Das Beispiel: Ein Bestellformular in HTML

Probieren wir's gemeinsam aus. Ich habe mir dieses Bestellformular für ein Produkt ausgedacht. Der Surfer soll die gewünschte Anzahl, Lieferart und Farbe wählen. Außerdem muss er Vor- und Zunamen, die Kundennummer und die E-Mail-Adresse eintragen. Auf die Angabe der Adresse habe ich verzichtet. Dazu dient die Kundennummer.

Wenn du dir ein Bild vom Formular machen möchtest, schaue einfach auf die Grafik. Nur nebenbei: Formulare werden auch gerne durch (unsichtbare) Tabellen ausgerichtet. Wenn du möchtest, kannst du dein Projekt wesentlich aufwändiger gestalten. Mir reichten allerdings 2 Linien.

Du findest das Projekt auch unter dem Namen *form1.htm* im Ordner *formular*. Schau dir den HTML-Quelltext des Formulars an:

```
<html>
<head>
<title>Bestellung</title>
</head>
<body bgcolor="white">
<h1>Produkt bestellen!</h1>
<form action="http://www.jchanke.de/php/test.php" method="post"
    name="abfrage" id="abfrage">
Das Produkt kostet EUR 75,-, die Versandkosten betragen EUR 12,-.
<hr width="420" align="left" />
Ich bestelle verbindlich <input type="text" size="2" maxlength="2" name="anzahl" id="anzahl"
    value="1" /> Stück des Produkts
<br /><input type="checkbox" name="express" id="express" value="ja" /> Expresslieferung (20
EUR Aufpreis)
<br />Ich möchte es in der Farbe
<input type="radio" name="sorte" id="sorte" value="schwarz" checked="checked" />Schwarz
<input type="radio" name="sorte" id="sorte" value="silber" />Silber
<input type="radio" name="sorte" id="sorte" value="gold" />Gold
<hr width="420" align="left" />
<br />Anrede: <select name="anrede" id="anrede" size="1">
<option value="Herr">Herr</option>
<option value="Frau">Frau</option>
</select>
<br />Vorname: <input type="text" name="vorname" id="vorname" />
<br />Nachname: <input type="text" name="name" id="name" />
<br />Kundennummer <input type="text" name="kundenr" id="kundenr" maxlength="8" />
<br />E-Mail-Adresse: <input type="text" name="email" id="email" />
<p><input type="reset" value="Formular leeren" />
<input type="submit" value="Daten abschicken" /></p>
</form>
</body>
</html>
```

Mit den vorhandenen HTML-Formularkenntnissen müsstest du dir die Funktionsweise schnell erschließen können. Ich habe versucht, die wichtigsten Formularfelder unterzubringen. Schaue bei Unklarheiten einfach im Referenzteil nach. Das Formular besitzt jedoch einige Schönheitsfehler, die wir gemeinsam beseitigen sollten.

Versehentlichen Reset und Submit verhindern

Den *Reset*-Knopf kennst du, er dient dazu, den Formularinhalt zu löschen und das Formular zurückzusetzen. Doch gerade bei längeren Formularen stellt dieser Button eine Gefahr da. Du hast ein meterlanges Formular ausgefüllt und drückst aus Versehen auf *Reset*? Zack – alle Daten sind weg, und du kannst von vorne mit dem Eintragen beginnen. Einverstanden, du könntest den Reset-Knopf einfach weglassen. Das ist eine gute Idee! (Wenn wir es im Beispiel trotzdem nicht so machen dann deshalb, weil ich dir an diesem Button etwas zeigen will!)

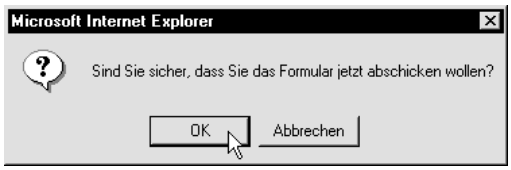
Ähnlich dumm ist es jedoch, wenn der Anwender das Formular nicht ausfüllt und trotzdem einfach so zum Spaß auf dem *Submit*-Knopf drückt. Deine CGI-Datenbank auf dem Server bzw. dein E-Mail-Postkasten wird mit unsinnigen, leeren Meldungen zugepflastert. Schluss mit diesem Leichtsinn! Bauen wir mit JavaScript einfach eine Routine ein, die das *reset*-Event und später auch das *submit*-Event überprüft.

Wichtig: Nach Klick auf *Reset* bzw. *Submit* wird ein *reset*- bzw. *submit*-Event an das Formular geschickt. Dieses kann man mit dem *onReset*- bzw. *onSubmit*-Event-Handler auffangen und an eine Funktion weiterleiten. Gibt diese Funktion den Wahrheitswert *true* (wahr) zurück, nimmt das Schicksal seinen Lauf. Erfolgt dagegen eine „*false*-Meldung", wird der Reset bzw. Submit verhindert.

Also, auf in den Kampf! Doch ehe es soweit ist, muss ich dir noch eine allerliebste neue Methode vorstellen, ohne die wir die Schlacht nicht schlagen werden.

Die confirm()-Box

Es handelt sich um die *confirm()*-Methode. Genau wie *alert()* wird dabei eine Dialogbox erzeugt. Genau wie bei *alert()* kannst du Textstrings und Variablen in dieser Box ausgeben. So wie in der Abbildung sieht unsere neckische *confirm()*-Box in der Praxis aus.



Was fällt dir auf? Richtig, im Gegensatz zu *alert()* besitzt *confirm()* zwei Schaltflächen, einmal *OK* und *Abbrechen*. Darüberhinaus verfügt *confirm()* über eine weitere interessante Besonderheit, die wir für unsere Zwecke schamlos ausnutzen. *confirm()* gibt je nach Klick den booleschen Wert (Wahrheitswert) *true* oder *false* zurück. Bei Klick auf *OK* lautet die Rückgabe logischerweise *true*, bei Klick auf *Abbrechen* dagegen *false*.

Das Skript zum Auffangen der Events

Und genau das brauchen wir! Denn mit *false* könnte man sowohl den Reset (Löschen des Inhalts) als auch den Submit (Abschicken) unterbinden. Wie soll das gehen? Angenommen, du notierst im `<form>`-Tag folgendes: `onReset="return false"`. Dann kannst hundertfach auf den Reset-Button drücken. Dann würde das Reset durch dieses *false* überhaupt nicht ausgeführt werden! Fazit: Wir brauchen nur noch eine Funktion, die entweder `true` oder `false` zurückgibt. Da hilft `confirm()`!

Bauen wir nun gemeinsam das Skript zusammen, unser „Sicherheitsnetz für das Formular". Wir setzen es in eine Funktion, die *onReset* „anspringen" wird. Diese Funktion taufe ich auf den Namen *sicher()*. Voilà, hier ist sie, setze die Funktion an die gewohnte Stelle im Dokument.

```
function sicher() {
var antwort;
antwort=confirm("Sind Sie sicher, dass Sie den Formularinhalt löschen wollen?");
return antwort;
}
```

Was macht das Skript? In der Variablen *antwort* wird der boolesche Wert *true* oder *false* aufgefangen. Je nachdem, für welchen Button sich der Nutzer entscheidet. Dieser Wert wird dann am Ende durch *return antwort* zurückgegeben. Denn mit dem Schlüsselwort *return* sorge ich dafür, dass die Funktion etwas zurückgibt!

Und wo rufen wir diese tolle Funktion nun auf? Natürlich im `<form>`-Tag, wie eben besprochen, und zwar mit `onReset`! Füge einfach folgende zusätzliche Attribute in das `<form>`-Tag ein:

```
onReset="return sicher()"
```

Das auf diese Weise verfeinerte Formular findest du auch in der Datei *form2.htm*.

Formularfelder auf Eingabe überprüfen

Immerhin haben wir sowohl den versehentlichen Reset als auch den Submit aufgefangen. Möglicherweise schickt der Kunde seine Daten im guten Glauben ab, hat aber vergessen, ein wichtiges Formularfeld auszufüllen? Ohne seine Kundennummer kennst du seine Adresse nicht! Und auch die anderen Felder sollten allesamt ausgefüllt werden. Und sicher würde sich der Kunde freuen, wenn er genau wüsste, was er für seine Bestellung im Einzelnen hinblättern muss. Das machen wir alles, ganz der Reihe nach.

Jetzt kommt onSubmit an die Reihe. Dafür bauen wir zusätzlich nun folgendes Attribut-Werte-Paar in das `<form>`-Tag ein: `onSubmit="return formtest()"`. Das gesamte `<form>`-Tag sieht nun so aus:

```
<form action="http://www.jchanke.de/php/test.php" method="post" name="abfrage"
    id="abfrage" onReset="return sicher()" onSubmit="return formtest()">
```

Als nächstes gilt es nun, die Funktion *formtest()* „zum Leben zu erwecken“. Ist der Submit gewünscht? Dann soll die Funktion *true* zurückgeben (`return true`). Das lösen wir gleich mit *confirm()*. Falls der Submit jedoch abgefangen werden soll, muss eine *false*-Meldung her (`return false`)!

Felder überprüfen, ob etwas eingetragen wurde

Wir bauen die Funktion Schritt für Schritt zu einer gewaltigen Routine aus. Zuerst beginnen wir ganz bescheiden damit, dass wir alle Felder auf Eingabe überprüfen. Im ersten Beispiel wurde die Kundennummer vergessen. Nach Klick auf unsere Submit-Schaltfläche springt *formtest()* an. Die vorläufige Routine dafür sieht so aus:

```
1   function formtest() {
2   var vorname, name, kundenr, email;
3   vorname=document.abfrage.vorname.value;
4   name=document.abfrage.name.value;
5   kundenr=document.abfrage.kundenr.value;
6   email=document.abfrage.email.value;
7   if (name=="" || vorname=="" || kundenr=="" || email=="") {
8   alert("Bitte füllen Sie alle Felder aus!")
9   return false;
10  }
11  return confirm("Sind Sie sicher, dass Sie das Formular jetzt abschicken wollen?");
12  }
```

Das Programm überprüft zuerst den Inhalt der Formularfelder mit einer *if*-Abfrage. Wir benötigen zuerst nur die Felder *vorname*, *name*, *kundenr* und *email*. *Zeile 3 bis 6* zeigen sehr schön, wie man Werte aus einfachen Text-Formularfeldern ausliest. Man beginnt bei *document*, verzweigt sich über den Formularnamen (*abfrage*) bis hin zum Namen des Formularfelds (in *Zeile 3* z.B. *vorname*) und ermittelt den *value*.

Zur if-Abfrage in Zeile 7: Die senkrechten Striche (||) sind der (Endweder-)Oder-Operator, der uns ermöglicht, gleich mehrere Fliegen mit einer Klappe zu schlagen. Es handelt sich um ein logisches Oder. Der zweite Operand wird nur dann ausgewertet, wenn der erste *false* ist. Die leeren Gänsefüßchen bedeuten nichts weiter als „leer“.

Wenn das erste Feld ausgefüllt ist, wird das zweite geprüft usw. Falls auch nur eines der Felder leer ist, hüpft eine *alert()*-Box hervor, und die Programmausführung wird beendet (*return false*). Dank unserers *false*-Wertes findet kein *Submit* statt. Ansonsten „rutschst“ das Skript tiefer und zeigt die *confirm*-Box aus Zeile 11. Hier wird der Benutzer gefragt, ob er das Formular wirklich abschicken will. Erinnerst du dich? Nur bei Klick auf OK wird true zurückgegeben (*return true*), ansonsten kommt es wieder zu einer *false*-Meldung und damit nicht zum Submit!

Den aktuellen Stand findest du übrigens in der Datei in *form3.htm*. Hier schnell ein paar Ideen zum Verfeinern:

Passwort-Check

Angenommen, du hast zwei Felder zur Eingabe des Passworts vorgesehen. Das zweite dient zur Bestätigung. Wie findest du heraus, ob Passwortfeld 1 und Passwortfeld 2 die gleichen Daten enthalten? Lies die Werte aus deinen Formularfeldern aus und vergleiche sie! Natürlich kannst du mit dieser Methode nur feststellen, ob sich der Surfer nicht verschrieben hat. Ob das Passwort tatsächlich korrekt ist, weißt du erst hinterher.

```
if (pw1!=pw2) {
alert("Bitte überprüfen Sie das Passwort!");
}
```

Postleitzahlen-Test

Wie schaut's mit der Postleitzahl aus? Diese muss in unseren Breiten 5 Zeichen lang sein. Lass dir von der *length*-Eigenschaft helfen. Die Prüfung könnte folgendermaßen aussehen:

```
if (plz.length<5) {
alert("Das ist keine gültige Postleitzahl");
}
```

Deiner Phantasie sind hier keine Grenzen gesetzt! Formuliere einfach die Bedingung, die du benötigst. Reagiere darauf. Hier noch einige Möglichkeiten, die dir beim Bau deiner individuellen Formulare sicher nützlich sein werden:

Test bei Verlassen: onBlur

Du kannst die jeweilige Kontrollfunktion übrigens auch direkt aus dem Formularfeld heraus „triggern". Nutze dafür den *onBlur*-Event-Handler. Im folgenden Beispiel wird bei Verlassen des E-Mail-Felds eine Funktion namens *mailcheck()* aufgerufen.

```
<input type="text" name="email" id="email" onBlur="mailcheck()" />
```

Ins Formularfeld zwingen: focus()-Methode

Du kannst auch den Fokus auf ein bestimmtes Feld setzen. Dafür nutzt du die von den Fenstern bekannte *focus()*-Methode. Diese setzt den Cursor in das gewünschte Formularfeld zurück. Im Beispiel ist es das E-Mail-Feld:

```
document.abfrage.email.focus();
```

Formularfeld markieren:select()-Methode

Ebenfalls nützlich ist in manchen Fällen die Möglichkeit, ein Formularfeld zu markieren. Das gelingt dir folgendermaßen:

```
document.abfrage.email.select();
```

Aber zurück zu unserem Beispiel! Als nächstes zeige ich dir, wie du hier einen E-Mail-Check einbaust!

E-Mail-Adresse auf Gültigkeit prüfen

Angenommen, du legst besonderen Wert auf die E-Mail-Adresse. Bisher prüfen wir lediglich, ob überhaupt etwas ins E-Mail-Feld eingetragen wurde: Der Surfer könnte aber auch irgendein Zeichen tippen, welches nichts mit einer E-Mail-Adresse zu tun hat.

Teste einfach, ob die Eingabe bestimmte Voraussetzungen erfüllt. Jede E-Mail-Adresse besitzt einen Klammeraffen (@) und einen Punkt(.). Außerdem sollte sie mindestens 7 Zeichen umfassen.

Nutze die schon kurz erwähnte *charAt()*-Methode und eine Zählschleife, um den Eingabestring auf das Vorhandensein des entsprechenden Zeichens zu testen. Hier zuerst die komplett aufgebohrte und wesentlich erweiterte Funktion mit E-Mail-Check „de luxe". Die Erklärung folgt danach.

```
1   function formtest() {
2   var vorname, name, kundenr, email, affe, punkt, i;
3   vorname=document.abfrage.vorname.value;
4   name=document.abfrage.name.value;
5   kundenr=document.abfrage.kundenr.value;
6   email=document.abfrage.email.value;
7   if (name=="" || vorname=="" || kundenr=="" || email=="") {
8   alert("Bitte füllen Sie alle Felder aus!")
9   return false;
10  }
11  affe=0;
12  punkt=0;
13  for (i=0; i<email.length; i++) {
14  if (email.charAt(i)=="@") {
15  affe=1;
16  }
```

```
17  if (email.charAt(i)==".") {
18  punkt=1;
19  }
20  }
21  if (affe!=1 || punkt!=1 || email.length<7) {
22  alert("Das ist keine gültige E-Mail-Adresse!");
23  document.abfrage.email.focus();
24  document.abfrage.email.select();
25  return false;
26  }
27  return confirm("Sind Sie sicher, dass Sie das Formular jetzt abschicken wollen?");
28  }
```

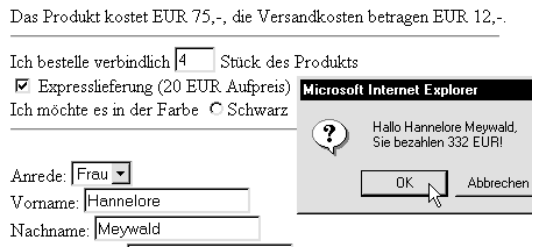
Im Beispiel nutzt du zwei Variablen als „Flag“. Das sind zwei „Fähnchen“ die du im Erfolgsfalle hisst und auswertest. Wenn dieser Schalter in unserem Beispiel auf 1 gesetzt wird, sind @ und Punkt vorhanden. Die neuen Variablen heißen im Beispiel *affe* (prüft, ob „@“ vorhanden ist) und *punkt* (prüft auf Vorhandensein von „.“).

In *Zeile 11* und *12* werden diese beide Variablen mit 0 initialisiert. Eine Zählschleife durchläuft die eingegebene Zeichenfolge von vorne bis hinten. Wenn das entsprechende Zeichen vorkommt (hier @), wird der Wert der Variablen auf 1 gesetzt. Das passiert in *Zeile 15*.

Eine weitere Zählschleife testet das Gleiche für das Punkt-Zeichen. Im Erfolgsfalle schaltet auch die *punkt*-Variable von 0 auf 1 um. Beachte bitte, dass du zum Abschluss zwei geschweifte Klammern setzen musst (*Zeile 19 und 20*). Einmal zum Beenden der zweiten *if*-Abfrage, dann zum Abschließen der Zählschleife.

In *Zeile 21* wird wieder eine if-Abfrage aktiv, die unser „Fähnchen“ testet und gleichzeitig prüft, ob auch 7 Zeichen Mindestlänge erreicht wurden. Wenn eine der Bedingungen nicht stimmt, folgt die obligatorische *alert()*-Box (*Zeile 22*) auf dem Fuß.

Mit *Zeile 23* setzt du den Cursor in das E-Mail-Feld zurück, *Zeile 24* sorgt dafür, dass es gleichzeitig markiert wird. Du findest den Stand bis hierher in der Datei *form4.htm*.



Das Ergebnis ausrechnen

Zum Schluss rechnest du noch das Ergebnis aus und zeigst es dem Surfer mitsamt Namen in einer *confim()*-Box an. Dafür verändern wir die *confirm()*-Box aus Zeile 27. Ergänze in der vorigen Version der *formtest()*-Funktion lediglich die Variable *ergebnis* in Zeile 2. Zeile 2 sieht jetzt so aus:

```
var vorname, name, kundenr, email, affe, punkt, i, ergebnis;
```

Lösche außerdem die *confirm()*-Box aus Zeile 27 und füge diesen Code ein:

```
ergebnis=document.abfrage.anzahl.value*75;
ergebnis=ergebnis+12;
if (document.abfrage.express.checked) {
ergebnis=ergebnis+20;
}
return confirm("Hallo, " + vorname + " " + name + ", \nSie bezahlen " + ergebnis + " EUR!");
```

Die erste Zeile liest den Wert des Feldes *anzahl* aus und multipliziert ihn mit 75. Das Ganze wird in der Variablen *ergebnis* gespeichert. Außerdem werden zu diesem Wert 12 dazugerechnet, die Versandkosten. Wenn der Anwender die Express-Lieferung ausgewählt hat, wird zum Ergebnis zusätzlich 20 addiert. Und jetzt gibt die neue *confirm()*-Box noch einmal Namen, Vornamen und Gesamtergebnis aus.

Du wunderst dich über das Zeichen *\n* in der *confirm()*-Box? Damit erzeugst du in einem Dialogfenster einen Zeilenvorschub. Das *\n* steht für new line. Du kannst sogar mit *\t* einen Tabsprung erzeugen. Du kannst das Zeichen aber auch weglassen, denn es ist eine rein optische Geschichte!

Das Ergebnis findest du in der Datei *form5.htm*. Nach diesem Muster kannst du weitergehende Formularauswertungen programmieren.