

9

Kapitel

Die Drag&Drop-API

Mit der Drag&Drop-API lassen sich Drag&Drop-Funktionen in HTML5 in standardisierter Weise ohne die Hilfe externer JavaScript-Bibliotheken wie jQuery oder MooTools umsetzen – man kann also im Browser Objekte mit der Maus durch die Gegend schieben und so mit Websites interagieren.

Das klingt erst einmal gut, die API hierfür stammt aber ursprünglich aus dem Internet Explorer 5 und gilt gemeinhin als missraten. Selbst HTML5-Editor Ian Hickson nahm das Wort „horrible“ in den Mund und gab zu, dass der vorrangige Grund für die Aufnahme dieser API in HTML5 schlicht der war, dass mit Safari und Mozilla bereits zwei Browser diese Technologie vom Internet Explorer übernommen haben.¹

Und ganz im Geiste der Standardisierung bestehender Techniken ist die Aufnahme der Drag&Drop-API in HTML geradezu zwingend², Qualität hin oder her.

¹ <http://twitter.com/Hixie/status/4075253361>

² Kritik zu diesem Vorgehen auch in Bezug auf Drag&Drop ist im Abschnitt 14.2.5 gesammelt.

Das Hauptproblem der API liegt darin, dass die diversen Funktionen und Events nicht das tun, was man aus der täglichen JavaScript-/DOM-Praxis kennt. Als Gesamtkunstwerk ist die API in sich aber durchaus schlüssig, es gibt nichts, was man nicht in den Griff bekommen könnte, und allen Marken zum Trotz ist der Drag&Drop-API von HTML5 eine Daseinsberechtigung nicht abzuspochen, da mit ihr Dinge möglich sind, die bei herkömmlichen, mit JavaScript programmierten Drag&Drop-Lösungen kaum machbar sind; und da sie bereits von Firefox, Safari, Chrome und Internet Explorer einigermaßen umfassend unterstützt wird, lohnt sich ein Blick allemal.

9.1 Die API im Überblick

Obwohl Microsoft die ursprüngliche Drag&Drop-API für den Internet Explorer erfunden hat, ist die Unterstützung im Sinne von HTML5 im Firefox am weitesten gediehen. Eigentlich sollte die API im IE alle Drag&Drop-Operationen managen, mit denen man markierten Text und Links umherziehen kann. Wenn man heute im Internet Explorer einen Link aus der Seite in die Adresszeile zieht, setzt man Funktionen eben dieser API ein, und weil Drag&Drop an sich ein angenehmes Feature ist, haben es die Macher von Mozilla und Webkit irgendwann adaptiert.

Nun wäre es für eine richtige Drag&Drop-Funktion etwas dürftig, wenn man nur Bilder und Text durch die Gegend ziehen könnte. Also erfand Ian Hickson spontan das HTML-Attribut `draggable`³, schrieb es in die HTML5-Spezifikationen und wartete darauf, dass Browserhersteller es einbauten. Bisher ist das nur im Firefox erfolgt, so dass er nun die Speerspitze der Drag&Drop-API-Implementierer darstellt, obwohl die Erfinder der API bei Microsoft sitzen. Seitens Webkit beschritt man einen ganz eigenen Weg und hielt es für eine gute Idee, den `draggable`-Status eines Elements *via* CSS festlegen zu können.⁴

Hinzu kommt, dass die Drag&Drop-API ausgeklügelte Formen des Datentransfers während der Drag&Drop-Operation möglich macht, was allein mit JavaScript-Bibliotheken so nicht möglich wäre (Details ab Seite 326). Hier bildet wiederum der IE in Sachen Unterstützung mit dem Firefox gemeinsam die Spitzengruppe, während es bei Webkit in den Details hakt. In Opera sind bisher keine Ansätze einer Umsetzung zu finden. Insgesamt ergibt sich also (Stand Anfang 2011, unverändert seit Anfang 2010) folgendes Bild:

³ Oder wie er es formulierte: „Yeah, I made up "draggable" because it seems there's no way to drag `_elements_` in IE, only selections.“ http://www.quirksmode.org/blog/archives/2009/09/the_html5_drag.html#c12540

⁴ Hier zeigt sich, dass das Monopol für seltsame Ideen in Sachen Webbrowser nicht bei Microsoft liegt. Irgendein Webkit-Programmierer muss der exklusiven Auffassung gewesen sein, es sei sinnvoll, das *Verhalten* von HTML-Elementen durch *Stylesheets* zu kontrollieren.

Funktion	Firefox 3.5 - 4.0	Safari 4 - 5	Chrome 3 - 8	Opera 10 - 11	IE 6-9
Drag & Drop für Links und Text	✓	✓	✓	-	✓
Drag & Drop für andere Elemente	✓	✓	✓	-	-
Unterstützung für Data Transfer	✓	z.T.	z.T.	-	✓

Tabelle 9.1:
Browser-Kompa-
tibilität zur HTML5-
Drag&Drop-API

Das sieht nun wieder nach Flickenteppich aus, und die API ist, wie gesagt, nicht einfach zu handhaben – dennoch hat sie ihre Berechtigung, denn in HTML5 kann Drag&Drop mehr als man vielleicht erwartet.

9.1.1 Anwendungsfälle für die API

Von einer Drag&Drop-Funktion in einem Betriebssystem auf dem Heimrechner erwartet man einiges: Sie soll erlauben, dass man Inhalte aus Programm A nach Programm B ziehen kann, selbst wenn weder A noch B den Programmierern des Systems bekannt waren. An eine Drag&Drop-Funktion im Web hat man wesentlich geringere Erwartungen und ist schon glücklich, wenn man sie innerhalb des Interface *einer* Website einigermaßen konsistent benutzen kann. Diese Erwartung kommt nicht von ungefähr, denn alles andere ist schwer bis gar nicht möglich – es sei denn, man nutzt die HTML5-Drag&Drop-API.

Mit einer standardisierten Drag&Drop-API werden Websites untereinander interoperabel. Website X könnte die Möglichkeit bereitstellen, JPG-Bilder per Drag&Drop irgendwohin zu ziehen, und Website Y für den Empfang von JPG-Bildern gerüstet sein; beide wären untereinander sofort kompatibel, ohne dass sich die Entwickler von X und Y hätten absprechen müssen. Auch wäre es möglich, Desktop- und Onlineapplikationen über diese Schnittstelle zu verzahnen, und statt einfacher JPGs ließen sich natürlich auch komplexere Datensätze verschieben, und zwar kreuz und quer über alle Browser. Aus Website X, die im Internet Explorer 8 angezeigt wird, ließen sich zum Beispiel Daten in den Firefox ziehen, wo Website Y wartet. In Zusammenarbeit mit der File API (vgl. Kapitel 10) ist es sogar möglich, Dateien aus dem Betriebssystem des Nutzers in das Browserfenster zu ziehen und dort zu verarbeiten. All das benötigt nur etwas (zugegebenermaßen konfuses) JavaScript.

Zwar lassen sich mit der Drag&Drop-API in HTML5 auch einfachere Operationen durchführen, nur ist das aufgrund der Natur der API nicht unbedingt ratsam. Wer die Browser-Browser- und Desktop-Browser Interoperabilität nicht braucht, sollte etwas anderes verwenden.

9.1.2 Anwendungsfälle für Alternativen

Mit einer modernen JavaScript-Bibliothek wie jQuery oder MooTools kann man mit wenigen Zeilen Code eine Drag&Drop-Anwendung schreiben, die in den meisten Browsern unterstützt wird. In MooTools sieht ein solches Skript, bei dem man das Element mit der ID `dragme` in eine beliebige Anzahl von Elementen mit der Klasse `dropme` ziehen kann und dann eine Erfolgsmeldung bekommt, beispielsweise wie folgt aus:

```
new Drag.Move('dragme', {
  droppables: '.dropme',
  onDrop: function(element, droppable, event){
    if(droppable){
      alert('Erfolg!');
    }
  }
});
```

Das sind 8 Zeilen JavaScript. Mit 8 Zeilen JavaScript ließen sich der HTML5-API noch nicht einmal eine vernünftige Fehlermeldung abringen. Derartig Simplex ist dort *ungleich* umständlicher zu realisieren, selbst wenn man die API in Kombination mit anderen JavaScript-Bibliotheken einsetzt. Zwar kommt man in Bezug auf die Ladezeit der Skripte in HTML5 in vielen Fällen mit ein paar Bytes weniger davon, da jQuery oder MooTools recht umfangreich sind – aber wann immer man einen Anwendungsfall hat, bei dem weder Interoperabilität noch Performancemaximierung um jeden Preis im Vordergrund stehen, fährt man mit anderen Lösungen besser. Andernfalls gilt: Ran an die API!

9.2 Eine erste einfache Drag-Anwendung

Um uns dem Potenzial der API schrittweise anzunähern, werden wir zunächst genau das tun, was wir nicht tun sollten: eine simple, lokale Drag&Drop-Anwendung bauen, die wir mit jeder JavaScript-Bibliothek einfacher erstellen könnten. Aber ganz ohne Grundlagen geht es nicht, und zudem ist es hilfreich, zu verstehen, *warum* Skripte von jQuery oder MooTools die bessere Wahl wären.

9.2.1 Ein Element ziehbar (draggable) machen

Das erste, was man für Drag&Drop braucht, ist etwas, das man ziehen kann. Mit der Tabelle aus Abschnitt 9.1 im Hinterkopf erinnern wir uns:

- Im Internet Explorer können nur Links und markierter Text gezogen werden.
- In Firefox (HTML-Attribut `draggable`) und Webkit-Browseren (CSS-Trick) können andere Elemente ziehbar gemacht werden.
- In Opera geht nichts von all dem.

Im Folgenden geht es zunächst um einen Anwendungsfall, der *nur* in Firefox und Webkit funktioniert, da wir ja das `draggable`-Attribut und den CSS-Hack kennenlernen wollen. Einen zum Internet Explorer kompatiblen Anwendungsfall spielen wir im Abschnitt 9.5 durch. Das folgende HTML5-Dokument ist die Ausgangsbasis:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Drag & Drop, Schritt 1</title>
    <style>
      body {
        padding:1em;
      }
      #log {
        border:1px solid #999;
        padding:1em;
        position:absolute;
        height:200px;
        overflow:auto;
        bottom:2em;
        left:2em;
        right:2em;
      }
      #dragme {
        display:block;
        padding:1em;
        width:8em;
        background:#CCC;
        border:0.125em solid black;
      }
      *[draggable=true] {
        -moz-user-select:none;
        -khtml-user-drag:element;
        cursor:move;
      }
    </style>
  </head>
  <body>
    <div id="log">
    </div>
    <div id="dragme">
    </div>
  </body>
</html>
```

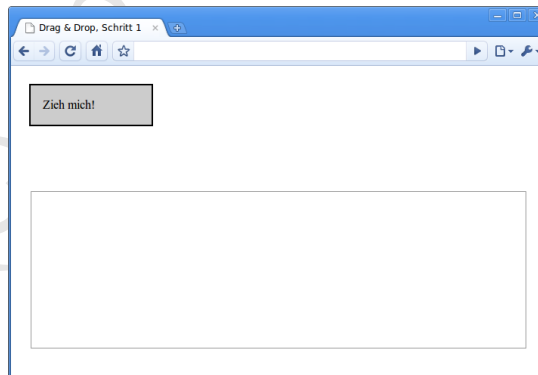
```

</style>
</head>
<body>
  <div id="dragme" draggable="true">Zieh mich!</div>
  <pre id="log"></pre>
  <script>
    // Elemente auswählen
    var dragme = document.getElementById('dragme');
    var logContainer = document.getElementById('log');
    // Log-Funktion
    function log(str){
      logContainer.innerHTML =
        str + '\n' + logContainer.innerHTML;
    }
  </script>
</body>
</html>

```

Ein gewöhnliches HTML5-Dokument mit Styles und einem Skript-Block am Ende – das `<pre>` nebst `log()`-Funktion ist zwar in Zeiten komfortabler Debugging-Konsolen etwas antiquiert, doch wir haben es mit archaischen Internet-Explorer-Versionen zu tun, die eine manuelle Lösung erfordern.

Abbildung 9.1:
Das Gerüst: Ein
Zielelement und der
Container für das
Log



In diesem Dokument wird die Drag&Drop-API zum Einsatz kommen, doch zuvor lohnt sich ein Blick auf die für Drag&Drop relevanten Details von HTML und CSS in unserem Beispiel.

Der HTML-Code im Detail

Das `<div>` mit der ID `dragme` ist unser Versuchsobjekt und wird durch die Vergabe des `draggable`-Attributs für Firefox 3.5+ und im Sinne der HTML5-Spezifikationen als ziehbares Objekt ausgezeichnet. Vorsicht ist geboten, weil `draggable` *kein* boolesches Attribut (wie etwa `checked` bei Checkbo-

zen) ist, sondern immer einen Wert haben muss. Mit `true` wird das Element als ziehbar markiert, mit `false` als nicht ziehbar; fehlt das `draggable`-Attribut, gelten für das Element die Standardregeln: Wenn das Element ein ``-Element oder ein `<a>`-Element mit `href`-Attribut ist, ist es ziehbar, andernfalls nicht. Das `<pre>`-Element dient uns als Logging-Konsole.

Die CSS-Tricks im Detail

Im CSS wird nicht nur das Element `#dragme` gestaltet, es werden auch drei CSS-Regeln für alle Elemente festgelegt, bei denen `draggable` auf `true` steht. Während die `cursor`-Regel nur für einen passenden Mauszeiger sorgt, sind die beiden anderen Kniffe für Firefox und Webkit-Browser gedacht.

Durch die Angabe `-moz-user-select:none`; wird im Firefox verhindert, dass der im Element stehende Text markiert werden kann. Das ist insofern sinnvoll, als die Drag&Drop-API mit dem IE 5 unter anderem explizit für das Ziehen von markiertem Text eingeführt wurde. Da wir es aber in diesem Fall nur auf das `<div>` als Element abgesehen haben und wir uns Konfusion ersparen wollen, kommt uns `-moz-user-select:none`; gelegen. Hingegen ist `-khtml-user-drag:element`; jener ominöse CSS-Trick, mit dem man Webkit-Browsern mitteilt, dass man gewisse Elemente ziehbar machen würde. Es ist quasi `draggable="true"` für Webkit in CSS-Form.

Das JavaScript im Detail

Unabhängig davon, ob und wie sinnvoll es ist, das Verhalten einer Website in den Stylesheets festzulegen, widmen wir uns nun der JavaScript-API. Als Hilfsmittel enthält das Beispiel die JavaScript-Funktion `log()`, mit der alle wichtigen Ereignisse aufgezeichnet werden können. Außerdem werden die Elemente `dragme` und `log` in Variablen gespeichert. Mit diesem Rüstzeug wagen wir uns an die API.

9.2.2 Drag-Events

Es gibt insgesamt sieben DOM-Events im Zusammenhang mit Drag&Drop, von denen sich drei auf das Element beziehen, das durch die Gegend geschoben wird. Die vier anderen sind für potenzielle Drop-Ziele zuständig. Einige dieser sieben Events verhalten sich etwas ungewohnt, denn ihre Standardfunktion besteht darin, Drag&Drop-Operationen *zu verhindern*. Richtig nutzbar macht man Drag&Drop also, indem man besagte Events so programmiert, dass sie sich selbst abbrechen, sobald sie stattfinden.⁵

⁵ Diese Konstruktion ist einer der Gründe, warum Ian Hickson die API „horrible“ findet und warum Browser-Guru Peter-Paul Koch (<http://www.quirksmode.org>) die Beschreibung „a steaming pile of bovine manure“ wählte.

Damit wir überhaupt dazu kommen, einen dieser Sonderlinge verwenden zu dürfen, brauchen wir zunächst ein gewöhnliches Exemplar dieser Gattung, mit dem wir eine Drag&Drop-Operation beginnen lassen können: `dragstart`.⁶

`dragstart` – Der Startschuss

Ein Element, das jenseits seiner Standardeigenschaften ziehbar sein soll (zur Erinnerung: alle `` und `<a>` mit `href`-Attribut sind – jedenfalls theoretisch – immer ziehbar, falls sie nicht explizit einen `draggable`-Wert `false` haben), braucht ein `dragstart`-Event, damit die Drag&Drop-Operation beginnen kann. Darin lässt sich der Datensatz festlegen, der mit diesem Element verknüpft ist – ohne Daten gibt es in HTML5 kein Drag&Drop.⁷

Datensätze für Drag&Drop-Operationen werden in einem `dataTransfer`-Objekt gespeichert, das Teil eines jeden Drag&Drop-Events ist. Was man damit anstellen kann, ist Gegenstand des Abschnitts 9.5 – für den Moment begnügen wir uns damit, dass wir eines brauchen und es mit Daten füttern müssen: `event.dataTransfer.setData('Text', 'test');` übernimmt das für uns.

Außerdem muss man dem `dataTransfer`-Objekt einen oder mehrere zulässige „Drag-Effekte“ mitgeben. Diese Effekte sind eigentlich gar keine, sondern dienen der Angabe verschiedener Arbeitsmodi für die API.⁸

Einer dieser Effekte/Modi ist `move`, das für unser einfaches Hin- und Herschiebeskript die richtige Wahl ist. Wir schreiben also das Folgende:

```
// Drag-Start-Event, setzt das dataTransfer-Objekt
dragme.ondragstart = function(event){
    event.dataTransfer.setData('Text', 'test');
    event.dataTransfer.effectAllowed = 'move';
};
```

Schon kann man das Objekt durch die Gegend ziehen! Aufgrund der Darstellung einer Drag&Drop-Operation in Chrome (die sich durch Unsichtbarkeit auszeichnet), sieht man das hier nur anhand des Cursors – in allen anderen Browsern hingegen erkennt man ein teiltransparentes Klon-Objekt. Um uns vom Funktionieren des o.g. Codes auch in Chrome zu überzeugen, bietet es sich an, die Drag&Drop-Operation zu loggen.

⁶ Zumindest im Firefox wird `dragstart` benötigt, und auch die HTML5-Spezifikationen sehen `dragstart` vor. Dass das Drag&Drop in Webkit auch ohne `dragstart` funktionieren würde, übersehen wir an dieser Stelle – Webstandards gehen vor.

⁷ Kleine Einschränkung der Vollständigkeit halber: In Webkit-Browsern ist das Datentransfermodell defekt, weshalb dort wohl auch Drag&Drop ohne `dragstart` funktioniert.

⁸ Ursprünglich legten diese „Effekte“ fest, welcher Cursor während der Operation zu sehen war. Im IE5 funktioniert das auch noch genau so, hat aber für moderne Browser keine Bedeutung.

drag und dragend – Bewegungsmelder und Zielflagge

Das drag-Event verhält sich wie das altbekannte mousemove, feuert also permanent, solange ein Element hin- und hergeschoben wird. Das lässt sich in Kombination mit unserem Logger zur Beweisführung einer funktionierenden Drag&Drop-Operation nutzen:

```
// Drag-Bewegungsmelder
dragme.ondrag = function(event){
    log('Operation läuft...');
};
```

Prompt füllt sich das Logbuch mit Einträgen. An dieser Stelle würde es sich anbieten, das dragstart-Event mit einem Log-Eintrag auszustatten, so dass sich folgender Code ergibt:

```
// Drag-Start-Event, setzt das dataTransfer-Objekt
dragme.ondragstart = function(event){
    event.dataTransfer.setData('Text', 'test');
    event.dataTransfer.effectAllowed = 'move';
    log('Beginne Drag-&-Drop-Operation!');
};
```

Schließlich gibt es mit dragend auch ein Event für das Ende von Drag-Operationen, das wir hier nur nutzen, um ins Logbuch zu schreiben:

```
// Drag-End-Event, meldet das Ende jeder Drag-Operation
dragme.ondragend = function(event){
    log('Operation beendet!');
};
```

Fertig ist ein einfaches Beispiel für die Benutzung der Drag&Drop-API in HTML5. Wenn wir unser Zielelement jetzt ein wenig hin- und herschieben, erkennen wir (neben dem visuellen Feedback in Firefox und Safari), wie sich das Log füllt:

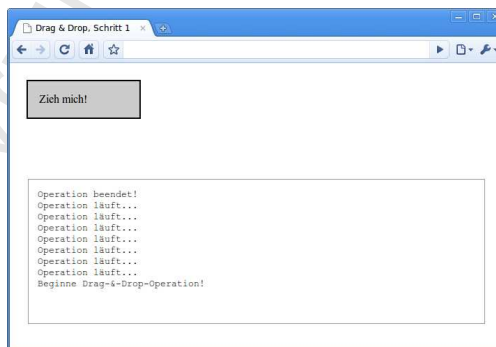


Abbildung 9.2:
Viele Zeilen im
Logbuch zeigen, dass
sich tatsächlich
etwas tut – sonst
sieht man (zumindest
in Chrome und IE)
nichts

Manuelles Drag&Drop-Feedback

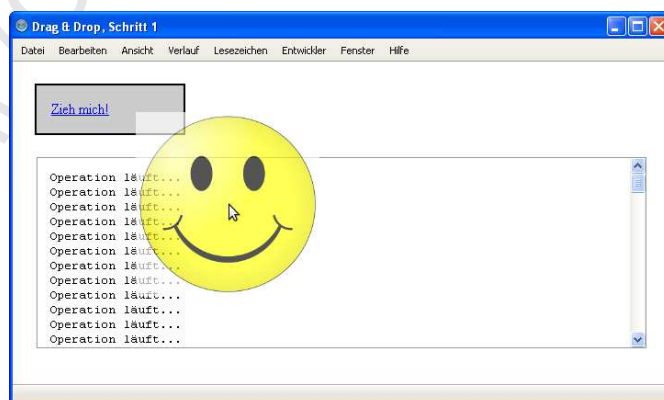
Diese unsichtbare Drag&Drop-Operation ist in Sachen Benutzerfreundlichkeit wenig überzeugend, und leider lässt sich das Problem zur Zeit in Chrome und IE nicht beheben. Allerdings stellt HTML5 zwei Methoden für das Feintuning des Drag&Drop-Feedbacks zur Verfügung, von denen eine bereits in Firefox, Chrome und Safari, die andere nur in Firefox unterstützt wird. Dabei hat man die Wahl, entweder ein Bild oder ein beliebiges anderes DOM-Element anzugeben, das während der Drag&Drop-Operation dem Cursor folgt.

Die Methode `setDragImage(element, x, y)` ist für den Einsatz von Grafiken gedacht. Als `element` ist ein ``-Element anzugeben, das, um `x` und `y` versetzt, hinter dem Mauszeiger erscheint. Angenommen wir hätten ein `200×200` Pixel großes Bild, das wir während der Operation mittig hinter unserem Cursor platzieren wollen, ginge das so:

```
// Drag-Feedback-Element
var img = new Image();
img.src = 'test.png';

// Drag-Start-Event, setzt das dataTransfer-Objekt
dragme.ondragstart = function(event){
    event.dataTransfer.setData('Text', 'test');
    event.dataTransfer.effectAllowed = 'move';
    event.dataTransfer.setDragImage(img, 100, 100);
    log('Beginne Drag-&-Drop-Operation!');
};
```

Abbildung 9.3:
Smiley als
Drag-Feedback
anstelle eines
automatisch
generierten Abbildes
des gezogenen
Objekts (funktioniert
in Firefox, Chrome
und Safari)



Die `addElement(element)`-Methode funktioniert bislang nur im Firefox. Sie zeigt auch ein Bild, generiert dieses aber aus der Darstellung eines DOM-Objekts im Browser:

```
// Drag-Feedback-Element
var el = document.createElement('span');
el.innerHTML = 'Drag & Drop läuft!';
document.body.appendChild(el);

// Drag-Start-Event, setzt das dataTransfer-Objekt
dragme.ondragstart = function(event){
    event.dataTransfer.setData('Text', 'test!');
    event.dataTransfer.effectAllowed = 'move';
    event.dataTransfer.addElement(el);
    log('Beginne Drag-&-Drop-Operation!');
};
```

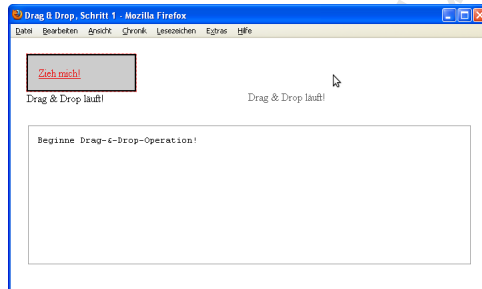


Abbildung 9.4:
Ein `` als
Drag-Feedback (links
unter dem
gezogenen Element
ist das Quell-Element
zu sehen)

Im Laufe einer Drag&Drop-Operation können die Drag-Feedback-Elemente modifiziert werden. Wenn es in den folgenden Abschnitten um die Programmierung von Drop-Zielen geht, wäre es eine Option, die Feedback-Elemente zu ändern, wenn zum Beispiel ein Element in eine Drop-Zone gezogen wird – das `src` eines Bildes oder die Klasse des Elements könnten ausgetauscht werden.

Das war nun relativ viel Arbeit für wenig Ertrag. Wir haben 20 Zeilen JavaScript geschrieben und die Hälfte von dem erreicht, was das MooTools-Codebeispiel auf Seite 306 leistet. Freilich könnte man Zeilen einsparen, wenn man für die API eine kleine Abstraktionsschicht schriebe, aber sonst bleibt es dabei, dass die Drag&Drop-API von HTML5 für so simple Aufgaben zu komplex ist. Kommen wir also zum „Drop“-Teil von Drag&Drop und damit zu den gewöhnungsbedürftigsten Aspekten der API.

9.3 Drag mit Drop

Eine Zone, in die ein Drag-Element hineingezogen werden kann, ist in HTML5 einfach ein weiteres Element mit ein wenig JavaScript. Für unser Beispiel bietet sich dafür ein weiteres `<div>`-Element an:

```
<div id="dropme">Hier hin!</div>
```