // Das letzte Argument fü addEventListener()
```

Ein Geschäft macht eine Menge auf der Website, und das obere Management wünscht sich eine grafische Darstellung der Webstatistiken. Die Backend-Programmierer können die Daten in Echtzeit liefern, möchten aber erst einmal sehen, ob Sie eine Möglichkeit finden, die Grafik im Browser darzustellen. Also haben sie Sie mit Testdaten versorgt. Unser Ziel ist es, die Testdaten in etwas umzuwandeln, das an Abbildung 1 auf der folgenden Seite erinnert.

Es gibt viele Möglichkeiten, **Diagramme** auf einer Webseite zu zeichnen. Manche Entwickler verwenden immer Flash, aber das hat den Nachteil, dass es auf einigen mobilen Geräten wie dem iPad oder iPhone nicht funktioniert. Es gibt serverseitige Lösungen, die gut funktionieren, aber für Echtzeitdaten zu rechenintensiv sein könnten. Eine standardbasierte clientseitige Lösung wie das `canvas`-Element ist eine ausgezeichnete Möglichkeit, solange wir darauf achten, dass es auch in älteren Browsern funktioniert. Sie haben bereits gesehen, wie Sie Quadrate zeichnen können. Aber um etwas Komplexes zu zeichnen, ist eine Menge mehr JavaScript erforderlich. Wir brauchen eine Bibliothek für die grafische Darstellung.

Von der Tatsache, dass HTML5 noch nicht überall verfügbar ist, haben sich die Entwickler der

RGraph-Bibliothek nicht abhalten lassen. Mit **RGraph** ist es geradezu lächerlich einfach, mit dem canvas-Element von HTML5 Diagramme zu zeichnen. Es handelt sich allerdings um eine reine JavaScript-Lösung, die dementsprechend nicht auf User Agents ohne JavaScript funktioniert. Aber das Gleiche gilt ja auch für das canvas-Element. Hier sehen Sie den Code für ein sehr einfaches Balkendiagramm:

```
<canvas width="500" height="250" id="test">[no canvas support]</canvas>
<script type="text/javascript" charset="utf-8">
var bar = new RGraph.Bar('test', [50,25,15,10]);
bar.Set('chart.gutter', 50);
bar.Set('chart.colors', ['red']);
bar.Set('chart.title', "A bar graph of my favorite pies");
bar.Set('chart.labels',["Banana Creme", "Pumpkin", "Apple","Cherry" ]);
bar.Draw();
</script>
```

Wir müssen lediglich einige **JavaScript-Arrays** anlegen, und schon zeichnet die Bibliothek das Diagramm im canvas-Element.

Daten mit HTML beschreiben

Wir könnten die Werte für die Browserstatistiken fest in den Java-Script-Code schreiben, aber dann können nur Benutzer mit JavaScript diese Werte sehen. Stattdessen schreiben wir die Daten als Text auf die Seite. Anschließend können wir die Daten mit JavaScript einlesen und die Diagramm-Bibliothek damit füttern.

```
<div id="graph_data">
<h1>Browser share for this site</h1>
<ul>
<li>
<p data-name="Safari 4" data-percent="15">
Safari 4 -15%</p>
</li>
<li>
<p data-name="Internet Explorer" data-percent="55">
Internet Explorer -55%</p>
</li>
<li>
<p data-name="Firefox" data-percent="14">
Firefox -14%</p>
</li>
<li>
<p data-name="Google Chrome" data-percent="16">
Google Chrome -16%</p>
</li>
</ul>
```

```
</div>
```

Wir verwenden **HTML5-Datenattribute**, um die Browsernamen und Prozentsätze zu speichern. Diese Informationen stehen zwar auch im Text, wir können aber so viel leichter programmgesteuert damit arbeiten, da wir keine Strings einlesen müssen.

Wenn Sie die Seite in Ihrem Browser öffnen oder einfach einen Blick auf Abbildung 2 werfen, sehen Sie, dass die Diagrammdaten hübsch lesbar angezeigt werden – auch ohne Diagramm. Das ist unser Alternativinhalt für mobile Geräte und andere Benutzer, für die entweder das `canvas`-Element oder JavaScript nicht verfügbar sind.

Jetzt machen wir aus diesem Markup ein Diagramm.

Ein Balkendiagramm aus HMTL erstellen

Wir verwenden ein Balkendiagramm, daher brauchen wir die Balkendiagrammbibliothek und die Hauptbibliothek von RGraph. Um die Daten aus dem Dokument auszulesen, verwenden wir **jQuery**. Im head der HTML-Seite müssen wir die erforderlichen Bibliotheken laden.

```
<script type="text/javascript" charset="utf-8"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
<script src="javascripts/RGraph.common.js"></script>
<script src="javascripts/RGraph.bar.js"></script>
```

Zum Erstellen des Diagramms müssen wir den Titel, die Beschriftungen und die Daten aus dem **HTML-Dokument** auslesen und an die RGraph-Bibliothek übergeben. RGraph erwartet sowohl für die Beschriftungen als auch für die Daten Arrays. Diese Arrays können wir mit jQuery schnell erstellen.

```
function canvasGraph(){
var title = $('#graph_data h1').text();
var labels = $("#graph_data>ul>li>p[data-name]").map(function(){
return $(this).attr("data-name");
});
var percents = $("#graph_data>ul>li>p[data-percent]").map(function(){
return parseInt($(this).attr("data-percent"));
});
var bar = new RGraph.Bar('browsers', percents);
bar.Set('chart.gutter', 50);
bar.Set('chart.colors', ['red']);
bar.Set('chart.title', title);
```

```
bar.Set('chart.labels', labels);
bar.Draw();
}
```

Als Erstes rufen wir in Zeile 2 den Text für die Kopfzeile ab. In Zeile 4 wählen wir anschließend alle Elemente mit dem Attribut `data-name` aus. Wir verwenden die `map`-Funktion von jQuery, um die Werte dieser Elemente in ein **Array** zu schreiben.

In Zeile 8 wenden wir dieselbe Logik an, um ein Array mit den Prozentzahlen anzulegen.

Nachdem wir die Daten gesammelt haben, ist es für RGraph ein Leichtes, unser Diagramm zu zeichnen.

Alternativen Inhalt anzeigen

Im Abschnitt Daten mit HTML beschreiben hätte ich das Diagramm auch zwischen dem öffnenden und dem schließenden `canvas`-Tag platzieren können. Dadurch würden diese Elemente in Browsern versteckt, die das `canvas`-Element unterstützen und in Browsern angezeigt, die es nicht unterstützen. Jedoch würde der Inhalt auch verborgen bleiben, wenn der Browser zwar das `canvas`-Element unterstützt, der Benutzer aber JavaScript deaktiviert hat.

jQuery CSS contra CSS

In diesem Artikel haben wir unsere Stilregeln mit jQuery auf die Elemente angewendet, während wir sie erstellt haben. Eine Menge dieser Stilinformationen, wie etwa die Farben der Beschriftungen und der Balken, sollten aber in einem separaten Stylesheet untergebracht werden. Umso wichtiger ist das, wenn Sie die Stilregeln unabhängig vom Skript ändern möchten. Für einen Prototyp ist unser bisheriger Ansatz in Ordnung, aber für eine Produktionsversion sollten Sie Darstellung, Verhalten und Inhalt strikt voneinander trennen.

Wir lassen die Daten einfach außerhalb des `canvas`-Elements stehen und verstecken sie anschließend mit jQuery, sobald wir überprüft haben, ob das `canvas`-Element existiert.

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
$('#graph_data').hide();
canvasGraph();
}
```

Damit ist unser Diagramm fertig, außer für diejenigen, deren Browser das `canvas`-Element nicht unterstützen.

Ausweichlösung

Beim Erstellen dieser Lösung haben wir bereits Ausweichlösungen für Barrierefreiheit und fehlendes **JavaScript** abgedeckt. Aber wir können ein alternatives Diagramm für Menschen erstellen, die zwar nicht über die Unterstützung für das `canvas`-Element verfügen, aber JavaScript verwenden können.

Es gibt tonnenweise Diagramm-Bibliotheken, von denen jede die Daten anders erfasst.

Balkendiagramme sind aber letztlich nur Rechtecke mit einer bestimmten Höhe, und wir haben alle Daten auf der Seite, um dieses Diagramm von Hand aufzubauen.

```
function divGraph(barColor, textColor, width, spacer, lblHeight){
$('#graph_data ul').hide();
var container = $("#graph_data");
container.css( {
"display": "block",
"position": "relative",
"height": "300px"
}
);
$("#graph_data>ul>li>p").each(function(i){
var bar = $("<div>" + $(this).attr("data-percent") + "%</div>");
var label = $("<div>" + $(this).attr("data-name") + "</div>");
var commonCSS = {
"width": width + "px",
"position": "absolute",
"left": i * (width + spacer) + "px" };
var barCSS = {
"background-color": barColor,
"color": textColor,
"bottom": lblHeight + "px",
"height": $(this).attr("data-percent") + "%"
};
varlabelCSS = {"bottom":"0","text-align":"center" };
bar.css( $.extend(barCSS, commonCSS) );
label.css( $.extend(labelCSS,commonCSS) );
container.append(bar);
container.append(label);
});
}
```

In Zeile 2 verstecken wir die ungeordnete Liste, damit die Textwerte nicht angezeigt werden. Anschließend nehmen wir das Element mit den Diagrammdaten und wenden einige grundlegende **CSS-Stilregeln** darauf an. In Zeile 7 legen wir für die Positionierung des Elements `relative` fest, wodurch wir unsere Balkendiagramme und Beschriftungen innerhalb dieses Containers absolut positionieren können.

Anschließend durchlaufen wir die Absätze der Aufzählungsliste (Zeile 11) und erstellen die Balken. In jedem Durchlauf für die Beschriftungen werden zwei `div`-Elemente erstellt: eins für den Balken und eins für die Beschriftung, die wir darunter positionieren. Mit ein bisschen Mathematik und ein bisschen jQuery sind wir also in der Lage, das Diagramm neu zu erfinden. Auch wenn es nicht genauso aussieht, so ist es doch ähnlich genug, um unser Konzept zu bestätigen.

Anschließend müssen wir es nur noch in unsere canvas-Prüfung einbauen:

```
var canvas = document.getElementById('browsers');
if (canvas.getContext){
$('#graph_data').hide();
canvasGraph();
}
else{
divGraph("#f00", "#fff", 140, 10, 20);
}
```

Sie können die Alternativversion in Abbildung 3 sehen. Mit einer Kombination aus JavaScript, HTML und CSS haben wir clientseitig ein Balkendiagramm und statistische Informationen zur Browsernutzung für jede beliebige Plattform bereitgestellt. Die Verwendung des canvas-Elements hat einen weiteren Vorteil – sie hat uns dazu gebracht, von Anfang an über eine Ausweichlösung nachzudenken, anstatt nachträglich etwas hineinzuquetschen. Das ist ein echtes Plus in puncto Barrierefreiheit.

Diese Methode bietet mit die größtmögliche Barrierefreiheit und Wandlungsfähigkeit,

Daten grafisch darzustellen. Sie können die visuelle Darstellung alternativ auch auf einfache Weise textbasiert anbieten. Auf diese Weise kann jeder die wichtigen Daten nutzen, die Sie bereitstellen.

Joe fragt ... Warum haben wir nicht ExplorerCanvas ausprobiert?

Explorer Canvas, von dem wir im Abschnitt Ausweichlösung gesprochen haben, und RGraph funktionieren wirklich gut zusammen. Die Distribution von RGraph enthält sogar eine Version von ExplorerCanvas. Allerdings funktioniert diese Kombination nur mit dem Internet Explorer 8. Für einen IE 7 oder ältere Versionen müssen Sie als Alternative eine Lösung wie unsere verwenden. Ich möchte Sie ermuntern, ExplorerCanvas im Auge zu behalten, da es ständig weiterentwickelt wird. Sie sollten auch mal darüber nachdenken, selbst an der Entwicklung mitzuwirken, damit Sie ExplorerCanvas für Ihr Projekt nutzen können.

Die Zukunft

Nachdem Sie jetzt ein bisschen über die Funktionsweise des canvas-Elements wissen, können Sie über weitere Einsatzmöglichkeiten nachdenken. Sie könnten damit ein Spiel, eine Benutzeroberfläche für einen Medienplayer oder eine tolle Bildergalerie schreiben. Sie brauchen lediglich etwas JavaScript und ein bisschen Fantasie, um mit dem Zeichnen loszulegen.

Im Moment hat Flash einen gewissen Vorsprung gegenüber dem canvas-Element, weil es weiter verbreitet ist. Aber wenn **HTML5** aufholt und canvas für ein breiteres Publikum zur Verfügung steht,

werden mehr und mehr Entwickler für einfache **2D-Grafiken** im Browser begeistert darauf zurückgreifen. Das `canvas`-Element erfordert keine zusätzlichen Plugins und benötigt weniger CPU-Leistung als Flash, insbesondere unter Linux und OS X. Zu guter Letzt bietet Ihnen das `canvas`-Element einen Mechanismus, um 2D-Grafiken auch in Umgebungen zu erstellen, in denen kein Flash verfügbar ist. Da das `canvas`-Element auf immer mehr Plattformen unterstützt wird, können Sie mit zunehmend mehr Geschwindigkeit und Funktionen rechnen und davon ausgehen, dass Sie von immer mehr Entwicklertools und Bibliotheken dabei unterstützt werden, außergewöhnliche Dinge zu tun.

Aber 2D-Grafiken sind noch nicht alles. Die `canvas`-Spezifikation wird irgendwann auch 3D-Grafiken unterstützen, und die Browserhersteller werden die Hardwarebeschleunigung implementieren. Mit dem `canvas`-Element wird es möglich sein, allein mit JavaScript verblüffende Benutzeroberflächen und packende Spiele zu schreiben.

Die Eigenschaft `lineWidth` ist uns schon begegnet. Sie gibt die Breite der Striche an, die mit `stroke()` und `strokeRect()` gezogen werden. Neben `lineWidth` (und natürlich `strokeStyle`) gibt es drei weitere Grafikattribute, die sich auf Striche auswirken.

Der Standardwert für die Eigenschaft `lineWidth` ist 1, und Sie können sie auf jede positive Zahl, auch Teilwerte, kleiner 1 setzen. (Striche, die weniger als ein Pixel breit sind, werden mit durchscheinenden Farben gezeichnet, damit sie weniger dunkel erscheinen als 1 Pixel breite Striche). Wenn Sie die Eigenschaft `lineWidth` vollständig verstehen wollen, ist es wichtig, dass Sie sich Pfade als unendlich dünne, eindimensionale Linien vorstellen. Die von der **Methode** `stroke()` gezeichneten Linien und Kurven sind über dem Pfad zentriert mit der halben `lineWidth` auf beiden Seiten. Wenn Sie einen geschlossenen Pfad zeichnen wollen, dessen Linie nur außerhalb des Pfads erscheinen soll, ziehen Sie zunächst den Pfad und füllen ihn dann mit einer undurchsichtigen Farbe, um den Teil des Strichs zu verbergen, der innerhalb des Pfads erscheint. Soll die Linie nur innerhalb eines geschlossenen Pfads sichtbar sein, rufen Sie erst die Methoden `save()` und `clip()` auf und dann `stroke()` und `restore()`.

Wie Sie wahrscheinlich an den skalierten Achsen in der rechten oberen Ecke von Canvas – Dimensionen und Koordinaten in Abbildung 1 erkennen können, wirkt sich die aktuelle Transformation auf die Strichbreite aus. Wenn Sie `scale(2,1)` aufrufen, um die x-Erstreckung zu skalieren, ohne dass sich das auf y auswirkt, werden vertikale Linien doppelt so breit gezeichnet wie horizontale Linien mit dem gleichen `lineWidth`-Wert. Es ist wichtig, dass Sie verstehen, dass die Strichbreite durch `lineWidth` und die aktuelle Transformation zu dem Zeitpunkt bestimmt wird, zu dem `stroke()` aufgerufen wird, nicht zu dem Zeitpunkt, an dem `lineTo()` oder eine andere Methode zur Pfadkonstruktion aufgerufen wird.

Die anderen drei Strichattribute wirken sich auf das Aussehen

der nicht verbundenden Enden von Pfaden und den Eckpunkten zwischen zwei Pfadsegmenten aus. Bei

schmalen Linien haben diese kaum Auswirkungen, zeigen aber einen erheblichen Unterschied, wenn breite Linien gezogen werden. Zwei dieser Eigenschaften werden in Abbildung 1 illustriert. Die Abbildung zeigt den Pfad als dünne schwarze Linie und den Strich als einen grauen Bereich, der ihn umgibt.

Die Eigenschaft `lineCap` gibt an, wie die Enden offener Teilpfade aussehen. Der Wert `butt` (der Standard) bedeutet, dass die Linie am Endpunkt abrupt endet. Der Wert `square` heißt, dass sich die Linie um die halbe Strichbreite über den Endpunkt erstreckt. Und der Wert `round` bedeutet, dass die Linie mit einem Halbkreis (mit der halben Strichbreite als Radius) über den Endpunkt gezogen wird.

Die Eigenschaft `lineJoin` gibt an, wie die Eckpunkte zwischen Teilpfadsegmenten verbunden werden. Der Standardwert ist miter und bedeutet, dass die äußeren Kanten der beiden Pfadsegmente verlängert werden, bis sie sich an einem Punkt treffen. Der Wert `round` bedeutet, dass der Eckpunkt abgerundet wird, und der Wert `bevel` heißt, dass der Anschlusspunkt mit einer geraden Linie abgeschnitten wird.

Die letzte Stricheigenschaft ist `miterLimit`, die nur in Kraft tritt, wenn `lineJoin` gleich miter ist. Wenn zwei Linien in spitzem Winkel zusammentreffen, kann die Eckverlängerung zwischen beiden sehr lang werden. Derart lange, spitze Verlängerungen sind visuell störend. Die Eigenschaft `miterLimit` richtet eine Obergrenze für die Verlängerung ein. Wenn die Verlängerung an einem Eckpunkt länger als die Strichbreite mal `miterLimit` ist, wird der Eckpunkt mit einer gerade abgeschnittenen Verbindung statt einer verlängerten Verbindung gezeichnet.

Auf den ersten Blick lautet die Antwort wohl eher Ja, denn die Einsatzmöglichkeiten für Text in **Canvas** sind begrenzt und beschränken sich auf das Formatieren und Positionieren von einfachen Zeichenketten. Fließtext mit automatischen Zeilenumbrüchen wird man ebenso vermissen wie Absatzformate oder auch die Selektierbarkeit bereits erstellter Texte.

Was bleibt, sind drei Attribute zur Bestimmung der Texteigenschaften, zwei Methoden zum Zeichnen von Texten und eine Methode zur Ermittlung der Textlänge einer Zeichenkette unter Berücksichtigung des aktuell eingestellten Formats. Das scheint nicht viel zu sein, doch auf den zweiten Blick wird klar, dass sich auch hinter vier Seiten Spezifikation gut durchdachte Details verstecken können.

Fonts

Die Definition des **Font-Attributs** verweist kurzerhand auf die CSS-Spezifikation und legt fest, dass `context.font` der gleichen Syntax unterliegt wie die CSS-`font`-Kurznotation.

```
context.font = [
CSS font Eigenschaft
]
```

Auf diese Weise können alle Font-Eigenschaften bequem in einem einzigen String spezifiziert werden. Tabelle zeigt die einzelnen Komponenten und listet deren mögliche Werte auf.

Eigenschaft	Werte
`font-style`	*normal, italic, oblique
`font-variant`	*normal, small-caps
`font-weight`	*normal, bold, bolder, lighter 100, 200, 300, 400, 500, 600, 700, 800, 900
`font-size`	xx-small, x-small, small, *medium, large, x-large, xx-large, larger, smaller em, ex, px, in, cm, mm, pt, pc, %
`line-height`	*normal, <Nummer>, em, ex, px, in, cm, mm, pt, pc, %
`font-family`	Schriftfamilie oder generische Schriftfamilie wie serif, sans-serif, cursive, fantasy, monospace

Beim Zusammensetzen des `font`-Attributs sind nur die Eigenschaften `font-size` und `font-family` zwingend anzugeben. Alle anderen können entfallen und nehmen dann ihre in der Tabelle durch einen Stern gekennzeichneten Standardwerte ein. Da Text in Canvas keine Zeilenumbrüche kennt, ist auch das Attribut `line-height` ohne Wirkung und wird in jedem Fall ignoriert. Das bereinigte Muster beim Zusammensetzen der Komponenten lautet damit:

```
context.font = [
font-style font-variant font-weight font-size font-family
]
```

Bezüglich der `font-family` gelten dieselben Regeln wie beim Definieren von Schriften in **Stylesheets**: Es dürfen beliebige Kombinationen von Schriftfamilien und/oder generischen Schriftfamilien ausgewiesen werden. Der Browser pickt sich dann die erste ihm bekannte Schrift aus dieser Prioritätenliste heraus.

> Völlige Unabhängigkeit vom Browser beziehungsweise der jeweiligen Plattform und ihren Schriften

erzielt man durch die Verwendung von Webfonts. Einmal mittels `@font-face` in einem Stylesheet eingebunden, stehen sie über den vergebenen Schriftnamen auch in Canvas als `font-family` zur Verfügung.

```
@font-face {
```

```
font-family: Scriptina;
src: url('fonts/scriptina.ttf');
}
```

Abbildung 1 zeigt kurze Beispiele gültiger CSS-`font`-Attribute und deren Darstellung in Canvas.

Die Quelle für den **Webfont** Scriptina im letzten Beispiel ist – eine übersichtlich aufbereitete Sammlung freier Fonts, die zum Download bereitstehen.

Zu dem Zeitpunkt, als dieses Artikel geschrieben wurde, unterstützte kein Browser `@font-face` ohne Probleme. So findet bei Firefox der Webfont Scriptina in der letzten Zeile nur dann den Weg in den Canvas, wenn er im HTML-Dokument zumindest einmal verwendet wird. Ebenso fehlt bei Firefox die korrekte Umsetzung von `small-caps`, weshalb auch das vorletzte Beispiel nicht richtig angezeigt wird.

Horizontaler Textanfasspunkt

Zum Festlegen des Textanfasspunktes von Canvas-Texten in horizontaler Richtung dient das Attribut `textAlign`.

```
context.textAlign = [
left | right | center | *start | end
]
```

Sind die Keywords `left`, `right` oder `center` noch vom **CSS-Attribut** `text-align` bekannt, handelt es sich bei `start` und `end` schon um CSS3-Erweiterungen, die die Laufrichtung des Textes in Abhängigkeit von der jeweiligen Sprache berücksichtigen. Schriften können nämlich nicht nur von links nach rechts, sondern wie im Fall von Arabisch oder Hebräisch, um nur zwei Beispiele zu nennen, auch von rechts nach links laufen.

Abbildung 2 präsentiert die horizontalen Textanfasspunkte für Schriften mit Textfluss `ltr` (*left to right*) und `rtl` (*right to left*) und verdeutlicht die Auswirkung der Richtungsabhängigkeit auf die Attribute `start` und `end`.

Info

Im Browser kann die Richtungsabhängigkeit eines Dokumentes über das globale Attribut `document.dir` geändert werden:

```
document.dir = [
*ltr | rtl
```

```
]
```

Vertikaler Textanfasspunkt
Den Textanfasspunkt in vertikaler Richtung und damit die Grundlinie, an der alle Glyphen ausgerichtet werden, bestimmt das dritte und letzte textbezogene Attribut, `textBaseline`.

```
context.textBaseline = [
top | middle | *alphabetic | bottom | hanging | ideographic
]
```

Oben, Mitte, alphabetisch, unten, hängend und ideografisch lautet die Übersetzung der gültigen `textBaseline`-Keywords, wobei die ersten vier davon wohl selbsterklärend sind. Eine hängende Grundlinie benötigen Devanagari, Gurmukhi und Bengali, drei indische Schriften, in denen die Sprachen Sanskrit, Hindi, Marathi, Nepali beziehungsweise Panjabi und Bengalisch geschrieben werden. Zur Gruppe der ideografischen Schriften zählen Chinesisch, Japanisch, Koreanisch und Vietnamesisch.
Text zeichnen und messen
Sind Font und Anfasspunkt erst einmal festgelegt, muss nur noch der Text gezeichnet werden. Ähnlich wie bei Rechtecken können Sie sich für eine Füllung und/oder Randlinie entscheiden und sogar die erlaubte Breite des Textes durch einen optionalen Parameter `maxwidth` beschränken.

```
context.fillText(text, x, y, maxwidth)
context.strokeText(text, x, y, maxwidth)
```

Zum Messen der Dimension eines Textes steht zu guter Letzt noch die Methode `measureText()` zur Verfügung, die zumindest die Breite unter Berücksichtigung des aktuellen Formats ermitteln kann. Im Beispiel aus Abbildung 3 wurde der Wert rechts unten (759) auf diese Art berechnet.

```
Textbreite = context.measureText(text).width
```

Die Bestimmung von Höhe oder Ursprungspunkt des Hüllrechtecks (Bounding-Box) ist zum derzeitigen Zeitpunkt nicht möglich, könnte aber in einer zukünftigen Version der Spezifikation ebenso implementiert werden wie mehrzeiliges Text-Layout. Vielversprechend hört sich die letzte Anmerkung im Textkapitel der **Canvas-Spezifikation** an, wonach in Zukunft durchaus auch Fragmente von Dokumenten (z. B. Absätze mit Formatierungen) über CSS den Weg nach Canvas finden könnten.

> Nicht erst in der Zukunft, sondern bereits jetzt bietet die **Canvas-API**

eine Vielzahl an Möglichkeiten, um mit rasterbasierten Formaten in Canvas zu arbeiten. Neben dem Einbinden von Bildern und Videos besteht sogar die Möglichkeit, auf jedes einzelne Pixel der Canvas-Fläche sowohl lesend als auch schreibend zuzugreifen.

Text zeichnen Sie in ein **Canvas** üblicherweise mit der Methode `fillText()`. Sie zeichnet Text in der Farbe (oder mit dem Verlauf oder dem Muster), die die Eigenschaft `fillStyle` definiert. Für besondere Effekte bei großen Textgrößen können Sie die Methode `strokeText()` nutzen, um den Umriss der einzelnen Textzeichen zu ziehen (ein Beispiel für umrissenen Text sehen Sie in Abbildung 4). Beide Methoden erwarten als erstes Argument den zu zeichnenden Text und als zweites und drittes Argument die x– und y-Koordinaten für den Text. Keine dieser Methoden wirkt sich auf den aktuellen Pfad oder den aktuellen Punkt aus.

Wie Sie in Canvas-Dimensionen und Koordinaten Abbildung 4 sehen können, wirkt sich die aktuelle Transformation auf den Text aus.

Die Eigenschaft `font` gibt die für den Text zu nutzende Schriftart an. Der Wert sollte ein String mit der gleichen Syntax wie für das **CSS**–`font`-Attribut sein. Einige Beispiele:

```
"48pt sans-serif"
"bold 18px Times Roman"
"italic 12pt monospaced"
// Größer und kleiner als der <canvas>-Font
"bolder smaller serif"
```

Die Eigenschaft `textAlign` gibt an, wie Text horizontal in Bezug auf die an `fillText()` oder `strokeText()` übergebene x-Koordinate ausgerichtet wird. Die Eigenschaft `textBaseline` legt fest, wie Text vertikal in Bezug auf die y-Koordinate ausgerichtet wird. Abbildung 4 illustriert die erlaubten Werte für diese Eigenschaften. Die dünne Linie neben dem Text ist die Basislinie, und das kleine Rechteck markiert den an `fillText()` übergebenen Punkt (x,y).

Der Standardwert für `textAlign` ist `start`. Beachten Sie, dass bei Links-rechts-Text die Ausrichtung start der Ausrichtung `left` entspricht und die Ausrichtung end der Ausrichtung `right`. Setzen Sie das `dir`-Attribut des `<canvas>`-Tags auf `rtl` (*right-to-left*), entspricht die Ausrichtung `start` der Ausrichtung `right` und die Ausrichtung end der Ausrichtung `left`.

Der Standardwert für `textBaseline` ist `alphabetic`, dieser ist für das lateinische Alphabet und ähnliche Schriftsysteme geeignet. Der Wert `ideographic` wird für ideografische **Schriften** wie das Chinesische und Japanische genutzt. Der Wert `hanging` ist für Devangari und ähnliche Schriftsysteme (die für viele in Indien gebräuchliche Sprachen verwendet werden) geeignet. Die Basislinien `top`, `middle` und `bottom` sind rein geometrische Basislinien, die auf dem „em-Quadrat“ der Schrift basieren.

`fillText()` und `strokeText()` akzeptieren ein optionales viertes Argument. Wird dieses angegeben, gibt es die maximale Breite des anzuzeigenden Texts an. Würde der Text, wenn er mit dem aktuellen Wert der Eigenschaft `font` gezeichnet wird, breiter werden als der angegebene Wert, wird er passend gemacht, indem er skaliert oder mit einer schmaleren oder kleineren Schrift gezeichnet wird.

Wenn Sie den **Text** selbst messen wollen, bevor Sie ihn zeichnen, übergeben Sie ihn an die Methode `measureText()`. Diese Methode liefert ein TextMetrics-Objekt, das die Maße des Texts angibt, wenn er auf Basis des aktuellen `font`-Werts gezeichnet wird. Als dies geschrieben wurde, war die einzige „Metrik“ im TextMetrics-Objekt die Breite. Den Raum, den ein String auf dem Bildschirm einnimmt, fragen Sie folgendermaßen ab:

```
var width = c.measureText(text).width;
```

Canvas-Transformationen manipulieren direkt das Koordinatensystem. So wird beim Verschieben eines Rechtecks nicht nur das Element selbst bewegt, sondern gleich das gesamte Koordinatensystem neu gesetzt und erst dann das Rechteck gezeichnet. Die drei einfachen Grundtransformationen sind `scale()`, `rotate()` und `translate()`.

```
context.scale(x, y)
context.rotate(angle)
context.translate(x, y)
```

Beim Skalieren über `scale()` benötigen wir zwei Multiplikanden als Argumente für die Größenänderung der x- und y-Dimension, Rotationen mit `rotate()` verlangen den Drehwinkel im Uhrzeigersinn in Radiant, und Verschiebungen durch `translate()` definieren Offsets in x– und y-Richtung in Pixel. Bei Kombination der Methoden müssen die einzelnen Transformationen in umgekehrter Reihenfolge ausgeführt werden – aus Sicht des **JavaScript-Codes** also quasi von hinten nach vorne gelesen werden:

Um zuerst zu skalieren und dann zu rotieren, schreiben wir:

```
context.rotate(0.175);
context.scale(0.75,0.75);
context.fillRect(0,0,200,150);
```

Wollen wir zuerst rotieren und dann verschieben, lautet der JavaScript-Code:

```
context.translate(100,50);
context.rotate(0.175);
context.fillRect(0,0,200,150);
```

Vorsicht ist in jedem Fall geboten, wenn Rotationen im Spiel sind, denn diese werden immer mit dem Ursprungspunkt 0/0 als Drehmittelpunkt ausgeführt. Als Faustregel für die Reihenfolge der JavaScript-Aufrufe gilt: `rotate()` ist meist die letzte Aktion. Abbildung 1 zeigt ein Beispiel, das alle drei Grundmethoden verwendet und unser Yosemite-Bild einmal aus anderer Perspektive, quasi als Sprungschanze darstellt.

Werfen wir einen kurzen Blick auf den ebenso kurzen **Quellcode** für Abbildung 1:

```
image.onload = function() {
var rotate = 15;
var scaleStart = 0.0;
var scaleEnd = 4.0;
var scaleInc = (scaleEnd-scaleStart)/(360/rotate);
var s = scaleStart;
for (var i=0; i<=360; i+=rotate) {
s += scaleInc;
context.translate(540,260);
context.scale(s,s);
context.rotate(i*-1*Math.PI/180);
context.drawImage(image,0,0,120,80);
context.setTransform(1,0,0,1,0,0);
}
};
```

Sobald das Bild geladen ist, definieren wir den Rotationswinkel `rotate` mit 15°, die Start- und Endskalierungen `scaleStart` mit 0.0 sowie `scaleEnd` mit 4.0 und daraus abgeleitet das Inkrement für die Skalierung `scaleInc` mit dem Ziel, innerhalb einer ganzen Umdrehung die Endskalierung 4.0 zu erreichen. In der `for`-Schleife rotieren wir dann das Bild gegen den Uhrzeigersinn jeweils um 15°, skalieren es von 0.0 bis 4.0 und setzen seine linke obere Ecke auf die Koordinate 540/260.

Offen bleibt, was es mit der Methode `setTransform()` am Ende der `for`-Schleife auf sich hat und um wen es sich bei dem Skispringer handelt, der sich wagemutig vom Taft Point in den Abgrund stürzt. Ersteres werden wir gleich klären, Zweiteres wird sich erst bei einem Blick in den Quellcode des Beispiels auflösen.

Neben den drei Grundtransformationen `scale()`, `rotate()` und `translate()` stellt **Canvas** noch zwei weitere Methoden zur Veränderung des Koordinatensystems und damit der sogenannten **Transformationsmatrix** bereit: `transform()` und das bereits in dem Code angesprochene `setTransform()`:

```
context.transform(m11, m12, m21, m22, dx, dy);
context.setTransform(m11, m12, m21, m22, dx, dy);
```

Beiden gemeinsam sind die Argumente m11, m12, m21, m22, dx und dy, die folgende Tranformationseigenschaften repräsentieren:

Komponente	Inhalt
m11	Skalierung in x-Richtung
m12	Horizontaler Scherfaktor
m21	Vertikaler Scherfaktor
m22	Skalierung in y-Richtung
dx	Verschiebung in x-Richtung
dy	Verschiebung in y-Richtung

Der Hauptunterschied zwischen beiden liegt darin, dass `transform()` die zum Zeitpunkt des Aufrufs bestehende **Transformationsmatrix** durch Multiplikation weiter verändert, wohingegen `setTransform()` die bestehende Matrix mit der neuen überschreibt.

Die drei Grundmethoden könnten ebenso als Attribute für `transform()` oder `setTransform()` formuliert werden und sind im Grunde genommen nichts anderes als bequeme Kürzel für entsprechende Matrixtransformationen. Tabelle zeigt diese Attribute und listet noch weitere nützliche Matrizen zum Spiegeln (`flipX/Y`) und Neigen (`skewX/Y`) auf. Gradangaben beim Neigen erfolgen wieder in Radiant.

Methode	Transformationsmatrix (m11, m12, m21, m22, dx, dy)
`scale(x, y)`	x,0,0,y,0,0
`rotate(angle)`	cos(angle),sin(angle),-sin(angle), cos(angle),0,0
`translate(x, y)`	1,0,0,1,x,y

Methode	Transformationsmatrix (m11, m12, m21, m22, dx, dy)
`flipX()`	-1,0,0,1,0,0
`flipY()`	1,0,0,-1,0,0
`skewX(angle)`	1,0,tan(angle),1,0,0
`skewY(angle)`	1,tan(angle),0,1,0,0)

Bevor wir uns einem ausführlichem Beispiel zuwenden, muss noch erwähnt werden, dass sowohl `getImageData()` als auch `putImageData()` gemäß Spezifikation von Transformationen unabhängig sind. Der Aufruf `getImageData`(0,0,100,100) greift immer auf das 100 x 100 Pixel große Quadrat in der linken oberen Ecke des Canvas zu, egal ob das **Koordinatensystem** verschoben, skaliert oder rotiert wurde. Ebenso verhält es sich bei `putImageData`(imagedata,0,0), wo wiederum die linke obere Ecke als Anfasspunkt zum Auftragen des Inhalts von `imagedata` dient.

Widmen wir uns jetzt dem angekündigten Beispiel, in dem wir alle gelernten Methoden zum Transformieren noch einmal anwenden. Abbildung 2 zeigt das ansprechende Resultat – eine Collage von drei Bildausschnitten unseres Yosemite-Bildes mit Spiegeleffekt im Pseudo-3D-Raum.

Beginnen wir mit dem Ausstanzen der drei quadratischen Ausschnitte für Taft Point, Merced River und El Capitan. Das Ergebnis speichern wir im Array `icons`.

```
var icons = [
clipIcon(image,0,100,600,600),
clipIcon(image,620,615,180,180),
clipIcon(image,550,310,400,400)
];
```

Das Zuschneiden und Anpassen der unterschiedlich großen Ausschnitte erledigt die Funktion `clipIcon()`. In ihr wird zuerst ein neuer **In-memory-Canvas** mit 320 x 320 Pixeln Größe erzeugt, auf den wir dann mit `drawImage()` das entsprechend verkleinerte (oder vergrößerte) Icon kopieren und mit einem 15 Pixel breiten, weißen Rahmen versehen.

```
var clipIcon = function(img,x,y,width,height) {
var cvs = document.createElement("CANVAS");
var ctx = cvs.getContext("2d");
cvs.width = 320;
cvs.height = 320;
ctx.drawImage(img,x,y,width,height,0,0,320,320);
ctx.strokeStyle = "#FFF";
ctx.lineWidth = 15;
ctx.strokeRect(0,0,320,320);
return cvs;
};
```

Für jeden dieser drei Ausschnitte erzeugen wir in einem zweiten Schritt den Spiegeleffekt und speichern ihn im Array `effects`.

```
var effects = [];
for (var i=0; i<icons.length; i++) {
effects  = createReflection(icons );
}
```

Die Hauptarbeit findet dabei in der Funktion `createReflection()` statt, deren leicht modifizierter Code einem Posting in *Charles Yings blog about art, music, and the art of technology* über den CoverFlow-Effekt des iPhones entstammt.

```
var createReflection = function(icon) {
var cvs = document.createElement("CANVAS");
var ctx = cvs.getContext("2d");
cvs.width = icon.width;
cvs.height = icon.height/2.0;
// flip
ctx.translate(0,icon.height);
ctx.scale(1,-1);
ctx.drawImage(icon,0,0);
// fade
ctx.setTransform(1,0,0,1,0,0);
ctx.globalCompositeOperation = "destination-out";
var grad = ctx.createLinearGradient(
0,0,0,icon.height/2.0
);
grad.addColorStop(0,"rgba(255,255,255,0.5)");
grad.addColorStop(1,"rgba(255,255,255,1.0)");
ctx.fillStyle = grad;
ctx.fillRect(0,0,icon.width,icon.height/2.0);
```

```
return cvs;
};
```

In `createReflection()` wird zuerst über einen weiteren In-memory-Canvas die untere Hälfte des in `icon` übergebenen Bildausschnitts umgeklappt. Erinnern wir uns an die Kürzel für Transformationsmatrizen, könnten wir das Umklappen mit der Matrix für `flipY()` realisieren. In diesem Fall verwenden wir allerdings eine weitere Variante zum Spiegeln, und zwar jene über die Methode `scale()`. Dabei entspricht `scale(1,-1)` der Methode `flipY()` und `scale(-1,1)` der Methode `flipX()`. Der Fade-out-Effekt entsteht durch eine Gradiente von semitransparentem Weiß zu opakem Weiß, die mithilfe der Compositing Methode `destination-out` über das Icon gelegt wird.

Damit sind die einzelnen Bildausschnitte definiert, und wir können mit dem Zeichnen beginnen. Eine **Gradiente** von Schwarz nach Weiß mit beinahe Schwarz in der Hälfte des Verlaufs erweckt den Eindruck eines Raums, in dem wir die drei Ausschnitte anschließend platzieren.

```
var grad = context.createLinearGradient(
0,0,0,canvas.height
);
grad.addColorStop(0.0,"#000");
grad.addColorStop(0.5,"#111");
grad.addColorStop(1.0,"#EEE");
context.fillStyle = grad;
context.fillRect(0,0,canvas.width,canvas.height);
```

Am einfachsten können wir das mittlere Bild vom Merced River über `setTransform()` positionieren und dann mit einem Spiegeleffekt zeichnen.

```
context.setTransform(1,0,0,1,440,160);
context.drawImage(icons[1],0,0,320,320);
context.drawImage(effects[1],0,320,320,160);
```

Die Breite des El-Capitan-Bildes skalieren wir für einen besseren 3D-Eindruck um den Faktor 0.9, neigen das Resultat mit der **Matrix** für `skeyY()` um 10° nach unten und positionieren das Resultat rechts von der Mitte.

```
context.setTransform(1,0,0,1,820,160);
context.transform(1,Math.tan(0.175),0,1,0,0);
context.scale(0.9,1);
context.drawImage(icons[2],0,0,320,320);
context.drawImage(effects[2],0,320,320,160);
```

Etwas komplizierter ist dann das Zeichnen des Taft-Point-Bildes links, denn nachdem beim Neigen die linke obere Ecke des Ausschnitts den Ankerpunkt bildet, müssen wir um 10° nach oben neigen und das Resultat dann wieder nach unten schieben. Der Satz des Pythagoras hilft uns beim Ermitteln des nötigen dy-Wertes: Er ergibt sich als Tangens des Drehwinkels in Radiant mal der Länge der Ankathete, die der Breite des Icons entspricht, also `Math.tan(0.175)*320`.

Zusätzlich muss noch die Skalierung der Bildbreite um 0.9 durch Verschiebung um 320*0.1 nach rechts ausgeglichen werden.

```
context.setTransform(1,0,0,1,60,160);
context.transform(1,Math.tan(-0.175),0,1,0,0);
context.translate(320*0.1,Math.tan(0.175)*320);
context.scale(0.9,1);
context.drawImage(icons[0],0,0,320,320);
context.drawImage(effects[0],0,320,320,160);
```

Damit wäre unser bisher schwierigstes **Canvas-Beispiel** geschafft – das Ergebnis kann sich sehen lassen und verlangt geradezu danach, im JPEG- oder PNG-Format gespeichert zu werden.

Beispiel demonstriert die Macht von Koordinatensystemtransformationen, indem mit den Methoden `translate()`, `rotate()` und `scale()` rekursiv ein Koch-Schneeflocken-Fraktal gezeichnet wird. Die Ausgabe dieses Beispiels sehen Sie in Abbildung 1, das Koch-Schneeflocken mit 0, 1, 2, 3 und 4 Rekursionsebenen zeigt.

Der Code, der diese Figuren erzeugt, ist elegant, aber sein Gebrauch rekursiver Koordinatensystemtransformationen macht ihn recht schwer verständlich. Selbst wenn Sie nicht allen Feinheiten folgen können, sollten Sie beachten, dass der Code nur einen einzigen Aufruf der Methode `lineTo()` enthält. Jedes Liniensegment in Abbildung 1 wird sögezeichnet:

```
c.lineTo(len, 0);
```

Der Wert der Variablen len ändert sich während der Ausführung des Programms nicht. Die Länge der Liniensegmente wird alsödurch Translations-, Rotations- und Skalierungsoperationen bestimmt.

Beispiel: Eine Koch-Schneeflocke mit Transformationen

```
var deg = Math.PI/180; // Zur Umwandlung von Grad in Radiant
```

```
// Ein Ebene-n-Koch-Schneeflocken-Fraktal im Kontext c
// zeichnen, mit der linken unteren Ecke bei (x,y) und der
// Seitenlänge len.
function snowflake(c, n, x, y, len) {
c.save();             // Aktuelle Transformation speichern
c.translate(x,y);     // An den Startpunkt verschieben
c.moveTo(0,0);        // Hier einen neuen Teilpfad beginnen
leg(n);               // Ersten Schenkel des Fraktals
                      // zeichnen
c.rotate(-120*deg);   // 120 Grad gegen die Uhr drehen
leg(n);               // Den zweiten Schenkel zeichnen
c.rotate(-120*deg);   // Wieder drehen
leg(n);               // Den letzten Schenkel zeichnen
c.closePath();        // Den Teilpfad schließen
c.restore();          // Die ursprüngliche Transformation
                      // wiederherstellen

// Einen einzelnen Schenkel einer Ebene-n-Koch-
// Schneeflocke zeichnen. Diese Funktion belässt den
// aktuellen Punkt am Ende der gerade gezeichneten Linie
// und verschiebt das Koordinatensystem so, dass der
// aktuelle Punkt (0,0) ist. Das heißt, dass Sie nach
// Zeichnung einer Linie leicht rotate() aufrufen können.
function leg(n) {
c.save();             // Aktuelle Transformation
                      // speichern
if (n == 0) {         // Nicht rekursiver Fall:
c.lineTo(len, 0);     // bloß eine horizontale Linie
                      // zeichnen
}
else { // Rekursiver Fall:
       // vier Teillinien dieser Form zeichnen: \/
c.scale(1/3,1/3);     // Unterlinien haben 1/3 der
                      // Größe
leg(n-1);             // Die erste Unterlinie
                      // zeichnen
c.rotate(60*deg);     // Um 60 Grad im Uhrzeiger-
                      // sinn drehen
leg(n-1);             // Die zweite Unterlinie
                      // zeichnen
c.rotate(-120*deg);   // Um 120 Grad zurückdrehen
leg(n-1);             // Dritte Unterlinie zeichnen
c.rotate(60*deg);     // Wiederzurück zur
                      // ursprünglichen Richtung
leg(n-1);             // Letzte Unterlinie zeichnen
}
```

```
c.restore();          // Die Transformation
                      // wiederherstellen
c.translate(len, 0);  // Zum Ende der Linie
                      // verschieben
}
}

// Schneeflocken-Fraktale der Ebene 0 bis 4 zeichnen
snowflake(c,0,5,115,125);     // Ein gleichseitiges Dreieck
snowflake(c,1,145,115,125);   // Ein sechseckiger Stern
snowflake(c,2,285,115,125);   // Ähnlich wie eine Schneeflocke
snowflake(c,3,425,115,125);   // Noch schneeflockiger
snowflake(c,4,565,115,125);   // Jetzt wird es richtig fraktal!
c.stroke();                   // Diesen komplizierten Pfad
                              // zeichnen
```

Die Methode `isPointInPath()` ermittelt, ob ein angegebener Punkt in den aktuellen Pfad (oder auf seine Grenzen) fällt, und liefert `true`, wenn das der Fall ist, andernfalls `false`. Der Punkt, den Sie dieser Methode übergeben, befindet sich im Standardkoordinatensystem und wird nicht transformiert. Das macht die Methode zur Treffererkennung geeignet, was bedeutet, zu prüfen, ob ein Mausklick über einer bestimmten Figur erfolgte.

Sie können die `clientX`– und `clientY`-Felder eines `MouseEventObjekts` allerdings nicht direkt an `isPointInPath()` übergeben. Erst müssen die Koordinaten des **Maus-Events** so übersetzt werden, dass sie sich auf das **Canvas**-Element und nicht auf das Window-Objekt beziehen. Außerdem müssen die Koordinaten des Maus-Events entsprechend skaliert werden, wenn die Bildschirmgröße des Canvas nicht den tatsächlichen Maßen entspricht. Beispiel 1 zeigt eine Hilfsfunktion, mit der Sie prüfen können, ob ein Maus-Event über dem aktuellen Pfad eintrat.

Beispiel 1: Prüfen, ob ein Maus-Event über dem aktuellen Pfad liegt

```
// Liefert true, wenn das Maus-Event über dem aktuellen Pfad
// im angegebenen CanvasRenderingContext2D-Objekt ist.
function hitpath(context, event) {
var canvas, bb, x, y;

// Das <canvas>-Element aus dem Kontext-Objekt abrufen
canvas = context.canvas;

// Canvas-Größe und Position abrufen
bb = canvas.getBoundingClientRect();

// Die Maus-Event-Koordinaten in Canvas-Koordinaten
// umwandeln
```

```
x = (event.clientX-bb.left) * (canvas.width/bb.width);
y = (event.clientY-bb.top) * (canvas.height/bb.height);

// isPointInPath() mit den transformierten Koordinaten
// aufrufen
return context.isPointInPath(x,y);
}
```

Folgendermaßen könnten Sie die Funktion `hitpath()` in einem Event-Handler nutzen:

```
canvas.onclick = function(event) {
if (hitpath(this.getContext("2d"), event) {
alert("Treffer!"); // Klick auf den aktuellen Pfad
}
};
```

Statt einer pfadbasierten Treffererkennung können Sie auch `getImageData()` nutzen, um zu prüfen, ob das Pixel unter der Maus gezeichnet wurde. Ist das **Pixel** (bzw. sind die Pixel) vollständig transparent, wurde an diesem Punkt nicht gezeichnet. Dann liegt also kein Treffer vor. Beispiel 2 zeigt, wie Sie eine derartige Treffererkennung umsetzen können.

Beispiel 2: Prüfen, ob ein Maus-Event über einem gemalten Pixel erfolgte

```
// Liefert true, wenn das angegebene Maus-Event über einem
// nicht transparenten Pixel liegt.
function hitpaint(context, event) {

// Maus-Event-Koordinaten in Canvas-Koordinaten umwandeln
var canvas = context.canvas;
var bb = canvas.getBoundingClientRect();
var x=(event.clientX-bb.left)*(canvas.width/bb.width);
var y=(event.clientY-bb.top)*(canvas.height/bb.height);

// Das Pixel (oder die Pixel, wenn mehrere Gerätepixel
// einem CSS-Pixel entsprechen)
// an diesen Koordinaten abrufen
var pixels = c.getImageData(x,y,1,1);

// Ist Alpha bei einem der Pixel ungleich null, true
// liefern
for(var i = 3; i < pixels.data.length; i+=4) {
if (pixels.data  !== 0) return true;
}
```

```
// Andernfalls war es kein Treffer.
return false;
}
```

Formulare mit HTML5

Ein Vorteil der neuen Elemente und Attribute bei **Formularen** ist, dass die Eingabe für Benutzer erleichtert wird (zum Beispiel wird ein Kalender zum Eingeben eines Datums angeboten). Ein weiterer großer Vorteil ist die Möglichkeit, den Formular-Inhalt bereits vor dem Abschicken überprüfen zu können und den Benutzer auf mögliche Fehler hinzuweisen. Jetzt werden Sie vielleicht sagen, dass das ein alter Hut ist, denn diese Form der Überprüfung kennt man bereits seit vielen Jahren. Das stimmt, aber bisher musste dieser Schritt immer mithilfe von selbst programmiertem JavaScript-Code erledigt werden. Durch jQuery und ähnliche Bibliotheken wurde diese Aufgabe zwar deutlich erleichtert und der Code wartbarer, aber es bleibt die Abhängigkeit von einer externen Bibliothek.

Mit **HTML5** ändert sich das grundlegend: Sie definieren die Vorgaben für die Eingabefelder in HTML, und der Browser überprüft, ob die Felder korrekt ausgefüllt wurden. Das ist ein großer Schritt vorwärts, der viele redundante Zeilen JavaScript-Code unnötig macht. Ein Minimalbeispiel wird Sie überzeugen:

```
<form method=get action=required.html>
<p><label>Ihre E-Mail-Adresse:
<input type=email name=email required>
</label>
<p><input type=submit>
</form>
```

Was passiert, wenn Sie das Formular in dem oben abgedruckten Listing ohne die Angabe einer E-Mail-Adresse abschicken, sehen Sie in Abbildung 1. Opera zeigt die Fehlermeldung: Sie müssen einen Wert eingeben. Wenn Sie die Opera-Benutzeroberfläche auf eine andere Sprache eingestellt haben, so erscheint diese Meldung in der entsprechenden Sprache. Natürlich kann man diese Fehlermeldungen auch noch mit JavaScript anpassen, mehr dazu erfahren Sie etwas später.

> Damit aber noch nicht genug:

Da das Feld vom Typ `email` definiert ist, meldet Opera auch einen Fehler, wenn keine gültige E-Mail-Adresse eingegeben wurde (vergleiche Abbildung 2).

Webkit-basierte Browser wie Google Chrome oder Safari unterstützen aktuell zwar die Überprüfung, geben aber keine Fehlermeldung aus. Sie umrahmen das ungültige Feld und positionieren den Cursor innerhalb des Feldes, um zumindest anzuzeigen, dass irgendetwas nicht stimmt.

Info

Bei aller Euphorie über die clientseitige Überprüfung von Formular-Eingaben dürfen Sie nicht vergessen, dass dieser Schritt die serverseitige Kontrolle nicht überflüssig machen kann. Ein potenzieller Angreifer kann diese Mechanismen mit wenig technischem Aufwand umgehen.

Das `invalid`-Event

Bei der Überprüfung des Formulars wird für **Elemente**, die einen ungültigen Inhalt haben, das Event `invalid` ausgelöst. Das können wir uns zunutze machen und individuell auf fehlerhafte Werte reagieren.

```
window.onload = function() {
var inputs = document.getElementsByTagName("input");
for (var i=0; i<inputs.length; i++) {
inputs .addEventListener("invalid", function() {
alert("Feld "+this.labels[0].innerHTML+" ist ungültig");
this.style.border = 'dotted 2px red';
}, false);
}
}
```

Nachdem die Seite geladen ist, wird eine Liste aller `input`-Elemente generiert. An jedes Element wird anschließend ein Event-Listener angehängt, der den Fehlerfall behandelt. Im vorliegenden Beispiel wird ein `alert`-Fenster geöffnet, und das Element erhält einen rot gepunkteten Rahmen. Für den Text im `alert`-Fenster wird die Beschriftung des `input`-Elements verwendet.

Bei Formularen mit vielen Eingabefeldern ist diese Vorgehensweise nicht ideal. Der Anwender muss für jede fehlerhafte Eingabe die OK-Schaltfläche anklicken und anschließend im Formular das betreffende Feld suchen und erneut ausfüllen. Manchmal wäre es günstiger, wenn der Anwender sofort nach dem Ausfüllen eine Benachrichtigung bekäme, sollte das Feld einen ungültigen Inhalt enthalten. Das wollen wir im nächsten Abschnitt probieren.

Die `checkValidity`-Funktion

Um die Überprüfung eines `input`-Elements auszulösen, wird die `checkValidity`-Funktion für dieses Element aufgerufen. Was normalerweise passiert, wenn das Formular abgeschickt wird, kann man aber auch „von Hand“ starten:

```
<input type=email name=email onchange="this.checkValidity();">
```

Gibt man eine ungültige E-Mail-Adresse ein und verlässt man das Eingabefeld (entweder mit der Tabulator-Taste oder durch einen Mausklick auf eine andere Stelle im Browser), so meldet der Browser (zurzeit zumindest Opera) den Fehler unmittelbar (vergleiche Abbildung 2). Noch eleganter wird die Fehlerbehandlung, wenn wir an das `onchange`-Event von allen `input`-Elementen eine Funktion zum Überprüfen der Eingabe hängen.

```
window.onload = function() {
var inputs = document.getElementsByTagName("input");
for (var i=0; i<inputs.length; i++) {
if (!inputs .willValidate) {
continue;
}
inputs .onchange = function() {
if (!this.checkValidity()) {
this.style.border = 'solid 2px red';
this.style.background = '';
} else {
this.style.border = '';
this.style.background = 'lightgreen';
}
}
}
}
```

In der bereits bekannten **Schleife** über alle `input`-Elemente wird als Erstes kontrolliert, ob das Element für eine Überprüfung zur Verfügung steht. Enthält `willValidate` nicht den Wert `true`, wird die Schleife mit dem nächsten Element fortgesetzt. Andernfalls wird das `onchange`-Event mit einer anonymen Funktion belegt, in der die `checkValidity`-Funktion aufgerufen wird. `this` bezieht sich innerhalb der anonymen Funktion auf das `input`-Element. Schlägt die Gültigkeitsprüfung fehl, so wird das Element mit einer roten Umrandung versehen; im anderen Fall wird der Hintergrund des Elements hellgrün eingefärbt. Das Zurücksetzen der Hintergrundfarbe beziehungsweise des Rahmens auf eine leere Zeichenkette ist notwendig, damit der Browser bei einer richtigen Eingabe nach einer falschen Eingabe die Formatierung wieder auf den Standardwert stellt. Abbildung 3 zeigt, wie die `checkValidity`-Funktion einen Fehler bei der Zeiteingabe anmahnt.

Wenn Sie die Fehlerbehandlung lieber noch interaktiver gestalten möchten,

können Sie statt des `onchange`-Events auch das in HTML5 neue `oninput`-Event verwenden. Anders als

onchange, das beim Verlassen des Feldes gestartet wird, kommt `oninput` nach jedem veränderten Zeichen zum Einsatz. Was bisher etwas mühsam mithilfe der Tastatur-Events `keyup` beziehungsweise `keydown` programmiert wurde, übernimmt jetzt das `oninput`-Event. Ein weiterer Vorteil von `oninput` ist, dass der Event-Listener nur einmal an das ganze Formular angehängt werden muss und nicht an jedes einzelne `input`-Element. Für das vorangegangene Beispiel könnte man damit auf den gesamten **JavaScript-Code** verzichten und die Formular-Definition wie folgt ändern:

```
<form method=get oninput="this.checkValidity();" action=checkValidity.html>
```

Man verzichtet damit zwar auf das Verändern von Rahmen und Hintergrundfarbe, verkürzt aber auch den Quelltext deutlich. Das unmittelbare Reagieren auf einen Tastendruck kann in manchen Fällen sehr hilfreich sein, beim Ausfüllen eines Formularfelds reicht es aber meist, wenn der Inhalt erst dann überprüft wird, wenn das Feld vollständig ausgefüllt wurde.

Fehlerbehandlung mit `setCustomValidity()`

Wenn Ihnen all die bisher vorgestellten Möglichkeiten zur Fehlerbehandlung noch nicht ausreichend erscheinen, können Sie sich auch selbst eine Funktion zur Überprüfung des Inhalts programmieren. Im folgenden Beispiel wird ein Eingabefeld vom Typ `email` definiert, wodurch der Browser schon die Überprüfung der gültigen E-Mail-Adresse übernimmt. Zusätzlich möchten wir aber noch drei E-Mail-Domains ausschließen.

```
var invalidMailDomains = [
'hotmail.com', 'gmx.com', 'gmail.com' ];
function checkMailDomain(item) {
for (var i=0; i<invalidMailDomains.length; i++) {
if (item.value.match(invalidMailDomains +'$')) {
item.setCustomValidity('E-Mail-Adressen von'+invalidMailDomains +' sind nicht
erlaubt.');
} else {
item.setCustomValidity('');
}
item.checkValidity();
}
}
```

Jedes Element im Array `invalidMailDomains` wird mit dem Wert des `input`-Elements verglichen. Die **JavaScript-Funktion** `match()` arbeitet mit regulären Ausdrücken, weshalb wir an den Domain-Namen noch ein $-Zeichen anhängen, das das Ende der Zeichenkette spezifiziert. Stimmen die Zeichenketten überein, so wird die `setCustomValidity`-Funktion aufgerufen und ihr die entsprechende Fehlermeldung übergeben. Handelt es sich nicht um einen Domain-Namen aus dem Array, wird `setCustomValidity()` mit einer leeren Zeichenkette aufgerufen. Intern wird dadurch die Variable `validationMessage` an das

`input`-Element angehängt, die Opera anschließend auch korrekt anzeigt (vergleiche Abbildung 4). Der abschließende Aufruf der `checkValidity`-Funktion löst die Überprüfung aus und führt zu der eben erwähnten Fehlermeldung.

Zusammenfassung der Gültigkeitsprüfungen

Tabelle zeigt eine Auflistung aller `input`-Attribute beziehungsweise Validierungsfunktionen, die bei der Gültigkeitsüberprüfung zur Verfügung stehen, und die Szenarien, in denen sie auftreten.

Attribut/Funktion	Problem
`required`	Es wurde kein Wert für das Feld eingegeben.
`type=email, url`	Der eingegebene Wert entspricht nicht dem verlangten Typ.
`pattern`	Der eingegebene Wert entspricht nicht dem geforderten Muster.
`maxlength`	Der eingegebene Wert ist länger als erlaubt.
`min, max`	Der eingegebene Wert ist zu klein bzw. zu groß.
`step`	Die verlangte Schrittweite beim eingegebenen Wert wurde nicht eingehalten.
`setCustomValidity()`	Die zusätzlich aufgestellten Kriterien für dieses Feld wurden nicht erfüllt.

Oder doch nicht prüfen? `formnovalidate`

Nun, da wir uns so ausführlich mit der Fehlerbehandlung beschäftigt haben, folgt die Erklärung, wie man sich an all den Regeln vorbeimogeln kann: mit dem Attribut `formnovalidate`. Im ersten Moment erscheint es vielleicht ein bisschen merkwürdig, all die mühevoll definierten Regeln einfach so beiseite zu lassen und das Formular auch ohne eine Prüfung abzuschicken. Die Spezifikation enthält dazu eine kurze Erklärung, die das Rätsel schnell löst. Der typische Anwendungsfall für das Überspringen der Prüfung ist ein Formular, das der Anwender nicht auf einmal ausfüllen kann oder will. Dadurch, dass man das `formnovalidate`-Attribut einer `submit`-Schaltfläche hinzufügt, kann der bisher eingegebene Inhalt zwischengespeichert werden.

Info

Beim Abschicken des Formulars mit `formnovalidate` werden die bereits ausgefüllten Felder an den Server gesendet. Um ein mögliches Zwischenspeichern muss sich die Server-Anwendung kümmern.

Stellen Sie sich vor, Sie füllen ein Support-Formular für Ihre defekte Digitalkamera aus.

Nachdem Sie ausführlich alle Angaben zu dem aufgetretenen Fehler gemacht haben, wird auf der Internet-Seite nach der Seriennummer der Kamera gefragt. Da die Sie die Kamera aber gerade nicht zur Hand haben und die mühevoll eingegebenen Informationen nicht verlieren möchten, klicken Sie auf die ZWISCHENSPEICHERN-Schaltfläche und können sich in Ruhe auf die Suche nach der Kamera begeben. Diese Schaltfläche wird wie folgt definiert:

```
<input type=submit formnovalidate value="Zwischenspeichern" name=save id=save>
```

Im abschließenden Beispiel wird die Idee mit dem **Support-Formular** vollständig ausgearbeitet.
Beispiel: Ein Support-Formular
In diesem Beispiel werden die bisher vorgestellten neuen Elemente und Attribute in einem Formular verwendet. Das Formular könnte, in einer erweiterten Form, auf der Webseite eines Elektronik-Verkäufers Verwendung finden.

Zu Beginn werden persönliche Informationen vom Klienten abgefragt (in diesem Beispiel nur der Name, eine E-Mail-Adresse, eine Telefon- und eine Faxnummer). Der zweite Teil des Formulars betrifft die technischen Daten und den Defekt des Geräts. Im untersten Teil der Webseite wird ein Fortschrittsbalken angezeigt, der den Anwender aufmuntern soll, das Formular fertig auszufüllen (vergleiche Abbildung 5).

Der **HTML-Code** für das Formular beginnt mit dem Laden einer externen JavaScript-Datei und dem bereits bekannten Aufruf `window.onload`.

```
<script src="support.js"></script>
<script>
window.onload = function() {
initEventListener();
}
</script>
```

Die `initEventListener`-Funktion läuft über alle `input`-Elemente und belegt das `onchange`-Event mit einer anonymen Funktion, die das entsprechende Element auf seine Gültigkeit überprüft.

```
function initEventListener() {
var inputs = document.getElementsByTagName("input");
for (var i=0; i<inputs.length; i++) {
if (!inputs .willValidate) {
continue;
}
```

```
inputs .onchange = function() {
this.checkValidity();
}
}
}
```

Der Event-Listener wird nur dann angehängt, wenn das Element eine Möglichkeit der Überprüfung hat. Im vorliegenden Beispiel haben die beiden Schaltflächen zum Absenden beziehungsweise zum Zwischenspeichern keine Überprüfungsmöglichkeit und bekommen daher kein `onchange`-Event. Wie im vorangegangenen Abschnitt schon erklärt wurde, ist die Überprüfung der einzelnen Formularfelder nach dem Ausfüllen des Feldes angenehmer als die Überprüfung des gesamten Formulars mit dem `oninput`-Event.

Um die Benutzerfreundlichkeit des Formulars zu verbessern,

wollen wir die als `required` markierten Elemente hervorheben, damit dem Anwender sofort klar wird, welches die wichtigen Felder sind. Glücklicherweise müssen wir dazu nicht jedes Element mit einem extra Stil versehen, **CSS3** bringt den neuen Selektor `:required` mit, der genau für diesen Fall gedacht ist. Die folgende Anweisung rahmt alle vorgeschriebenen Elemente mit oranger Farbe ein.

```
:required {
border-color: orange;
border-style: solid;
}
```

Die Definition der einzelnen `input`-Felder birgt keine großen Überraschungen. E-Mail-Adresse und Telefonnummer haben ihre eigenen Typen und sind vorgeschrieben; das Datum, an dem der Defekt auftrat, ist vom Typ `date` und kann daher mit einem Kalenderfenster ausgewählt werden. Das zweispaltige Layout im oberen Teil der Webseite wird mit `div`-Elementen erreicht, die nebeneinander liegen. Trotzdem möchten wir, dass Anwender, die die Tabulatortaste zum Weiterspringen verwenden, das Formular von oben nach unten ausfüllen und nicht, der HTML-Logik folgend, zuerst die linke und dann die rechte Spalte.

Erreicht wird das mithilfe des `tabindex`-Attributs, wodurch ein Drücken der Tabulatortaste in einem Feld den Cursor auf das Feld mit dem nächsthöheren `tabindex`-Wert stellt.

```
<div style="float:left">
<p><label>Ihr Name:
<input tabindex=1 type=text required autofocus placeholder="Max Mustermann"
name=name>
</label>
```

```
<p><label>E-Mail
<input tabindex=3 type=email name=email required>
</label>
</div>
<div style="float:left;margin-left:10px;">
<p><label>Telefonnummer
<input tabindex=2 type=tel name=tel required>
</label>
<p><label>Faxnummer
<input tabindex=4 type=tel name=fax>
</label>
</div>
```

Etwas spannender wird es bei den `textarea`-Feldern. Viel Neues brachte HTML5 für diesen Typ nicht, aber wie Sie in Abbildung 5 sehen können, enthält jedes Textfeld eine kleine grafische Anzeige oberhalb, die darstellt, wie viele Zeichen in dem Feld noch getippt werden können. Sicherlich haben Sie es gleich erkannt: Wir verwenden dazu das neue `meter`-Element, das uns bereits aus Anzeigen von Messgrößen mit meter, vertraut ist.

```
<p><label>Fehlermeldung
<textarea placeholder="Lens. Camera restart." name=errmsg required rows=5
cols=50 title="maximal 200 Zeichen">
</textarea>
</label>
<meter value=0 max=200 tabindex=-1></meter>
```

Das `meter`-Element wird mit einem Maximalwert von 200 initialisiert, exakt dem Wert, der im `title`-Attribut der `textarea` als Maximum angegeben ist. Gibt ein Anwender mehr als die vorgeschriebenen Zeichen ein, wird das `meter`-Element rot und warnt vor dem zu langen Text. Der Browser wird den zu langen Text aber trotzdem abschicken, da wir die `textarea` nicht limitiert haben. Es handelt sich hier also mehr um einen Hinweis, als um eine strikte Vorgabe. Die JavaScript-Funktion zum Aktualisieren der `meter`-Elemente lautet `updateTAMeters()` und wird für alle `textareas` ausgeführt:

```
function updateTAMeters() {
var textfs = document.getElementsByTagName("textarea");
for(var i=0; i<textfs.length; i++) {
textfs .labels[0].nextSibling.value = textfs .textLength;
}
}
```

Der Vorteil der Schleife ist, dass wir nun beliebig viele `textarea`-Elemente hinzufügen können, und sofern sie ein `meter`-Element besitzen, werden diese automatisch aktualisiert. Um das zu erreichen, müssen wir zu einem Trick aus der DOM-Kiste greifen: Die in dem oben stehenden Code fett gedruckte Zuweisung greift auf die DOM-Funktion `nextSibling` zu, einen Verweis auf das folgende Element. Führen wir uns

zum besseren Verständnis noch einmal den HTML-Code für das Textfeld und den Status-Balken vor Augen. Das `textarea`-Element ist von einem `label`-Element eingeschlossen, auf das das gesuchte `meter`-Element folgt. Um vom `textarea`-Element auf das `meter`-Element zu kommen, verwenden wir die `labels`-Eigenschaft des Textfeldes. Dabei handelt es sich um ein NodeList-Array, von dem uns das erste Element (also das mit dem Index 0) interessiert, weil das darauffolgende Element (der `nextSibling`) das `meter`-Element ist.

> Bei genauerer Betrachtung ist die Vorgehensweise also gar nicht so kompliziert,

sie birgt aber ihre Tücken. Sollte sich zwischen dem abgeschlossenen `label`-Element und dem `meter`-Element ein Leerzeichen oder ein Zeilenumbruch verirren, dann funktioniert unsere Status-Anzeige nicht mehr. Der `next-Sibling` ist dann nämlich ein **Text-Element**, und in der `for`-Schleife erreichen wir das `meter`-Element nicht mehr.

Als Nächstes wollen wir uns um die Fortschrittsanzeige am Ende des Formulars kümmern. Leicht zu erraten war, dass es sich dabei um ein `progress`-Element handelt, spannender wird, wie sich das Aktualisieren dieses Elements in JavaScript elegant ausdrücken lässt. Zuerst sehen Sie hier den HTML-Code für das Element:

```
<label>Fortschritt:
<progress id=formProgress value=0 tabindex=-1></progress>
</label>
```

Das `progress`-Element bekommt eine `id`, einen Anfangswert von 0 (`value`) und einen negativen `tabindex` zugewiesen, was dazu führt, dass das Element nie mit der Tabulator-Taste angesprungen wird. Um alles Weitere kümmert sich die JavaScript-Funktion `updateProgress()`.

```
function updateProgress() {
var req = document.querySelectorAll(":required");
count = 0;
for(var i=0; i<req.length; i++) {
if (req .value != '') {
count++;
}
}
var pb = document.getElementById("formProgress");
pb.max = req.length;
pb.value = count;
}
```

Da sich der Fortschrittsbalken nur auf diese Elemente beziehen soll, die unbedingt eingegeben werden müssen, verwenden wir die Funktion `querySelectorAll()` und übergeben ihr die Zeichenkette `:required`. Als Ergebnis erhalten wir eine NodeList, die nur Elemente beinhaltet, die das `required`-Attribut gesetzt haben. Anschließend läuft eine Schleife über diese Elemente und überprüft, ob das `value`-Attribut nicht mit einer leeren Zeichenkette übereinstimmt. Trifft diese Bedingung zu (das bedeutet, es wurde schon ein Wert eingegeben), so wird die **Zählervariable** `count` um einen Wert erhöht. Abschließend wird der Maximalwert für das `progress`-Element auf die Anzahl aller `required`-Felder gesetzt und der Wert (`value`) auf die Zahl der nicht leeren Elemente.

Zum Abschicken des Formulars stehen zwei Schaltflächen zur Verfügung: eine mit der Beschriftung ZWISCHENSPEICHERN und eine mit dem Schriftzug ABSCHICKEN. Die Zwischenspeichern-Funktion wurde bereits in Abschnitt Oder doch nicht prüfen? `formnovalidate`, erklärt, neu ist hier das Attribut `accesskey`.

```
<p><input accesskey=Z type=submit formnovalidate value="Zwischenspeichern [Z]"
name=save id=save>
<input accesskey=A type=submit name=submit id=submit value="Abschicken [A]">
```

Zwar sind Tastaturkürzel nicht neu in HTML5, viel Verwendung fanden sie bisher aber nicht. Ein Problem mit Tastaturkürzeln ist, dass sie auf unterschiedlichen Plattformen mit unterschiedlichen Tastaturkombinationen aktiviert werden und man nie so genau weiß, welche Taste man jetzt zu dem Kürzel drücken muss. Die **HTML5-Spezifikation** hat auch hierzu einen Vorschlag parat: Der Wert des `accessKeyLabel` soll eine Zeichenkette zurückgeben, die dem korrekten Wert auf der verwendeten Plattform entspricht. Diesen Wert könnte man dann in der Beschriftung der Schaltfläche oder in deren `title`-Attribut verwenden. Leider war zu dem Zeitpunkt, als dieses Buch geschrieben wurde, kein Browser in der Lage, diese Zeichenkette auszugeben.

> So viel zu den neuen Möglichkeiten, die HTML5 für Formulare vorgesehen hat.

Für Webentwickler brechen angenehmere Zeiten an, da sie sich nicht mehr mit JavaScript-Bibliotheken für gängige Eingabeelemente wie zum Beispiel Datum und Uhrzeit herumschlagen müssen. Vor allem im Bereich der mobilen Endgeräte, auf denen die Texteingabe meist nicht so angenehm wie am Computer ist, werden die neuen Formular-Funktionen für ein angenehmeres Arbeiten sorgen. Auch die Formular-Überprüfung im Browser wird wesentlich zu einem übersichtlicheren und damit leichter wartbaren Code beitragen. Dabei sollten Sie nicht vergessen, dass die clientseitige Überprüfung kein Sicherheitsgewinn für die Server-Anwendung ist, da es einem Angreifer leicht möglich ist, diese Prüfungen zu umgehen.

Sollten Sie jetzt Lust bekommen haben, das neu erworbene Wissen über Formulare auf Ihrer Webseite auszuprobieren, so können Sie das bereits heute bedenkenlos tun. Die **Syntax** der neuen Elemente und Attribute ist so aufgebaut, dass auch ältere Browser keine Fehler produzieren. Zwar kommen Benutzer von diesen Browsern nicht in den Genuss der neuen Eingabeelemente und Funktionen, eine Texteingabe ist aber allemal möglich.

Falls Sie jemals eine komplexere Benutzeroberfläche gestaltet haben, wissen Sie, was die einfachen **HTML-Formularelemente** für eine Einschränkung darstellen. Sie müssen sich mit Textfeldern, Auswahlmenüs, Optionsschaltflächen, Kontrollkästchen und manchmal sogar mit den noch klobigeren Mehrfach-Auswahllisten herumärgern, die Sie dann auch noch Ihren Benutzern erklären müssen. („Halten Sie die `Strg`-Taste gedrückt, und klicken Sie auf die gewünschten Einträge. Außer Sie verwenden einen Mac. In diesem Fall drücken Sie die `Cmd`-Taste.

Also tun Sie, was alle guten Webentwickler tun – Sie nehmen Prototype, jQuery oder basteln Ihre eigenen Steuerelemente und Funktionen aus einer Kombination von **HTML**, **CSS** und **JavaScript**. Aber wenn Sie sich ein Formular mit Slidern, Kalendersteuerelementen, Spinboxen, Autovervollständigungsfeldern und visuellen Editoren ansehen, merken Sie schnell, was Sie sich da für einen Albtraum eingebrockt haben. Sie müssen sicherstellen, dass die von Ihnen eingefügten Steuerelemente keine Konflikte mit den anderen Steuerelementen von Ihnen oder aus anderen JavaScript-Bibliotheken auf der Seite verursachen. Sie können Stunden damit verwenden, einen Kalender-Picker zu implementieren, nur um dann festzustellen, dass die Prototype-Bibliothek Probleme hat, weil jQuery die Funktion `$()` übernommen hat. Also verwenden Sie die Methode `noConict()` von jQuery. Allerdings stellen Sie dann fest, dass der von Ihnen verwendete Farbwähler nicht mehr funktioniert, weil das Plugin nicht sorgfältig genug programmiert wurde.

Falls Sie jetzt lächeln, liegt das daran, dass Sie das alles schon erlebt haben. Falls Sie vor Wut kochen, dann aus demselben Grund. Es gibt jedoch Hoffnung. In diesem Artikel erstellen wir einige Webformulare mit neuen Arten von Formularfeldern. Und wir implementieren dabei auch Autofokus und Platzhaltertexte.

> Zum Abschluss zeigen wir Ihnen, wie Sie mit dem neuen Attribut `contenteditable` jedes beliebige HTML-Feld

in ein Steuerelement umwandeln, das Ihre Benutzer bearbeiten können.

Im Einzelnen behandeln wir die folgenden Funktionen:

E-Mail-Feld `[<input type="email">]`
Zeigt ein Formularfeld für E-Mail-Adressen an. [O10.1, IOS]

URL-Feld `[<input type="url">]`
Zeigt ein Formularfeld für URLs an. [O10.1, IOS]

Telefonnummern-Feld `[<input type="tel">]`
Zeigt ein Formularfeld für Telefonnummern an. [O10.1, IOS]

Suchfeld `[<input type="search">]`
Zeigt ein Formularfeld für Suchschlüsselwörter an. [C5, S4, O10.1, IOS]

Slider (Bereich) `[<input type="range">]`

Zeigt ein Slider-Steuerelement an. [C5, S4, O10.1]

Zahl `[<input type="number">]`
Zeigt ein Formularfeld für Zahlen an, häufig als Spinbox. [C5, S5, O10.1, IOS]

Datumsfelder `[<input type="date">]`
Zeigt ein Formularfeld für Datumswerte an. Unterstützt date, month oder week. [C5, S5, O10.1]

Datumswerte mit Uhrzeit `[<input type="datetime">]`
Zeigt ein Formularfeld für Datumswerte mit Uhrzeit an. Unterstützt `datetime`, `datetime-local` oder `time`. [C5, S5, O10.1]

Farbe `[<input type="color">]`
Zeigt ein Feld zum Auswählen von Farben an. [C5,S5] (Chrome 5 und Safari 5 verstehen das `color`-Feld, zeigen aber kein spezielles Steuerelement dafür an.)

Autofokus `[<input type="text" autofocus>]`
Bietet die Möglichkeit, einem bestimmten Formularelement automatisch den Fokus zu erteilen. [C5, S4]

Unterstützung für Platzhalter `[<input type="email" placeholder="me@example.com">]`
Ermöglicht die Anzeige von Platzhaltertext in einem Formularfeld. [C5, S4, F4]

Unterstützung für In-Place-Editing `[<p contenteditable>loremipsum</p>]`
Unterstützung für das In-Place-Editing von Inhalten im Browser. [C4, S3.2, IE6, O10.1]

Fangen wir damit an, etwas über die extrem nützlichen neuen Formularfeldtypen zu lernen.

Daten mit neuen Eingabefeldern beschreiben
In **HTML5** werden mehrere neue Typen von Eingabefeldern eingeführt, mit denen Sie den Datentyp besser beschreiben können, den Ihre Benutzer eingeben sollen. Zusätzlich zu den standardmäßigen Textfeldern, Optionsschaltflächen und Kontrollkästchen können Sie Elemente wie E-Mail-Felder, Kalender, Farbwähler, Spinboxen und Slider verwenden. Browser bieten den Benutzern für diese neuen Felder besser funktionierende Steuerelemente an – auch ohne JavaScript. Mobile Geräte und virtuelle Tastaturen für Tablet-PCs und Touchscreens können für diese Feldtypen unterschiedliche Tastaturlayouts anzeigen. So zeigt beispielsweise der iPhone-Browser Mobile Safari ein anderes Tastaturlayout an, wenn Benutzer Daten vom Typ URL oder `email` eingeben, wodurch Sonderzeichen wie (@), (`.`), (`:`) und (`/`) leichter zugänglich sind.
Verbessertes Formular für unser Projekt
Ein Geschäft arbeitet an einer neuen Webanwendung für das Projektmanagement, die es Entwicklern und Managern erleichtern soll, bezüglich der aktuellen Projekte auf dem neuesten Stand zu bleiben. Jedes Projekt hat einen Namen, eine Kontakt-E-Mail-Adresse sowie eine Test-URL, damit Manager während der Entwicklungsphase eine Vorschau der Website ansehen können. Es gibt auch Felder für das Startdatum, für die Priorität sowie für die geschätzte Stundenzahl bis zur Fertigstellung des Projekts. Außerdem möchte

der Entwicklungsmanager die Möglichkeit haben, jedem Projekt eine Farbe zuzuweisen, damit er in Berichten das jeweilige Projekt schnell erkennen kann.

Bauen wir also mit den neuen HTML5-Feldern ein Gerüst der Seite für die Projekteinstellungen.

Das grundlegende Formular

Wir erstellen ein einfaches **HTML-Formular**, das eine POST-Anfrage sendet. Nachdem es mit dem Namensfeld nichts Besonderes auf sich hat, verwenden wir dafür ein `text`-Feld.

```
<form method="post" action="/projects/1">
<fieldset id="personal_information">
<legend>Project Information</legend>
<ol>
<li>
<label for="name">Name</label>
<input type="text" name="name" autofocus id="name">
</li>
<li>
<input type="submit" value="Submit">
</li>
</ol>
</fieldset>
</form>
```

Beachten Sie, dass wir die Beschriftungen für das Formular in eine geordnete Liste schreiben. Beschriftungen sind von grundlegender Bedeutung bei der Erstellung barrierefreier Formulare. Das Attribut `for` referenziert die `id` des entsprechenden Formularelements. Dies hilft Bildschirmlesegeräten bei der Erkennung der Felder auf einer Seite. Die geordnete Liste bietet eine gute Möglichkeit, die Felder aufzulisten, ohne auf eine komplizierte Tabelle oder `div`-Strukturen zurückzugreifen. Außerdem können Sie das Markup auf diese Weise auch in der Reihenfolge schreiben, in der Ihre Besucher die Felder ausfüllen sollen.

Slider mithilfe von `range` erstellen

Slider werden häufig verwendet, damit Benutzer einen numerischen Wert bequem vergrößern oder verkleinern können. Dies ist eine ausgezeichnete Lösung, damit Manager schnell die Priorität eines Projekts visualisieren und anpassen können. Einen Slider implementieren Sie mit dem Typ `range`.

```
<label for="priority">Priority</label>
<input type="range" min="0" max="10" name="priority" value="0" id="priority">
```

Fügen Sie diesen Code genau wie das vorherige Feld in einem `li`-Element in das Formular ein.

Chrome und Opera implementieren beide ein Slider-Widget, das folgendermaßen aussieht:

Beachten Sie, dass wir den Wertebereich des Sliders mit `min` und `max` festgelegt haben. Dadurch wird auch der Wertebereich des Formularfelds eingegrenzt.

Zahlen und Spinboxen

Wir verwenden sehr häufig Zahlen. Und obwohl das Eingeben von Zahlen relativ einfach ist, können Spinboxen kleinere Anpassungen erleichtern. Eine Spinbox ist ein Steuerelement mit Pfeilen, über die der Wert im Feld erhöht oder verringert werden kann. Wir verwenden eine Spinbox für die geschätzte Stundenzahl. Auf diese Weise können die Stunden einfach angepasst werden.

```
<label for="estimated_hours">Estimated Hours</label>
<input type="number" name="estimated_hours" min="0" max="1000"
id="estimated_hours">
```

Opera unterstützt das Spinbox-Steuerelement. Es sieht folgendermaßen aus:

Standardmäßig können auch Werte direkt in die Spinbox eingetippt werden. Ebenso wie beim `range`-Slider können wir einen Mindest- und einen Maximalwert festlegen. Der so festgelegte Wertebereich gilt jedoch nicht für Werte, die direkt in das Feld eingetippt werden.

Beachten Sie auch, dass Sie die Größe der Schritte beim Erhöhen oder Verringern des Wertes festlegen können, indem Sie den Parameter `step` angeben. Dieser beträgt standardmäßig 1, kann aber ein beliebiger numerischer Wert sein.

Datumswerte

Die Erfassung des Startdatums für das Projekt ist ziemlich wichtig, wir möchten es daher so einfach wie möglich gestalten. Der `input type date` ist dafür die perfekte Wahl.

```
<label for="start_date">Start date</label>
<input type="date" name="start_date" id="start_date" value="2010-12-01">
```

Als dieser Artikel geschrieben wurde, war Opera der einzige Browser, der einen vollwertigen Kalender-Picker unterstützt.

Hier ein Beispiel für die Implementierung:

Safari 5.0 zeigt ein Feld, das dem `number`-Feld ähnlich ist und über Pfeile verfügt, mit denen das Datum erhöht oder verringert werden kann. Wird es leer gelassen, steht der Standardwert auf „1582“. Andere Browser rendern ein Textfeld.

E-Mail

Die **HTML5-Spezifikation** legt fest, dass der `input type email` für die Anzeige einer einzelnen E-Mail-Adresse oder einer Liste von E-Mail-Adressen entwickelt wurde. Er ist daher die perfekte Wahl für unser E-Mail-Feld.

```
<label for="email">Email contact</label>
<input type="email" name="email" id="email">
```

Auf mobilen Geräten kommt der größtmögliche Nutzen dieser Art von Formularfeld zum Tragen, da das Layout der virtuellen Tastatur angepasst und so die Eingabe von E-Mail-Adressen entsprechend vereinfacht wird.

URL

Es gibt auch einen eigenen Feldtyp für URLs. Das ist besonders praktisch, wenn Ihre Besucher ein iPhone verwenden. Denn dann wird ein ganz anderes Tastaturlayout mit Hilfsschaltflächen für die schnelle Eingabe von Webadressen angezeigt, ganz ähnlich wie die Tastatur zur Eingabe einer URL in die Adressleiste von Mobile Safari. Das Feld für die Eingabe der Test-URL lässt sich mit dem folgenden Code einfach einfügen:

```
<label for="url">Staging URL</label>
<input type="url" name="url" id="url">
```

Virtuelle Tastaturen zeigen für diesen Feldtyp ebenfalls ein anderes Layout an.

Farbe

Zu guter Letzt müssen wir noch eine Möglichkeit finden, Farbcodes einzugeben. Dafür verwenden wir den Typ `color`.

```
<label for="project_color">Project color</label>
<input type="color" name="project_color" id="project_color">
```

Als dieser Artikel geschrieben wurde, zeigte kein Browser ein Farbwähler-Steuerelement an, aber das soll Sie nicht davon abhalten, es zu verwenden. Sie schreiben korrekten Markup für die Beschreibung Ihres

Inhalts, und das wird sich in Zukunft als praktisch erweisen. Vor allem, wenn Sie Ausweichlösungen bereitstellen müssen.

Opera unterstützt bereits jetzt die meisten der neuen Steuerelemente, wie Sie in Abbildung 4 sehen können. Öffnen Sie die Seite dagegen in Firefox, Safari oder Google Chrome, werden Sie keinen nennenswerten Unterschied erkennen. Das müssen wir ändern.

Ausweichlösungen

Browser, die die neuen Typen nicht verstehen, verwenden stattdessen den Typ `text`. Ihre Formulare bleiben also trotzdem verwendbar. Außerdem können Sie eines der jQueryUI- oder YUI-Widgets mit dem Feld verknüpfen, um es entsprechend anzupassen. Wenn im Laufe der Zeit immer mehr Browser diese Steuerelemente unterstützen, können Sie die JavaScript-Krücken nach und nach entfernen.

Den Farbwähler ersetzen

Den Farbwähler können wir mit **jQuery** und den Attribut-Selektoren von **CSS3** einfach suchen und ersetzen. Wir suchen alle `input`-Felder vom Typ `color` und wenden das jQuery-Plugin „SimpleColor“ darauf an.

```
if (!hasColorSupport()){
$('input[type=color]').simpleColor();
}
```

Da wir die neuen Formulartypen in unserem Markup verwenden, brauchen wir keinen zusätzlichen Klassennamen oder anderes Markup einfügen, um Farbwähler zu kennzeichnen. Attributselektoren und HTML5 ergänzen sich prima.

Wir möchten das Farbwähler-Plugin nicht verwenden, wenn der Browser es nativ unterstützt. Also ermitteln wir mit JavaScript, ob der Browser Eingabefelder unterstützt, deren type den Wert `color` hat.

```
function hasColorSupport(){
input = document.createElement("input");
input.setAttribute("type", "color");
var hasColorType = (input.type !== "text");
// Lösung für teilweise Implementierung in Safari/Chrome
if(hasColorType){
var testString = "foo";
input.value=testString;
hasColorType = (input.value != testString);
}
return(hasColorType);
}
```

Zunächst erstellen wir mit JavaScript ein Element und legen das `type`-Attribut auf den Wert `color` fest. Anschließend lesen wir das `type`-Attribut wieder aus, um festzustellen, ob wir das Attribut erfolgreich setzen konnten. Wenn wir den Wert `color` zurückbekommen, wird dieser Typ unterstützt. Falls nicht,

müssen wir unser Skript anwenden.

Interessant wird es in Zeile 6. Safari 5 und Google Chrome 5 haben den Typ `color` teilweise implementiert. Sie unterstützen das Feld, zeigen aber kein Farb-Widget an, und so erhalten wir trotzdem nur ein Textfeld. Daher legen wir in unserer Prüfmethode einen Wert für das Eingabefeld fest und überprüfen, ob der Wert erhalten bleibt. Falls nein, können wir davon ausgehen, dass der Browser einen Farbwähler implementiert hat, da sich das Eingabefeld nicht wie ein Textfeld verhält.

Der vollständige Code zum Ersetzen das Farbwählers sieht folgendermaßen aus:

```
if (!hasColorSupport()){
$('input[type=color]').simpleColor();
}
```

Diese Lösung funktioniert, ist aber ein bisschen wackelig. Sie richtet sich nur an bestimmte Browser und funktioniert auch nur für das `color`-Steuerelement. Andere Steuerelemente haben wieder ganz andere Macken, mit denen Sie sich ebenfalls auskennen müssen. Glücklicherweise gibt es eine Alternative.

Modernizr

Die Modernizr-Bibliothek kann die Unterstützung für viele HTML5- und CSS3-Funktionen überprüfen. Sie implementiert zwar nicht die fehlende Funktionalität, bietet aber mehrere Mechanismen für die Ermittlung von Formularfeldern, die ähnlich wie unsere Lösungen, aber zuverlässiger funktionieren.

Bevor Sie Modernizr in Ihren Projekten einsetzen, sollten Sie sich etwas Zeit nehmen zu verstehen, wie er funktioniert. Ob Sie den Code nun selbst geschrieben haben oder nicht, sobald Sie ihn in Ihrem Projekt einsetzen, sind Sie auch dafür verantwortlich. Modernizr war nicht von Anfang an so weit, mit der teilweisen Unterstützung des Farbfelds in Safari klarzukommen. Wenn die nächste Version von Chrome oder Firefox herauskommt, müssen Sie unter Umständen selbst eine Lösung basteln. Wer weiß, vielleicht können Sie ja genau diese Lösung zu Modernizr beitragen!

> Ausweichlösungen für Steuerelemente wie den Datums-Picker und den Slider implementieren Sie auf dieselbe Weise.

Slider und Datums-Picker sind als Komponenten in der jQuery UI-Bibliothek enthalten. Sie binden die **jQuery UI-Bibliothek** in die Seite ein und überprüfen, ob der Browser das Steuerelement nativ unterstützt. Falls nein, verwenden Sie stattdessen die JavaScript-Version.

Nach und nach können Sie die JavaScript-Steuerelemente ausrangieren und ausschließlich auf die Steuerelemente des Browsers zurückgreifen. Aufgrund der Komplexität bei der Ermittlung dieser Typen

wird Ihnen Modernizr eine große Hilfe sein. Allerdings werden wir auch weiterhin unsere eigenen Prüftechniken schreiben, damit Sie sehen können, wie sie funktionieren.

Neben den neuen Formularfeldtypen werden in HTML5 auch einige andere Attribute für Formularfelder eingeführt, die dabei helfen können, die Benutzerfreundlichkeit zu verbessern. Sehen wir uns als Nächstes `autofocus` an.

Mit `autofocus` zum ersten Feld springen

Sie können die Dateneingabe deutlich beschleunigen, wenn Sie schon beim Laden der Seite den Cursor des Benutzers im ersten Feld des Formulars platzieren. Viele Suchmaschinen machen das mit **JavaScript**, aber in HTML5 ist diese Funktionalität jetzt Teil der Sprache.

Dafür müssen Sie lediglich das Attribut `autofocus` zu einem beliebigen Formularfeld hinzufügen, genau wie wir das schon im Abschnitt Daten mit neuen Eingabefeldern beschreiben gemacht haben.

```
<label for="name">Name</label>
<input type="text" name="name" autofocus id="name">
```

Damit das Attribut `autofocus` zuverlässig funktioniert, dürfen Sie es nur einmal pro Seite verwenden. Wenn Sie es auf mehr als ein Feld pro Seite anwenden, setzt der Browser den Cursor auf das zuletzt „autofokussierte“ Formularfeld.

Ausweichlösung

Wir überprüfen mit JavaScript, ob das `autofocus`-Attribut vorhanden ist. Wenn der Browser keine Unterstützung für `autofocus` bietet, setzen wir den Fokus mit jQuery. Das ist bestimmt die einfachste Ausweichlösung, die es gibt.

```
function hasAutofocus() {
var element = document.createElement('input');
return 'autofocus' in element;
}
$(function(){
if(!hasAutofocus()){
$('input[autofocus=true]').focus();
}
});
```

Fügen Sie einfach dieses JavaScript in Ihre Seite ein, und Sie haben jederzeit `autofocus`-Unterstützung, wenn Sie sie brauchen.

Mit `autofocus` erleichtern Sie Ihren Benutzern die Arbeit mit Ihren Formularen, unmittelbar nachdem sie geladen wurden. Aber Sie möchten ihnen auch etwas klarer sagen, welche Art von Informationen sie eingeben sollen.

Anzeigen von Messgrößen mit `meter`

Mithilfe des `meter`-Elements wird der Anteil an einer gewissen Größe grafisch dargestellt. Denken Sie zum Beispiel an die Tankanzeige in Ihrem Auto: Die Nadel zeigt den aktuellen Füllstand im Tank irgendwo zwischen 0 und 100 Prozent an. Bisher wurden solche grafischen Darstellungen in **HTML** unter anderem mit verschachtelten `div`-Elementen codiert, eine relativ unelegante Lösung, die wohl etwas am Sinn des `div`-Elements vorbeigeht. Außerdem lässt sich eine Statusanzeige auch grafisch, als Bild darstellen. Dabei können freie Webservices herangezogen werden, wie zum Beispiel die Google Chart API. Alle diese Möglichkeiten werden Sie im folgenden Beispiel sehen.

Die Verwendung des `meter`-Elements ist sehr einfach: Über das `value`-Attribut wird der gewünschte Wert eingestellt; alle anderen Attribute sind optional. Wird kein `min`– und `max`-Wert eingestellt, so verwendet der Browser 0 beziehungsweise 1 für diese Attribute. Folgendes `meter`-Element zeigt also ein halb volles Element an:

```
<meter value=0.5></meter>
```

Außer `value`, `min` und `max` gibt es noch die Attribute `low`, `high` und `optimum`, wobei der Browser diese Werte in der Darstellung mit einbeziehen kann. So zeigt zum Beispiel Google Chrome (im Sommer 2010 der einzige Browser, der das `meter`-Element darstellen konnte) den ansonsten grünen Balken in Gelb an, wenn der `optimum`-Wert überschritten wird.

Im folgenden Beispiel wird der Anteil der vergangenen Tage im aktuellen Jahr grafisch dargestellt. Die Webseite soll die Ausgabe auf vier verschiedene Arten visualisieren: als Text mit einer Angabe in Prozent, mithilfe des neuen `meter`-Elements, durch verschachtelte `div`-Elemente und als Grafik, die durch den Webservice von Googles Chart API erzeugt wird. Das Resultat sehen Sie in Abbildung 1.

Der **HTML-Code** für das Beispiel enthält die noch leeren Elemente, die mithilfe von JavaScript befüllt werden:

```
<h2>Text</h2>
<p><output id=op></output>
<span id=opText></span>
% des Jahres sind vorbei.</p>
<h2>Das neue <span class=tt>meter</span>-Element</h2>
<meter value=0 id=m></meter>
<h2>Verschachtelte <span class=tt>div</span>-Elemente</h2>
<div id=outer style="background:lightgray;width:150px;">
<div id=innerDIV>
</div>
</div>
<h2>Googles Chart API</h2>
```

```
<img id=google src="/">
<p id=googleSrc class=tt></p>
```

Für die Textausgabe verwenden wir zusätzlich das in Abschnitt Berechnungen mit `output`, vorgestellte `output`-Element. Als Erstes wird in Java-Script aber das aktuelle Datum erzeugt und das `meter`-Element initialisiert:

```
var today = new Date();
var m = document.getElementById("m");
m.min = new Date(today.getFullYear(), 0, 1);
m.max = new Date(today.getFullYear(), 11, 31);
// m.optimum = m.min-m.max/2;
m.value = today;
```

Die **Variable** `today` enthält die Anzahl an Millisekunden seit dem Beginn der UNIX-Epoche (dem 1.1.1970). Damit unser `meter`-Element eine vernünftige Skala erhält, wird der `min`-Wert auf den 1. Januar des aktuellen Jahres eingestellt, der `max`-Wert entsprechend auf den 31. Dezember. Der Wert des `meter`-Elements wird in der letzten Zeile des Listings eingestellt, und die grafische Anzeige ist komplett. Wer den hier ausgeklammerten `optimum`-Wert aktiviert (in diesem Fall die Mitte des Jahres), wird je nachdem, ob das Script in der ersten oder in der zweiten Jahreshälfte aufgerufen wird, eine entsprechende Veränderung der Anzeige erkennen. Wunderbar, wie einfach das neue Element zu verwenden ist.

Doch kommen wir nun zu den restlichen Elementen auf unserer HTML-Seite. Das mit der ID `op` gekennzeichnete `output`-Element wollen wir mit dem Prozentwert der vergangenen Tage belegen. Die Prozentrechnung wird mit `Math.round()` von ihren Kommastellen befreit, eine Genauigkeit, die für unser Beispiel ausreichend ist. Anschließend wird der `span`-Bereich (`opText`) mit diesem Wert belegt.

```
var op = document.getElementById("op");
op.value =
Math.round(100/(m.max-m.min)*(m.value-m.min));
var opText = document.getElementById("opText");
opText.innerHTML = op.value;
var innerDIV = document.getElementById("innerDIV");
innerDIV.style.width=op.value+"%";
innerDIV.style.background = "green";
```

Der Rest dieses Beispiels hat zwar nichts mehr mit neuen **HTML5-Techniken** zu tun, wird aber aus Gründen der Vollständigkeit auch noch erklärt. Die verschachtelten `div`-Elemente wollen wir ebenfalls mit dem Prozentwert befüllen. Die Idee dahinter ist simpel: Ein erster `div`-Bereich wird mit einer fixen Breite in HTML definiert (hier 150px). Ein darin verschachteltes `div`-Element wird mit der Breite von der berechneten Prozentzahl angegebenen und mit grüner Hintergrundfarbe gefüllt – ein einfacher Trick mit guter Wirkung. Abschließend wollen wir noch die Google Chart API mit einbeziehen. Beim Aufruf des

Webservice müssen die Größe der Grafik (chs, hier 200×125 Pixel), der Typ der Grafik (cht, hier gom, Google-O-Meter) und die darzustellenden Daten (chd, hier der Prozentwert `op.value`) übergeben werden:

```
var google = document.getElementById("google");
google.src = "http://chart.apis.google.com/chart?chs=200x125&cht=gom&
chd=t:"+op.value;
var gSrc = document.getElementById("googleSrc");
gSrc.innerHTML = google.src;
```

Fortschrittsanzeige mit `progress`

`progress` funktioniert ähnlich wie das eben vorgestellte `meter`-Element, mit dem Unterschied, dass es den Fortschritt eines laufenden Prozesses darstellt. Mögliche Prozesse sind ein Datei-Upload, den der Benutzer auslöst, oder der Download von externen Bibliotheken, wenn eine Applikation diese benötigt.

Für ein kurzes Beispiel wollen wir aber keine Dateien hochladen oder große Datenmengen herunterladen, es reicht, wenn wir uns selbst eine Aufgabe stellen und diese zu 100 Prozent erfüllen. Im Folgenden werden zehn Eingabeelemente vom Typ `checkbox` definiert, und sobald alle aktiviert sind, soll der Fortschrittsbalken 100 % anzeigen.

```
<h1>Bitte aktivieren Sie alle Checkboxen</h1>
<form method=get>
<input type=checkbox onchange=updateProgress()>
<input type=checkbox onchange=updateProgress()>
<!-- und 8 weitere -->
<p>Fortschritt:
<progress value=0 max=10 id=pb></progress>
</form>
```

Das `progress`-Element wird mit einem Wert von 0 und einem Maximalwert von 10 initialisiert. Sobald ein Eingabeelement aktiviert wird, ruft es die Funktion `updateProgress()` auf, die wie folgt aussieht:

```
function updateProgress() {
var pb = document.getElementById("pb");
var ip = document.getElementsByTagName("input");
var cnt = 0;
for(var i=0; i<ip.length; i++) {
if (ip .checked == true) {
cnt++;
}
}
pb.value = cnt;
}
```

Die Variable `ip` enthält eine NodeList mit allen `input`-Elementen. Jedes dieser Elemente wird in der `for`-Schleife auf seinen Zustand überprüft. Sollte dieser aktiviert sein (`checked == true`), so erhöht sich die Zählervariable `cnt` um den Wert 1. Abschließend wird der Wert des `progress`-Elements auf den Wert der Zählervariable gestellt.

Auswahllisten mit `datalist`

Eine sehr häufig gewünschte neue Funktion für **Formulare** ist ein Aufklappmenü, das um eigene Einträge erweitert werden kann. Da das altbekannte `select`-Element auf die als `option`-Elemente angegebenen Werte beschränkt ist, ersannen Webentwickler verschiedene JavaScript-Kunstgriffe, durch die Textfelder um eine erweiterbare Auswahlliste ergänzt werden können.

Die **HTML5-Spezifikation** beinhaltet eine sehr elegante Lösung für dieses Problem. Das neue `datalist`-Element wurde so definiert, dass es als Container für das schon bekannte `option`-Element dient. Jedem `input`-Element kann nun ein `datalist`-Element zugewiesen werden, das bei Bedarf die Auswahlmöglichkeiten anzeigt. Browser, die das `datalist`-Element nicht unterstützen, zeigen nur das leere Textfeld an.

Code unten zeigt die Verwendung des neuen Elements. Das `input`-Element wird vom Typ `text` definiert, und das Attribut `list` verweist auf die `id` des `datalist`-Elements (in diesem Fall homepages).

Das `autofocus`-Attribut positioniert die Schreibmarke beim Laden der Seite automatisch innerhalb des Textfeldes (vergleiche Fokussieren mit autofocus) und sorgt, zumindest beim Browser Opera, dafür, dass sich die Auswahlliste öffnet.

Für die `option`-Elemente innerhalb der `datalist` ist es ausreichend, das `value`-Attribut zu befüllen. Weitere Attribute und ein Text-Node sind zwar möglich, werden aber bei dieser Verwendung nicht benötigt. Beim Anklicken der SUBMIT-Schaltfläche wird dem Inhalt des Textfeldes die Zeichenkette http:// vorangestellt und der Browser an die so entstandene URL umgeleitet (`window.location`).

```
<form>
<p>
<label for=url>Goto</label>
http://<input type=text id=url name=homepage list=hompages autofocus>
<datalist id=hompages>
<option value=www.google.com>
<option value=html5.komplett.cc/code>
<option value=slashdot.org>
<option value=heise.de>
</datalist>
<input type=submit onclick="window.location =
'http://'+document.getElementById('url').value; return false;">
</form>
```

Wenn Sie ältere Browser ebenfalls mit einer Auswahlliste ausstatten möchten, ohne den HTML-Code zu duplizieren, können Sie auf folgenden Trick zurückgreifen. Da Browser, die das `datalist`-Element unterstützen, ein eingeschlossenes `select`-Element ignorieren, zeigen sie das neue **HTML5-Auswahlelement** an. Ältere Browser hingegen zeigen zu dem Textfeld eine Auswahlliste mit vorgegebenen Links an, die bei einer Änderung der Auswahl in das Textfeld eingefügt werden.

```
<datalist id=hompages>
<select name=homepage onchange="document.getElementById('url').value =
document.forms[0].homepage[1].value">
<option value=www.google.com>www.google.com
<option value=html-info.eu/code>html-info.eu/code
<option value=slashdot.org>slashdot.org
<option value=heise.de>heise.de
</select>
</datalist>
```

Wie in diesem Code zu sehen ist, müssen die `option`-Elemente mit einem Text-Node versehen werden, da das „alte“ `select`-Element nicht den Inhalt des `value`-Attributs anzeigt, sondern den Text. Das `onchange`-Event in dem `select`-Element setzt den aktuellen Text des Auswahlmenüs in das Textfeld ein (vergleiche Abbildung 3).

Kryptografische Schlüssel mit keygen

Das keygen-Element hat bereits eine lange Geschichte im Browser Mozilla Firefox (enthalten seit Version 1.0), trotzdem meldete Microsoft große Bedenken bei der Implementierung in HTML5 an. keygen wird zur Erzeugung von kryptografischen Schlüsseln verwendet, und so kompliziert das klingt, so kompliziert ist es leider auch.

Ganz einfach gesprochen ist die Idee hinter diesem Element folgende: Der Browser erzeugt ein Schlüsselpaar, das aus einem öffentlichen Schlüssel (public key) und einem privaten Schlüssel (private key) besteht. Der öffentliche Schlüssel wird mit den anderen Formulardaten verschickt und steht anschließend der Server-Anwendung zur Verfügung, während der private Schlüssel im Browser gespeichert bleibt. Nach diesem Schlüsselaustausch haben Server und Browser die Möglichkeit, verschlüsselt zu kommunizieren, und zwar ohne SSL-Zertifikate. Das klingt nach einer praktischen Lösung für die lästigen selbst signierten Zertifikate, die die Browser immer beanstanden müssen, ist es aber leider nicht, denn die Identität des Servers kann nur aufgrund eines Zertifikats gewährleistet werden, das von einer vertrauenswürdigen Zertifizierungsstelle (der Certificate Authority, CA) unterschrieben worden ist.

Da keygen SSL nicht ablösen kann, wofür soll das neue Element dann verwendet werden? Wie die Dokumentation von Mozilla erklärt, hilft das keygen-Element, ein Zertifikat zu erstellen, das vom Server unterschieben werden kann (signed certificate). Um diesen Schritt ganz sicher zu gestalten, ist es

normalerweise notwendig, dass der Antragsteller persönlich bei der Behörde erscheint. Da das Ausstellen von signierten Zertifikaten eher etwas für Experten ist, werden wir die Beschreibung dieses Elements und seiner Attribute eher kurz halten.

Folgendes kurze HTML-Dokument erzeugt eine keygen-Schaltfläche:

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>keygen Demo</title>
<form method=post action=submit.html>
<keygen id=kg challenge=hereismychallenge name=kg>
<input type=submit>
</form>
```

Außer den bekannten **Attributen** wie `autofocus`, `disabled`, `name` und `form` besitzt das keygen-Element zwei spezielle Attribute: `keytype` und `challenge`. Vor allem `keytype` ist interessant, da der Browser anhand dieses Eintrags entscheidet, ob er die Funktion dieses Elements unterstützt. Momentan gibt es nur einen gültigen `keytype`, und zwar `rsa`, ein kryptografisches System, das im Jahr 1977 am Massachusetts Institute of Technology (MIT) entwickelt wurde. Wird kein `keytype` angegeben (wie im vorangegangenen Beispiel), wird als Standardwert `rsa` verwendet. Die Spezifikation sieht auch vor, dass ein Browser überhaupt keinen `keytype` unterstützen muss, was wohl auf das Veto von Microsoft gegen das Element zurückgeht. Das optionale `challenge`-Attribut erhöht die Sicherheit beim Schlüsselaustausch. Für weiterführende Informationen verwenden Sie bitte die Links am Ende dieses Abschnitts.

Unterstützt der Browser die RSA-Schlüsselerzeugung, so kann er dem Benutzer eine Auswahlliste für die Länge und damit die Sicherheit des Schlüssels anbieten (vergleiche Abbildung 4).

Das Resultat nach dem Abschicken dieses Formulars zeigt Abbildung 5: Die POST-Variable kg enthält den zur Verschlüsselung notwendigen öffentlichen Schlüssel.

Berechnungen mit `output`

„Das `output`-Element enthält das Ergebnis einer Berechnung“. So lautet die sehr knappe Erklärung in der HTML5-Spezifikation, und genau das findet man auch auf den meisten Internet-Seiten, die das neue Element beschreiben. Das klingt sehr vernünftig, aber was für eine Art von Berechnung ist damit gemeint? Wieso braucht es dazu ein eigenes Element?

In der Regel handelt es sich dabei um Berechnungen, die aus Eingabefeldern auf einer Webseite zustande kommen. Ein Beispiel, das vielleicht allen geläufig ist, wäre ein elektronischer Einkaufswagen, in dem die Stückzahl für die Produkte in einem `input`-Feld eingestellt werden kann. Mithilfe des optionalen `for`-Attributs lässt sich festlegen, welche Felder in die Berechnung mit einfließen. Dabei werden ein oder mehrere `id`-Attribute anderer Felder des Dokuments referenziert.

Um das `output`-Element auszuprobieren, wollen wir so einen kleinen Einkaufswagen programmieren, in dem drei verschiedene Produkte vorhanden sind. Die Stückzahl jedes dieser Produkte kann mithilfe eines Eingabefeldes verändert werden. Gleichzeitig wird unter dem Einkaufswagen die Anzahl der Waren und die Gesamtsumme angezeigt. Abbildung 6 zeigt einen Warenkorb mit fünf Einträgen.

Der Code für das Beispiel ist schnell erklärt: Um bei jeder Änderung der Stückzahl die `output`-Elemente zu aktualisieren, verwenden wir das `oninput`-Event des Formulars:

```
<form oninput="updateSum();">
<table>
<tr>
<th>Produkt<th>Preis (€)<th>Stückzahl
<tr>
<td>Tastatur<td class=num id=i1Price>29.90<td>
<input name=i1 id=i1 type=number min=0 value=0 max=99>
<tr>
<td>Maus<td class=num id=i2Price>19.90<td>
```

Die `output`-Elemente sind im Anschluss an die Tabelle mit den Produkten definiert und verweisen über das `for`-Attribut auf die IDs der `input`-Felder:

```
<p>Sie haben
<output name=sumProd for="i1 i2 i3" id=sumProd></output>
Produkte im Einkaufswagen und müssen
<output name=sum for="i1 i2 i3" id=sum></output>
€ bezahlen.
```

Im **JavaScript-Code** läuft eine Schleife über alle `input`-Elemente. Sie zählt die Stückzahlen zusammen und errechnet den Gesamtpreis.

```
function updateSum() {
var ips = document.getElementsByTagName("input");
```

```
var sum = 0;
var prods = 0;
for (var i=0; i<ips.length; i++) {
var cnt=Number(ips .value);
if (cnt > 0) {
sum += cnt * Number(document.getElementById(
ips .name+"Price").innerHTML);
prods += cnt;
}
}
document.getElementById("sumProd").value = prods;
document.getElementById("sum").value = sum;
}
```

Den Preis des Produkts holen wir uns direkt aus der Tabelle. Dabei verwenden wir den `innerHTML`-Wert der entsprechenden Tabellenspalte und wandeln diesen mit der JavaScript-Funktion `Number()` in eine Zahl um. Gleiches gilt auch für den Wert im `input`-Feld (`ips .value`), denn ohne diese Umwandlung würde JavaScript die Zeichenketten addieren, was nicht zu dem gewünschten Ergebnis führt. Abschließend werden die errechneten Werte in die `value`-Attribute der `output`-Elemente eingesetzt.

Egal ob Sie einen Flug buchen, Ihre Bankgeschäfte online abwickeln oder einen Suchbegriff bei Google eingeben – ohne Formularfelder wären diese Dienste nicht nutzbar. Seit der Version 2.0 von HTML aus dem Jahr 1995 sind die meisten Elemente für interaktive **Formulare** unverändert, was einerseits für ein sehr vorausschauendes Design von Tim Berners Lee spricht; andererseits hat sich dadurch aber auch ein großer Nachholbedarf entwickelt. Die HTML5-Spezifikation widmet dem Thema Formulare einen großen Abschnitt und wird die Arbeit von Webentwicklern drastisch erleichtern.

Die **HTML5-Spezifikation** wertet das `input`-Element deutlich auf, indem es für das `type`-Attribut mehrere neue Werte vorsieht. Die neuen Typen wie `date`, `color` oder `range` machen es einerseits für Browserhersteller möglich, bedienerfreundliche Eingabeelemente zur Verfügung zu stellen, andererseits kann der Browser sicherstellen, dass die Eingaben vom gewünschten Typ sind. Erkennt ein Browser den type des `input`-Elements nicht, so fällt er auf `type=text` zurück und zeigt ein Textfeld an, was in jedem Fall hilfreich ist. Dieses Verhalten zeigen auch ältere Browser, wodurch dem Einsatz der neuen Typen nichts im Wege steht.

> Den größten Nutzen werden wohl die Typen für Datum und Uhrzeit mit sich bringen.

Aktuell kursieren unzählige verschiedene Versionen von mehr oder weniger gelungenen JavaScript-Kalendern im Internet. Egal ob Sie einen Flug buchen, ein Hotel reservieren oder sich bei einer Tagung anmelden, die komfortable Eingabe eines Datums ist ein Problem, das bisher immer Handarbeit verlangte. Natürlich bieten JavaScript-Bibliotheken wie jQuery fertige Kalender an, aber eigentlich sollte diese Funktion vom Browser direkt unterstützt werden.
Im Sommer 2010 gab es nur einen Desktop-Browser, der ein grafisches Bedienungselement für die

Datumseingabe mitlieferte, nämlich Opera. Abbildung 1 zeigt den aufgeklappten Kalender, der beim Anklicken eines `input`-Elements vom Typ date erscheint. Aber der Reihe nach – zuerst verschaffen wir Ihnen in Tabelle 1 einen Überblick über die neuen Typen und zeigen Ihnen dann in Abbildung 1 deren Umsetzung im Opera-Browser.

Typ	Beschreibung	Beispiel
`tel`	Text ohne Zeilenumbrüche	`+49 6473 3993443`
`search`	Text ohne Zeilenumbrüche	`suchbegriff`
`url`	eine absolute URL	http://www.example.com
`email`	eine gültige E-Mail-Adresse	user@host.com
`datetime`	Datum und Uhrzeit (immer in der UTC-Zeitzone)	`2010-08-11T11:58Z`
`date`	Datumsangabe ohne Zeitzone	`2010-08-11`
`month`	Monatsangabe ohne Zeitzone	`2010-08`
`week`	Jahr und Woche im Jahr ohne Zeitzone	`2010-W32`
`time`	Uhrzeit ohne Zeitzone	`11:58`
`datetime-local`	Datum und Uhrzeit ohne Angabe einer Zeitzone	`2010-08-11T11:58:22.5`
`number`	Zahl	`9999 oder 99.2`
`range`	Numerischer Wert eines Wertebereichs	33 oder `2.99792458E8`
`color`	Hexadezimale Darstellung von RGB-Werten im sRGB-Farbraum	#eeeeee

Die Input-Typen `tel` und `search`

`tel` und `search` unterscheiden sich nicht wesentlich von normalen Textfeldern. In beiden sind **Zeichenketten** ohne Zeilenumbrüche zulässig. Auch Telefonnummern sind nicht auf Zahlen beschränkt, da hier oft Klammern oder das Pluszeichen verwendet wird. Bei `tel` könnte der Browser Vorschläge aus dem lokalen Adressbuch anbieten, eine Situation, die vor allem auf Mobiltelefonen sehr nützlich sein kann. Der `search`-Typ wurde eingeführt, damit der Browser die Sucheingabe in einem konsistenten Layout zur jeweiligen Plattform gestalten kann. Zum Beispiel sind Benutzer des Mac OS X-Betriebssystems an abgerundete Ecken bei Suchfeldern gewohnt.

Die Input-Typen `url` und `email`

Bei `url` und `email` ist außer möglichen Vorschlägen auch eine Prüfung der Syntax möglich. Da es sowohl für E-Mail-Adressen als auch für Internet-Adressen in Form von URLs konkrete Vorschriften gibt, kann der Browser bereits während der Eingabe eine Rückmeldung über mögliche Fehler geben.

Datum und Uhrzeit mit `datetime`, `date`, `month`, `week`, `time` und `datetime-local`

Die Datums- und Uhrzeitformate bedürfen einer genaueren Betrachtung. `datetime` enthält Datumsangabe und Uhrzeit, wobei als Zeitzone immer UTC verwendet wird. Laut Spezifikation kann der Browser den Anwender auch eine andere Zeitzone auswählen lassen, der Wert des `input`-Elements muss aber in UTC umgerechnet sein.

Die Regeln für Zeitangaben beim `datetime`-Attribut des `time`-Elements, die wir in Das Element time, erklären, treffen auch hier zu – mit der einzigen Ausnahme, dass die Zeichenkette immer mit Z, dem Bezeichner für UTC, abgeschlossen werden muss.

Bei `date` und `month` fällt die Angabe der Uhrzeit und der Zeitzone weg. Für `date` wird in der Spezifikation außerdem erwähnt, dass es sich um eine gültige Tagesangabe innerhalb des ausgewählten Monats handeln muss, wobei Schaltjahre mit einzubeziehen sind. Jahr, Monat und Tag sind mit einem Minuszeichen zu trennen, wobei die Jahresangabe mindestens vier Stellen enthalten und größer als 0 sein muss. Damit sind, anders als in dem etwas ausführlicheren ISO-Standard 8601, keine Zeitpunkte vor Christus darstellbar.

Der Typ `week` wird als Woche im Jahr dargestellt, und ihm wird zwingend das Jahr vorangestellt. Als Trennzeichen zwischen Jahr und Woche dient abermals das Minuszeichen. Damit die Angabe nicht mit der von `month` verwechselt werden kann, muss der Woche das Zeichen W vorangestellt werden.

`datetime-local` funktioniert analog zu dem oben beschriebenen `datetime`, mit dem einzigen Unterschied, dass die Angabe der Zeitzone entfällt.

Opera verwendet für die Auswahl aller Datumsangaben ein Kalenderfenster; die Angaben zur Uhrzeit können manuell eingegeben oder über Pfeiltasten am Rand verändert werden (vergleiche Abbildung 1).

Die Input-Typen `number` und `range`

Die Typen `number` und `range` verlangen, dass die Eingabe in einen numerischen Wert umgewandelt werden kann, wobei auch die Notation für Gleitkommazahlen (zum Beispiel 2.99792458E8) gültig ist. Für

den range-Typ enthält die **Spezifikation** die Anmerkung, dass der genaue Wert nicht entscheidend ist. Es handelt sich um eine ungefähre Angabe, die vom Anwender gut mit einem Schieberegler eingestellt werden kann. Sowohl Webkit-basierte Browser wie Safari und Google Chrome als auch Opera verwenden zur Darstellung dieses Typs einen Schieberegler (vergleiche Abbildung 1 und Abbildung 2).

Der Input-Typ color

Leider gänzlich ohne Implementierung ist der neue Typ color. Ähnlich wie bei dem Typ date gibt es auch hier schon etliche Versuche in JavaScript; im Vergleich zum Datum ist die Farbauswahl aber wesentlich seltener notwendig. Zukünftige Implementierungen im Browser werden aber wahrscheinlich einen Farbwähler vorsehen, wie man ihn von Bildbearbeitungsprogrammen her kennt. Der Wert für das input-Element muss die 8-Bit-Rot-, -Grün- und -Blau-Werte in hexadezimaler Notation mit führendem #-Zeichen enthalten. Die Farbe Blau wird in dieser Notation zum Beispiel als #0000ff geschrieben.

Die neuen Input-Typen im Einsatz

Genug der Theorie. In einem ersten Beispiel werden alle neuen Elemente untereinander dargestellt. Da das allein noch keine besondere Herausforderung darstellt, soll jedes Element noch auf seine Funktion geprüft werden. Der Trick dabei ist, dass der Browser den Typ eines unbekannten Elements auf text setzt, und diese Eigenschaft können wir in **JavaScript** abfragen:

```
<script>
window.onload = function() {
inputs = document.getElementsByTagName("input");
for (var i=0; i<inputs.length; i++) {
if (inputs .type == "text") {
inputs .value = "nicht erkannt";
}
}
}
</script>
```

Sobald die Webseite vollständig geladen ist, läuft eine Schleife über alle input-Elemente, in der deren type-Attribute analysiert werden. Sofern das type-Attribut dem Standard-Typ text entspricht, wird dessen Wert auf nicht erkannt gesetzt. Der **HTML-Code** für die neuen input-Elemente sieht folgendermaßen aus:

```
<fieldset>
<legend>Neue Input-Typen</legend>
<p><label for=tel>tel</label>
<input type=tel id=tel name=tel>
<p><label for=search>search</label>
```

```
<input type=search id=search name=search>
<p><label for=url>url</label>
<input type=url id=url name=url>
<p><label for=email>email</label>
```

Wie das Ergebnis dieses Tests auf einem Android-Mobiltelefon ausfällt, zeigt Abbildung 3. Der Webkit-basierte Browser des Systems (links) gibt zwar vor, die Typen `tel`, `search`, `url` und `email` zu kennen, leistet aber bei der Eingabe der Telefonnummer über die Tastatur (Mitte) keine besondere Hilfe. Opera Mini in Version 5.1 (rechts) gibt direkt zu, dass es keinen der neuen Typen unterstützt.

Das ist eine enttäuschende Bilanz für die sonst so modernen mobilen Browser. Auf dem iPhone sieht die Sache etwas besser aus: So passt das Smartphone zumindest die Software-Tastatur so an, dass bei der Eingabe von Telefonnummern ein Zahlenfeld erscheint, und bei dem `input`-Typ `email` wird das @-Zeichen auf der Tastatur hinzugefügt.

Noch etwas besser funktioniert das mit BlackBerry, dem Betriebssystem des kanadischen Smartphone-Herstellers Research in Motion (RIM), dessen Geräte vor allem in Nordamerika weit verbreitet sind. Wie Abbildung 4 zeigt, werden sowohl `tel` als auch `number` und Datumstypen unterstützt, wobei vor allem Letztere grafisch sehr ansprechend aufbereitet sind. Unter der Haube arbeitet Webkit, wobei die Software um diese Funktionen erweitert wurde.

Neben neuen Elementen und vielen neuen Typen für das `input`-Element bietet HTML5 auch einige neue **Attribute** für Formular-Elemente.

Fokussieren mit `autofocus`
Google überraschte viele Anwender vor Jahren mit einem einfachen Trick, der die Suchseite deutlich komfortabler machte: Beim Laden der Seite positionierte sich der Cursor automatisch im Suchfeld. Dadurch konnte man unmittelbar den Suchbegriff eingeben, ohne erst mit der Maus das Eingabefeld aktivieren zu müssen. Was bisher mit einem kurzen JavaScript-Schnipsel erledigt wurde, kann in **HTML5** mit dem `autofocus`-Attribut erreicht werden.

```
<input type=search name=query autofocus>
```

Wie alle Attribute vom Typ `boolean` kann das Attribut auch als `autofocus="autofocus"` geschrieben werden (vergleiche dazu Das Element `time`). Laut Spezifikation darf nur ein Element einer Webseite das `autofocus`-Attribut enthalten.

Für ältere Browser stellt `autofocus` kein Hindernis dar, weil sie das unbekannte Attribut einfach ignorieren. Den Gewinn an Benutzerfreundlichkeit haben freilich nur neue Browser.
Platzhalter-Text mit `placeholder`
Eine weitere Verbesserung der Benutzbarkeit von HTML-Formularen erreicht man durch das neue

placeholder-Attribut.

```
<p><label for=email>Ihre E-Mail-Adresse:</label>
<input type=email name=email id=email placeholder="user@host.com">
<p><label for=birthday>Ihr Geburtstag</label>
<input type=date name=birthday id=birthday placeholder="1978-11-24">
```

Der Wert von placeholder kann dem Benutzer einen kurzen Hinweis darauf geben, wie das Feld auszufüllen ist, und sollte nicht als Ersatz für das label-Element verwendet werden. Das bietet sich vor allem bei solchen Feldern an, wo ein bestimmtes Eingabeformat erwartet wird. Der Browser zeigt den Hinweistext innerhalb eines nicht aktiven Eingabefelds an. Sobald das Feld aktiviert wird und den Fokus erhält, wird der Text nicht mehr angezeigt (vergleiche Abbildung 1).

Verpflichtende Felder mit required

required ist ein boolean-Attribut, das mit einem Wort schon alles über seine Funktion verrät: Ein **Formular-Element**, dem dieses Attribut zugewiesen ist, muss ausgefüllt werden. Wenn ein required-Feld beim Abschicken des Formulars leer ist, so erfüllt es nicht die erforderten Vorgaben, und der Browser muss darauf entsprechend reagieren.

Noch mehr neue Attribute für das input-Element

Das input-Element wurde nicht nur durch neue Typen aufgewertet, sondern auch durch neue Attribute, die die Handhabung von Formularen erleichtern.

Attribut	Typ	Beschreibung
list	String	Verweis auf die ID eines datalist-Elements mit Vorschlägen
min	Numerisch/Datum	Minimalwert für numerische Felder und Datumsfelder
max	Numerisch/Datum	Maximalwert für numerische Felder und Datumsfelder
step	Numerisch	Schrittweite für numerische Felder und Datumsfelder
multiple	Boolean	Mehrfachauswahl möglich
autocomplete	Enumerated (on/off/default)	Vorausfüllen von Formularfeldern mit gespeicherten Daten

Attribut	Typ	Beschreibung
pattern	String	Regulärer Ausdruck zum Überprüfen des Werts

Dem `list`-Attribut verweist auf das `datalist`-Element, das mögliche Einträge als Vorschläge bereitstellt.

`min`, `max` und `step` eignen sich nicht nur für numerische Felder; auch bei Datums- und Zeitangaben können diese Attribute verwendet werden.

```
<p><label for=minMax>Zahlen zwischen 0 und 1:</label>
<input type=number name=minMax id=minMax min=0 max=1 step=0.1>
<p><label for=minMaxDate>Datum mit Wochenschritten:</label>
<input type=date name=minMaxDate id=minMaxDate min=2010-08-01 max=2010-11-11
step=7>
<p><label for=minMaxTime>Zeit mit Stundenschritten:</label>
<input type=time name=minMaxTime id=minMaxTime min=14:30 max=19:30 step=3600>
```

In Browsern, die den `input`-Typ `number` unterstützen, wird das erste `input`-Element (`id=minMax`) jeweils um den Wert von 0.1 erhöht. Das funktioniert durch den Klick auf die Pfeiltasten am Ende des Textfeldes oder durch das Drücken der Pfeiltasten auf der Tastatur. Das Element mit der ID `minMaxDate` springt jeweils um sieben Tage weiter. Opera zeigt dabei in dem Kalender nur jene Tage als aktiv an, die dem Wochenzyklus entsprechen. Google Chrome bietet zum Einstellen dieses Elements die gleiche Navigation wie beim `input`-Typ `number`: zwei Pfeiltasten, die das Datum um sieben Tage vor oder zurück stellen. Bei dem dritten `input`-Element in diesem Beispiel wird die Schrittweite mit 3600 angegeben, was dazu führt, dass die Zeitangabe jeweils um eine Stunde vor oder zurückgestellt wird. Obwohl in der **Spezifikation** erwähnt ist, dass die Eingabeelemente für Zeitangaben normalerweise mit einer Genauigkeit von Minuten arbeiten, interpretieren sowohl Opera als auch Google Chrome diese Angabe als Sekunden.

Die Mehrfachauswahl ist uns allen vom Kopieren von Dateien her bekannt; im Browser gibt es diese Möglichkeit jetzt auch. Wollte man bisher mehrere Dateien auf einer Webseite laden, so musste man für jede Datei ein `input`-Feld vorsehen. Das `multiple`-Attribut ermöglicht es, im Dateidialog mehrere Dateien zu markieren. Für das `select`-Element war die `multiple`-Option schon immer vorgesehen, neu ist die Verwendung für Eingabefelder vom Typ `email`. Im Sommer 2010 konnte aber keiner der gängigen Desktop-Browser diese Funktion für `email`-Typen umsetzen.

> Moderne Browser verfügen über eine Funktion, durch die Formular-Eingaben gespeichert werden,

damit sie bei einem neuerlichen Zugriff auf das Formular als Hilfe beim Ausfüllen dienen. Was meist sehr praktisch ist, kann bei sicherheitskritischen Eingabefeldern auch unerwünscht sein (die Spezifikation erwähnt hier als Beispiel die Abschusscodes von Nuklearwaffen). Das `autocomplete`-Attribut wurde

eingeführt, damit Webentwickler dieses Verhalten steuern können. Wird ein Element mit dem Attribut `autocomplete="off"` versehen, so bedeutet das, dass die einzugebende Information vertraulich ist und nicht im Browser gespeichert werden soll. Enthält ein Formular-Element keinen Hinweis, ob `autocomplete` ein- oder ausgeschaltet sein soll, so ist der Standardwert, dass Vorschläge angezeigt werden sollen. Das `autocomplete`-Attribut kann auch auf das ganze Formular angewendet werden, indem man es dem `form`-Element zuweist.

Um eine sehr flexible Überprüfung der Eingabe zu ermöglichen, wurde das `pattern`-Attribut eingeführt. Durch die Angabe eines regulären Ausdrucks wird das Formularfeld auf eine Übereinstimmung geprüft. Reguläre Ausdrücke stellen eine sehr mächtige, aber leider auch nicht ganz einfache Methode zur Behandlung von Strings dar. Stellen Sie sich vor, Sie suchen eine Zeichenkette, die mit einem Großbuchstaben beginnt, auf den eine beliebige Anzahl von Kleinbuchstaben oder Zahlen folgt, und die auf .txt endet. Mit einem `regexp` (eine Kurzform für Regular Expression, d. h. regulärer Ausdruck) ist das kein Problem:

```
[A-Z]{1}[a-z,0-9]+\.txt
```

Beim Einsatz von regulären Ausdrücken im `pattern`-Attribut ist zu beachten, dass das Suchmuster immer auf den gesamten Inhalt des Feldes zutreffen muss. Außerdem wird in der Spezifikation vorgeschlagen, dass das `title`-Attribut dazu verwendet wird, dem Anwender einen Hinweis zu geben, wie das Format der Eingabe ist. Opera und Google Chrome zeigen diese Informationen dann in Form eines Tool-Tipps an, sobald sich der Mauszeiger über dem Feld befindet. Nach so viel Theorie folgt nun endlich ein kurzes Beispiel:

```
<p><label for=pattern>Ihr Nickname:</label>
<input type=text pattern="[a-z]{3,32}" placeholder="johnsmith" name=pattern
id=pattern
title="Bitte nur Kleinbuchstaben, min. 3, max. 32!">
```

Die Vorgabe für das `pattern` lautet, dass die **Zeichenkette** nur Zeichen zwischen a und z (also Kleinbuchstaben) enthalten darf (`[a-z]`) und davon mindestens 3 und höchstens 32. Umlaute und andere Sonderzeichen sind damit nicht erlaubt, was für einen Benutzernamen, wie in oben stehendem Beispiel, auch ganz gut sein kann. Wollte man zumindest die deutschen Umlaute und das scharfe ß mit einbeziehen, müsste man die Gruppe um diese erweitern: `[a-zäöüß]`.

Mit **Platzhaltertext** können Sie Benutzern Anweisungen geben, womit sie das entsprechende Feld ausfüllen sollen. Abbildung 1 zeigt ein Registrierungsformular mit Platzhaltertexten. Dieses Formular bauen wir jetzt nach.

Ein einfaches Registrierungsformular
Benutzer der Support-Website müssen zunächst ein Konto erstellen. Eines der größten Probleme bei der Registrierung ist, dass Benutzer immer wieder versuchen, unsichere Kennwörter zu wählen. Wir werden unsere Benutzer mit Platzhaltertext an die Hand nehmen und ihnen mitteilen, welche Anforderungen wir

an Kennwörter haben. Der Konsistenz halber werden wir auch die anderen Felder mit Platzhaltertexten versehen.

Zum Hinzufügen von Platzhaltertexten müssen Sie lediglich für jedes Eingabefeld das `placeholder`-Attribut angeben:

```
<input id="email" type="email" name="email" placeholder="user@example.com">
```

Das Markup für unser **Formular** mit Platzhaltertexten für jedes Feld sieht in etwa so aus:

```
<form id="create_account" action="/signup" method="post">
<fieldset id="signup">
<legend>Create New Account</legend>
<ol>
<li>
<label for="first_name">First Name</label>
<input id="first_name" type="text" autofocus="true" name="first_name"
placeholder="'John'">
</li>
<li>
<label for="last_name">Last Name</label>
<input id="last_name" type="text" name="last_name" placeholder="'Smith'">
</li>
<li>
<label for="email">Email</label>
<input id="email" type="email" name="email" placeholder="user@example.com">
</li>
<li>
<label for="password">Password</label>
<input id="password" type="password" name="password" value="" autocomplete="off"
placeholder="8-10 characters" />
</li>
<li>
<label for="password_confirmation">Password Confirmation</label>
<input id="password_confirmation" type="password" name="password_confirmation"
value="" autocomplete="off" placeholder="Type your password again" />
</li>
<li><input type="submit" value="Sign Up"></li>
</ol>
```

```
</fieldset>
</form>
```

Autovervollständigung verhindern
Vielleicht ist Ihnen aufgefallen, dass wir für die Kennwortfelder des Formulars das Attribut `autocomplete` angegeben haben. In **HTML5** wird das Attribut `autocomplete` eingeführt, um Webbrowsern mitzuteilen, dass sie nicht versuchen sollen, die Daten für dieses Feld automatisch auszufüllen. Einige Browser merken sich Daten, die die Benutzer zuvor eingetippt haben. Aber in manchen Fällen möchten wir den Browser anweisen, das lieber nicht zu tun.

Da wir wieder einmal eine geordnete Liste als Container für unsere Formularfelder verwenden, schreiben wir ein bisschen CSS, damit das Formular besser aussieht.

```
fieldset {
width: 216px;
}
fieldset ol {
list-style:none;
padding:0;
margin:2px;
}
fieldset ol li {
margin:0 0 9px 0;
padding:0;
}
/* Jedes Eingabefeld erhält eine eigene Zeile */
fieldset input {
display:block;
}
```

Safari, Opera und Chrome zeigen nun in den Formularfeldern einen hilfreichen Text für die Benutzer an. Als Nächstes bringen wir Firefox und den Internet Explorer dazu, ebenfalls mitzuspielen.
Ausweichlösung
Mit JavaScript können Sie Platzhaltertext ohne größeren Aufwand in **Formularfelder** schreiben. Sie testen den Wert jedes Formularfelds. Und wenn er leer ist, belegen Sie den Wert mit dem Platzhalter. Erhält das Formularfeld den Fokus, löschen Sie den Wert wieder. Und wenn das Feld den Fokus verliert, überprüfen Sie den Wert erneut. Falls er sich geändert hat, lassen Sie ihn unverändert. Und falls der Wert leer ist, ersetzen Sie ihn wieder durch den Platzhaltertext.

Die Unterstützung für `placeholder` ermitteln Sie genauso wie die Unterstützung für `autofocus`.

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

Dann schreiben Sie das JavaScript, um die Änderungen zu verarbeiten. Um das zum Laufen zu bekommen, verwenden wir eine Lösung, die auf der Arbeit von Andrew January und anderen basiert. Wir füllen die Werte aller Formularfelder mit dem Text, der im Attribut `placeholder` abgelegt ist. Wählen die Benutzer ein Feld aus, entfernen wir den Text aus dem Feld. Das Ganze verpacken wir in ein jQuery-Plugin, damit wir dieses Verhalten einfach auf unser Formular anwenden können.

```
(function($){
$.fn.placeholder = function() {
function valueIsPlaceholder(input) {
return ($(input).val() == $(input).attr("placeholder"));
}
return this.each(function() {
$(this).find(":input").each(function() {
if($(this).attr("type") == "password") {
var new_field = $("<input type='text'>");
new_field.attr("rel", $(this).attr("id"));
new_field.attr("value", $(this).attr("placeholder"));
$(this).parent().append(new_field);
new_field.hide();
function showPasswordPlaceHolder(input) {
if( $(input).val() == "" || valueIsPlaceholder(input) ) {
$(input).hide();
$('input[rel=' + $(input).attr("id") + ']').show();
};
};
new_field.focus(function() {
$(this).hide();
$('input#' + $(this).attr("rel")).show().focus();
});
$(this).blur(function() {
showPasswordPlaceHolder(this, false);
});
showPasswordPlaceHolder(this);
}else{
// Wert wird durch den Platzhaltertext ersetzt.
// Durch den optionalen Parameter "reload" wird
// verhindert, dass FF und IE die Werte der Felder
// zwischenspeichern.
function showPlaceholder(input, reload) {
```

```
if( $(input).val() == "" ||
( reload && valueIsPlaceholder(input) ) ) {
$(input).val($(input).attr("placeholder"));
}
};
$(this).focus(function() {
if($(this).val() == $(this).attr("placeholder")) {
$(this).val("");
};
});
$(this).blur(function() {
showPlaceholder($(this), false)
});
showPlaceholder(this, true);
};
});
// Verhindern, dass Formulare die Standardwerte übermitteln
$(this).submit(function() {
$(this).find(":input").each(function() {
if($(this).val() == $(this).attr("placeholder")) {
$(this).val("");
}
});
});
});
};
})(jQuery);
```

In diesem Plugin stehen einige interessante Dinge, die Sie wissen sollten. In Zeile 46 laden wir den Platzhaltertext erneut in die Felder, wenn diese keinen Wert haben oder die Seite neu geladen wurde. Firefox und andere Browser speichern die **Formularwerte**. Wir belegen das `value`-Attribut mit dem Platzhalter und möchten auf keinen Fall, dass dieser versehentlich zum Wert des Benutzers wird. Wenn wir die Seite laden, übergeben wir `true` an diese Methode, wie Sie in Zeile 61 sehen.

jQuery-Plugins
Sie können **jQuery** um eigene Plugins erweitern. Dazu fügen Sie eigene Methoden zur jQuery-Funktion hinzu, und schon steht Ihr Plugin nahtlos allen Entwicklern zur Verfügung, die Ihre Bibliothek einbinden. Hierzu ein triviales Beispiel, das eine JavaScript-Alertbox anzeigt:

```
jQuery.fn.debug = function() {
return this.each(function() {
alert(this.html());
```

```
});
```

Wenn Sie nun für jeden Absatz einer Seite ein Pop-up-Feld anzeigen möchten, geht das so:

```
$("p").debug();
```

jQuery-Plugins sind so konzipiert, dass sie eine Sammlung von jQuery-Objekten durchlaufen. Gleichzeitig liefern sie diese Objektsammlung auch zurück, damit mehrere Methoden miteinander verkettet werden können. Da unser debug-Plugin ebenfalls die jQuery-Sammlung zurückliefert, können wir in derselben Zeile gleich noch mit der `css`-Methode von jQuery die Textfarbe der Absätze ändern:

```
$("p").debug().css("color", "red");
```

Wir werden im Laufe mehrere Male **jQuery-Plugins** für Ausweichlösungen einsetzen, um unseren Code klar zu strukturieren. Weitere Informationen dazu können Sie auf der Dokumentations-Website von jQuery erfahren.

Kennwortfelder verhalten sich ein bisschen anders als andere Formularfelder. Also müssen wir auch anders damit umgehen. Werfen Sie einen Blick auf Zeile 12. Wir überprüfen, ob ein Kennwortfeld vorhanden ist, und müssen seinen Typ in ein reguläres Textfeld ändern, damit der Wert nicht mit Sternchen maskiert wird. Manche Browser melden Fehler, wenn Sie versuchen, Kennwortfelder zu konvertieren. Also müssen wir die Kennwortfelder durch Textfelder ersetzen. Wir werden die Felder austauschen, während die Benutzer damit interagieren.

Durch diesen Trick werden die Werte der Formulare verändert. Im Zweifel möchten Sie aber verhindern, dass die Platzhalter zum Server gelangen. Nachdem wir den Platzhalter-Code nur einsetzen können, wenn JavaScript aktiviert ist, können wir auch mit JavaScript die Übermittlung des Formulars überprüfen. In Zeile 66 fangen wir daher die Übermittlung des Formulars ab und löschen alle Werte aus allen Eingabefeldern, die den Platzhalterwerten entsprechen.

Nachdem wir jetzt alles als Plugin zusammengefasst haben, können wir das Plugin auf der Seite aufrufen, indem wir es mit dem Formular verknüpfen:

```
$(function(){
function hasPlaceholderSupport() {
var i = document.createElement('input');
return 'placeholder' in i;
}
if(!hasPlaceholderSupport()){
$("#create_account").placeholder();
// Ende der Ausweichlösung für Platzhalter
$('input[autofocus=true]').focus();
```

```
};
});
```

Damit haben wir eine ziemlich gute Lösung, die Ihnen **Platzhaltertexte** als brauchbare Option für Ihre Webanwendungen zur Verfügung stellt, egal welchen Browser Ihre Benutzer verwenden.

Im Web ist standardmäßig alles eckig. Formularfelder, Tabellen und sogar die Abschnitte von Webseiten sehen klobig und kantig aus. Deshalb haben viele Designer im Laufe der Jahre auf die verschiedensten Techniken zurückgegriffen, um Elementen runde Ecken zu verpassen und so die Oberflächen ein bisschen weicher zu gestalten.

CSS3 bietet eine einfache Möglichkeit, Ecken abzurunden, die von Firefox und Safari bereits seit langer Zeit unterstützt wird. Leider ist der Internet Explorer noch nicht mit an Bord. Aber das können wir leicht ändern.

Ein Anmeldeformular auflockern

Die Modelle und Entwürfe für Ihr aktuelles Projekt zeigen Formularfelder mit abgerundeten Ecken. Zunächst runden wir die Ecken nur mit CSS3 ab. Das Ergebnis soll so aussehen wie in Abbildung 1.

Für das Anmeldeformular schreiben wir ganz einfaches **HTML**.

```
<form action="/login" method="post">
<fieldset id="login">
<legend>Log in</legend>
<ol>
<li>
<label for="email">Email</label>
<input id="email" type="email" name="email">
</li>
<li>
<label for="password">Password</label>
<input id="password" type="password" name="password" value=""
autocomplete="off"/>
</li>
<li>
<input type="submit" value="Log in">
</li>
</ol>
</fieldset>
</form>
```

Wir stylen das Formular noch ein bisschen, damit es besser aussieht.

```
fieldset {
width:216px;
border:none;
background-color: #ddd;
}
fieldset legend {
background-color: #ddd;
padding: 0 64px 0 2px;
}
fieldset>ol {
list-style: none;
padding: 0;
margin: 2px;
}
fieldset>ol>li {
margin: 0 0 9px 0;
padding: 0;
}
/* Jedes Eingabefeld erhält eine eigene Zeile */
fieldset input {
display: block;
}
input {
width: 200px;
background-color: #fff;
border: 1px solid #bbb;
}
input[type="submit"] {
width: 202px;
padding: 0;
background-color: #bbb;
}
```

Diese einfachen Stilregeln entfernen die Aufzählungszeichen von der Liste und gewährleisten, dass alle Eingabefelder dieselbe Größe haben. Nachdem wir damit fertig sind, können wir unsere Elemente abrunden.

Browserspezifische Selektoren

Da die **CSS3-Spezifikation** noch nicht endgültig ist, haben die Browserhersteller selbst einige Funktionen hinzugefügt und den Namen jeweils das Präfix ihrer eigenen Implementierungen vorangestellt. Durch diese Präfixe können Browserhersteller frühzeitig Funktionen einführen, bevor diese Teil einer endgültigen Spezifikation werden. Und da sie nicht der eigentlichen Spezifikation folgen, können die Browserhersteller die tatsächliche Spezifikation und ihre eigene Version parallel implementieren. In den meisten Fällen

entspricht die Version mit dem Hersteller-Präfix der CSS-Spezifikation, aber gelegentlich auch nicht. Für Sie bedeutet das leider, dass Sie den Rahmenradius für jeden Browsertyp gesondert deklarieren müssen.

Firefox verwendet diesen Selektor:

```
-moz-border-radius: 5px;
```

WebKit-basierte Browser wie Safari und Chrome nutzen den folgenden Selektor:

```
-webkit-border-radius: 5px;
```

Um alle Eingabefelder unseres Formulars abzurunden, benötigen wir eine **CSS-Regel** wie die folgende:

```
input, fieldset, legend {
border-radius: 5px;
-moz-border-radius: 5px;
-webkit-border-radius: 5px;
}
```

Fügen Sie das in Ihre Datei style.css ein, und die Ecken sind rund.

Ausweichlösung
Nun funktioniert alles in Firefox, Safari und Google Chrome. Aber wie Sie wissen, funktioniert es nicht im Internet Explorer. Und natürlich wissen Sie, dass es auch im Internet Explorer funktionieren muss und Sie daher etwas implementieren müssen, das dem so nahe wie möglich kommt.

Webentwickler runden Ecken nun schon seit geraumer Zeit mit Hintergrundbildern und anderen Techniken ab, aber wir möchten es möglichst einfach halten. Wir können mit JavaScript den Radius der Ecken ermitteln und eine ganze Reihe von Abrundungstechniken anwenden. In diesem Beispiel verwenden wir das `jQuery-Plugin` „Corner“ sowie eine Abwandlung des Corner-Plugins, die auch Textfelder abrundet.
Unterstützung für abgerundete Ecken ermitteln
Wir binden die **jQuery-Bibliothek** und das Plugin ein und überprüfen, ob der Browser das Attribut unterstützt. Falls nicht, aktivieren wir das Plugin. In diesem Fall müssen wir das Vorhandensein der CSS-Eigenschaft `border-radius`, aber auch browserspezifischer Präfixe wie `webkit` und `moz` ermitteln.

Erstellen Sie corner.js, und fügen Sie die folgende Funktion ein:

```
function hasBorderRadius(){
var element = document.documentElement;
```

```
var style = element.style;
if (style){
return typeof style.borderRadius == "string" ||
typeof style.MozBorderRadius == "string" ||
typeof style.WebkitBorderRadius == "string" ||
typeof style.KhtmlBorderRadius == "string";
}
return null;
}
```

Nun können wir überprüfen, ob es bei unserem Browser an Unterstützung für abgerundete Ecken hapert. Schreiben wir also den Code für das eigentliche Abrunden der Ecken. Erfreulicherweise gibt es ein Plugin, das wir als Ausgangspunkt verwenden können.

jQuery Corners

jQuery Corners ist ein kleines Plugin, das Ecken dadurch abrundet, dass die jeweiligen Elemente in zusätzliche `div`-Tags verpackt und so gestylt werden, dass das Zielelement abgerundet aussieht. Es funktioniert aber nicht mit Formularfeldern. Mit ein bisschen Fantasie und ein bisschen jQuery können wir dieses Plugin aber dazu bringen, dass es auch das tut.

Entscheiden, ob sich der Aufwand lohnt

In unserem Beispiel möchte der Kunde, dass die abgerundeten Ecken wirklich in allen Browsern funktionieren. Allerdings sollten Sie solche Funktionen nach Möglichkeit immer optional halten. Natürlich argumentieren manche Leute, dass es einen realen Nutzen bringt, wenn das Aussehen des Formulars aufgelockert wird. Trotzdem sollten Sie sich zunächst einen Eindruck davon verschaffen, wie viele Menschen überhaupt Browser verwenden, die die CSS-basierte Abrundung nicht unterstützen. Wenn Ihre Besucher hauptsächlich Safari und Firefox benutzen, lohnt sich unter Umständen der zeitliche Aufwand zum Schreiben und Pflegen eines Skripts zur Überprüfung und für die Ausweichlösung nicht.

Als Erstes verweisen Sie von Ihrer HTML-Seite auf **jQuery Corners**. Bei dieser Gelegenheit binden Sie auch gleich noch die Datei corner.js mit ein.

```
<script src="jquery.corner.js" charset="utf-8" type='text/javascript'></script>
<script src="corner.js" charset="utf-8" type='text/javascript'></script>
```

Nun müssen wir lediglich den Code schreiben, der die Abrundung aufruft.

Unser formCorners-Plugin

Wir schreiben ein jQuery-Plugin, damit wir die Funktion zum Abrunden einfach auf alle Formularfelder anwenden können. Die Erstellung von jQuery-Plugins haben wir bereits in Platzhaltertext für Hinweise im Abschnitt Ausweichlösung besprochen und müssen sie daher nicht noch einmal erklären.Gehen Sie einfach den Code für das Plugin Schritt für Schritt durch (es baut zum Teil auf einer Lösung von Tony Amoyal auf).

Fügen Sie Folgendes in die Datei corners.js ein:

```
(function($){
$.fn.formCorner = function(){
return this.each(function() {
var input = $(this);
var input_background = input.css("background-color");
var input_border = input.css("border-color");
input.css("border", "none");
var wrap_width = parseInt(input.css("width")) + 4;
var wrapper = input.wrap("<div></div>").parent();
var border = wrapper.wrap("<div></div>").parent();
wrapper.css("background-color", input_background)
.css("padding", "1px");
border.css("background-color",input_border)
.css("width", wrap_width + "px")
.css('padding', '1px');
wrapper.corner("round 5px");
border.corner("round 5px");
});
};
})(jQuery);
```

Wir nehmen ein **jQuery-Objekt**, das entweder ein Element oder eine Sammlung von Elementen sein kann, und verpacken es in zwei `div`-Tags, die wir anschließend abrunden. Zuerst geben wir dem inneren `div` dieselbe Farbe wie dem Hintergrund des ursprünglichen Eingabefelds und deaktivieren den Rahmen des eigentlichen Formularfelds. Anschließend verschachteln wir das Feld in ein anderes Feld, das die Rahmenfarbe des ursprünglichen Eingabefelds als Hintergrundfarbe sowie ein bisschen Padding erhält. Und dieses Padding ergibt die Rahmenlinie. Stellen Sie sich einfach zwei Blätter Buntpapier vor: ein grünes, das 10 cm breit ist, und ein rotes, das 8 cm breit ist. Wenn Sie das kleinere auf das größere legen, sehen Sie um das rote Stück Papier einen grünen Rahmen herum. So funktioniert das.

Die Abrundung aufrufen

Mit dem Plugin und unserer Prüfbibliothek können wir nun die Abrundung starten.

Fügen Sie Folgendes in die Datei corners.js ein:

```
$(function(){
if(!hasBorderRadius()){
$("input").formCorner();
$("fieldset").corner("round 5px");
$("legend").corner("round top 5px cc:#fff");
```

```
}
});
```

Wir runden die drei Formularfelder und die Feldgruppe ab. In Zeile 5 runden wir schließlich den oberen Teil der Legende ab und legen fest, dass für den Ausschnitt der Ecken Weiß verwendet werden soll. Das Plugin verwendet die Hintergrundfarbe des übergeordneten Elements als Farbe für den Ausschnitt, aber das ist hier nicht richtig.

Wenn der Browser die Eigenschaft `border-radius` unterstützt, führt er unser Plugin aus. Falls nicht, verwendet er das CSS, das wir vorhin eingefügt haben.

Ein kleiner IE-Trick

Der IE behandelt Legenden ein bisschen anders. Wir können einen Trick einsetzen, durch den der IE die Legende der Feldgruppe ein bisschen nach oben schiebt, sodass sie genauso aussieht wie in Firefox und Chrome.

```
<link rel="stylesheet" href="style.css" type="text/css" media="screen">
<!--[if IE]>
<style>fieldset legend{margin-top: -10px }</style>
<![endif]-->
```

Jetzt sieht unser Formular auf allen wichtigen Browsern relativ ähnlich aus. Die Internet Explorer-Version sehen Sie in Abbildung 2.

Abgerundete Ecken lassen Ihre Oberflächen weicher erscheinen und sind extrem einfach zu verwenden. Sie sollten diese Technik aber wie jede andere Gestaltungsmöglichkeit mit Bedacht und nicht übermäßig verwenden.

Abgerundete Ecken genießen natürlich eine Menge Ansehen, aber das ist nur der Anfang. Wir können noch viel mehr mit **CSS3** machen: Elemente mit Schlagschatten vom restlichen Inhalt abheben, Hintergründe mit Verläufen definierter aussehen lassen und Elemente mit Transformationen rotieren. Kombinieren wir einige dieser Techniken, um ein Banner zusammenzubasteln.

Der Grafikdesigner hat ein PSD geschickt, das wie die Abbildung 1 aussieht. Wir können das Namensschild, den Schatten und sogar die Transparenz vollständig in CSS erstellen. Vom Grafikdesigner brauchen wir lediglich das Hintergrundbild mit den Menschen.

Die grundlegende Struktur

Beginnen wir mit der grundlegenden **HTML-Struktur** der Seite:

```
<div id="conference">
```

```
<section id="badge">
<h3>Hi, My Name Is</h3>
<h2>Barney</h2>
</section>
<section id="info">
</section>
</div>
```

Diesen Code stylen wir folgendermaßen:

```
#conference {
background-color: #000;
width: 960px;
float: left;
background-image: url('images/awesomeconf.jpg');
background-position: center;
height: 240px;
}
#badge {
text-align: center;
width: 200px;
border: 2px solid blue;
}
#info {
margin: 20px;
padding: 20px;
width: 660px;
height: 160px;
}
#badge, #info {
float: left;
background-color: #fff;
}
#badge h2 {
margin: 0;
color: red;
font-size: 40px;
}
#badge h3 {
margin: 0;
background-color: blue;
color: #fff;
}
```

Sobald wir dieses Stylesheet auf unsere Seite anwenden, werden unser Namensschild und der Inhaltsbereich wie in Abbildung 2 nebeneinander angezeigt. Jetzt kann es mit dem Styling des Namensschilds losgehen.

Einen Verlauf hinzufügen

Dem Ansteckschild können wir mehr Struktur verpassen, indem wir den weißen Hintergrund durch einen leichten Verlauf von Weiß nach Hellgrau ersetzen. Verläufe funktionieren in Firefox, Safari und Chrome. Aber die Implementierung in Firefox ist anders. Chrome und Safari verwenden die WebKit-Syntax, den ursprünglichen Vorschlag. Für Firefox benötigen wir eine **Syntax**, die dem `W3C-Vorschlag` ähnelt. Wieder einmal verwenden wir die Browser-Präfixe, die Sie im Scharfe Ecken abrunden Abschnitt Browserspezifische Selektoren kennengelernt haben.

```
#badge {
background-image: -moz-linear-gradient(
top, #fff, #efefef
);
background-image: -webkit-gradient(
linear,left top, left bottom,
color-stop(0, #fff),
color-stop(1, #efefef)
);
background-image: linear-gradient(
top, #fff, #efefef
);
}
```

Firefox verwendet die Methode `-moz-linear-gradient`, mit der wir den Anfangspunkt des Verlaufs, gefolgt von der Startfarbe und der Zielfarbe, angeben.

Für WebKit-basierte Browser können wir Farbstopps angeben. In unserem Beispiel soll der Verlauf von Weiß bis Grau gehen. Für zusätzliche Farben müssen wir lediglich einen weiteren Farbstopp definieren.

Schatten für das Ansteckschild

Wir können es leicht so aussehen lassen, als ob das Ansteckschild oberhalb des Banners liegt, indem wir einen Schlagschatten hinzufügen. Üblicherweise würden wir den Schatten in Photoshop zum Bild hinzu- oder als Hintergrundbild einfügen. In CSS3 können wir aber einfach mit der Eigenschaft `box-shadow` einen Schatten für Elemente definieren.

Für den Schatten des Namensschilds brauchen wir nur die folgende Regel in unser **Stylesheet** einzufügen:

```
#badge {
-moz-box-shadow: 5px 5px 5px #333;
-webkit-box-shadow: 5px 5px 5px #333;
-o-box-shadow: 5px 5px 5px #333;
box-shadow: 5px 5px 5px #333;
}
```

Die Eigenschaft `box-shadow` erwartet vier Parameter. Der erste ist der horizontale Versatz: Eine positive Zahl bedeutet, dass der Schatten zur rechten Seite des Objekts fällt. Bei einer negativen Zahl fällt der Schatten zur linken Seite. Der zweite Parameter gibt den vertikalen Versatz an: Positive Zahlen lassen den Schatten unterhalb der Box erscheinen, negative Zahlen oberhalb.

Der dritte Parameter gibt den Weichzeichnungsradius an. Beim Wert 0 wird der Schatten sehr scharf dargestellt, bei höheren Werten wird der Schatten entsprechend weichgezeichnet. Der letzte Parameter bestimmt die Farbe des Schattens.

Mit diesen Werten müssen Sie experimentieren, bis Sie ein Gefühl dafür entwickeln, wie sie funktionieren, und die passenden Werte finden. Für die Arbeit mit Schatten sollten Sie erforschen, wie Schatten in der physischen Welt funktionieren.

Textschatten
Zusätzlich zu Elementen können Sie auch Text ganz einfach Schatten werfen lassen. Das funktioniert genauso wie `box-shadow`.

```
h1 {
text-shadow: 2px 2px 2px #bbbbbb;
}
```

Sie geben den X- und Y-Versatz an sowie den Grad der Weichzeichnung und die Farbe des Schattens. IE 6, 7 und 8 bieten mit dem Filter „Shadow“ dieselbe Möglichkeit.

```
filter: Shadow(Color=#bbbbbb,
Direction=135,
Strength=3);
```

Das ist derselbe Ansatz wie beim Schlagschatten für Elemente. Schatten ergänzen Texte um einen hübschen Effekt, aber zu starke Schatten schaden der Lesbarkeit.

Schnappen Sie sich eine Taschenlampe, und beleuchten Sie Objekte damit, oder gehen Sie vor die Tür, und beobachten Sie, wie die Sonne Gegenstände Schatten werfen lässt. Diese Perspektive ist sehr wichtig.

Inkonsistente Schatten lassen Ihre Oberflächen schnell verwirrend erscheinen, insbesondere dann, wenn Sie für mehrere Elemente die Schatten falsch zeichnen. Die einfachste Variante besteht darin, für jeden Schatten dieselben Einstellungen zu verwenden.

Das Namensschild drehen

Mit **CSS3-Transformationen** können Sie Elemente genauso wie mit den Vektorgrafikprogrammen Flash, Illustrator oder Inkscape drehen, skalieren und verzerren. Elemente heben sich dadurch deutlicher ab, und die Webseiten sehen nicht ganz so „kastenförmig“ aus. Wir drehen das Namensschild ein bisschen, damit es aus der geraden Kante des Banners ausbricht.

```
#badge {
-moz-transform: rotate(-7.5deg);
-o-transform: rotate(-7.5deg);
-webkit-transform: rotate(-7.5deg);
-ms-transform: rotate(-7.5deg);
transform: rotate(-7.5deg);
}
```

Drehungen sind mit CSS3 ziemlich einfach. Wir müssen lediglich den Drehwinkel angeben, und los geht es. Alle Elemente innerhalb des rotierten Elements werden mitgedreht.

Die Drehung ist genauso simpel wie die abgerundeten Ecken, aber auch damit sollten Sie es nicht übertreiben. Das Ziel beim Interface-Design besteht darin, Oberflächen benutzerfreundlicher zu machen. Wenn Sie Elemente mit viel Inhalt drehen, sollten Sie sicherstellen, dass Ihre Benutzer den Inhalt auch lesen können, ohne den Kopf zu weit zur Seite legen zu müssen!

Transparente Hintergründe

Grafikdesigner verwenden schon seit einer ganzen Weile semitransparente Ebenen hinter Text. Dafür erstellen sie entweder ein komplettes Bild im Photoshop oder legen mit CSS ein transparentes PNG über ein Element. In CSS3 können wir Hintergrundfarben mit einer neuen Syntax definieren, die auch Transparenz unterstützt.

Als Neuling in der Webentwicklung lernen Sie zunächst, Farben mit hexadezimalen Farbcodes zu definieren: Sie geben die Menge an Rot, Grün und Blau in Zahlenpaaren. 00 bedeutet „ganz aus“, und FF bedeutet „ganz an“. FF0000 entspricht der Farbe Rot und bedeutet „Rot ganz an, Blau ganz aus und Grün ganz aus“.

In CSS3 werden die Funktionen `rgb` und `rgba` eingeführt. Die `rgb`-Funktion funktioniert wie ihr hexadezimales Gegenstück, jedoch verwenden Sie Werte zwischen 0 und 255 für die jeweiligen Farben. Rot wird beispielsweise als `rgb(255,0,0)` definiert.

Die `rgba`-Funktion arbeitet auf die gleiche Weise wie die `rgb`-Funktion, erwartet aber einen vierten Parameter, der die Deckkraft zwischen 0 und 1 angibt. Für den Wert 0 wird überhaupt keine Farbe angezeigt, die Farbe ist vollkommen transparent. Mit der folgenden Stilregel machen wir den weißen Kasten semitransparent:

```
#info {
background-color: rgba(255,255,255,0.95);
}
```

Wenn Sie mit solchen Transparenzwerten arbeiten, können sich die Kontrasteinstellungen der Benutzer auf das Endergebnis auswirken. Sie sollten unbedingt auf verschiedenen Bildschirmen mit den Werten experimentieren, um konsistente Ergebnisse zu gewährleisten.

Während wir am Infobereich unseres Banners arbeiten, können wir auch gleich die Ecken abrunden:

```
#info {
moz-border-radius: 12px;
webkit-border-radius: 12px;
o-border-radius: 12px;
border-radius: 12px;
}
```

Damit sieht unser Banner in Safari, Firefox und Chrome ziemlich gut aus. Als Nächstes implementieren wir ein Stylesheet für den Internet Explorer.

Ausweichlösung
Die in diesem Abschnitt verwendeten Techniken funktionieren wunderbar im IE 9, sind aber auch mit Internet Explorer 6, 7 und 8 möglich! Dafür müssen wir die DirectX-Filter von Microsoft verwenden. Das bedeutet, dass wir auf einen bedingten Kommentar zurückgreifen und ein IE-spezifisches Stylesheet laden müssen. Außerdem müssen wir mit JavaScript das `section`-Element erstellen, damit wir es mit CSS stylen können, da diese Versionen des IE das Element nicht nativ erkennen.

```
<!--[if lte IE 8]>
<script>
document.createElement("section");
</script>
<link rel="stylesheet" href="ie.css" type="text/css" media="screen">
<![endif]-->
</head>
<body>
<div id="conference">
<section id="badge">
<h3>Hi, My Name Is</h3>
<h2>Barney</h2>
</section>
<section id="info">
```

```
</section>
</div>
</body>
</html>
```

Die DirectX-Filter funktionieren in IE 6, 7 und 8. Aber im IE 8 werden die Filter anders aufgerufen. Daher müssen Sie jeden Filter zweimal deklarieren. Sehen wir uns zunächst an, wie wir Elemente drehen können.

Drehen

Wir können Elemente mit diesen Filtern drehen. Aber es reicht nicht aus, einfach nur den Winkel vorzugeben. Für den gewünschten Effekt müssen wir den Matrixfilter verwenden und den Cosinus bzw. Sinus des gewünschten Winkels angeben. Genauer gesagt müssen wir den Cosinus, den negativen Sinus, den Sinus und nochmals den Cosinus angeben:

```
filter: progid:DXImageTransform.Microsoft.Matrix(
sizingMethod='auto expand',
M11=0.9914448613738104,
M12=0.13052619222005157,
M21=-0.13052619222005157,
M22=0.9914448613738104
);
-ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
sizingMethod='auto expand',
M11=0.9914448613738104,
M12=0.13052619222005157,
M21=-0.13052619222005157,
M22=0.9914448613738104
)";
```

Kompliziert? Ja, vor allem, wenn Sie sich das vorherige Beispiel genauer ansehen: Erinnern Sie sich, dass unser ursprünglicher Winkel minus 7,5 Grad beträgt? Entsprechend brauchen wir für den negativen Sinus einen positiven Wert, und unser Sinus erhält einen negativen Wert.

Mathe ist kompliziert. Machen wir lieber einen Verlauf.

Farbverläufe

Der Verlaufsfilter des IE funktioniert wie im Standard vorgesehen, allerdings haben Sie eine ganze Menge mehr Tipparbeit. Eigentlich geben Sie nur die Start- und die Endfarbe an – und schon wird der Verlauf angezeigt.

```
filter: progid:DXImageTransform.Microsoft.gradient(
startColorStr=#FFFFFF, endColorStr=#EFEFEF
);
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
startColorStr=#FFFFFF, endColorStr=#EFEFEF
```

```
)";
```

Anders als bei den anderen Browsern wenden Sie im IE den Verlauf direkt auf das **Element** an statt auf die Eigenschaft `background-image`.

Wir wenden den Filter gleich noch einmal für die transparente Hintergrundfarbe unseres Infobereichs an.

Transparenz
Sie können dem Verlaufsfilter erweiterte Hexadezimalwerte für die Start- und Endfarbe übergeben, wobei die ersten beiden Stellen den Transparenzgrad angeben:

```
background: none;
filter:
progid:DXImageTransform.Microsoft.gradient(
startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
);
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
)";
```

Die achtstelligen Hexcodes funktionieren ähnlich wie die `rgba`-Funktion, jedoch steht der Transparenzwert an erster Stelle, nicht am Ende. In Wahrheit haben wir es also mit Alpha, Rot, Grün und Blau zu tun.

Wir müssen die Hintergrundeinstellungen des Elements entfernen, damit das im IE 7 funktioniert. Falls Sie das Stylesheet mittippen, haben Sie sicherlich festgestellt, dass es noch nicht funktioniert. Aber das können wir ändern.
Alles zusammengenommen
Eines der schwierigeren Probleme mit den IE-Filtern besteht darin, dass wir sie nicht stückchenweise definieren können. Um mehrere Filter auf ein einzelnes Element anzuwenden, müssen wir die Filter als mit Kommata getrennte Liste definieren. So sieht das tatsächliche IE-Stylesheet aus:

```
#info{
background: none;
filter:
progid:DXImageTransform.Microsoft.gradient(
startColorStr=#BBFFFFFF, endColorStr=#BBFFFFFF
);
-ms-filter: "progid:DXImageTransform.Microsoft.gradient(
startColorStr='#BBFFFFFF', EndColorStr='#BBFFFFFF'
)";
}
```

```
#badge{
filter:
progid:DXImageTransform.Microsoft.Matrix(
sizingMethod='auto expand',
M11=0.9914448613738104,
M12=0.13052619222005157,
M21=-0.13052619222005157,
M22=0.9914448613738104
),
progid:DXImageTransform.Microsoft.gradient(
startColorStr=#FFFFFF, endColorStr=#EFEFEF
),
progid:DXImageTransform.Microsoft.Shadow(
color=#333333, Direction=135, Strength=3
);
-ms-filter: "progid:DXImageTransform.Microsoft.Matrix(
sizingMethod='auto expand',
M11=0.9914448613738104,
M12=0.13052619222005157,
M21=-0.13052619222005157,
M22=0.9914448613738104
),
progid:DXImageTransform.Microsoft.gradient(
startColorStr=#FFFFFF, endColorStr=#EFEFEF
),
progid:DXImageTransform.Microsoft.Shadow(
color=#333333, Direction=135, Strength=3
)";
}
```

Das ist eine Menge **Code** für das gewünschte Ergebnis, beweist aber, dass es möglich ist, diese Funktionen zu verwenden. Wenn Sie einen Blick auf Abbildung 3 werfen, sehen Sie, dass wir unserem Ziel ziemlich nahe kommen.

Jetzt müssen wir nur noch die Ecken des info-Abschnitts abrunden. Wie das geht, können Sie im Scharfe Ecken abrunden nachlesen.

Auch wenn diese Filter unhandlich und ein bisschen seltsam sind, sollten Sie sie in eigenen Projekten weiter erforschen, weil Sie dadurch IE-Benutzern ein ähnliches Ergebnis bieten können.

Denken Sie daran: Die Effekte, die wir in diesem Artikel untersucht haben, betreffen nur die Präsentation. Beim ursprünglichen Stylesheet haben wir sorgfältig Hintergrundfarben ausgesucht, mit denen der Text gut lesbar ist. Auch Browser, die kein CSS3 verstehen, können die Seite also auf lesbare Art und Weise

darstellen.

Geolocation mit HTML5

Die Verwendung des kurzen Beispiels macht ausschließlich auf mobilen Geräten Sinn. Zwar kann der Anwendung zu Demonstrationszwecken auch künstlich „Beine gemacht“ werden, das Erfolgserlebnis stellt sich wahrscheinlich aber erst bei einer präzisen Positionsbestimmung mittels GPS und einem Browser ein, der in Bewegung ist. Versuchsanordnung für das folgende Beispiel war ein Android-Mobiltelefon, das die **HTML-Seite** während einer Fahrt auf der Autobahn anzeigte.

Wie in Abbildung 1 zu sehen ist, werden die jeweils letzten fünf ermittelten Positionen auf der Straßenkarte von **Google Maps** markiert. Sobald der Beobachter den dargestellten Bereich der Karte verlässt, wird die Karte um den nächsten Punkt zentriert. Der Aufruf der **Geolocation-API** wird wieder in `window.onload` ausgeführt und sieht folgendermaßen aus:

```
var watchID = navigator.geolocation.watchPosition(
moveMe, posError, {enableHighAccuracy: true}
);
```

Die eigentliche Arbeit findet in der Funktion `moveMe()` statt:

```
function moveMe(position) {
latlng = new google.maps.LatLng(
position.coords.latitude,
position.coords.longitude);
bounds = map.getBounds();
map.setZoom(16);
if (!bounds.contains(latlng)) {
map.setCenter(latlng);
}
if (marker.length >= maxMarkers) {
m = marker.shift();
if (m) {
m.setMap();
}
}
marker.push(new google.maps.Marker({
position: latlng, map: map,
title: position.coords.accuracy+"m lat: "
+position.coords.latitude+" lon: "+
```

```
position.coords.longitude
}));
}
```

Die Variable `latlng` wird als LatLng-Objekt aus der **Google Maps API** erzeugt, wobei diesem Objekt die aktuellen Koordinaten übergeben werden. Sollte die aktuelle Position außerhalb des dargestellten Bereichs sein (`!bounds.contains(latlng)`), wird die Karte über dem aktuellen Punkt neu zentriert. Wie das Array `marker` wurde auch die Variable `maxMarkers` am Anfang des Scripts global definiert und mit dem Wert 5 belegt. Enthält das Array `marker` mehr als fünf Elemente, so wird das erste Element mit der `shift`-Funktion aus dem Array entfernt und anschließend durch den Aufruf von `setMap()` ohne weitere Parameter von der Karte gelöscht. Abschließend wird dem Array ein neues Objekt vom Typ Marker an der aktuellen Position hinzugefügt.

Im folgenden Beispiel wird die aktuelle Position auf einer Karte von **OpenStreetMap** dargestellt und mit einem Marker gekennzeichnet. Außerdem werden unterschiedliche Layer und eine Navigationsleiste von OpenStreetMap angezeigt. Abbildung 1 zeigt den Mapnik-Layer der OpenStreetMap mit dem Positions-Icon in der Mitte des Browsers.

Wie schon im Online-Kartendienste, werden hier die Daten des OpenStreetMap-Projekts mithilfe der OpenLayers-Bibliothek dargestellt. Nachdem die beiden dazu notwendigen JavaScript-Dateien geladen sind, wird in diesem Beispiel die Karte initialisiert und werden die gewünschten Steuerelemente hinzugefügt.

```
// Karte initialisieren und Navigation hinzufügen
var map = new OpenLayers.Map ("map");
map.addControl(new OpenLayers.Control.Navigation());
map.addControl(new OpenLayers.Control.PanZoomBar());
```

Neben dem Navigationselement mit den vier Pfeilen wird die Zoom-Leiste an die Karten-Variable (`map`) angehängt. Anschließend wird das Auswahl-Element für die unterschiedlichen Layer erzeugt (`Control.LayerSwitcher`) und werden mehrere Layer zu der Karte hinzugefügt. Der Funktionsaufruf von `map.addLayers()` nimmt dabei ein Array von neu erzeugten Karten-Objekten entgegen.

```
// Layerauswahl mit vier Kartentypen
map.addControl(new OpenLayers.Control.LayerSwitcher());
map.addLayers([
new OpenLayers.Layer.OSM.Mapnik("Mapnik"),
new OpenLayers.Layer.OSM.Osmarender("Osmarender"),
new OpenLayers.Layer.OSM.Maplint("Maplint"),
new OpenLayers.Layer.OSM.CycleMap("CycleMap")
]);
```

Abschließend erhält die Karte noch einen Layer für den Marker:

```
var markers = new OpenLayers.Layer.Markers("Markers");
map.addLayer(markers);
```

Das Success-Callback nach einer erfolgreichen **Positionsbestimmung** sieht so aus:

```
function(pos) {
var ll = new OpenLayers.LonLat(
pos.coords.longitude,
pos.coords.latitude).transform(
new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject()
);
map.setCenter (ll,zoom);
markers.addMarker(
new OpenLayers.Marker(
ll,new OpenLayers.Icon(
'http://www.openstreetmap.org/openlayers/img/marker.png')
)
);
},
```

Wie Ihnen schon aus Online-Kartendienste, bekannt ist, müssen die Koordinaten aus dem geografischen Koordinatensystem (Lat/Lon) in das Spherical Mercator-System transformiert werden (mehr dazu finden Sie im Positionsausgabe im Browser). Abschließend wird der Marker `ll` an die ermittelte Position gesetzt, wobei das entsprechende Icon direkt von den OpenStreetMap-Servern geladen wird.Die **Geolocation-Spezifikation** sieht einen weiteren Aufruf vor, der sich vor allem bei bewegten Objekten anbietet: `navigator.geolocation.watchPosition()`. Das folgende Beispiel zeigt anhand der Google Maps API, wie sich eine Positionsänderung grafisch darstellen lässt.

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: „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.

> Geolocation-API wird 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 Artikel sehen werden.

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 Artikel 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 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 Artikels 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.

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 1.

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 2 aufgeführten Eigenschaften.

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 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 Station 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 Position Options-Objekt. Es gibt drei Eigenschaften, die Sie auf einem `PositionOptions`-Objekt setzen können (siehe Tabelle 3). Alle Eigenschaften sind optional; Sie können alle setzen, einige oder keine.

Eigenschaft	Typ	Standard	Anmerkungen
`enableHighAccuracy`	`bolean`	`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 `enable HighAccuracy: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“). Aber verzweifeln Sie nicht! Gears 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 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 gears_init.js. Es initialisiert Gears, falls es installiert ist. 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.

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.');
}
```

Die Idee zum abschließenden Beispiel entstand bei einem Auslandsaufenthalt mit einem neuen Smartphone: Das Programm ist ein digitales (Reise-)Tagebuch, das jeden Eintrag automatisch mit geografischen Koordinaten versieht und alle Einträge auf einer Karte anzeigen kann. Die hohen Daten-Roaming-Gebühren in Europa machten bald die Einbeziehung einer weiteren Technologie aus dem **HTML5-Umfeld** erforderlich, um die Kosten zu senken: Web Storage. Mithilfe der Web Storage API werden die Einträge lokal, in einem persistenten Speicher gehalten, wodurch die Anwendung auch ohne bestehende Datenverbindung funktioniert.

Bedienung
Die Anwendung ist sehr einfach aufgebaut (siehe Abbildung 1): Im linken oberen Bereich befindet sich das Textfeld zur Eingabe der Notiz. Durch das in **HTML5** neu eingeführte `placeholder`-Attribut zeigt der Browser eine Aufforderung zum Eingeben einer neuen Nachricht. Sind bereits Einträge vorhanden, erscheint im rechten Bereich ein Kartenausschnitt von Google Maps. Unterhalb folgt die Liste der Einträge, wobei außer dem Nachrichtentext noch die Position, der Zeitpunkt der Eingabe und eine Entfernung zum aktuellen Standort angegeben wird. Außerdem besteht die Möglichkeit, Nachrichten zu löschen oder den Standort vergrößert auf Google Maps darzustellen. Wie in Abbildung 1 zu sehen ist, wird bei vergrößerter Darstellung die Position mit einem Google-typischen Marker gekennzeichnet. Der Kreis um den Punkt bezeichnet die Genauigkeit der Positionsmessung.

Da man beim Entwickeln der Anwendung seine Position nicht ständig ändert,

hat sich das in Technischer Hintergrund der Positionsbestimmung, vorgestellte Firefox-Add-On Geolocator

als sehr nützlich erwiesen. Durch die Möglichkeit, mehrere Positionen in dem Add-On zu speichern, kann man die Anwendung auch von zu Hause aus testen. Idealerweise aber kommt die Anwendung auf einem Smartphone mit GPS-Unterstützung zum Einsatz. Sowohl Android-basierte Telefone als auch das iPhone erfüllen die notwendigen Voraussetzungen.

Um die Anwendung gleich ausprobieren zu können, gibt es die Möglichkeit, einen Testdatensatz einzuspielen. Es handelt sich dabei um zum Teil frei erfundene Einträge und um solche, die der Autor während der Entwicklung selbst aufgenommen hat.

Wichtige Code-Fragmente

Der **HTML-Code** für die Anwendung stellt einige `div`-Container-Elemente bereit, in denen später die Nachrichten (`id='items'`) und die Karte (`id='map'`) dargestellt werden. Wie schon eingangs erwähnt wurde, enthält das `textarea`-Element das neue `placeholder`-Attribut, das Anwendungen deutlich benutzerfreundlicher machen kann. Den drei `button`-Schaltflächen wird der entsprechende `onclick`-Event-Listener direkt zugewiesen.

```
<body>
<h1>Geonotes</h1>
<div class='text_input'>
<textarea style='float:left; margin-right:30px;' placeholder='Your message here
...' cols="50" rows="15" id="note">
</textarea>
<div class='map' id='map'></div>
<div style='clear:both;' id='status'></div>
<button style='float:left; color:green;' id='save'
onclick='saveItem()'>Save</button>
<button onclick='drawAllItems()'>Draw all items on map</button>
<button onclick='importDemoData()'>Import Demo Data</button>
</div>
<div class='items' id='items'></div>
```

Wesentlich interessanter als die wenigen Zeilen HTML-Code ist der dazugehörige **JavaScript-Code**. Zuerst werden eine Hilfsfunktion und drei globale Variablen definiert:

```
function $(id) {
return document.getElementById(id);
}
var map;
var my_pos;
var diaryItem = { id: 0, pos: 0, ts: 0, msg: ''
}
```

Die Funktion $ ist Ihnen bereits aus Positionsausgabe im Browser, bekannt. Sie erspart Ihnen auch hier Tipparbeit und trägt zur Lesbarkeit des Codes bei.

Die Variable `map` dient als Referenz auf den HTML-Bereich, in dem die **Google-Maps**-Darstellung Platz finden wird. `my_pos` wird zur Berechnung der Entfernung benötigt und enthält die aktuelle Position, von der aus das Script aufgerufen wird. `diaryItem` stellt die Struktur dar, nach der die einzelnen Einträge aufgebaut sind. Jeder Tagebucheintrag enthält eine ID (`id`), Informationen zur Position (`pos`), einen Zeitstempel (`ts`) sowie die eingegebene Nachricht aus dem Textfeld (`msg`).

Sobald die Seite vollständig geladen ist, beginnen die Bestimmung der aktuellen Position und die Anzeige bestehender Einträge:

```
window.onload = function() {
if (navigator.geolocation) {
navigator.geolocation.getCurrentPosition(
function(pos) {
my_pos = pos;
showItems();
},
posError,
{ enableHighAccuracy: true, maximumAge: 60000 }
);
}
showItems();
if (localStorage.length > 0) {
drawAllItems();
}
}
```

Für den bereits bekannten Aufruf von `getCurrentPosition()` wird die Option `enableHighAccuracy` aktiviert. Die maximale Zeit für die Wiederverwendung einer bereits ermittelten Position ist eine Minute. Bei einer erfolgreichen Positionsbestimmung wird die vorher definierte globale Variable `my_pos` mit den Werten aus der soeben bestimmten Position belegt und anschließend die Funktion `showItems()` aufgerufen. Ein Fehler bei der Positionsbestimmung führt zum Aufruf von `posError()`, einer Funktion, die die entsprechende Fehlermeldung in einem Dialogfenster ausgibt. Ist die Anzahl der Elemente im `localStorage` größer als 0, wird außerdem noch die Funktion `drawAllItems()` ausgeführt, die vorhandene Einträge auf Google Maps darstellt.

Die `showItems`-Funktion setzt eine Zeichenkette aus allen Einträgen zusammen und belegt anschließend das HTML-Element mit der ID `items` damit.

```
function showItems() {
var s = '<h2>'+localStorage.length+' Items for '
```

```
+location.hostname+'</h2>';
s+= '<ul>';
var i_array = getAllItems();
for (k in i_array) {
var item = i_array[k];
var iDate = new Date(item.ts);
s+= '<li>';
s+= '<p class="msg">'+item.msg+'</p>';
s+= '<div class="footer">';
s+= '<p class="i_date">'+iDate.toLocaleString();
+'</p>'; ...
$('items').innerHTML = s+'</ul>';
```

Die **Variable** `i_array` wird mit dem Ergebnis der Funktion `getAllItems()` befüllt, die den `localStorage` ausliest, anschließend die Inhalte als Objekte in einem Array zurückgibt und außerdem eine Sortierung der Objekte nach dem Datum vornimmt.

```
function getAllItems() {
var i_array = [];
for (var i=0;i<localStorage.length;i++) {
var item = JSON.parse(
localStorage.getItem(localStorage.key(i))
);
i_array.push(item);
}
i_array.sort(function(a, b) {
return b.ts - a.ts
});
return i_array;
}
```

Der Aufruf `localStorage.getItem()` holt ein Element aus dem persistenten Speicher, das anschließend mit der Funktion `JSON.parse` in ein **JavaScript-Objekt** umgewandelt wird. Voraussetzung dafür ist, dass das Objekt beim Abspeichern mit `JSON.stringify` in eine Zeichenkette umgewandelt wurde (siehe weiter unten). Die Objekte werden mit `i_array.push()` an das Ende des Arrays `i_array` gehängt und im nächsten Schritt nach dem Datum sortiert. Um der JavaScript-Funktion `sort` mitzuteilen, nach welchen Kriterien sortiert werden soll, wird sie mit einer anonymen Funktion erweitert. Um eine zeitliche Sortierung der Objekte zu ermöglichen, wird die Variable `ts` aus den Objekten ausgewertet. Sie enthält die Zahl der Millisekunden seit dem 1.1.1970, ein Wert, der durch die JavaScript-Funktion `new Date().getTime()` erzeugt wird. Liefert die anonyme Funktion einen negativen Wert zurück, so wird a hinter b angereiht, und bei einem positiven Wert kommt a vor b.

Jetzt bleibt noch die Frage zu klären, wie neue Einträge erzeugt und gespeichert werden. Die Funktion

`saveItem()` übernimmt diesen Teil und beginnt mit der Initialisierung einer lokalen Variable `d`, der die Struktur `diaryItem` zugewiesen wird.

```
function saveItem() {
var d = diaryItem;
d.msg = $('note').value;
if (d.msg == '') {
alert("Empty message");
return;
}
d.ts = new Date().getTime();
d.id = "geonotes_"+d.ts;
if (navigator.geolocation) {
$('status').innerHTML = '<span style="color:red">'
+'getting current position / item unsaved</span>';
navigator.geolocation.getCurrentPosition(
function(pos) {
d.pos = pos.coords;
localStorage.setItem(d.id, JSON.stringify(d));
$('status').innerHTML =
'<span style="color:green">item saved. Position'
+' is: '+pos.coords.latitude
+','+pos.coords.longitude+'</span>';
showItems();
},
posError,
{ enableHighAccuracy: true, maximumAge: 60000 }
);
} else {
// alert("Browser does not support Geolocation");
localStorage.setItem(d.id, JSON.stringify(d));
$('status').innerHTML =
"Browser does not support Geolocation/item saved.";
}
showItems();
}
```

Sollte das Textfeld leer sein (`d.msg = ''`), wird ein entsprechendes Dialogfenster angezeigt und die Funktion mit `return` abgebrochen. Andernfalls wird der Zeitstempel auf den aktuellen Millisekundenwert gesetzt und die ID des Eintrags aus der Zeichenkette `geonotes_` und dem Zeitstempel zusammengesetzt. Sollten von einem Server mehrere Anwendungen auf den `localStorage` zugreifen, so kann man über die vorangestellte Zeichenkette eine Unterscheidung der Daten vornehmen. Nach erfolgreicher Positionsbestimmung wird die Variable `pos` innerhalb des `diaryItem`-Objekts mit Koordinaten und den dazugehörigen Meta-Informationen befüllt und anschließend über `JSON.stringify()` als JSON-

Zeichenkette im `localStorage` gespeichert.

Sollte der Browser keine Unterstützung für die **Geolocation-API** haben, speichert die Anwendung den Text dennoch und weist darauf hin, dass die entsprechende Unterstützung fehlt. Der abschließende Aufruf von `showItems()` sorgt dafür, dass die Liste der Nachrichten aktualisiert wird.

Die Geolocation-API wurde zwar aus dem Kern der **HTML5-Spezifikation** entfernt und befindet sich nach der `W3C-Nomenklatur` erst in einem frühen Stadium, sie ist aber vor allem bei mobilen Browsern schon weitgehend implementiert. Ein Grund für die rasche Umsetzung liegt sicher in der Kürze und der Abstraktion der Schnittstelle: Mit nur drei JavaScript-Aufrufen wird der gesamte Funktionsumfang abgedeckt. Die Spezifikation schreibt nicht vor, auf welche Art der Browser die Position zu bestimmen hat, sondern nur das Format, in dem das Ergebnis zurückgegeben wird.

Nach einer kurzen Einführung in das Wesen geografischer Daten stellen wir die neuen Funktionen anhand von mehreren kurzen Beispielen vor. Wenn Sie die Beispiele mit einem Smartphone ausprobieren, sollte sich schnell der Aha-Effekt einstellen.

Ein Wort zu geografischen Daten

Vielleicht ist Ihnen schon einmal eine Koordinatenangabe in der Form N47 16 06.6 E11 23 35.9 begegnet. Die Position wird in Grad-Minuten-Sekunden angegeben. In diesem Fall befindet sich der gesuchte Ort bei 47 Grad, 16 Minuten und 6,6 Sekunden nördlicher Breite und 11 Grad, 23 Minuten und 35,9 Sekunden östlicher Länge. Man bezeichnet solche Angaben als geografische Koordinaten. Leider haben diese Werte den großen Nachteil, dass man nur schwer mit ihnen rechnen kann, was nicht nur daran liegt, dass Menschen gewohnt sind, im Dezimalsystem zu denken. Da die Koordinaten eine Position auf dem Rotationsellipsoid Erde beschreiben, muss bei der Berechnung von Entfernungen die Krümmung der Oberfläche miteinbezogen werden.

> Um diesen Zustand zu vereinfachen, werden in der Praxis meist projizierte Koordinaten verwendet.

Dabei wird das Rotationsellipsoid in Streifen zerschnitten, in denen die Entfernung zwischen Punkten linear gemessen werden kann. Viele Länder verwenden ein eigenes, auf die lokalen Bedürfnisse abgestimmtes Koordinatensystem. In Österreich werden zum Beispiel Daten meist im Bundesmeldenetz referenziert. Alle gängigen Koordinatensysteme sind mit einer numerischen Kennung versehen, ihrem **EPSG-Code** (verwaltet von der European Petroleum Survey Group).

Natürlich kann die `Geolocation-API` nicht alle Koordinatensysteme berücksichtigen. Die X- und Y-Koordinaten werden daher nicht projiziert, sondern in geografischen Koordinaten in Dezimalgrad angegeben. Als geodätisches Referenzsystem schreibt der Standard das weit verbreitete **World Geodetic System** 1984 (WGS84) vor. Es beschreibt im Wesentlichen das darunterliegende Referenzellipsoid. Der Z-Wert wird in Metern über diesem Ellipsoid angegeben. Damit lässt sich jeder Punkt auf der Erde und im erdnahen Raum mit ausreichender Genauigkeit beschreiben.

Online-Kartendienste
Zur Darstellung von geografischen Daten im Browser gibt es mehrere Möglichkeiten: SVG eignet sich durch das flexible Koordinaten-System sehr gut, und mit `canvas` könnten die Daten als Rasterbild gezeichnet werden. Am bequemsten ist es aber, eine bereits bestehende JavaScript-Bibliothek zu Hilfe zu nehmen. Von den im Internet frei verfügbaren Bibliotheken beschreiben wir im Folgenden Google Maps und OpenStreetMap genauer. Da der Kartendienst von Microsoft, Bing Maps, nur nach vorheriger Registrierung verwendet werden kann, gehen wir hier nicht weiter darauf ein.

Die zwei hier vorgestellten Bibliotheken verwenden zur Anzeige eine Mischung aus Raster- und Vektordaten. Um schnelle Ladezeiten zu ermöglichen, werden die Rasterbilder in Kacheln zerlegt und für alle Zoomstufen im Voraus berechnet. Dadurch entsteht der schrittweise Bildaufbau. Vektorinformationen werden je nach Browser in SVG oder für den Internet Explorer im Microsoftspezifischen Format VML angezeigt.
Google Maps
Google Maps ist zweifellos der am weitesten verbreitete Kartendienst im Internet. Viele Firmen verwenden den kostenlosen Service, um den eigenen Firmenstandort kartografisch darzustellen. Doch **Google Maps** hat weit mehr zu bieten, als nur Positionsmarker auf eine Karte zu setzen. Wie die Webseite Case Studies verrät, setzen über 150.000 Webseiten Google Maps ein, unter ihnen auch große Unternehmen wie die New York Times.

> Die aktuelle Version der Bibliothek, V3, unterscheidet sich stark von vorangegangenen Versionen:

Zur Verwendung wird kein API-Schlüssel mehr benötigt (also keine Registrierung bei Google), und die Bibliothek wurde für die Verwendung auf mobilen Endgeräten optimiert. Wie so oft bei Produkten von Google ist der Komfort in der Programmierung sehr hoch. Für eine einfache Straßenkarte von Mitteleuropa reichen diese wenigen Zeilen **HTML** und **JavaScript** aus:

```
<html>
<head>
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true"></script>
<script type="text/javascript">
window.onload = function() {
var map =
new google.maps.Map(document.getElementById("map"),
{ center: new google.maps.LatLng(47,11),
zoom: 7,
mapTypeId: google.maps.MapTypeId.ROADMAP
}
);
```

```
}
</script>
<body>
<div id="map" style="width:100%; height:100%"></div>
```

Beim Laden der Bibliothek muss der `sensors`-Parameter angegeben werden. Wenn dieser Wert auf `true` steht, kann das Endgerät seine Position bestimmen und der Anwendung mitteilen. Das ist vor allem bei mobilen Geräten (zum Beispiel Smartphones mit GPS) von Bedeutung. Ist die gesamte Seite geladen (`window.onload`), wird ein neues Objekt vom Typ `google.maps.Map` erzeugt, dessen Konstruktor als ersten Parameter das **HTML-Element** entgegennimmt, das zur Anzeige der Karte vorgesehen ist. Der zweite Parameter bestimmt als Liste von Optionen, wie und was auf der Karte dargestellt werden soll. In diesem Fall wird der Kartenmittelpunkt auf 47 Grad Nord und 11 Grad Ost mit der Zoomstufe 7 gesetzt (Zoomstufe 0 entspricht der Ansicht der gesamten Erde) und der Kartentyp über die Konstante `google.maps.MapTypeId.ROADMAP` als Straßenkarte festgelegt.

Info

Da der Konstruktor des Kartenobjekts einen Verweis auf den Inhalt der HTML-Seite enthält, darf er erst aufgerufen werden, wenn die Webseite geladen ist – also zum Zeitpunkt `window.onload`.

OpenStreetMap / OpenLayers

OpenStreetMap startete im Jahr 2004 mit dem sehr ambitionierten Ziel, eine umfassende und freie Plattform für `Geodaten` weltweit zu werden. Anknüpfend an die erfolgreiche Methode von Wikipedia wollte man es den Benutzern einfach machen, geografische Elemente im persönlichen Umfeld aufzunehmen und im Internet zu speichern. Wenn Sie bedenken, wie ungleich schwieriger es ist, Geodaten zu editieren, ist der bisherige Stand des Projekts beeindruckend.

Tausende User haben ihre GPS-Aufzeichnungen auf die Plattform OpenStreetMap hochgeladen und dort korrigiert und kommentiert. Zudem nahm man bereits vorhandene Geodaten, die über eine entsprechende Lizenz verfügten, mit in den Datenbestand auf (zum Beispiel den *TIGER-Datensatz* der US-Bundesstaaten oder die Landsat 7-Satellitenbilder).

> Im Umfeld des Projekts entstanden unterschiedliche Werkzeuge,

mit denen man Daten von den Servern von OpenStreetMap herunterladen und – eine entsprechende Berechtigung vorausgesetzt – hinaufladen und dort speichern kann. Die offenen Schnittstellen machen es für Software-Entwickler einfach, ihre Produkte an das System anzubinden.

Eine wesentliche Komponente für den Erfolg von **OpenStreetMap** ist eine einfache Möglichkeit für Webentwickler, Karten in ihre Webseiten einzubauen. Darum kümmert sich das Projekt **OpenLayers**. Die JavaScript-Bibliothek ist nicht auf den Einsatz mit OpenStreetMap beschränkt, kann aber in diesem Zusammenspiel ihre Stärken sehr gut ausspielen. Mit OpenLayers können Sie auch auf die Karten von Google, Microsoft, Yahoo und unzähligen anderen Geodiensten (auf der Basis der Standards WMS und

WFS) zugreifen.

Ein minimales Beispiel für eine Straßenkarte von Mitteleuropa mit OpenLayers und OpenStreetMap sieht folgendermaßen aus:

```
<!DOCTYPE html>
<html>
<head>
<title>Geolocation - OpenLayers / OpenStreetMap</title>
<script src="http://www.openlayers.org/api/OpenLayers.js"></script>
<script src="http://www.openstreetmap.org/openlayers/OpenStreetMap.js"></script>
<script>
window.onload = function() {
var map = new OpenLayers.Map("map");
map.addLayer(new
OpenLayers.Layer.OSM.Osmarender("Osmarender"));
var lonLat = new OpenLayers.LonLat(11, 47).transform(
new OpenLayers.Projection("EPSG:4326"),
map.getProjectionObject()
);
map.setCenter (lonLat,7);
}
</script>
<body>
<div id="map" style="top: 0; left: 0; bottom: 0; right: 0; position:
fixed;"></div>
</body>
</html>
```

Für dieses Beispiel muss einerseits die JavaScript-Bibliothek von OpenLayers und andererseits die Bibliothek von OpenStreetMap geladen werden.

Dem `OpenLayers.Map`-Objekt wird ähnlich wie bei Google Maps ein HTML-`div`-Element zur Darstellung zugewiesen und ein Layer vom Typ `Osmarender` hinzugefügt, der Standard-Kartenansicht von OpenStreetMap (OSM). Hier kommt eine Besonderheit von OpenStreetMap ins Spiel: Wie schon in Ein Wort zu geografischen Daten, erwähnt wurde, müssen dreidimensionale Informationen projiziert werden, um sie am Bildschirm in 2D darzustellen. Während Google Maps den Anwender nicht mit diesen Details belästigt und man ganz einfach die X- und Y-Koordination in Dezimalgrad angibt, verlangt OpenLayers, dass Angaben in Dezimalgrad zuerst in das entsprechende Koordinatensystem projiziert werden. Intern verwendet OpenLayers genauso wie Google Maps, Yahoo! Maps und Microsoft Bing Maps für die Kartendarstellung eine Projektion, die als **Spherical Mercator** (EPSG-Code 3785) bezeichnet wird. In Spherical Mercator werden Koordinaten nicht in Dezimalgrad, sondern in Metern verwaltet, weshalb die hier verwendeten Gradangaben mit dem Aufruf `transform()` unter Angabe des EPSG-Codes der gewünschten Projektion (EPSG:4326) in das in der Karte verwendete Koordinatensystem (ermittelt durch die Funktion `map.getProjectionObject()`) konvertiert werden müssen.

Info

Wird der für **HTML5** korrekte DOCTYPE am Beginn des Dokuments verwendet, muss das **HTML-Element**, in dem die Karte angezeigt wird, eine Positionierung von `fixed` oder `absolute` aufweisen. Anderenfalls zeigt die OpenLayers-Bibliothek gar nichts an. Interessanterweise fällt diese Einschränkung weg, wenn kein DOC-TYPE angegeben wird.
Um die **Geolocation**-Funktion des Browsers zu testen, reicht folgender Java-Script-Code aus:

```
function $(id) { return document.getElementById(id); }
window.onload = function() {
if (navigator.geolocation) {
navigator.geolocation.getCurrentPosition(
function(pos) {
$("lat").innerHTML = pos.coords.latitude;
$("lon").innerHTML = pos.coords.longitude;
$("alt").innerHTML = pos.coords.altitude;
},
function() {},
{enableHighAccuracy:true, maximumAge:600000}
);
}
else {
$("status").innerHTML =
'No Geolocation support for your Browser';
}
}
```

In der ersten Zeile des Listings wird eine Hilfsfunktion mit dem Namen $ definiert, die eine verkürzte Schreibweise der Funktion `document.getElementById()` erlaubt (ähnlich einem Alias). Dieser Trick wurde aus der bekannten **jQuery-Bibliothek** übernommen und ist für das vorliegende Beispiel sehr angenehm, da die zu befüllenden Elemente auf der Webseite alle mit einem ID-Attribut gekennzeichnet sind.

Wie schon in den vorangegangenen Beispielen (Online-Kartendienste) stellt `window.onload` sicher, dass die Inhalte der Webseite vollständig geladen sind, bevor Referenzen auf HTML-Elemente gesetzt werden. Die erste `if`-Abfrage überprüft, ob der Browser die Geolocation API unterstützt. Sollte dies nicht der Fall sein, wird eine entsprechende Meldung in das Element mit der ID status geschrieben. Andernfalls kommt die eigentliche Funktion zur Bestimmung der Position zum Einsatz: `navigator.geolocation.getCurrentPosition()`.

> Beim Aufruf dieser Funktion muss der Browser laut Spezifikation nachfragen, ob es gewünscht ist,

dass die aktuelle Position ermittelt und der Webseite bekannt gegeben wird.

- Dem Funktionsaufruf werden drei Argumente übergeben:
- eine Funktion, die nach erfolgreicher Positionsbestimmung ausgeführt werden soll (Success-Callback),
- eine Funktion, die auf Fehler nach einer gescheiterten Positionsbestimmung reagieren kann (Error-Callback) sowie
- Wertepaare, die die Art der Positionsbestimmung beeinflussen.

Laut Spezifikation sind die beiden letzten Argumente optional, das Success-Callback muss immer angegeben werden. Um den JavaScript-Ablauf nicht zu blockieren, muss `getCurrentPosition()` asynchron, sozusagen im Hintergrund ausgeführt werden, und erst nachdem die Position bekannt ist oder ein Fehler aufgetreten ist, kann die entsprechende Callback-Funktionen aufgerufen werden.

In diesem sehr kurzen Beispiel sind beide Callback-Funktionen als anonyme Funktionen implementiert, wobei der Fehlerfall nicht weiter behandelt wird. Das Wertepaar `enableHighAccuracy: true` weist den Browser an, eine möglichst genaue Positionsbestimmung durchzuführen. Bei einem Android-Mobiltelefon bewirkt diese Einstellung zum Beispiel die Aktivierung des internen **GPS-Sensors** (mehr dazu finden Sie in Abschnitt Technischer Hintergrund der Positionsbestimmung). `maximumAge` legt schließlich die Zeit in Millisekunden fest, in der eine bereits bestimmte Position wieder verwendet werden darf. Nach Ablauf dieser Zeitspanne muss die Position neu bestimmt werden – in unserem Fall alle zehn Minuten.

Nach erfolgreicher Positionsbestimmung enthält die Variable `pos` des Success-Callbacks im sogenannten Position-Interface Angaben zur Koordinate (`pos.coords`) sowie einen Zeitstempel in Millisekunden seit 1970 (`pos.timestamp`). Abbildung 2 zeigt die verfügbaren Attribute mit ihren jeweiligen Werten, sofern welche vorhanden sind.

Zusätzlich zu `latitude`, `longitude` und `altitude` liefert `pos.coords` auch noch Informationen zur Genauigkeit der Position (`accuracy`, `altitudeAccuracy`) sowie zu möglicher Geschwindigkeit (`speed`) und Richtung (`heading`). Während Google-Chrome sich auf die in der Spezifikation geforderten Attribute beschränkt, gibt Firefox eine ganze Reihe zusätzlicher Informationen aus – unter anderem sogar Angaben zur Adresse, wie der Code zeigt, einen Auszug aus dem Ergebnis von `JSON.stringify(pos)` bietet:

```
{"coords":
..// ...
"address":
{"streetNumber":"6","street":"Postgasse",
"premises":null,"city":"Murnau am Staffelsee",
"county":"Garmisch-Partenkirchen","region":"Bavaria",
"country":"Germany","countryCode":"DE",
"postalCode":"82418","contractID":"",
```

```
"classDescription":"wifi geo position address object",
// ...
},
// ...
}
```

Erstaunlich viele Informationen, die der Browser hier zur Verfügung stellt! Woher diese Informationen kommen, erklärt der folgende Abschnitt.

Technischer Hintergrund der Positionsbestimmung

Wenn Sie im Ausland die Seite Google aufrufen, werden Sie mehr oder weniger überrascht feststellen, dass Sie zur entsprechenden Landesseite von Google umgeleitet werden. Das funktioniert auch ohne einen Geolocation-fähigen Browser: Google bedient sich eines einfachen Tricks und bestimmt den Aufenthaltsort anhand der IP-Adresse.

Browser, die die **Geolocation-API** unterstützen, können eine wesentlich höhere Genauigkeit erreichen, indem sie auf weitere technische Möglichkeiten zurückgreifen. Aktuell sind folgende Methoden im Einsatz:

- 1. Bei PCs mit kabelgebundenem Internetanschluss wird die Position anhand der IP-Adresse bestimmt. Diese Bestimmung ist erwartungsgemäß ziemlich ungenau.
- 2. Bei einer vorhandenen Wireless-LAN-Verbindung kann eine wesentlich präzisere Positionsbestimmung durchgeführt werden. Google sammelte dazu weltweit Daten von öffentlichen und privaten WLANs.
- 3. Verfügt die Hardware über einen Mobilfunk-Chip (zum Beispiel bei einem Smartphone), wird versucht, die Position innerhalb des Mobilfunk-Netzwerks zu errechnen.
- 4. Verfügt die Hardware außerdem über einen GPS-Sensor, kann die Position noch genauer bestimmt werden. GPS ist ein satellitengestütztes Positionierungsverfahren und kann bei günstigen Verhältnissen (außerhalb von Gebäuden, wenig Abschattung des Horizonts, ...) selbst mit billigen Sensoren eine Genauigkeit im Meterbereich erreichen.

Nur der GPS-Sensor funktioniert offline, die Methoden 1-3 benötigen Internet-Zugriff und werden durch einen Serverdienst umgesetzt. Diese Serverdienste gibt es von Google (**Google Location Service**, verwendet in Firefox, Chrome und Opera) und von einer weiteren amerikanischen Firma namens Skyhook Wireless (verwendet in Safari und in frühen Versionen von Opera).

> Aber wie kommen diese Dienstleister zu den Standort-Informationen von Wireless- und Mobilfunk-Netzwerken?

Parallel zu den Fotos, die Google für den Dienst Street View aufnimmt, speichern die Google-StreetView-Aufnahmefahrzeuge auch Informationen zu öffentlichen und privaten WLANs ab. Nachdem im Frühjahr 2010 bekannt wurde, dass dabei nicht nur die MAC-Adresse und die SSID des WLANs, sondern auch Nutzdaten mitprotokolliert wurden, fiel ein schlechtes Licht auf Google, und der Konzern musste sich

mehrmals öffentlich entschuldigen.

Damit aber noch nicht genug: Sofern der Browser Zugriff auf die Informationen eines Mobilfunk-Netzwerks oder Wireless-LAN-Routers hat, werden diese bei jedem Aufruf des Dienstes mitgesendet. Für Google betrifft das vor allem Mobilfunkgeräte mit dem Betriebssystem Android, Skyhook profitiert hier von den iPhone-Benutzern. Die Kombination der vorgestellten Methoden führt zu einem sehr großen Datensatz von Geodaten, über den diese beiden Dienstleister verfügen und den sie durch Crowdsourcing ständig aktualisieren (auch wenn die Benutzer als Datenlieferanten davon nichts bemerken).

Für Firefox gibt es mit dem Add-On Geolocator eine Erweiterung, die vor allem beim Entwickeln von Anwendungen sehr hilfreich ist. Es ermöglicht die Eingabe von Positionen, die Firefox auf den Geolocation-API-Aufruf zurückgibt. Mithilfe eines Pulldown-Menüs kann der Standort ausgewählt werden, ohne jedes Mal den Online-Dienst von Google zu Hilfe nehmen zu müssen.

Struktur in HTML5

Wir machen heutzutage eine Menge mit Ajax in unseren Webanwendungen. Typischerweise lösen wir irgendeinen visuellen Effekt aus, um den Benutzer darauf hinzuweisen, dass sich etwas auf der Seite verändert hat. Personen, die einen Screenreader verwenden, sind aber aus naheliegenden Gründen nicht dazu in der Lage, solche visuellen Hinweise zu erkennen. Die **WIA-ARIA**-Spezifikation bietet eine Alternativlösung, die derzeit in IE, Firefox und Safari mit verschiedenen beliebten Bildschirmlesegeräten funktioniert.

Der Kommunikationsdirektor möchte eine neue Homepage. Sie soll Links auf die Abschnitte „Services“, „Contact“ und „About“ enthalten. Er besteht darauf, dass die Seite nicht gescrollt werden soll, weil „viele Leute es hassen, zu scrollen“. Er möchte, dass Sie einen Prototyp für die Seite mit einem horizontalen Menü erstellen, über das per Klick der Hauptinhalt der Seite geändert werden kann. Das ist einfach zu implementieren, und mit dem Attribut `aria-live` können wir sogar etwas tun, das bisher nicht so ohne Weiteres möglich war – eine derartige Oberfläche screenreader freundlich implementieren.

Erstellen wir eine einfache Oberfläche wie in Abbildung 1. Wir stellen den gesamten Inhalt auf die Homepage. Falls JavaScript verfügbar ist, blenden wir alles bis auf den ersten Eintrag aus. Wir lassen die Navigationslinks mit Seitenankern auf den jeweiligen Abschnitt verweisen und verwenden **jQuery**, um diese Links in Events umzuwandeln, die den Hauptinhalt austauschen. Leute mit JavaScript sehen, was unser Direktor möchte, und Menschen ohne JavaScript können trotzdem den gesamten Inhalt der Seite sehen.

Die Seite erstellen

Wir beginnen damit, eine einfache **HTML5-Seite** zu erstellen und fügen unseren Abschnitt „Welcome“ ein, der standardmäßig angezeigt wird, wenn Benutzer die Seite besuchen. Hier sehen Sie den Code für die Seite mit der Navigationsleiste und den Jump-Links:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>HTML-Info</title>
<link rel="stylesheet" href="style.css" type="text/css">
</head>
<body>
<header id="header">
<h1>HTML-Info</h1>
<nav>
<ul>
<li><a href="#welcome">Welcome</a></li>
<li><a href="#services">Services</a></li>
<li><a href="#contact">Contact</a></li>
<li><a href="#about">About</a></li>
</ul>
</nav>
</header>
<section id="content" role="document" aria-live="assertive" aria-atomic="true">
<section id="welcome">
<header>
<h2>Welcome</h2>
</header>
<p>The welcome section</p>
</section>
</section>
<footer id="footer">
<p>© 2010 HTML-Info</p>
<nav>
<ul>
<li><a href="http://www.html-info.eu/">Home</a></li>
<li><a href="about">About</a></li>
<li><a href="terms.html">Terms of Service</a></li>
<li><a href="privacy.html">Privacy</a></li>
</ul>
</nav>
</footer>
</body>
</html>
```

Der Abschnitt „Welcome“ erhält die ID welcome, die dem Anker in der Navigationsleiste entspricht. Die übrigen Seitenabschnitte können wir auf dieselbe Weise deklarieren.

```
<section id="services">
<header>
<h2>Services</h2>
</header>
<p>The service section</p>
</section>
<section id="contact">
<header>
<h2>Contact</h2>
</header>
<p>The contact section</p>
</section>
<section id="about">
<header>
<h2>About</h2>
</header>
<p>The about section</p>
</section>
```

Unsere vier Inhaltsbereiche verpacken wir in das folgende **Markup**:

```
<section id="content" role="document" aria-live="assertive" aria-atomic="true">
```

Die Attribute in dieser Zeile teilen Screenreadern mit, dass dieser Seitenbereich aktualisiert wird.

Höfliche, aber bestimmte Aktualisierung
Es gibt zwei verschiedene Methoden, um den Benutzer bei der Verwendung von `aria-live` auf Änderungen der Seite hinzuweisen. Die Methode `polite` wurde entwickelt, um den Arbeitsfluss des Benutzers nicht zu unterbrechen. Wenn der Screenreader beispielsweise einen Satz vorliest, während ein Bereich der Seite aktualisiert wird und der Modus `polite` gewählt ist, liest das Bildschirmlesegerät den aktuellen Satz zu Ende. Ist dagegen der Modus `assertive` festgelegt, wird für diesen Inhalt eine hohe Priorität angenommen: Der **Screenreader** hört auf, den aktuellen Satz vorzulesen und beginnt mit dem Vorlesen des neuen Inhalts. Es ist wirklich wichtig, dass Sie sich bei der Entwicklung Ihrer Website für die richtige Art der Unterbrechung entscheiden. Der übermäßige Einsatz von `assertive` kann dazu führen, dass Ihre Benutzer die Orientierung verlieren und verwirrt werden. Verwenden Sie daher `assertive` nur, wenn es absolut notwendig ist. In unserem Fall ist es die richtige Entscheidung, weil wir den anderen Inhalt ausblenden.
Alles vorlesen lassen
Der zweite Parameter, `aria-atomic=true`, weist den Screenreader an, den gesamten Inhalt des veränderten Abschnitts zu lesen. Wenn wir dafür den Wert `false` angeben, weisen wir damit den Screenreader an, nur veränderte Knoten vorzulesen. Wir ersetzen den gesamten Inhalt, deshalb ist es sinnvoll, das Bildschirmlesegerät anzuweisen, auch alles vorzulesen. Wenn wir dagegen nur ein einzelnes

Listenelement ersetzen oder etwas mit Ajax zu einer Tabelle hinzufügen, wäre false die richtige Entscheidung.

Bereiche ausblenden

Zum Ausblenden der Bereiche müssen wir ein bisschen **JavaScript** schreiben und in unsere Seite einfügen. Wir erstellen hierzu eine Datei mit dem Namen application.js und binden diese Datei ebenso wie die jQuery-Bibliothek in unsere Seite ein.

```
<script type="text/javascript" charset="utf-8"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js">
</script>
<script type="text/javascript" charset="utf-8" src="javascripts/application.js">
</script>
```

Unsere Datei application.js enthält ein einfaches Skript:

```
// Unterstützung der HTML5-Strukturelemente für IE 6, 7 und 8
document.createElement("header");
document.createElement("footer");
document.createElement("section");
document.createElement("aside");
document.createElement("article");
document.createElement("nav");
$(function(){
$("#services, #about, #contact").hide().addClass("hidden");
$("#welcome").addClass("visible");
$("nav ul").click(function(event){
target = $(event.target);
if(target.is("a")){
event.preventDefault();
if ( $(target.attr("href")).hasClass("hidden") ){
$(".visible").removeClass("visible")
.addClass("hidden")
.hide();
$(target.attr("href"))
.removeClass("hidden")
.addClass("visible")
.show();
};
};
});
});
```

In Zeile 11 blenden wir die Abschnitte „services“, „about“ und „contact“ aus. Außerdem wenden wir darauf

die Klasse `hidden` an und weisen in der nächsten Zeile dem Abschnitt welcome die Klasse `visible` zu. Auf diese Weise ist es wirklich einfach festzustellen, welche Abschnitte beim Umschalten aktiviert oder deaktiviert werden müssen.

In Zeile 14 fangen wir sämtliche Klicks auf die Navigationsleiste ab und ermitteln in Zeile 17, welches Element angeklickt wurde. Wenn der Benutzer auf einen Link geklickt hat, prüfen wir, ob der entsprechende Abschnitt versteckt ist. Über das `href`-Attribut des geklickten Links können wir mit **jQuery-Selektoren** den entsprechenden Abschnitt finden, wie Sie in Zeile 19 sehen.

Ist der Abschnitt ausgeblendet, zeigen wir den gewählten Abschnitt an und blenden alle anderen aus. Das war's. Der Screenreader sollte die Änderungen dieses Bereichs feststellen.

Ausweichlösung

Wie auch die Rollen kann diese Lösung sofort mit den neuesten Versionen von Screenreadern verwendet werden. Wenn Sie sich an einige bewährte Regeln halten, zum Beispiel an den Verzicht auf aufdringliches JavaScript, haben wir eine einfache Implementierung, die für ein durchaus breites Publikum funktioniert. Ältere Browser und Bildschirmlesegeräte ignorieren die zusätzlichen Attribute, sodass es auch keine Gefahr darstellt, wenn wir sie in unseren Markup einfügen.

Die Zukunft

HTML5 und die **WIA-ARIA-Spezifikation** haben den Weg für ein wesentlich barrierefreieres Web geebnet. Mit der Möglichkeit, veränderliche Bereiche der Seite zu kennzeichnen, können Entwickler umfangreichere JavaScript-Anwendungen entwickeln, ohne sich allzu viele Gedanken über Barrierefreiheit machen zu müssen.

Zu Beginn würde ich gern mit Ihnen über ein ernst zu nehmendes Problem sprechen, von dem heutzutage viele Webentwickler betroffen sind: Die Divitis greift um sich – ein chronisches Syndrom, das dazu führt, dass Webentwickler Elemente in zusätzliche `div`-Tags mit IDs wie zum Beispiel `banner`, `sidebar`, `article` und `footer` verpacken. Dieses Syndrom ist in hohem Maße ansteckend. Divitis verbreitet sich extrem schnell von Entwickler zu Entwickler. Und da divs für das bloße Auge unsichtbar sind, bleiben insbesondere leichte Fälle von Divitis unter Umständen jahrelang unbemerkt.

Ein häufiges Symptom von Divitis sieht so aus:

```
<div id="navbar_wrapper">
<div id="navbar">
<ul>
<li><a href="/">Home</a></li>
<li><a href="/">Home</a></li>
</ul>
</div>
</div>
```

Hier wurde eine ungeordnete Liste, die ihrerseits ja bereits ein **Blockelement** ist, in zwei `div`-Tags eingebettet, die auch wiederum Blockelemente sind. Die `id`-Attribute dieser Wrapper-Elemente erklären zwar, welche Aufgabe sie haben. Aber Sie können zumindest einen dieser Wrapper entfernen und zum

gleichen Ergebnis kommen. Diese übermäßige Verwendung von Markup führt zu aufgeblähten Seiten, die schwierig zu gestalten und zu pflegen sind.

Es gibt jedoch Hoffnung: Die HTML5-Spezifikation liefert ein Heilmittel in Form von neuen semantischen Tags, die ihren Inhalt beschreiben. Weil so viele Entwickler Seitenleisten, Kopfzeilen, Fußzeilen und Abschnitte in ihren Designs verwenden, werden mit der HTML5-Spezifikation spezielle Tags eingeführt, um Seiten in solche logischen Abschnitte zu unterteilen. Machen wir uns mit diesen neuen Elementen an die Arbeit. Mit HTML5 können wir der Divitis noch in unserer Generation ein Ende setzen.

Neben den neuen strukturellen Tags werden wir auch das `meter`-Element besprechen und Ihnen zeigen, wie Sie in **HTML5** mit den neuen benutzerdefinierten Attributen Daten in Elemente einbetten können, ohne dafür Klassen oder existierende Attribute zu missbrauchen. Mit anderen Worten werden wir herausfinden, wie wir das richtige Tag für die richtige Aufgabe einsetzen.

> In einem Blog finden Sie jede Menge Inhalte, die nach strukturiertem Markup schreien.

Sie brauchen Kopfzeilen, Fußzeilen, mehrere Navigationsmöglichkeiten (Archive, Blogrolls und interne Links) und natürlich Artikel oder Beiträge. Basteln wir mit HTML5 ein Mock-up für die Startseite des Blogs.

In Abbildung 1 sehen Sie, worum es geht. Es handelt sich um eine typische Blog-Struktur, mit einer Hauptkopfzeile und einer horizontalen Navigation darunter. Im Hauptbereich erhält jeder Artikel eine Kopf- und eine Fußzeile. Artikel können außerdem einen Zitatkasten oder Sekundärinhalte enthalten. Zum Abschluss erhält die Seite eine Fußzeile für Kontakt- und Copyright-Informationen. An der Struktur an sich ist nichts außergewöhnlich. Allerdings bauen wir sie diesmal nicht aus einer Menge `div`-Tags auf, sondern verwenden spezielle Tags, um die jeweiligen Abschnitte zu beschreiben.

Wenn wir fertig sind, erhalten wir etwas, das in etwa wie Abbildung 2 aussieht.

Auf den Doctype kommt es an

Wir möchten die neuen Elemente von HTML5 verwenden. Deshalb müssen wir Browsern und Validierern mitteilen, welche Tags wir verwenden. Erstellen Sie eine neue Seite mit dem Namen index.html, und schreiben Sie die folgende einfache HTML5-Vorlage in die Datei.

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>HTML-Info Blog</title>
</head>
<body>
</body>
</html>
```

Sehen Sie sich mal den Doctype in Zeile 1 an. Mehr brauchen wir für den **HTML5-Doctype** nicht zu schreiben. Wenn Sie regelmäßig Webseiten erstellen, sind Sie wahrscheinlich eher mit diesen langen, schwer zu merkenden Doctypes für XHTML vertraut:

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

Zum Vergleich noch einmal der HTML5-Doctype:

```
<!DOCTYPE HTML>
```

Das ist doch viel einfacher und viel leichter zu merken!

Ein Doctype hat zweierlei Aufgaben. Erstens hilft er Validierern festzustellen, welche Validierungsregeln für die Validierung des Codes herangezogen werden müssen. Außerdem zwingt der Doctype den Internet Explorer 6, 7 und 8, in den „Standards Mode“ zu wechseln, was von entscheidender Bedeutung ist, wenn Sie Webseiten erstellen, die in allen Browsern funktionieren sollen. Der HTML5-Doctype erfüllt beide Anforderungen und wird sogar vom Internet Explorer 6 erkannt.

Kopfzeilen
Kopfzeilen (nicht zu verwechseln mit Überschriften wie etwa `h1`, `h2` und `h3`) können alle möglichen Arten von Inhalten enthalten – vom Unternehmenslogo bis hin zum Suchfeld. Die Kopfzeile unseres Blogs soll zunächst nur den Titel des Blogs enthalten.

```
<header id="page_header">
<h1>HTM;L-Info Blog!</h1>
</header>
```

Sie sind nicht auf eine Kopfzeile pro Seite beschränkt. Jeder einzelne Abschnitt oder Artikel kann eine eigene Kopfzeile enthalten. Da kann es hilfreich sein, Ihre Elemente mit dem `id`-Attribut eindeutig zu kennzeichnen, wie ich das in Zeile 1 getan habe. Mit einer eindeutigen ID sind Elemente mit **CSS** einfach zu gestalten oder mit JavaScript ausfindig zu machen.
Semantisches Markup

Semantisches Markup dient dazu, Ihren Inhalt zu beschreiben. Wenn Sie bereits seit mehreren Jahren Webseiten entwickeln, teilen Sie Ihre Seiten wahrscheinlich in verschiedene Abschnitte wie `header`, `footer` und `sidebar` auf. Dadurch können Sie die einzelnen Abschnitte der Seite leichter identifizieren, wenn Sie Stylesheets und andere Formatierungen darauf anwenden.

Semantisches Markup macht es Maschinen und Menschen gleichermaßen leicht, die Bedeutung und den Kontext des Inhalts zu verstehen. Die neuen Markup-Tags von HTML5 wie `header` und `nav` sollen Ihnen genau dabei helfen.

Fußzeilen

Das `footer`-Element definiert Fußzeileninformationen für ein Dokument oder einen angrenzenden Abschnitt. Fußzeilen auf Websites kennen Sie bereits. Üblicherweise stehen darin Informationen wie das Copyright-Datum oder darüber, wem die Website gehört. Der Spezifikation entsprechend können wir auch mehrere Fußzeilen in einem Dokument haben. Das bedeutet also, dass wir auch innerhalb unserer Blog-Artikel Fußzeilen verwenden können.

Für den Moment definieren wir einfach eine einfache Fußzeile für unsere Seite. Da wir mehr als eine Fußzeile verwenden können, geben wir dieser genauso wie der Kopfzeile eine ID. Dadurch können wir diese bestimmte Fußzeile eindeutig identifizieren, wenn wir das Element und seine Kinder stylen möchten.

```
<footer id="page_footer">
<p>© 2010 HTML-Info.</p>
</footer>
```

Die Fußzeile enthält lediglich ein Copyright-Datum. Jedoch können Fußzeilen genau wie Kopfzeilen auch andere Elemente enthalten, wie etwa Navigationselemente.

Navigation

Die Navigation ist für den Erfolg einer Website entscheidend. Ihre Besucher werden nicht lange bleiben, wenn sie nicht das finden, was sie suchen. Daher ist es absolut sinnvoll, dass die Navigation ein eigenes **HTML-Tag** erhält.

Fügen wir einen Navigationsabschnitt in die Kopfzeile unseres Dokuments ein. Wir erstellen Links auf die Homepage des Blogs, das Archiv, eine Liste mit den Verfassern von Beiträgen zum Blog und einen Link auf die Kontaktseite.

```
<header id="page_header">
<h1>HTML-Info Blog!</h1>
<nav>
<ul>
<li><a href="/">Latest Posts</a></li>
<li><a href="archives">Archives</a></li>
```

```
<li><a href="contributors">Contributors</a></li>
<li><a href="contact">Contact Us</a></li>
</ul>
</nav>
</header>
```

Ihre Seite kann nicht nur mehrere Kopf- und Fußzeilen, sondern auch mehrere Navigationselemente enthalten. Die Navigation befindet sich häufig in der Kopfzeile und in der Fußzeile, die Sie nun explizit kennzeichnen können. Die Fußzeile unseres Blogs braucht Links auf die Homepage von HTML-Info, eine Seite über das Unternehmen sowie Links auf die Nutzungsbedingungen und Datenschutzrichtlinien. Diese Links fügen wir in der Seite als eine weitere ungeordnete Liste im Element `footer` ein.

```
<footer id="page_footer">
<p>© 2010 HTML-Info.</p>
<nav>
<ul>
<li><a href="http://www.html-info.eu/">Home</a></li>
<li><a href="about">About</a></li>
<li><a href="terms.html">Terms of Service</a></li>
<li><a href="privacy.html">Privacy</a></li>
</ul>
</nav>
</footer>
```

Das Aussehen der beiden Navigationsleisten passen wir später mit CSS an. Achten Sie daher zunächst nicht so sehr auf die Optik. Die Aufgabe dieser neuen Elemente ist es, den Inhalt zu beschreiben, nicht aber, das Aussehen des Inhalt zu bestimmen.

Abschnitte und Artikel

Abschnitte sind die logischen Bereiche einer Seite. Deshalb gibt es nun das Element `section`, um das oft missbrauchte `div`-Tag bei der Beschreibung logischer Abschnitte einer Seite abzulösen.

```
<section id="posts">
</section>
```

Übertreiben Sie es aber nicht mit den Abschnitten. Setzen Sie sie ein, um Ihren Inhalt logisch zu gliedern. Hier haben wir einen Abschnitt erstellt, der alle Beiträge in einem Blog enthält. Jedoch soll nicht jeder Beitrag einen eigenen Abschnitt erhalten. Dafür gibt es ein besser geeignetes Tag.

Artikel

Das `article`-Tag ist das perfekte Element, um den Inhalt einer **Webseite** zu beschreiben. Mit den vielen

Elementen auf einer Seite – Kopfzeilen, Fußzeilen, Navigationselemente, Werbung, Widgets, Blogrolls und Lesezeichen für soziale Medien – könnten Sie glatt vergessen, dass die Benutzer Ihre Website wegen des von Ihnen angebotenen Inhalts besuchen. Das `article`-Tag hilft Ihnen dabei, diesen Inhalt zu beschreiben.

Jeder unserer Artikel besteht aus einer Kopfzeile, dem eigentlichen Inhalt und einer Fußzeile. So definieren wir einen vollständigen Artikel:

```
<article class="post">
<header>
<h2>How Many Should We Put You Down For?</h2>
<p>Posted by Brian on <time datetime="2010-10-01T14:39">October 1st, 2010 at
2:39PM</time></p>
</header>
<p>The first big rule in sales is that if the person leaves empty-handed,
they're likely not going to come back. That's why you have to be somewhat
aggressive when you're working with a customer, but you have to make sure you
don't overdo it and scare them away.</p>
<p>One way you can keep a conversation going is to avoid asking questions that
have yes or no answers. For example, if you're selling a service plan, don't
ever ask "Are you interested in our 3 or 5 year service plan?" Instead, ask "Are
you interested in the 3 year service plan or the 5 year plan, which is a better
value?" At first glance, they appear to be asking the same thing, and while a
customer can still opt out, it's harder for them to opt out of the second
question because they have to say more than just "no."</p>
<footer>
<p><a href="comments"><i>25 Comments</i></a>...</p>
</footer>
</article>
```

Joe fragt …

Was ist der Unterschied zwischen Artikeln und Abschnitten?

Stellen Sie sich einen Abschnitt als logischen Teil eines Dokuments vor. Einen Artikel können Sie sich als den eigentlichen Inhalt vorstellen, wie etwa einen Artikel in einer Zeitschrift, einen Beitrag in einem Blog oder eine aktuelle Meldung.

> Die neuen Tags beschreiben genau den Inhalt, den sie enthalten.

Abschnitte können mehrere Artikel enthalten, und Artikel können aus mehreren Abschnitten bestehen. Ein Abschnitt ist wie der Sportteil einer Zeitung. Der Sportteil enthält viele Artikel. Jeder dieser Artikel kann

wiederum aus mehreren eigenständigen Abschnitten bestehen. Bestimmte Abschnitte wie Kopfzeilen und Fußzeilen erhalten eigene Tags. Ein Abschnitt ist ein allgemein gehalteneres Element, mit dem Sie andere Elemente logisch gruppieren können.

Bei semantischem Markup geht es darum, die Bedeutung Ihres Inhalts zu vermitteln.

In einem Artikel können wir die **Elemente** `header` und `footer` verwenden, wodurch es wesentlich einfacher wird, diese bestimmten Abschnitte zu beschreiben. Wir können unseren Artikel auch mit dem Element `section` in mehrere Abschnitte unterteilen.

Das aside-Tag und Seitenleisten

Manche Inhalte ergänzen etwas zum eigentlichen Inhalt, wie etwa Zitatkästen, Diagramme, weiterführende Gedanken oder entsprechende Links. Mit dem neuen `aside`-Tag können Sie diese Elemente kenntlich machen.

```
<aside>
<p>"Never give someone a chance to say no when selling your product."</p>
</aside>
```

Das Zitat schreiben wir in ein `aside`-Element. Das `aside`-Element verschachteln wir wiederum in den Artikel und platzieren es damit nahe beim zugehörigen Inhalt.

So sieht unser vollständiger Abschnitt mit dem `aside`-Element aus:

```
<section id="posts">
<article class="post">
<header>
<h2>How Many Should We Put You Down For?</h2>
<p>Posted by Brian on <time datetime="2010-10-01T14:39">October 1st, 2010 at
2:39PM</time></p>
</header>
<aside>
<p>"Never give someone a chance to say no when selling your product."</p>
</aside>
<p>The first big rule in sales is that if the person leaves empty-handed,
they're likely not going to come back.
That's why you have to be somewhat aggressive when you're working with a
customer, but you have to make sure you don't overdo it and scare them away.</p>
<p>One way you can keep a conversation going is to avoid asking questions that
have yes or no answers. For example, if you're selling a service plan, don't
ever ask "Are you interested in our 3 or 5 year service plan?" Instead, ask "Are
```

```
you interested in the 3 year service plan or the 5 year plan, which is a better
value?" At first glance, they appear to be asking the same thing, and while a
customer can still opt out, it's harder for them to opt out of the second
question because they have to say more than just "no."</p>
<footer>
<p><a href="comments"><i>25 Comments</i></a>...</p>
</footer>
</article>
</section>
```

Nun müssen wir nur noch den Abschnitt für die Seitenleiste einfügen.

aside-Elemente sind keine Seitenleisten
Unser Blog enthält eine rechte Seitenleiste mit den Archiven für den Blog. Falls Sie jetzt glauben, wir könnten das `aside`-Tag verwenden, um die Seitenleiste unseres Blogs zu definieren, liegen Sie leider falsch. Sie könnten das zwar so machen, aber es widerspricht dem Sinn der Spezifikation. `aside` wurde entwickelt, um die Inhalte anzuzeigen, die zu einem Artikel gehören: die richtige Stelle also, um entsprechende Links, ein Glossar oder einen Zitatkasten anzuzeigen.

Als Markup für unsere Seitenleiste mit der Liste der bisherigen Archive verwenden wir ein weiteres `section`-Tag sowie ein `nav`-Tag:

```
<section id="sidebar">
<nav>
<h3>Archives</h3>
<ul>
<li><a href="2010/10">October 2010</a></li>
<li><a href="2010/09">September 2010</a></li>
<li><a href="2010/08">August 2010</a></li>
<li><a href="2010/07">July 2010</a></li>
<li><a href="2010/06">June 2010</a></li>
<li><a href="2010/05">May 2010</a></li>
<li><a href="2010/04">April 2010</a></li>
<li><a href="2010/03">March 2010</a></li>
<li><a href="2010/02">February 2010</a></li>
<li><a href="2010/01">January 2010</a></li>
</ul>
</nav>
</section>
```

Damit haben wir schon unsere Blog-Struktur. Jetzt können wir damit beginnen, die neuen Elemente zu

stylen.

Styling

Genau wie auf `div`-Tags können wir auch auf die neuen Elemente Stilregeln anwenden. Zunächst erstellen wir ein neues Stylesheet mit dem Namen style.css. Anschließend verknüpfen wir es mit unserem **HTML-Dokument**, indem wir im Header einen Link auf das Stylesheet einfügen:

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

Als Erstes zentrieren wir den Inhalt der Seite und legen einige grundlegenden Schriftarten fest:

```
body {
width: 960px;
margin: 15px auto;
font-family: Arial, "MS Trebuchet", sans-serif;
}
p {
margin: 0 0 20px 0;
}
p, li {
line-height: 20px;
}
```

Dann definieren wir die Breite der Kopfzeile:

```
header#page_header {
width: 100%;
}
```

Für die Navigationslinks wandeln wir die Listen mit Aufzählungszeichen in horizontale Navigationsleisten um:

```
header#page_header nav ul, #page_footer nav ul {
list-style: none;
margin: 0;
padding: 0;
}
#page_header nav ul li, footer#page_footer nav ul li {
padding: 0;
margin: 0 20px 0 0;
display: inline;
}
```

Der Abschnitt posts muss gefloatet werden und eine feste Breite erhalten. Außerdem müssen wir auch den Callout im Artikel floaten. Bei der Gelegenheit vergrößern wir gleich noch die Schrift für den Callout.

```
section#posts {
float: left;
width: 74%;
}
section#posts aside {
float: right;
width: 35%;
```

Messwerte und Fortschrittsbalken
Wenn Sie ein Spendenbarometer oder einen Fortschrittsbalken für den Upload in einer Webanwendung implementieren möchten, sollten Sie einen Blick auf die Elemente `meter` und `progress` werfen, die mit HTML5 eingeführt wurden.

Mit dem Element `meter` können wir einen festen Punkt auf einer Messskala mit einem Mindest- und einem Höchstwert semantisch beschreiben. Damit Ihr Messwert mit der Spezifikation harmoniert, sollten Sie ihn nicht für beliebige Minimal- oder Maximalwerte wie Höhe oder Gewicht verwenden, außer Sie haben einen bestimmten Grenzwert festgelegt. Wenn wir beispielsweise auf einer Website zum Sammeln von Spenden zeigen möchten, wie weit wir noch von dem Ziel $5.000 entfernt sind, lässt sich das so beschreiben:

```
<section id="pledge">
<header>
<h3>Our Fundraising Goal</h3>
</header>
<meter title="USD" id="pledge_goal" value="2500" min="0"
max="5000">$2500.00</meter>
<p>Help us reach our goal of $5000!</p>
</section>
```

Das Element `progress` ist einem Messwert sehr ähnlich, wurde aber dafür entwickelt, einen aktiven Fortschritt darzustellen, zum Beispiel beim Hochladen einer Datei. Ein Messwert zeigt dagegen eine Messung an, die nicht mehr verändert wird, wie etwa einen Schnappschuss des Speicherplatzes, der auf einem Server einem bestimmten Benutzer noch zur Verfügung steht. Das Markup für einen Fortschrittsbalken ist dem eines `meter`-Elements aber sehr ähnlich:

```
<progress id="progressbar" max=100><span>0</span>%</progress>
```

Die Elemente `meter` und `progress` lassen sich noch in keinem Browser darstellen. Sie können jedoch mit **JavaScript** die Werte aus dem `meter`-Element auslesen und selbst abbilden, das Element `meter` oder `progress` also dafür verwenden, die Daten semantisch zu beschreiben.

```
margin-left: 5%
font-size: 20px;
line-height: 40px;
}
```

Außerdem müssen wir die Seitenleiste floaten und eine feste Breite dafür festlegen:

```
section#sidebar {
float: left;
width: 25%;
}
```

Und wir müssen die Fußzeile definieren. Wir löschen die Floats in der Fußzeile, sodass sie im unteren Teil der Seite zu liegen kommt.

```
footer#page_footer {
clear: both;
width: 100%;
display: block;
text-align: center;
}
```

Das sind nur die wichtigsten Stilregeln in Kürze. Ich bin mir sicher, dass Sie das alles noch viel, viel besser aussehen lassen können.

Ausweichlösung
Das funktioniert bereits alles wunderbar in Firefox, Chrome und Safari. Allerdings werden die Leute im Management nicht allzu glücklich sein, wenn sie das Chaos sehen, das der Internet Explorer aus unserer Seite macht. Der Inhalt wird zwar wunderbar dargestellt. Aber da der IE diese Elemente nicht kennt, kann er auch keine Stilregeln darauf anwenden. Die Seite erinnert deshalb eher an die Mitte der Neunziger-Jahre.

Mit dem IE können wir diese Elemente nur stylen, wenn wir sie mit JavaScript als Teil des Dokuments definieren. Wie sich herausstellt, ist das wirklich einfach. Wir fügen den **Code** in den Abschnitt head der Seite ein, sodass er ausgeführt wird, bevor der Browser irgendwelche Elemente rendert. Außerdem

schreiben wir den Code in einen bedingten Kommentar – eine besondere Art von Kommentar, die nur der Internet Explorer verarbeitet.

```
<!--[if lt IE 9]>
<script type="text/javascript">
document.createElement("nav");
document.createElement("header");
document.createElement("footer");
document.createElement("section");
document.createElement("aside");
document.createElement("article");
</script>
<![endif]-->
```

Solche Kommentare gelten nur für Versionen des Internet Explorer, die älter als Version 9.0 sind. Wenn wir die Seite jetzt noch einmal laden, wird sie richtig dargestellt.

> Allerdings machen wir uns dadurch von JavaScript abhängig. Das sollten Sie bedenken.

Für den verbesserten Aufbau und die Lesbarkeit des Dokuments lohnt sich das durchaus. Außerdem gibt es keine Probleme bezüglich der Barrierefreiheit, da der Inhalt nach wie vor angezeigt und von einem Bildschirmlesegerät vorgelesen werden kann. Die Darstellung wirkt lediglich hoffnungslos altmodisch für Benutzer, die JavaScript absichtlich deaktiviert haben.

Mit diesem Ansatz können Sie auch wunderbar zusätzliche Elemente unterstützen oder verstehen, wie Sie eine entsprechende Unterstützung schreiben können.

Die brillante Lösung HTMLShiv von Remy Sharp treibt die Dinge noch wesentlich weiter, ist aber wahrscheinlich eher dann angebracht, wenn Sie eine Ausweichlösung für viele zusätzliche Elemente integrieren möchten.

„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“.
- 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“.
- 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“.
- 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“.

Modernizr: Eine Bibliothek zur HTML5-Erkennung

Modernizr ist eine unter der MIT-Lizenz stehende Open Source-JavaScript-Bibliothek, die die Unterstützung für viele **HTML5**– und **CSS3-Funktionen** prüft. Sie sollten immer die aktuellste Version einsetzen. Fügen Sie dazu zu Beginn Ihrer Seite folgendes `<script>`-Element ein:

```
<!DOCTYPE html> <html>
<head>
<meta charset=ütf-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, 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 (HTML5-Spezifikation – canvas-Element) 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“). Wenn Ihr Browser die Canvas-API unterstützt, hat das von ihr erstellte DOM-Objekt zur Repräsentation eines <canvas>-Elements eine `getContext()`-Methode. 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, Pfaden, Verläufen und Mustern. Die externe explorercanvas-Bibliothek, die die Canvas-API in Microsofts Internet Explorer bereitstellt, erkennt es nicht.

Statt diese Funktion selbst zu schreiben, können Sie Modernizr 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“. Falls Ihr Browser die Canvas-API unterstützt, bietet das DOM-Objekt, das zur Repräsentation eines `<canvas>`-Elements genutzt wird, eine `getContext()`-Methode. 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“) 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.

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! nennt und HTML5-Video nutzt, so es verfügbar ist, in älteren Browsern aber auf Quick-Time 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“). 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“) nutzen, um die Unterstützung für HTML5-Video zu prüfen:

```
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“). 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=ävc1.42E01E, 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=ävc1.42E01E, 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.

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 Video 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"');
}
```

Statt eigenhändig diese Funktionen zu schreiben, nutzen Sie Modernizr, um die Unterstützung verschiedener **HTML5-Videoformate** zu prüfen:

```
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 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“). 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“) 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 (Same-Origin-Restriction) (d.h. eine Beschränkung auf den gleichen Urheber).

Web Worker

Web Worker (HTML-Spezifikation – Web Worker) 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"). 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") 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“). 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“). 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“) 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 standardisiert, die unabhängig von der HTML5 Working Group ist. Aber ich werde 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“). 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“) 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. Google 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.

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">

<input type="number">

<input type="range">

<input type="color">

<input type="tel">

<input type=ürl">

<input type="email">

<input type="date">

<input type="month">

<input type="week">

<input type="time">

<input type="datetime">
```

```
<input type="datetime-local">
```

Die Prüfung der Unterstützung der **HTML5-Eingabetypen** nutzt Erkennungstechnik Nummer 4 (siehe dazu den Abschnitt „Erkennungstechniken“). 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“) 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.

Die Prüfung der Platzhalterunterstützung nutzt Erkennungstechnik Nummer 2 (siehe dazu den Abschnitt

„Erkennungstechniken“). 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“) 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“). 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“) 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!
}
```

Microdaten

Microdaten (HTML-Spezifikation – Microdata) 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. Sie können 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“). 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.

Dieser Artikel 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.

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“ 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. Dort finden Sie auch 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 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. 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“. 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.

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. 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.

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, 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.

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:
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).
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 und definiert den Akzeptierungsprozess (Akzeptierungsprozess).

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"`

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"`

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"`

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"`

ist die zweitbeliebteste Link-Relation 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.

`rel="license"`

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"`
„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 aufgeben 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"`

„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.

`rel="pingback"`

gibt die Adresse eines „Pingback-Servers“ an. Wie in der Pingback-Spezifikation 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"`

„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"`

„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"`

„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.

`rel="tag"`

„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 diesen Artikel 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` 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> schreiben, schließt das span den Absatz nicht, weil Firefox weiß, dass <p> ein Blockelement ist, das das Inline-Element `<span>` enthalten kann. Das `<span>` 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 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 );
}
</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 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.

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)
```

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 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 Webaim 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>§</p>
<p>© 2001–9 <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>§</p>
<p>© 2001–9 <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.

Fat Footers (umfangreiche Fußleisten) sind aktuell der letzte Schrei. Schauen Sie sich nur die Fußleiste der W3C-Site 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.

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>
```

Mit CSS können Sie nicht nur vorhandene Elemente stylen, sondern auch Inhalte in ein Dokument einfügen. Es gibt einige Fälle, in denen die Erzeugung von Inhalten mit **CSS** sinnvoll ist. So wäre es zum Beispiel naheliegend, beim Ausdruck einer Seite die URLs von Hyperlinks neben dem jeweiligen Text mit auszugeben. Wenn Sie ein Dokument auf dem Bildschirm ansehen, können Sie einfach den Mauszeiger über den Link bewegen und sehen in der Statusleiste, wohin der Link führt. Auf dem Ausdruck einer Seite haben Sie dagegen keine Ahnung, wohin die Links verweisen.

Das Geschäft arbeitet an einer neuen Seite für Formulare und Richtlinien. Ein Mitglied des Gremiums für die Neugestaltung besteht darauf, bei jedem Treffen ein Exemplar der Website auszudrucken. Dieser Kollege möchte genau wissen, wohin die Links auf der Seite führen, damit er feststellen kann, ob sie

verändert werden müssen. Mit ein bisschen CSS können wir diese Funktionalität in IE 8, Firefox, Safari und Chrome hinbekommen. Und mit proprietärem JavaScript funktioniert es auch im IE 6 und 7.

Die Seite selbst enthält derzeit nichts außer einer Liste von Links. Bei Gelegenheit wird sie in eine entsprechende Vorlage eingesetzt.

```
<ul>
<li><a href="travel/index.html">Travel Authorization Form</a></li>
<li><a href="travel/expenses.html">Travel Reimbursement Form</a></li>
<li><a href="travel/guidelines.html">Travel Guidelines</a></li>
</ul>
</body>
```

Auf einem Ausdruck dieser Seite können Sie bisher nicht erkennen, wohin diese Links führen. Das werden wir ändern.

Das CSS

Wenn wir Stylesheets in eine Seite einbinden, können wir den Medientyp angeben, für den die Stilregeln gelten sollen. Meistens verwenden wir den Typ `screen`. Wir können aber auch den Typ `print` angeben, um ein **Stylesheet** zu definieren, das nur für den Ausdruck der Seite geladen wird (oder wenn der Benutzer die Druckvorschau aufruft).

```
<link rel="stylesheet" href="print.css" type="text/css" media="print">
```

Anschließend erstellen wir das Stylesheet print.css mit dieser einfachen Regel:

```
a:after {
content: " (" attr(href) ") ";
}
```

Dadurch wird hinter den Text jedes Links in Klammern der Wert des **Attributs** `href` hinzugefügt. Wenn Sie die Seite in einem modernen Browser ausdrucken, sieht das so aus:

Wenn Sie das Ganze in Aktion sehen möchten, ohne Papier zu verbrauchen, können Sie die Druckvorschau des Browsers verwenden, die ebenfalls dieses Stylesheet aufruft.

Damit haben wir alles im Griff, außer den Internet Explorer 6 und 7. Sollen wir uns darum als Nächstes kümmern?

Ausweichlösung
Der Internet Explorer verfügt über einige JavaScript-Events, von denen ich mir wünsche, dass alle Browser sie übernehmen: `onbeforeprint` und `onafterprint`. Mithilfe dieser Events können wir den Text des Hyperlinks ändern, wenn der **Ausdruck** angestoßen wird, und anschließend die Änderung wieder rückgängig machen, wenn der Druck abgeschlossen ist. Unsere Benutzer werden den Unterschied nicht bemerken.

Wir müssen lediglich eine Datei mit dem Namen print.js anlegen und diesen Code einfügen:

```
$(function() {
if (window.onbeforeprint !== undefined) {
window.onbeforeprint = ShowLinks;
window.onafterprint = HideLinks;
}
});
function ShowLinks() {
$("a").each(function() {
$(this).data("linkText",$(this).text());
$(this).append(" (" + $(this).attr("href") + ")");
});
}
function HideLinks() {
$("a").each(function() {
$(this).text($(this).data("linkText"));
});
}
```

Anschließend binden wir die Datei in unsere Seite ein. Wir brauchen diese Lösung lediglich für den IE 6 und 7, deshalb können wir einen bedingten Kommentar verwenden. Der Code setzt **jQuery** voraus, also müssen wir unbedingt auch die jQuery-Bibliothek einbinden.

```
<script charset="utf-8" src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2
jquery.min.js' type='text/javascript'>
</script>
<!--[if lte IE 7]>
<script type="text/javascript" src="print.js"></script>
<![endif]-->
</head>
<body>
<h1>Forms and Policies</h1>
<ul>
<li><a href="travel/index.html">Travel Authorization Form</a></li>
<li><a href="travel/expenses.html">Travel Reimbursement Form</a></li>
```

```
<li><a href="travel/guidelines.html">Travel Guidelines</a></li>
</ul>
```

Sobald das **JavaScript** eingebunden ist, werden die URLs von Links auf allen Browsern ausgedruckt. Sie können dieses Stylesheet für den Druck als Grundlage für ein umfassenderes verwenden und zum Beispiel dieses Verhalten nur auf bestimmte Links anwenden.

Im Printbereich werden seit Langem Spalten verwendet – und von vielen Webdesignern neidisch beäugt. Schmale Spalten erleichtern das Lesen, und seitdem Bildschirme immer breiter werden, suchen Entwickler nach Möglichkeiten, die Spalten auf eine angenehme Breite zu begrenzen. Schließlich möchte niemand Textzeilen quer über den ganzen Monitor lesen – genauso wie niemand Textzeilen über die gesamte Breite einer Zeitung lesen möchte. In den vergangenen zehn Jahren gab es einige ziemlich smarte Lösungen. Aber keine ist so einfach wie die Methode der **CSS3-Spezifikation**.

Spalten spalten
Das Unternehmen verwendet zufällig ein beliebtes webbasiertes E-Mail-System für Newsletter. E-Maibasierte Newsletter sehen bei Weitem nicht so gut aus und sind schwer zu pflegen. Es wurde daher beschlossen, den Newsletter auf die Intranetseite zu stellen und den Mitarbeitern nur eine E-Mail mit einem Link zu schicken, damit Sie den Newsletter im Browser ansehen können. Ein Gerüst dieses Newsletters sehen Sie in Abbildung 1.

Die neue Kommunikationdirektorin, die Erfahrung im Printbereich hat, möchte, dass der **Newsletter** mehr wie ein echter Newsletter aussieht – mit zwei Spalten statt einer.

Falls Sie jemals versucht haben, Text mit divs und floats in mehrere Spalten aufzuteilen, wissen Sie, wie schwer das sein kann. Die erste große Hürde besteht darin, dass Sie den Text von Hand aufteilen müssen. In DTP-Software wie zum Beispiel InDesign können Sie die Textfelder so miteinander verknüpfen, dass der Text vom ersten Feld automatisch in das nächste fließt, wenn das erste Textfeld vollständig mit Text gefüllt ist. Exakt so etwas gibt es für das Web noch nicht. Aber wir haben etwas, das wirklich sehr gut und einfach funktioniert. Wir können den Inhalt eines Elements auf mehrere gleich breite Spalten aufteilen.

Wir beginnen mit dem Markup für den Newsletter. Das **HTML** ist ziemlich einfach. Da sich der Inhalt ohnehin ändern wird, sobald wir es geschrieben haben, verwenden wir einfach Blindtext. Falls Sie sich fragen, warum wir nicht die neuen **HTML5-Markup-Elemente** wie zum Beispiel `section` für den Newsletter verwenden: Das liegt daran, dass unsere Ausweichlösung mit diesen Elementen nicht mit dem Internet Explorer kompatibel ist.

```
<body>
<div id="header">
<h1>HTML-Info Newsletter</h1>
<p>Volume 3, Issue 12</p>
```

```
</div>
<div id="newsletter">
<div id="director_news">
<div>
<h2>News From The Director</h2>
</div>
<div>
<p>Lorem ipsum dolor...</p>
<div class="callout">
<h4>Being HTML-Info</h4>
<p>"Lorem ipsum dolor sit amet..."</p>
</div>
<p>Duis aute irure...</p>
</div>
</div>
<div id="htmlinfo_bits">
<div>
<h2>Quick Bits of HTML-Info</h2>
</div>
<div>
<p>Lorem ipsum...</p>
<p>Duis aute irure...</p>
</div>
</div>
<div id="birthdays">
<div>
<h2>Birthdays</h2>
</div>
<div>
<p>Lorem ipsum dolor...</p>
<p>Duis aute irure...</p>
</div>
</div>
</div>
<div id="footer">
<h6>Send newsworthy things to <a href="mailto:news@html-info.eu">news@html-
info.eu</a></h6>
</div>
</body>
```

Um den Newsletter auf zwei Spalten aufzuteilen, müssen wir lediglich Folgendes in unser Stylesheet einfügen:

```
#newsletter{
```

```
-moz-column-count: 2;
-webkit-column-count: 2;
-moz-column-gap: 20px;
-webkit-column-gap: 20px;
-moz-column-rule: 1px solid #ddccb5;
-webkit-column-rule: 1px solid #ddccb5;
}
```

Wie Sie in Abbildung 2 erkennen, sieht unser Newsletter schon viel besser aus. Wenn wir zusätzlichen Inhalt einfügen, verteilt der Browser den Inhalt automatisch gleichmäßig auf die Spalten. Beachten Sie auch, wie die gefloateten Elemente in die Spalten fließen, in die sie gehören.

Joe fragt …

Kann ich für jede Spalte eine andere Breite angeben?

Nein, alle Spalten müssen dieselbe Breite haben. Ich war zunächst auch etwas überrascht, also habe ich die Spezifikation gründlich überprüft. Als ich dieses Buch geschrieben habe, war keine Möglichkeit vorgesehen, verschiedene Spaltenbreiten anzugeben.

Wenn man aber darüber nachdenkt, wie Spalten herkömmlicherweise verwendet werden, ist das auch sinnvoll. Spalten sind genauso wenig wie Tabellen als Trick gedacht, um schnell eine Seitenleiste zu basteln. Spalten sollen das Lesen von langen Textpassagen erleichtern, und gleich breite Spalten sind nun mal perfekt dafür.

Ausweichlösung

CSS3-Spalten funktionieren im Internet Explorer der Version 8 und davor leider nicht. Also verwenden wir den jQuery-Plugin Columnizer als Ausweichlösung. Mit Columnizer können wir unseren Inhalt mit dem folgenden einfachen Code gleichmäßig aufteilen:

```
$("#newsletter").columnize({ columns: 2 });
```

Menschen ohne JavaScript müssen mit einer Textspalte auskommen, können den Inhalt aber trotzdem lesen, weil wir ihn linear aufgebaut haben. Auch sie sind also versorgt. Allerdings können wir mit Java-Script die Unterstützung bestimmter Elemente durch den Browser ermitteln: Wenn wir eine existierende CSS-Eigenschaft abfragen, erhalten wir einen Leerstring. Erhalten wir dagegen einen Nullwert, ist die Eigenschaft nicht verfügbar.

```
<script charset="utf-8"
src='http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js'
```

```
type='text/javascript'> </script>
<script charset="utf-8" src="javascripts/autocolumn.js"
type='text/javascript'></script>
<script type="text/javascript">
function hasColumnSupport(){
var element = document.documentElement;
var style = element.style;
if (style){
return typeof style.columnCount == "string" ||
typeof style.MozColumnCount == "string" ||
typeof style.WebkitColumnCount == "string" ||
typeof style.KhtmlColumnCount == "string";
}
return null;
}
$(function(){
if(!hasColumnSupport()){
$("#newsletter").columnize({ columns: 2 });
}
});
</script>
```

Wir prüfen also zunächst, ob Spalten unterstützt werden. Falls nicht, wenden wir unser Plugin an.

Laden Sie die Seite im Internet Explorer neu, und der Newsletter wird zweispaltig angezeigt. Das Ergebnis ist zwar nicht perfekt (siehe Abbildung 3). Aber mit ein bisschen **CSS** oder **JavaScript** können Sie die Elemente zurechtbiegen, die noch nicht optimal aussehen. Diese Übung überlasse ich Ihnen.

Nutzen Sie bedingte Kommentare wie im Abschnitt JavaScript verwenden, um sich bei Bedarf bestimmte Versionen des Internet Explorer vorzuknöpfen.

Die Aufteilung von Inhalten auf mehrere Spalten kann diese besser lesbar machen. Sollte Ihre Seite aber länger sein, finden es Ihre Benutzer unter Umständen nervig, wieder ganz an den Anfang scrollen zu müssen. Setzen Sie diese Funktion daher mit Vorsicht ein.

Viele der neuen Elemente in **HTML5** helfen Ihnen dabei, Inhalte genauer zu beschreiben. Dies wird umso entscheidender, wenn andere Programme Ihren Code übersetzen. Manche Menschen verwenden zum Beispiel Bildschirmlesegeräte oder Screenreader, um sich die grafischen Inhalte des Bildschirms in Text übersetzen und laut vorlesen zu lassen. Screenreader interpretieren den Text auf dem Bildschirm und das entsprechende Markup, um Links, Bilder und andere Elemente zu erkennen. Diese Geräte haben erstaunliche Fortschritte gemacht, hinken aber immer ein bisschen hinter den neuesten Trends her. Live-Bereiche auf Seiten, in denen durch Polling oder Ajax-Anfragen der Inhalt verändert wird, sind schwierig zu

erkennen. Aufwendige Seiten sind teilweise schwierig zu navigieren, da der Screenreader eine Menge Text vorlesen muss.

„Accessibility for Rich Internet Applications (WIA-ARIA) ist eine Spezifikation zur Verbesserung der Barrierefreiheit von Webseiten, insbesondere von Webanwendungen. Sie ist besonders nützlich, wenn Sie Anwendungen mit JavaScript-Steuerelementen und Ajax entwickeln. Manche Teile der WIA-ARIA-Spezifikation wurden in HTML5 eingeführt, während andere außen vor bleiben und die **HTML5-Spezifikation** ergänzen können. Viele Bildschirmlesegeräte verwenden bereits Funktionen der WIA-ARIA-Spezifikation, dazu gehören JAWS, Window Eyes und sogar die integrierte Voice Over-Funktion von Apple. ARIA führt auch zusätzliches Markup ein, das unterstützende Technologien verwenden können, um aktualisierbare Bereiche zu erkennen.

In diesem Artikel sehen wir uns an, wie HTML5 die Erfahrung jener Besucher verbessern kann, die unterstützende Geräte verwenden.

Das Beste daran ist:

Für die in diesem Artikel beschriebenen Techniken benötigen wir keine Ausweichlösung, da viele Screenreader jetzt schon diese Technologien nutzen können.

Dazu gehören:

- Das **role**-Attribut `[<div role="document">]`
  Gibt die Aufgabe eines Elements für den Screenreader an. [C3, F3.6, S4, IE8, O9.6]
- **aria-live** `[<div aria-live="polite">]`
  Kennzeichnet einen Bereich, der automatisch aktualisiert wird, möglicherweise durch Ajax. [F3.6 (Windows), S4, IE8]
- **aria-atomic** `[<div aria-live="polite" aria-atomic="true">]`
  Gibt an, ob der gesamte Inhalt eines Live-Bereichs oder nur die veränderten Elemente vorgelesen werden sollen. [F3.6 (Windows), S4, IE8]

Die meisten Websites haben eine typische Struktur gemeinsam: Es gibt einen Kopfbereich, einen Navigationsbereich, den Hauptinhalt und eine Fußzeile. Genauso ist auch der Code der meisten Websites aufgebaut: `linear`. Leider bedeutet das, dass ein Bildschirmlesegerät unter Umständen die gesamte Website auch in dieser Reihenfolge vorlesen muss. Da sich auf den meisten Websites derselbe Kopfbereich und dieselbe Navigation auf jeder Seite wiederholen, müssen die Benutzer sich diese Elemente jedes Mal wieder anhören, um von einer Seite zur nächsten zu navigieren.

> Die bevorzugte Abhilfe besteht in einem versteckten Link „Navigation überspringen“,

den Screenreader laut vorlesen und der auf einen Anker im Hauptinhalt verweist. Allerdings ist diese Funktionalität nicht von Haus aus integriert – und nicht jeder weiß, wie das funktioniert, oder denkt überhaupt daran.

Mit dem neuen **HTML5-Attribut** `role` können wir jedem Element auf Ihrer Seite eine „Zuständigkeit“ zuweisen. Bildschirmlesegeräte können dann ganz einfach die Seite einlesen und diese Zuständigkeiten kategorisieren und einen einfachen Index für die Seite erstellen. So werden beispielsweise alle Elemente mit der Rolle `navigation` auf der Seite gesucht und den Benutzern erklärt, damit sie schnell durch die Anwendung navigieren können.

Diese Rollen stammen aus der WIA-ARIA-Spezifikation und wurden in die HTML5-Spezifikation integriert. Es gibt zwei bestimmte Rollenklassifizierungen, die Sie bereits jetzt einsetzen können: Landmark- und Dokument-Rollen.

Landmark-Rollen

Landmark-Rollen kennzeichnen „interessante Punkte“ auf Ihrer Website, wie etwa Banner, den Suchbereich oder die Navigation, die Screenreader schnell erkennen können.

Rolle	Verwendung
`banner`	Kennzeichnet den Bannerbereich Ihrer Seite.
`search`	Kennzeichnet den Suchbereich Ihrer Seite.
`navigation`	Kennzeichnet Navigationselemente auf Ihrer Seite.
`main`	Kennzeichnet die Stelle, an der der Hauptinhalt Ihrer Seite beginnt.
`contentinfo`	Gibt an, wo Informationen über den Inhalt vorhanden sind, zum Beispiel Copyright-Informationen und das Veröffentlichungsdatum.
`complementary`	Kennzeichnet Inhalte auf einer Seite, die den Hauptinhalt ergänzen, aber auch eine eigenständige Bedeutung haben.
`application`	Kennzeichnet einen Seitenbereich, der statt einem Webdokument eine Anwendung enthält.

Wir können einige dieser Rollen auf die Webseite-Vorlage anwenden, die wir im Abschnitt Eine Webseite mit semantischem Markup neu definieren erarbeitet haben.

Für den Kopfbereich vergeben wir die Rolle `banner`:

```
<header id="page_header" role="banner">
```

```
<h1>HTML-Info Blog!</h1>
</header>
```

Wir müssen lediglich `role="banner"` zu dem vorhandenen `header`-Tag hinzufügen.

Unsere Navigation können wir auf dieselbe Weise kennzeichnen:

```
<nav role="navigation">
<ul>
<li><a href="/">Latest Posts</a></li>
<li><a href="/archives">Archives</a></li>
<li><a href="/contributors">Contributors</a></li>
<li><a href="/contact">Contact Us</a></li>
</ul>
</nav>
```

Die HTML5-Spezifikation besagt, dass manche Elemente standardmäßige Rollen haben, die nicht überschrieben werden können. Das Element `nav` muss die Rolle `navigation` haben und daher technisch gesehen nicht damit gekennzeichnet werden. Nicht alle Bildschirmlesegeräte akzeptieren diesen Standard, aber viele verstehen die ARIA-Rollen.

Unsere Haupt- und Seitenleistenbereiche lassen sich wie folgt kennzeichnen:

```
<section id="posts" role="main">
</section>
<section id="sidebar" role="complementary">
<nav> <h3>Archives</h3> <ul>
<li><a href="2010/10">October 2010</a></li>
<li><a href="2010/09">September 2010</a></li>
<li><a href="2010/08">August 2010</a></li>
<li><a href="2010/07">July 2010</a></li>
<li><a href="2010/06">June 2010</a></li>
<li><a href="2010/05">May 2010</a></li>
<li><a href="2010/04">April 2010</a></li>
<li><a href="2010/03">March 2010</a></li>
<li><a href="2010/02">February 2010</a></li>
<li><a href="2010/01">January 2010</a></li>
</ul>
</nav>
</section>
<!-- Seitenleiste -->
```

Die Informationen zur Veröffentlichung und zum Copyright in unserem Fußbereich kennzeichnen wir mit der Rolle `contentinfo`:

```
<footer id="page_footer" role="contentinfo">
<p>© 2010 HTML-Info.</p>
</footer>
<!-- Fußzeile -->
```

Wenn es in unserem Blog eine Suchfunktion gäbe, könnten wir diesen Bereich ebenso kennzeichnen. Nachdem wir jetzt die Orientierungspunkte gekennzeichnet haben, gehen wir einen Schritt weiter und identifizieren einige der Dokumentelemente.

Rollen für die Dokumentstruktur

Die Rollen für die Dokumentstruktur helfen Screenreadern dabei, die Teile mit statischen Inhalten leichter zu erkennen, wodurch sich der Inhalt für die Navigation leichter strukturieren lässt.

Rolle	Verwendung
`document`	Kennzeichnet einen Bereich mit Dokumentinhalt, im Gegensatz zu Anwendungsinhalten.
`article`	Kennzeichnet eine Zusammenstellung, die einen unabhängigen Dokumentteil darstellt.
`definition`	Kennzeichnet die Definition eines Begriffs oder Themas.
`directory`	Kennzeichnet eine Liste von Referenzen auf eine Gruppe, wie etwa ein Inhaltsverzeichnis. Für statischen Inhalt.
`heading`	Kennzeichnet eine Überschrift für einen Abschnitt einer Seite.
`img`	Kennzeichnet einen Abschnitt, der Bildelemente enthält. Dabei kann es sich ebenso um Bildelemente wie um Beschriftungen und beschreibenden Text handeln.
`list`	Kennzeichnet eine Gruppe nicht interaktiver Listenelemente.
`listitem`	Kennzeichnet ein einzelnes Mitglied einer Gruppe nicht interaktiver Listenelemente.

Rolle	Verwendung
`math`	Kennzeichnet einen mathematischen Ausdruck.
`note`	Kennzeichnet Inhalt, der ausgeklammert ist oder den Hauptinhalt der Ressource ergänzt.
`presentation`	Kennzeichnet Inhalt, der der Darstellung dient und daher von unterstützenden Technologien ignoriert werden kann.
`row`	Kennzeichnet eine Zeile von Zellen in einem Gitter.
`rowheader`	Kennzeichnet eine Zelle mit Header-Informationen für eine Zeile in einem Gitter.

Viele Dokumentrollen sind implizit durch **HTML-Tags** definiert, wie etwa `article` und `header`. Allerdings ist die Rolle `document` nicht implizit definiert. Und das ist eine sehr wichtige Rolle, insbesondere für Anwendungen, in denen dynamischer und statischer Inhalt gemischt werden. So kann zum Beispiel in einem webbasierten E-Mail-Client dem Element, das den Body der E-Mail enthält, die Rolle `document` zugewiesen werden. Das ist nützlich, weil Bildschirmlesegeräte oft verschiedene Methoden für die Navigation mit der Tastatur anbieten. Wenn der Fokus des Screenreaders auf einem Anwendungselement liegt, müssen Tastenanschläge unter Umständen an die Webanwendung durchgereicht werden. Befindet sich der Fokus dagegen auf statischem Inhalt, können die Tastenbelegungen des Screenreaders andere Aufgaben übernehmen.

In unserem Blog können wir die Rolle `document` auf das `body`-Element anwenden:

```
<body role="document">
```

Joe fragt …

Brauchen wir diese Landmark-Rollen für Elemente wie `nav` und `header`?

Die Landmark-Rollen erscheinen im ersten Moment vielleicht redundant, aber sie bieten die Flexibilität, die Sie dann benötigen, wenn Sie die neuen Elemente nicht verwenden können.

Mit der Rolle `search` können Sie Ihre Benutzer nicht nur zu dem Seitenbereich führen, der das Suchfeld enthält, sondern auch zu Links auf eine Sitemap, zu einer Dropdown-Liste mit Quick-Links oder zu anderen Elementen, die Ihren Benutzern helfen können, Informationen schnell zu finden. Sie sind also nicht darauf beschränkt, Ihre Benutzer lediglich zum Suchfeld selbst zu führen.

Mit der Spezifikation werden eine Menge mehr Rollen eingeführt als neue Elemente und

Formularsteuerelemente.

Dies hilft, dafür zu sorgen, dass Bildschirmlesegeräte Ihre Seite als statischen Inhalt behandeln.

Ausweichlösung

Diese Rollen können bereits mit den neuesten Browsern und den neuesten Screenreadern verwendet werden. Sie können also ab sofort damit arbeiten. Browser, die die Rollen nicht unterstützen, ignorieren sie einfach. Sie können damit also auch nur den Menschen helfen, die sie verwenden können.