

Web 2.0 mit ASP.NET 4.0 und AJAX

Praxiseinstieg in die Erstellung
interaktiver Websites



AJAX

AJAX ist ein Synonym für *Asynchronous JavaScript and XML*, mithin also für die Kombination dreier Techniken, genauer, einer Technologie (asynchrone Datenübertragung), einer Scriptsprache (JavaScript implizit DOM) sowie einer deklarativen Formatsprache (XML – Extensible Markup Language). Eigentlich könnte AJAX auch »AJAXON« heißen, kann beim Einsatz von AJAX alternativ zu XML auch JSON als JavaScript-basiertes kompakteres Rückgabeformat verwendet werden (siehe Abschnitt 1.4). Aber der Reihe nach!

Zurückzuführen ist der Begriff AJAX auf Jesse James Garrets im Jahre 2005 verfassten Artikel »AJAX: A New Approach to Web Application« (<http://www.adaptivepath.com/publications/essays/archives/000385.php>), in dem sich Garret wie folgt äußert:

I needed something shorter than »Asynchronous JavaScript+CSS+DOM+XMLHttpRequest« to use when discussing this approach with clients.

Für die Diskussionen mit seiner Kundschaft benötigte Garret demnach etwas Kürzeres als Asynchronous JavaScript+CSS+DOM+XMLHttpRequest. Heraus kam AJAX – und ein Hype, der bis heute ungebrochen ist.

Garrets Zitat verdeutlicht aber auch, dass bei AJAX weniger die Technologie der asynchronen Datenübertragung an sich entscheidend ist (gleichwohl ist sie Voraussetzung für das grundsätzliche Funktionieren von AJAX), noch die verwendeten Sprachen. Erst das Zusammenspiel gerade *dieser* Sprachen mit eben *dieser* Technologie ermöglicht jene AJAX-spezifischen, optischen Effekte, von denen in den nächsten Kapiteln – im Besonderen im Praxisteil des Buches – einige vorgestellt werden sollen.

Dennoch sollte nicht unerwähnt bleiben, dass die hinter AJAX stehende Technologie deutlich älter ist als die Begrifflichkeit selbst. Beispielsweise verfügt Microsofts **Exchange** (Outlook Web Access), Release 5.5, bereits über vergleichbare technologische Konzepte. Als weiteres Beispiel sei der **Web Messenger** (ebenfalls ein Produkt der Firma Microsoft) genannt. Daneben existieren weitere Ansätze einer asynchronen Kommunikation zwischen Client und Server, von denen sich jedoch bislang kein einziger durchsetzen konnte. Es scheint, als bliebe AJAX – zumindest vorerst – das Maß aller »asynchronen Dinge«.

1.1 AJAX aus optischer Sicht

Internetseiten, die eindrucksvoll die visuellen Möglichkeiten von AJAX dokumentieren, gibt es zwischenzeitlich viele. Eine sehr spezielle und nicht minder populäre AJAX-Anwendung stellen die Google Maps unter <http://maps.google.de/> dar. Wie war es bislang in der »Online-Kartografie«? Sie klickten auf einen der vier den Kartenausschnitt umgebenden Pfeile, die Karte verschwand und eine neue wurde geladen. Desgleichen, wenn Sie den Ausschnitt vergrößern oder verkleinern wollten. Nicht so bei den Google Maps: Hier können Sie ohne Reload der Seite in die Karte hineinzoomen und sie bequem via Drag&Drop verschieben. Es muss allerdings nicht immer Google sein. Zahlreiche, der in den meisten kommunalen Webauftritten integrierten Stadtpläne wurden zwischenzeitlich auf Grundlage von AJAX realisiert (siehe Abbildung 1.1).



Abb. 1.1: »AJAX-getunter« Stadtplan unter <http://www.siegen.de>

Suchen Sie dagegen Seiten, die vollständig auf AJAX basieren, werden Sie an dem Dashboard Netvibes unter <http://www.netvibes.com/> nicht vorbeikommen. Die Seite bietet Features, die ohne AJAX kaum oder gar nicht zu realisieren wären. Netvibes ist AJAX pur, und ein exzellentes Beispiel für die Web2.o-Philosophie. So

können Sie – via Drag&Drop – die Komponenten der Startseite festlegen, deren Reihenfolge verändern und einzelne Elemente wieder löschen. Ebenso sind Sie es, der bestimmt, welche Newsfeeds angezeigt werden sollen. Selbst der Header der Seite kann auf intuitive Art geändert werden: Mauszeiger auf die Überschrift bewegen, Klick mit der linken Maustaste und einfach in das aktivierte Eingabefeld schreiben.

Blenden Sie (z.B. im Internet Explorer) Standardschaltflächen, Adress- und Statusleiste aus, und nur noch wenig erinnert Sie daran, mit einer Internetseite beschäftigt zu sein.

Hinweis

Durch AJAX wird das Submit-Reload-Konzept klassischer Webseiten aufgehoben, das heißt, Änderungen am Seiteninhalt führen nicht dazu, dass eine vollständig neue Seite, mit einem entsprechend neuen URL (Uniform Resource Locator – zu Deutsch: einheitlicher Quellenordner) jedes Mal nachgeladen werden muss. Unabhängig vom verwendeten Browser folgt daraus, dass sowohl die Lesezeichen-Funktion (Bookmark) als auch der ZURÜCK-Button nicht mehr funktionieren (siehe auch Abschnitt 1.6).

Es bleibt abzuwarten, wann es zu einer diesbezüglichen Anpassung der Browser kommt. Lösungsansätze sind aber bereits vorhanden. Suchanfragen zum Beispiel sollen innerhalb unsichtbarer Inline Frames (IFrames) ausgeführt werden, was zu einem Eintrag in der History des Browsers führt. Damit würde die ZURÜCK-Schaltfläche wieder funktionieren.

Bleiben wir noch einen Augenblick beim Browser. Sie kennen die Autovervollständigen-Funktion der Adressleiste. Selbst dieses Feature lässt sich in die Onlinewelt übertragen – aber eben nur unter Zuhilfenahme von AJAX. Stellen Sie sich »Autovervollständigen« nach klassischem Reloadingschema vor! Die manuelle Eingabe selbst längerer Wörter und Wortkombinationen würde Ihnen möglicherweise schnell sympathischer sein.

Dem Begriff Onlineanwendung am ehesten Rechnung trägt jedoch die Seite <http://www.glify.com>, auf der Sie wunderbar intuitiv Flow Charts, Floor Plans, Diagramme und einiges mehr erstellen können. Einfach die gewünschten Objekte mit der Maus auf die Arbeitsfläche ziehen (die Objekte werden dabei automatisch skaliert). So gelangen Sie Stück für Stück zu einer höchst ansprechenden Darstellung dessen, was auch immer Sie schematisch darstellen wollen.



Abb. 1.2: »Autovervollständigen« bei YouTube (<http://www.youtube.com>)



Abb. 1.3: Diagrammerstellung online und mit Genuss: gifly

1.2 AJAX aus technischer Sicht

Bei AJAX wird die Anforderung über ein zwischengeschaltetes JavaScript-Programm ausgeführt, woraus sich eine Abkopplung von der eigentlichen Seitenanfrage ergibt. Es werden also nicht mehr die in einer Seite enthaltenen Informationen vollständig an den Server übermittelt, sondern lediglich der für die Anforderung relevante Teil. Somit entfällt das Senden der Statusdaten (die durchaus über 100 Kilobyte groß werden können). Die Seite wird einmal aufgebaut, und dann lediglich an den Stellen aktualisiert, wo Inhalte tatsächlich variieren sollen. So viel zum technischen Prinzip.

Wie aber läuft ein AJAX-gestütztes Programm im Detail ab?

1.2.1 Ablaufverfolgung

Unter Zuhilfenahme des `XmlHttpRequest-Object` stellt JavaScript eine Serveranfrage. Im Falle von originärem AJAX bezieht sich die auf eine XML-Datei. Mittels HTTP-Protokoll wird die XML-Datei vom Server an den Browser zurückgesendet und von JavaScript empfangen. Mit Hilfe des `Document Object Moduls (DOM)`, greift JavaScript auf die Inhalte der XML-Datei zu, um die XML-Markups auszulesen. Ebenfalls via JavaScript und des DOMs wird auf das aktuell im Browser befindliche HTML-Dokument zugegriffen, in dem die Daten der XML-Datei an den relevanten Stellen eingefügt werden. Die Änderungen werden ohne Neuaufbau der Seite angezeigt.

Hinweis

Auch der Begriff AJAX birgt Missverständnisse, wird doch der zwingende Einsatz von XML suggeriert. Wie bereits erwähnt, kann bei der Verwendung von asynchronem JavaScript aber auf XML zugunsten des mit weniger Overhead (Verwaltungsdaten) ausgestatteten JSON verzichtet werden. Aber nicht nur das: Selbst PHP- oder ASPX-Dateien können von JavaScript/DOM ausgelesen werden, zumindest dann, wenn sie entsprechende Formatelemente enthalten. Entscheidend ist, was zurückgegeben werden soll. Im Falle von einigen Zeilen Text wäre XML als Format der Rückgabe ebenso übertrieben wie JSON.

Bis hierhin sind Sie bereits mehrfach auf den Begriff »asynchron« gestoßen. Im Kontext der Kommunikation zwischen Server und Client werden die Begriffe synchron und asynchron diametral anders verwendet, als es die lexikalische Definition vorsieht.

1.2.2 Synchroner Kommunikation

Zunächst zur synchronen Kommunikation. Die beteiligten Prozesse (Kommunikationspartner) synchronisieren in dem Sinne beim Senden oder Empfangen von Daten, als dass jeweils ein Prozess so lange blockiert, bis der andere Prozess abgeschlossen ist. Handelt es sich bei der Daten versendenden Stelle beispielsweise um den Client, verbleibt dort solange der beteiligte Prozess in Wartestellung, wie es nicht zu einer Empfangsbestätigung durch die Serverseite gekommen ist.

1.2.3 Asynchrone Kommunikation

Anders sieht die Sache bei der asynchronen Datenübertragung, sprich asynchronen Kommunikation aus. Hier erfolgt das Empfangen und Senden von Daten zeitlich versetzt, ohne Blockieren der beteiligten Prozesse. Die Datenübertragung erfolgt also im Hintergrund, ohne dass der Benutzer einer Webseite vom Transfer etwas mitbekommt. Erst dadurch ist es möglich, eine Seite partiell durch spezifische Anfragen an den Server zu aktualisieren. Damit reduziert sich der Zeitaufwand merklich, der für gewöhnlich bei der Nutzung einer Webanwendung anfällt.

1.2.4 Das Herzstück von AJAX – XMLHttpRequest-Object

Würden die an AJAX beteiligten Sprachen und Technologien hierarchisch angeordnet, stünde das XMLHttpRequest-Object ganz oben. Denn zunächst ungeachtet, ob die Datenübertragung synchron oder asynchron verläuft, bedarf es erst einmal einer Möglichkeit, per JavaScript HTTP-Anfragen stellen zu können. Diese Aufgabe übernimmt die Programmierschnittstelle XMLHttpRequest-Object. Konkret gesagt: Bevor es zu einer Kommunikation zwischen Client und Server kommen kann, muss eine XMLHttpRequest-Instanz erzeugt werden. Im Falle, dass im Browser XMLHttpRequest als natives Objekt implementiert ist, geschieht dies über:

```
new XMLHttpRequest-Object()
```

Die Instanziierung von XMLHttpRequest ist die Stelle, wo browserabhängiger Code geschrieben werden muss (und glücklicherweise, zumindest im Normalfall, die einzige). Bevor auf die Codierung eingegangen wird, zunächst ein kurzer Blick in die Historie beziehungsweise die Verfügbarkeit des XMLHttpRequest-Object in den einzelnen Browsern.

Ursprünglich wurde das XMLHttpRequest-Object von Microsoft als ActiveX-Komponente für den Internet Explorer 5 implementiert. Genauer existieren gleich mehrere ActiveX-Komponenten, von denen allerdings nur zwei für AJAX relevant sind: Msxml2.XMLHTTP, zu erzeugen über

```
new ActiveXObject("Msxml2.XMLHTTP"),
```

und das ältere `Microsoft.XMLHTTP`, zu instanziiieren via

```
new ActiveXObject("Microsoft.XMLHTTP").
```

Seit Version 7 des Internet Explorers steht das `XMLHttpRequest-Object` auch als »natürliches« Objekt zur Verfügung. Die Instanziierung erfolgt durch:

```
new XMLHttpRequest()
```

In den anderen Browsern wurde das `XMLHttpRequest-Object` bereits früher implementiert. Einen Überblick gibt nachstehende Tabelle:

Browser	XMLHttpRequest-Object verfügbar ab Version
Internet Explorer	7.0
Netscape	7.0
Mozilla	1.0
Firefox	1.0
Safari	1.2
Camino	1.0
Konquereor	3.2
SeaMonkey	1.0
Opera	7.6

Tabelle 1.1: Verfügbarkeit des `XMLHttpRequest-Objects` in unterschiedlichen Browsern

Detaillierte Informationen zu den Schnittstellen der diversen `XMLHttpRequest-Implementierungen` finden sie auf den Webpräsenzen der Browserhersteller.

Browserabhängigkeit versus JavaScript

Seien Sie nicht enttäuscht wegen der Browserabhängigkeit des `XMLHttpRequest-Object`, denn diese kann durch wenige Zeilen JavaScript-Code auf unkomplizierte Weise relativiert werden. Dazu wird die Ergebnisinstanz in einer Variablen mit dem Namen `anfrage` gespeichert. Eine `try-catch`-Anweisung hilft uns (indirekt) festzustellen, um welchen Browser es sich handelt beziehungsweise welche Art der Erstellung verwendet werden muss.

```
<head>
<script type="text/javascript">
```

```
/*<![CDATA[*]  
//XMLHttpRequest-Object erstellen  
var anfrage = null;  
  
try  
{  
    anfrage = new XMLHttpRequest();  
}  
catch (e)  
{  
    try  
    {  
        anfrage = new ActiveXObject("Msxml2.XMLHTTP");  
    }  
    catch (e)  
    {  
        try  
        {  
            anfrage = new ActiveXObject("Microsoft.XMLHTTP");  
        }  
        catch (e)  
        {  
            anfrage = null;  
        }  
    }  
}  
}  
/*]]>*/  
</script>  
</head>
```

Listing 1.1: Erzeugung einer Instanz von XMLHttpRequest-Object über eine try-catch-Anweisung

Zunächst wird versucht, über den Konstruktor XMLHttpRequest() eine Instanz zu erstellen (try()). Schlägt der Versuch fehl (beispielsweise weil das Objekt in älteren Versionen des Internet Explorers nicht existiert), wird der aufgetretene Fehler in Richtung des ersten catch »geworfen«. Dort geht es um die »ActiveX-Variante« des XMLHttpRequest-Object. Gelingt auch hier die Bildung einer Instanz nicht, erfolgt ein »Wurf« zur zweiten catch-Direktive. Da insgesamt drei Möglichkeiten der Erstellung einer Instanz existieren, muss in der Regel spätestens hier die Instanziierung erfolgen. Ist dies nicht der Fall, greift das dritte und letzte catch, und es wird statt einer Objektinstanz lediglich null in der Variablen anfrage gespeichert.

Methoden und Eigenschaften in XMLHttpRequest

Nun wissen Sie, auf welchem Wege eine Instanz des Objekts XMLHttpRequest erstellt wird, ferner wie die Abhängigkeit der Programmierschnittstelle vom verwendeten Browser teilweise aufgehoben werden kann.

Kommen wir zu den im Objekt XMLHttpRequest verfügbaren Methoden und Eigenschaften. Für die auf Basis von AJAX erstellten Anwendungen sind zunächst nur drei relevant:

- Zur Initialisierung einer Verbindung die Methode `open()`. Diese erwartet drei Parameter:
 - Die Methode der Übertragung
 - Den Namen der angefragten Datei
 - Ob die Übertragung asynchron (`true`) oder synchron (`false`) erfolgen soll. Wird der dritte Parameter weggelassen, werden die Daten, wie bei AJAX prinzipiell üblich, asynchron übertragen (wenn Sie zu den »Puristen« unter den Webentwicklern zählen, wollen Sie vermutlich auf `true` dennoch nicht verzichten).
- Um auf Zustandsänderungen innerhalb der Anfrage reagieren zu können, existiert die Eigenschaft `onreadystatechange`, der als Wert der Name der aufzurufenden Funktion übergeben wird. Bitte beachten Sie: Bei der asynchronen Kommunikation informiert der Server den Client ständig über Zustandsänderungen, von denen es bis zur vollständigen Abarbeitung der Anfrage gleich mehrere gibt.
- Zum tatsächlichen Senden der Anfrage nehmen Sie die Methode `send()`. Als Parameter erwartet `send()` die zu sendenden Daten. Im Falle, dass als Übertragungsmethode GET zum Einsatz kommt, wird als Parameter `null` angegeben, da die Daten an den URL angehängt werden.

1.3 Ein Beispiel – Auswahlbestätigung

Anhand eines kleinen Beispiels soll das grundsätzliche Wirkprinzip von AJAX sowohl auf der programmiertechnischen als auch der optischen Ebene gezeigt werden.

Die Anforderung an unseren exemplarischen Code ist denkbar einfach: In einer Dropdown-Liste wird ein Eintrag ausgewählt und – ergänzt um einen kurzen Begleitsatz – bestätigt.

Auf nachstehendes HTML-Listing zur Darstellung eines minimalistischen Formulars soll hier nicht näher eingegangen werden. Nur so viel: Erstmalig werden Sie mit einer ASPX-Seite (siehe Anhang A ASP.NET) in Kontakt kommen, denn die

Verarbeitung der Auswahl erfolgt in der Seite `formularausgabe.aspx`. Im folgenden HTML-Listing entdecken Sie die Datei unschwer wieder:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Eingabebestätigung</title>
</head>
<body>
  <label style="font-weight: bold; font-size: 20px">
    Welchen Eindruck haben Sie bisher von diesem Buch?
  </label>
  <br/>
  <br/>
  <form action="formularausgabe.aspx" method="get">
    <select name="eindruck" size="2">
      <option value="informativ" selected="selected" >Informativ</option>
      <option value="langatmig">Langatmig</option>
      <option value="praxisbezogen">Praxisbezogen</option>
      <option value="theorieelastig">Theorieelastig</option>
    </select>
    <br/>
    <br/>
    <input type="submit" value="Abschicken" />
  </form>
</body>
</html>
```

Listing 1.2: Die Datei `formulareingabe.html`

Klassische Formularverarbeitung mit ASP.NET

Um den Unterschied zwischen AJAX-gestützter und klassischer Formularverarbeitung zu verdeutlichen, soll es im Weiteren zunächst um die klassische Formularverarbeitung mit ASP.NET gehen. In diesem Falle wird über den SUBMIT-Schalter (Button ABSCHICKEN) unser Formular an die im `<Form>`-Tag dem Attribut `action` übergebene Datei `formularauswertung.aspx` übermittelt. Die Übertragung der Daten erfolgt via `post`-Methode, also im Inhaltsbereich der HTTP-Anforderung (HTTP-Body).

Die clientseitige Anforderung der ASP.NET-Seite leitet der Internet Information Service (IIS) an den ASP.NET-Worker-Prozess weiter, der die Seite ausführt. Das

Ergebnis sendet der Server als HTML-Dokument an den Client zurück. Innerhalb der vom Browser empfangenen Seite werden die Informationen eingelesen. Im konkreten Beispiel durch die Zeile:

```
Response.Write(Request.QueryString("Name"))
```

Das Request-Objekt ist, wie das Response-Objekt auch, integraler Bestandteil von ASP.NET.

Hier die vollständige Datei `formularausgabe.aspx`:

```
<% @Page Language="C#" Debug="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
  <title>Formularwerte ausgeben</title>
</head>
<body>
  <b/>
  Bis zu dieser Stelle empfinden Sie dieses Buch als
  <br/>
  <br/>
  <label style="font-style: italic; background-color: red; color: white;
font-size: 18px">
  <%
    Response.Write(Request.QueryString("eindruck"))
  %>
  </label>
  &nbsp;
  <br/>
  <br/>
  Vielen Dank für Ihre Angabe!
  </b>
</body>
</html>
```

Listing 1.3: Die Datei `formularausgabe.aspx`

Entscheidend ist, dass nach jedem Klick auf den Button ABSENDEN eine neue Antwortseite aufgebaut wird, nämlich jene durch ASP.NET generierte.

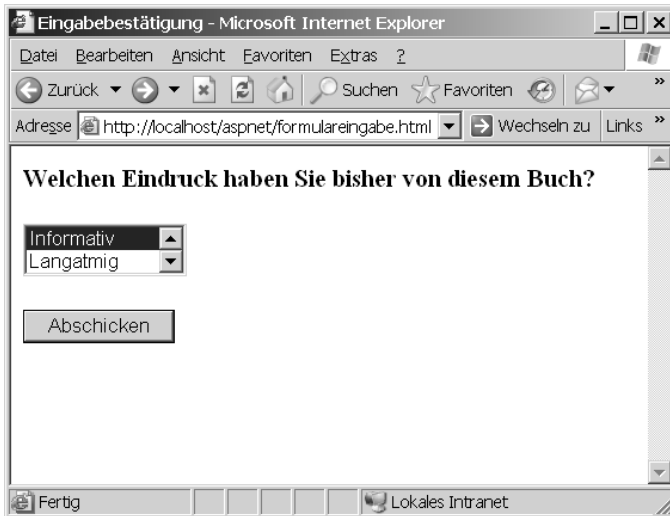


Abb. 1.4: Die HTML-Eingabeseite

Und die Ausgabeseite sehen Sie in Abbildung 1.5.

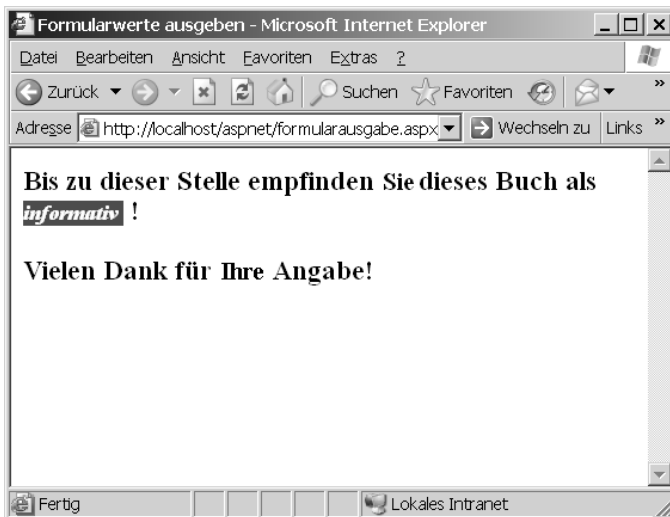


Abb. 1.5: Die ASPX-Ausgabeseite

Formularverarbeitung mit AJAX

Wie Sie bereits aus Abschnitt 1.1, *AJAX aus optischer Sicht wissen*, führt die Verwendung von AJAX dazu, dass das Ergebnis einer Formularanfrage im selben Fenster erscheint wie das Formular selbst. Die bestehende Seite wird also lediglich um neue Inhalte ergänzt. Um dies zu realisieren, muss sich der AJAX-Programmierer um das Versenden von Anfragen an den Server in Eigenregie kümmern. Eine Aufgabe, die ansonsten der Browser automatisch erledigt.

Hinweis

Die Anfrage wie auch die Prüfung, ob sie erfolgreich war, wird unter Verwendung von JavaScript erstellt, desgleichen die Verarbeitung der durch den Server gegebenen Antwort. Genau genommen jedoch richtet JavaScript selbst nicht die Anfrage an den Server, sondern weist lediglich dem Browser eine spezifische Anfrage zur Weiterleitung an den Server zu (für das grundlegende Verständnis des AJAX-Konzepts ist dies unerheblich, soll der Vollständigkeit halber gleichwohl nicht unerwähnt bleiben).

Kommen wir zurück zur Beispielanwendung. In der Dropdown-Liste wurde ein Eintrag ausgewählt, auf die eine JavaScript-Funktion reagieren soll. Die Funktion (im Folgenden `inquiry()` genannt) schickt eine Anfrage mit der getroffenen Auswahl an den Server. Fortlaufend erfolgt eine Rückmeldung des Servers, wie weit die Anfrage fortgeschritten ist. Ist die Bearbeitung abgeschlossen, übermittelt der Server an den Client eine Antwort, in deren Header sich u.a. der Statuscode befindet, der die Informationen darüber enthält, ob es bei der Datenübertragung zu Fehlern gekommen ist. Die Daten selbst befinden sich im Hauptteil der Antwort.

Die Funktion `inquiry()` wird über den Eventhandler `onchange`, bei Änderung der Auswahl, aufgerufen. Die Übergabe des Wertes aus dem ausgewählten `select`-Feld erfolgt über `this.value`. Damit entfällt in Listing 1.2 die Verwendung des `SUBMIT`-Schalters. Ein Formular wird ebenfalls nicht mehr benötigt. Dafür aber ein `div`-Tag, das um eine ID zu ergänzen ist. Doch dazu später mehr.

Listing 1.4 zeigt den geänderten HTML-Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
  <title>Eingabebestätigung</title>
</head>
<body>
  <label style="font-weight: bold; font-size: 20px">
```

```
        Welchen Eindruck haben Sie bisher von diesem Buch?  
</label>  
<br/>  
<br/>  
<select name=eindruck size="2" onchange="inquiry(this.value)">  
<option value="informativ" selected="selected" >Informativ</option>  
<option value="langatmig">Langatmig</option>  
<option value="praxisbezogen">Praxisbezogen</option>  
<option value="theorielastig">Theorielastig</option>  
</select>  
<br/>  
<br/>  
<div ID="ausgeben"></div>  
</form>  
</body>  
</html>
```

Listing 1.4: An die AJAX-Erfordernisse angepasster HTML-Code

Wie sieht die Funktion `inquiry()` im Einzelnen aus? Geschrieben wird die Funktion unterhalb des `XMLHttpRequest-Object`s, das Ihnen in Abschnitt 1.2.4 detailliert vorgestellt wurde:

```
function inquiry(wahl) {  
  
    if(anfrage != null) {  
        var url="formularausgabe.aspx";  
        var parameter = "?eindruck="+encodeURIComponent(wahl);  
        url += parameter;  
        anfrage.open("GET", url, true);  
        anfrage.onreadystatechange=dispay;  
        anfrage.send(null);  
    }  
    else  
    {  
        alert("XMLHttpRequest-Object konnte nicht erstellt werden");  
    }  
}
```

Listing 1.5: Die Funktion `inquiry()`

Haben Sie es bemerkt? Richtig! Bei unserer Funktion `inquiry(wahl)` wird `GET` als Methode der Übertragung verwendet. Demnach werden die Daten an den URL angehängt.

Keine Gretchenfrage – aber auch nicht unwichtig: GET oder POST?

Wann welche Übertragungsart zum Einsatz kommt, ist in der Tat keine Gretchenfrage, wohl aber eine des korrekten Programmierens, gerade auch in Zeiten vermehrter Verwendung von Proxyservern. So dürfen GET-Requests von Proxyservern gecacht werden. Eine Angelegenheit mit zwei Seiten, führt das Caching doch um den Preis zu einer Reduzierung der Serverlast, dass auch alte Daten zurück zum Client geschickt werden. Das Caching kann zwar – auch durch ASP.NET – unterbunden werden, was nichts daran ändert, dass die Verwendung des GET-Kommandos auf Nicht-Formularseiten beschränkt sein sollte. Ab HTML 4.0 wird dies sogar empfohlen.

GePOSTete Formulare werden dagegen vom »Proxy« nicht gecacht. In ASP.NET schlägt sich dies in der `OutputCache`-Direktive wieder, die per Default nur via `Get` verschickte Seiten cacht.

Damit im Erfolgsfalle die im Listenfeld gewählte Option auf derselben Seite angezeigt werden kann, wird eine weitere JavaScript-Funktion – wir nennen sie `display()` – benötigt. Wenn sich der Zustand der Anfrage ändert, soll diese Funktion aufgerufen werden. Dies wurde innerhalb der Funktion `inquiry()` durch die Zeile

```
anfrage.onreadystatechange=display;
```

definiert. So einfach sind die Dinge jedoch nicht, existieren doch gleich mehrere Bereitschaftszustände. Das Ziel soll aber sein, die Bestätigung unserer Wahl erst dann auszugeben, wenn der Server die benötigte Antwort zurückgesendet hat, das heißt, die Anfrage vollständig abgearbeitet wurde.

Auszulesen sind die Bereitschaftszustände über `readyState`. Zurückgeliefert wird eine Zahl von 1 bis einschließlich 4, die den entsprechenden Zustand angibt. Nachstehende Tabelle erklärt, was sich hinter den Zahlen verbirgt.

readyState	Bedeutung
0	Die Verbindung wurde noch nicht initialisiert.
1	Die Verbindung wird über <code>open</code> initialisiert.
2	Die Anfrage wurde gesendet.
3	interaktiv, Serverantwort empfangen
4	abgeschlossen

Tabelle 1.2: Werte für `readyState` und ihre Bedeutung

Fügen wir also eine entsprechende `if`-Abfrage ein, können wir getrost davon ausgehen, eine vollständige Antwort vom Server erhalten zu haben.

```
if (anfrage.readyState == 4) { /*Dies wird nicht die letzte Abfrage sein ...*/ }
```

Die Serverantwort besteht aus einem Header, der Angaben über Erfolg oder Misserfolg der Anfrage enthält (Statuscode), sowie möglichen Informationen im Body. Auf den Statuscode können Sie ebenfalls codeintern zugreifen, und zwar über `anfrage.status`. Hier sollte nicht eine Information wie beispielsweise `404 Not Found` ausgegeben werden, die nichts anderes bedeutet, als dass die Seite nicht gefunden wurde. Gewünscht ist der Statuscode `200`, als Beweis einer erfolgreich »aufgespürten« Seite.

Die Frage, ob eine Seite gefunden wurde oder nicht, führt beinahe zwangsläufig zu einer zweiten `if`-Abfrage:

```
if (anfrage.status == 200){/*trifft auch dies zu, erfolgt die Textausgabe*/ }
```

Die beiden Abfragen sollten ineinander verschachtelt werden, denn nur wenn eine Anfrage tatsächlich abgearbeitet wurde (`readyState == 4`), lässt sich auch der Statuscode ermitteln.

Bitte vergegenwärtigen Sie sich, dass `readyState` und `status` Eigenschaften des `XMLHttpRequest-Objects` sind. Genauso wie `responseText`, mit der sich auf im Body der Serverantwort enthaltene Daten zugreifen lässt.

Nun zur Implementierung der Funktion `display()`. In unserem Beispiel beinhaltet `responseText` den via HTML ausgegebenen Satz zuzüglich dem gewählten Eintrag aus der Dropdown-Liste. Die Serverantwort weisen wir einer Variablen mit dem Namen `inhalt` zu:

```
var inhalt = anfrage.responseText;
```

Mit dem, was der Server zurückgibt, muss das mit `ID="ausgeben"` versehene `div`-Element aktualisiert werden. Damit hätten wir der Funktion `display()` die nötigen »Zutaten« beigegeben – außer der wichtigsten: eine Zeile JavaScript-Code, mit dem die ID angesprochen werden kann:

```
document.getElementById("ausgeben");
```

Um den Inhalt tatsächlich ändern zu können, wird ferner eine Eigenschaft `innerHTML` benötigt:

```
document.getElementById("ausgeben").innerHTML=inhalt;
```

Damit wäre die Funktion `display()` vollständig:

```
function display() {  
  
    if(anfrage.readyState== 4){  
  
        if (anfrage.status==200){  
  
            var inhalt = anfrage.responseText;  
            document.getElementById("ausgeben").innerHTML = inhalt;  
        }  
    }  
}
```

Listing 1.6: Die Funktion `display()` zur Erzeugung der partiellen Ausgabe

Bleibe nur noch die Frage zu klären, an welcher Stelle die Funktionen `inquiry()` und `display()` innerhalb des HTML-Listings eingebaut werden sollen. Wie Sie sich sicher denken werden, finden die Funktionen als Teil des JavaScript-Blocks direkt unterhalb des `XMLHttpRequest`-Objects ihren Platz. Die HTML-Datei sieht damit wie folgt aus:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de" lang="de">  
<head>  
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />  
    <title>Eingabebestätigung</title>  
    <script type="text/javascript">  
/*<![CDATA[*/  
//XMLHttpRequest erzeugen:  
  
    var anfrage = null;  
  
    try  
    {  
        anfrage = new XMLHttpRequest();  
    }  
    catch(e)  
    {  
        try  
        {  
            anfrage = new ActiveXObject("Msxml2.XMLHTTP");  
        }  
        catch(e)
```

```
        {
            try {
                anfrage = new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch(e)
            {
                anfrage = null;
            }
        }
    }

function inquiry(wahl) {

    if(anfrage != null) {

        var url="formularausgabe.aspx";
        var parameter = "?eindruck="+encodeURIComponent(wahl);
        url += parameter;
        anfrage.open("GET", url, true);
        anfrage.onreadystatechange=display;
        anfrage.send(null);
    }
    else
    {
        alert("XMLHttpRequest-Objekt konnte nicht erstellt werden");
    }
}

function display() {

    if(anfrage.readyState== 4) {

        if (anfrage.status==200) {
            var inhalt = anfrage.responseText;
            document.getElementById("ausgeben").innerHTML = inhalt;
        }
    }
}
/*]]>*/
</script>
</head>
<body>
    <label style="font-weight: bold; font-size: 21px">
        Welchen Eindruck haben Sie bisher von diesem Buch?
    </label>
```

```

<br/>
<br/>
<select name="eindruck" size="2" onchange="inquiry(this.value)">
  <option value="informativ">Informativ</option>
  <option value="langatmig">Langatmig</option>
  <option value="praxisbezogen">Praxisbezogen</option>
  <option value="theorieelastig">Theorieelastig</option>
</select>
<br/>
<br/>
  <div id="ausgeben"></div>
</body>
</html>

```

Listing 1.7: Die HTML-Datei `formulareingabe_ajax.html` in ihrer Gesamtheit

Rufen wir die Seite im Browser auf (siehe Abbildung 1.6).

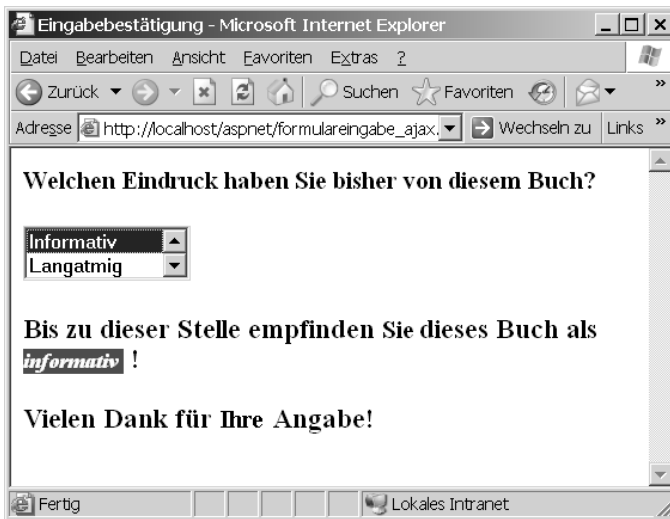


Abb. 1.6: ... und was Sie im Browser erwartet

Viel zu experimentieren gibt es in unserem Beispiel nicht. Das, was Sie im Dropdown-Listenfeld auswählen, wird unterhalb des Formulars angezeigt – und augenscheinlich nur das. Das Formular selbst erfährt ebenso wenig einen Reload wie die Überschrift der Beispielseite. Trotzdem hat die elektronische Buchbewertung eine Schwäche. Die Antwort des Servers besteht nämlich nicht nur aus dem, was im Dropdown-Listenfeld angeklickt wurde, sondern überflüssigerweise und immer

wieder aufs Neue auch aus dem Begleittext, der durchaus aber auch statisch im »HTML-Geflecht« untergebracht werden könnte. Machen Sie es besser als ich und ändern Sie – als kleine Vorübung zum Praxisteil des Buches – die relevanten Listings ab.

1.4 XML oder JSON?

Betrachten wir es von der praktischen Seite: In der Einführung zum Kapitel AJAX wurde bereits darauf hingewiesen, dass JSON (JavaScript Object Notation) gegenüber XML das kompaktere, vor allem mit weniger Overhead belastete Rückgabeformat ist. Gerade jener, durch die notwendigen Markups zur Auszeichnung der Struktur eines XML-Dokuments entstehende Overhead wirkt sich nachteilig auf Übertragungsgeschwindigkeit und Datenvolumen aus.

Zwischen JSON und XML existiert jedoch ein grundsätzlicherer Unterschied, der die Verwendung von JSON (zunächst) einschränkt: XML ist eine Auszeichnungssprache (Markup-Language), während JSON lediglich ein Format zum Austausch von Daten ist.

JSON stellt die Informationen in Schlüssel-Wert-Paaren dar. Schlüssel und Wert werden dabei durch einen Doppelpunkt voneinander getrennt und jeweils zwischen Anführungszeichen gestellt. Zwischen Listen von Schlüssel-Wert-Paaren steht ein Komma. Allgemein sind die als Objekte zu betrachtenden Schlüssel-Wert-Paare in geschweiften Klammern einzufassen. Demnach:

```
{"Schlüssel1": "Wert1", "Schlüssel2": "Wert2"}
```

Beispiel:

```
{"Buchtitel": "WEB2.0 mit ASP.NET und AJAX", "Verlag": "mitp"}
```

Kommen Arrays ins Spiel, sind diese in eckigen Klammern einzukapseln. In der JSON-Notation besteht ein Array aus einer Zeichenkette, einer Zahl oder aber einem booleschen Wert, sprich `true`, `false` oder `null`. Was die Darstellung von Zahlen anbelangt, so können diese durch die Angabe eines Exponenten `e` oder `E`, gefolgt von `+` oder `-` und der Ziffernfolge 0–9 dargestellt werden (die Basis bzw. eine Zahl ohne Exponentialdarstellung wird selbstverständlich auch durch eine Folge der Ziffern 0–9 gebildet).

Um den syntaktischen Unterschied zwischen XML und JSON augenscheinlich zu machen, stellen wir ein paar Daten zunächst in XML und anschließend in JSON dar. Die Informationen sollen der Rubrik »Allgemeine Datensammelwut im Internet« entstammen. Und da ist (auch für finstere Gestalten) nichts interessanter als die Geheimnisse Ihrer Kreditkarte:

```
<?xml version="1.0" encoding="UTF-8"?>
<daten>
  <karte>
    <kreditkarte>Bankavia</kreditkarte>
    <nummer>1966-0202-2008</nummer>
    <inhaber>
      <name>Klappert</name>
      <vorname>Uwe</vorname>
      <geschlecht>männlich</geschlecht>
      <präferenzen>
        <primär>Programmieren</primär>
        <sekundär>Schreiben</sekundär>
      </präferenzen>
      <alter>43</alter>
    </inhaber>
    <finanzstatus>
      <deckung>72</deckung>
      <währung>Euro</währung>
    </finanzstatus>
  </karte>
</daten>
```

Listing 1.8: »Datenklausur« in XML

Die JSON-Variante:

```
{
  "daten" : {
    "Kreditkarte" : "Bankavia",
    "Nummer" : "1966-0202-2008",
    "Karteninhaber" : {
      "Name" : "Klappert",
      "Vorname" : "Uwe",
      "Geschlecht" : "männlich",
      "Präferenzen" : [
        "Programmieren",
        "Schreiben"
      ],
      "Alter" : 43,
      "Deckung" : 72,
      "Währung" : "EURO"
    }
  }
}
```

Listing 1.9: ... und eine mögliche JSON-Variante

Hinweis

Sollten Sie Ihre Entscheidung pro oder contra JSON einzig vor dem Hintergrund des bei XML reichlich anfallenden Overheads treffen, muss diese Entscheidung möglicherweise bald revidiert werden: Das u.a. für Standardisierungen zuständige W3C (World Wide Web Consortium) untersucht seit einiger Zeit Möglichkeiten der Speicherung von XML-Inhalten im Binär- statt des bislang üblichen Textformats. Zahlreiche Binärformate für XML existieren bereits jetzt und erlauben durchaus die Prognose eines künftig schnelleren XML.

1.5 Frameworks auch für AJAX

Selbst unser, in aller erdenklichen Einfachheit gehaltenes Beispiel macht eines deutlich: Der Zugriff auf die einzelnen Elemente einer Internetseite via DOM ist ein recht mühseliges Unterfangen. Und selbst die Erstellung des XMLHttpRequest-Object in relativer Abhängigkeit vom Browser ist keine wirklich aufwandlose Angelegenheit. Da hilft nur eines: Frameworks in Gestalt von wieder verwendbaren JavaScript-Bibliotheken, die Ihnen die Arbeit mit AJAX mehr als nur erleichtern.

1.5.1 Prototype

Ob Prototype wirklich der Prototyp der AJAX-Frameworks ist; die Frage muss offenbleiben. Unbestreitbar dient Prototype als Basisframework weiterer AJAX-Frameworks, wie **script.aculo.us** (über das Sie im nächsten Unterkapitel mehr erfahren werden) sowie **rico**.

Prototype selbst ist ein kostenloses, umfangreiches JavaScript-Framework, das sowohl verschiedene Programmiererleichterungen als auch Möglichkeiten zur Verfügung stellt, JavaScript-Code zu vereinfachen. Darüber hinaus ist Prototype plattformunabhängig.

Von Sam Stephenson im Jahre 2005 entwickelt, ist Prototype Bestandteil des quell-offenen Web Application Frameworks **Ruby on Rails**.

Aktuell in der Version 1.6.1 erhältlich steht Prototype unter <http://www.prototypejs.org> zum Herunterladen bereit. Nach erfolgreichem Download muss die Bibliothek zuerst in die Datei eingebunden werden, die das AJAX-Script enthält. Befindet sich die Datei `prototype.js` zum Beispiel im Unterordner `prototype` und der wiederum in einem Ordner `frameworks`, so wäre im Header der (HTML-) Datei zu ergänzen:

```
<script type="text/javascript" src="frameworks/prototype/prototype.js">
```

Ruby on Rails

Ein sehr spezielles Framework, von David Heinemeier Hansson in der Programmiersprache **Ruby** geschrieben, und erstmalig ein Jahr vor der Erstveröffentlichung des Prototype-Frameworks publiziert. Ruby on Rails, kurz Rails oder RoR, basiert auf den Grundsätzen »Don't Repeat Yourself« (DRY) sowie »Convention Over Configuration« (COC). Statt also einer veränderbaren Konfiguration sind bestimmte Regeln für die Konvention von Objektamen einzuhalten. Das Zusammenspiel der verschiedenen Objekte ergibt sich dabei aus den Konventionen. Dieses Prinzip ermöglicht eine eindeutig schnellere Umsetzung von Anforderungen, und damit eine Zeitersparnis bei der Entwicklung vor allem webbasierter Software.

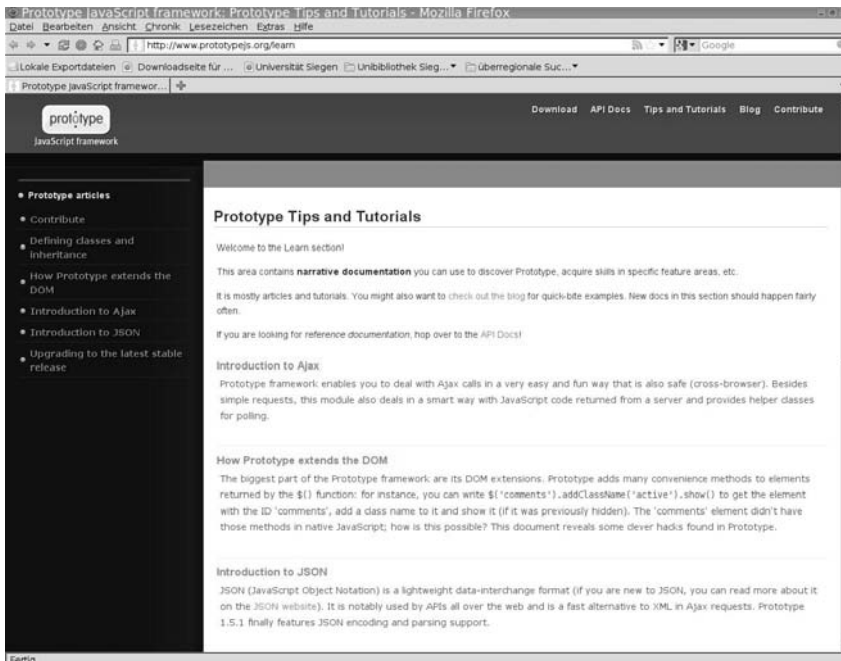


Abb. 1.7: Schlicht, aber nicht unsympathisch – Prototype im Web

Prototype beinhaltet zahlreiche Funktionen für die Entwicklung von JavaScript-Programmen. Die Vielfalt reicht von Kurzbefehlen (Shortcuts) bis hin zu komplexen Funktionen für beispielsweise das XMLHttpRequest-Object. Einige werden Ihnen im Folgenden vorgestellt:

Wie Sie im Abschnitt *Formularverarbeitung mit AJAX* auf Seite 33 erfahren haben, wird, um das DOM-Element einer Internetseite anzusprechen, die Funktion `document.getElementById("element_id")` verwendet. Im Prototype-Framework existiert eine Funktion `$()`, die die originäre Funktion beträchtlich verkürzt:

```
$("element_id")
```

Um die CSS-formatierte Textfarbe eines Elements zu ändern, muss geschrieben werden:

```
$("element_id").style.color = "#fcfcfc"
```

Beziehungsweise:

```
$("element_id").setStyle({color: "#fcfcfc"})
```

Eine Weiterentwicklung der `$()`-Funktion stellt die `$F()`-Funktion dar. Der Buchstabe »F« steht dabei für »Form«, das heißt, mit `$F()` kann der Wert eines Formelements angesprochen werden. Handelt es sich bei diesem Formelement um ein Dropdown-Listefeld (wie in unserem Beispiel), wird mittels `$F()` der aktuell ausgewählte Eintrag geliefert. Bei einem Textfeld werden die im Feld befindlichen Daten ausgegeben:

```
$F("input_element_id")
```

Neben grundlegender Funktionalität bietet Prototype auch Unterstützung für objektorientiertes Programmieren. So wird mit `Class.create()` eine neue Klasse angelegt:

```
var ExampleClass = Class.create({
  initialize: function(){
    this.data = "A first and simple Example Class";
  }
});
```

Listing 1.10: Beispielklasse unter Einsatz von Prototype

Das Prototype-Framework wurde auch mit dem Ziel einer Vereinfachung des Umgangs mit dem `XMLHttpRequest-Object` entwickelt. Ferner geht es darum, eine auf die unterschiedlichen Browser »zugeschnittene« Programmierung zu vermeiden.

Der Abruf serverseitiger Information wird durch Prototype auf zwei Arten realisiert:

- Durch `Ajax.Request`, das eine unformatierte XML-Ausgabe liefert.
- Durch `Ajax.Updater`, das die Server-Response direkt in ein DOM-Objekt schreibt.

Nachstehendes Listing verdeutlicht das Vorgehen bei einem `Ajax.Request`-Aufruf:

```
var content = $H({
  v1: $("input_element_id")
});
var url = "http://www.examplepage.com/page.aspx";
var request = new Ajax.Request(url, {
  method: "post";
  parameters: content;
  start: showResponse;
});
```

Listing 1.11: Wertermittlung und Ausgabe durch `Ajax.Request` und `showResponse`

Was passiert? `Ajax.Request` liest den Wert eines Textfeldes (oder eines Drop-down-Listenfeldes) aus. Im Anschluss erfolgt über

```
var url = http://www.example.com/page.aspx;
```

(Zuweisung der Seite an die Variable `url`) der Aufruf der Webseite vom Server. Die Formulardaten werden mit `post` versendet. Nach erledigter Anfrage wird schließlich die Funktion `showResponse()` aufgerufen.

1.5.2 script.aculo.us

Prototype ist überaus nützlich, wenn es um die Reduzierung von Programmieraufwand geht. Die für AJAX wichtigen, visuellen Effekte lassen sich damit jedoch nicht erzeugen. Hier kommt **script.aculo.us** ins Spiel, ein wahrer Zauberer, wenn es darum geht, einer desktopähnlichen Anwendung die eben desktopeigenen Effekte, wie beispielsweise Drag&Drop, hinzuzufügen. Darüber hinaus hält `script.aculo.us` neben anderen GUI-Elementen auch diverse Möglichkeiten zur problemlosen Erzeugung sortierbarer Listen parat. Des Weiteren können Sie ohne viel Aufwand Google-Suggest-ähnliche Funktionen implementieren. Und was es auch gibt, ist eine breite Palette zum Teil beeindruckender optischer Effekte.

Zu verdanken haben Sie die »bunte Vielfalt« Thomas Fuchs, der `script.aculo.us` aus dem Digital-Asset-Management-Tool **fluxium** heraus kreierte. Die Erstvorstellung des Frameworks erfolgte, wie auch bei Prototype, im Jahre 2005. Das Framework wird von Ruby on Rails ebenso genutzt wie von teils namhaften Organisationen und Firmen. Exemplarisch seien NASA, Apple und CNN genannt.

Besuchen Sie script.aculo.us unter <http://script.aculo.us>, und lassen Sie sich von einer Internetseite begrüßen, die Sie vielleicht ein wenig an das berühmte Intro der James-Bond-Filme erinnert (siehe Abbildung 1.8). Möglich (auch) dank script.aculo.us!

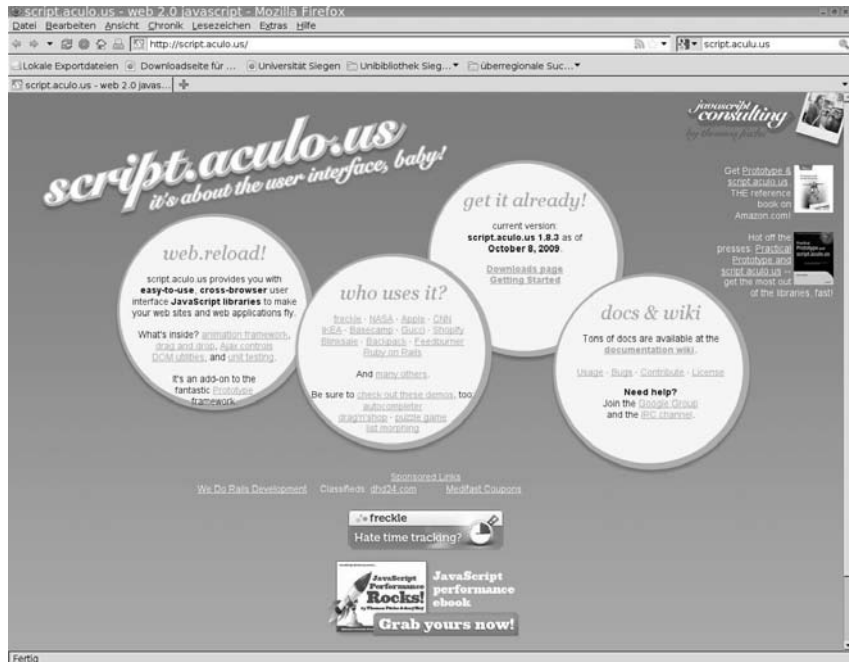


Abb. 1.8: Eine runde Sache – script.aculo.us im World Wide Web

Hier können Sie sich das aktuelle Release (1.8.3) in verschiedenen Formaten gepackt (denn auch script.aculo.us ist selbstredend plattformunabhängig) zum Nulltarif herunterladen. Da script.aculo.us Prototype voraussetzt, muss zuerst Prototype und dann script.aculo.us in Ihre Dokumente eingebunden werden. Aussehen könnte das beispielsweise so:

```
<script type="text/javascript" src="scriptaculous/lib/prototype.js"></script>
<script type="text/javascript" src="scriptaculous/src/scriptaculo.js"></script>
```

Schauen wir uns ein wenig genauer an, was dieses erstaunliche Framework bietet und wie Sie von den Möglichkeiten Gebrauch machen können. Zunächst auf der

visuellen Ebene. Hier existieren so genannte Kerneffekte, die miteinander kombiniert weitere Effekte ergeben. Zu den Kerneffekten zählen unter anderem:

- **Opacity**, ändert die Transparenz eines Objekts.
- **Fade**, blendet ein Objekt aus (das Pendant zu **fade** ist **Appear**, bei dem ein Element eingeblendet wird).
- **Move**, verschiebt beliebige Elemente per JavaScript.
- **MoveBy**. Eine Spezialisierung des Effekts **Move**. Bei **MoveBy** erfolgt die Bewegung relativ zur Ausgangsposition.
- **Highlight**, hebt ein Element durch einen animierten Wechsel der Hintergrundfarbe hervor.

Hinweis

Script.aculo.us ist nicht als abgeschlossen zu betrachten. Gewiefte Programmierer können dem Framework weitere Effekte hinzufügen, beispielsweise durch neue Kombinationen von Kerneffekten.

Um einen Effekt nutzen zu können, muss dem Element, auf dem der Effekt angewendet werden soll, eine ID zugeordnet werden. Für die Aktivierung des Effekts genügt eine Zeile Code. Vergessen Sie bitte nicht, dass der Effekt auf ein DOM-Element angewendet wird.

```
new Effect.Effekt_Name('id_of_Element');
```

Sie haben sogar die Möglichkeit, Dauer und Reichweite des ausgewählten Effekts Ihren Wünschen anzupassen. Nehmen wir an, es ginge darum, eine Grafik auszublenzen, demnach also um den Effekt **Fade**:

```
new Effect.Fade('id_of_Element', {duration:2.0, from:0.0 to:0.7});
```

Durch die exemplarische Parameterfestlegung wird erreicht, dass Ihre Grafik zwar ausgeblendet wird, jedoch nur bis zu 70% und das bei einer Deckkraft (**duration**) von 20%.

Als Pseudoeffekt ist der Effekt **Parallel** zu bezeichnen. Doch ist »pseudo« in diesem Falle nicht gleich »unwichtig«, denn genau mit diesem Pseudoeffekt werden Sie in die glückliche Lage versetzt, Effekte miteinander kombinieren zu können. Nehmen wir **Opacity** und **Move**, die für Transparenz und Verschiebung zuständige Effekte. Unser Ziel soll sein, die Transparenz des Objekts zu verändern, während sich das Objekt bewegt (sozusagen zur Laufzeit also). Mit **Parallel** ist die Aufgabe schnell und ohne viel Aufwand erledigt:

```
new Effect.Parallel(  
  [new Effect.Opacity(this {sync: true}),  
   new Effect.Move(this {sync: true})]  
);
```

Listing 1.12: Parallelisierung von Kerneffekten

1.6 Probleme und Kritik bei AJAX

Mag das, was mit AJAX möglich ist, gerade in Verbindung mit Frameworks wie Prototype und script.aculo.us die Optik einer Webpräsenz auch noch so nahe an eine typische Desktopanwendung rücken; von einer (technischen) Perfektionierung des World Wide Web (WWW) durch die AJAX-Technologie kann wiederum keine Rede sein. Denn auch bei AJAX gibt es Nachteile – und die sind nicht zu unterschätzen.

1.6.1 Belastende Anfragen – Polling

Es wurde bereits darauf hingewiesen, dass im Gegensatz zu einem Request-Response-Aufruf bei einer AJAX-Anwendung permanent der Stand der Bearbeitung beim Server erfragt wird. Wie leicht nachzuvollziehen ist, führt dies zu einer ganz anderen Auslastung des Webservers. Bezeichnet wird das Problem als Polling. Dieses Buch kann Ihnen leider keine Möglichkeit liefern, das Polling zu umgehen. Aber vielleicht haben Sie eine Idee ...

1.6.2 Barrieren durch AJAX

In Deutschland müssen durch die öffentliche Hand betriebene Webseiten für Menschen mit Behinderungen benutzbar sein (bei Betriebssystemen wie Windows und den großen Linux-Distributionen – in Maßen – längst eine Selbstverständlichkeit). Webseiten, die solche Kriterien erfüllen, werden als barrierefrei bezeichnet. Hier gibt es bei AJAX Nachholbedarf.

Stichwort: Drag&Drop. Ein Effekt, der oft und gerne benutzt wird, leider aber für Personen, die lediglich die Tastatur für ihre Arbeit am PC benutzen können, nicht anwendbar. Ein Ansatz zur Umgehung des Problems könnte darin bestehen, die zu verschiebenden Elemente einer Seite zunächst wie einen Hotkey anzusprechen und anschließend mittels Pfeiltasten über den Bildschirm zu bewegen. Warum sollte ein AJAX der nächsten Generation die Pfeiltasten nicht ebenso selbstverständlich nutzen, wie es die Computerspiele der ersten Generation taten?

Wirklich problematisch ist der Einsatz von AJAX im Zusammenhang mit Screenreadern, Programmen also, die Sehbehinderten den Inhalt einer Seite vorlesen. Bekannte Screenreader, wie »IBM Homepage Reader« oder »JAWS« sind zwar für die Auswertung von JavaScript gerüstet, hochgradig defizitär ist jedoch, dass

genannte Leseprogramme den Benutzer nicht informieren, wenn sich AJAX-bedingt die Inhalte einer Seite geändert haben.

Man muss es ganz klar sagen: Dem partiellen Austausch von Seiteninhalten gehört die Zukunft. Und hier wird es auch keine, womöglich noch nostalgiebedingten, Rückschritte geben. Die Zahl derjenigen aber, die trotz – oder gerade wegen! – körperlicher Einschränkungen einen Arbeitsplatz im PC-Umfeld haben, ist nicht klein, und dementsprechend ernst ist das Problem der Barrierefreiheit zu nehmen.

Interessante Diskussionsbeiträge zu diesem Thema finden Sie unter <http://www.vorsprungdurchwebstandards.de/theory/ajax-vs-accessibility/>. Gut informiert werden Sie auch durch den Accessibility-Blog von <http://www.einfach-fuer-alle.de/>. Daneben verweist die Seite immer wieder auf entsprechende Links (<http://www.einfach-fuer-alle.de/blog/tag/AJAX>). Reinschauen lohnt sich!

1.6.3 Verwendungstauglich? AJAX und Usability

Kritiker von AJAX stellen die Benutzerfreundlichkeit entsprechender Seiten immer wieder mit dem Argument in Frage, der Benutzer sei das klassische Submit-Reload-Schema gewöhnt. Und es stimmt: Gewöhnung ist alles – sagt zumindest der Volksmund. Wenn aber Gewöhnung alles ist, gebührt der Umgewöhnung mindestens derselbe Platz auf der Bewertungsskala menschlicher Anpassungsleistung. Die Frage ist nur, ob die im Falle von AJAX tatsächlich notwendig ist.

Bedenken Sie: Die meisten Menschen, die sich beruflich oder privat (oder beides) im Internet tummeln, arbeiten auch mit dem Betriebssystem oder sogar mit komplexen Desktopanwendungen. Zumindest aber werden Ordner geöffnet, verschoben, gelöscht, neu benannt oder in Richtung Papierkorb bewegt. Alles das können Sie ebenso mit einer entsprechend performanten AJAX-Anwendung. Der User betritt somit also keine wirkliche »Terra incognita«, wenn er beispielsweise eine Community besucht, bei der der Progressionsbalken des Browsers so gut wie nichts zu tun bekommt, weil die Seite nur einmal geladen und danach lediglich teilweise erneuert wird.

Ein Weiteres: Auch wenn Sie in diesem Buch immer wieder auf Formulierungen wie »partieller Seitenaustausch« stoßen, ist das keine Erfindung von AJAX, ebenso wenig wie Drag&Drop im Browser. Lange bevor Sie zum ersten Male auf den Begriff AJAX stießen, ohne dass damit ein Putzmittel oder eine Fußballmannschaft gemeint war, haben Sie vermutlich Ihre Internetseiten mit Frames oder gar Inline Frames versehen. Was das im Prinzip war: partieller Seitenaustausch. Wohlverstanden: *im Prinzip*. Und sicher gab es auch bei Ihnen schon experimentierfreudige Feierabende, an denen Sie die Eingabeformulare einer Internetseite mit Hilfe von JavaScript über den Bildschirm schoben.

1.6.4 Die Sprache zum Ein- und Ausschalten – JavaScript

Wäre JavaScript vor einigen Jahren nicht als weit offen stehende Pforte für allerlei unliebsames Virenetier entlarvt worden, gäbe es vielleicht zwischenzeitlich keine Möglichkeit mehr, Java Script in den gängigen Browsern zu deaktivieren. Die Option besteht aber nach wie vor. Und mit ihr ein Problem, denn AJAX »lebt« von JavaScript. Was also tun, wenn Ihnen die Sicherheit Ihrer Festplatte wichtiger ist als eine fortschrittliche Webseite im schicken AJAX-Gewand – auf die Sie trotzdem nicht verzichten wollen?

Viele Möglichkeiten haben Sie nicht. Zwei seien Ihnen genannt:

- Sie können über Ihre Internetseite einen Tickertext schicken, der den Benutzer höflich, aber bestimmt an die Notwendigkeit einer eingeschalteten JavaScript-Funktion erinnert. Zum längeren Verbleib auf der Seite motiviert ein solcher jedoch nicht.
- Alternativ besteht die Möglichkeit der Einrichtung eines `noscript`-Bereichs, der nur von Browsern ausgelesen werden kann, bei denen JavaScript deaktiviert ist. Zwischen `<noscript>` und `</noscript>`, oder an anderer Stelle, wäre vielleicht auch der Hinweis auf eine »Out-of-AJAX«-Version der Webpräsenz sinnvoll. Falls eine solche existiert ...

Hinweis

Das Problem mit JavaScript ist jedoch weitreichender, denn die Robots der großen Suchmaschinen (Yahoo, Google et cetera), quittieren gegenüber JavaScript-erzeugten Inhalten mitunter ihren Dienst. Konsequenz: Entsprechende Inhalte werden – wenn überhaupt – nur unzureichend erfasst.