7 Formulare überprüfen

In Sachen Formular-Handling hat ASP.NET eine ganze Reihe von neuen Konzepten eingeführt, wovon Sie sich insbesondere in den beiden vorhergehenden Kapiteln überzeugen konnten. Die einzige Aufgabe, die wirklich noch mühsam von Hand durchgeführt werden musste, war die Vollständigkeitsüberprüfung. Allerdings haben wir immer darauf hingewiesen, dass es hierzu noch eine Vereinfachung gibt.

Die Rede ist von ASP.NET Validation Controls. Das sind ASP.NET Controls, die den Zweck haben, die Eingaben in einem Formularfeld zu überprüfen. Möglichkeiten der Überprüfung gibt es viele. Hier einige Beispiele:

- Das Formularfeld muss ausgefüllt werden.
- Die Eingabe im Formularfeld muss bestimmte Kriterien erfüllen (beispielsweise eine Zahl sein).
- Die Eingabe im Formularfeld muss eine bestimmte Länge haben.
- Die Eingabe muss einem bestimmten Muster entsprechen (Stichwort reguläre Ausdrücke).
- Die Eingabe wird mit einer frei zu bestimmenden Überprüfungsfunktion validiert.

7.1 Worum geht es?

Als Einführung in die Thematik zunächst ein kleines Beispiellisting. Sie sehen dort ein Formular mit einem Textfeld und ein ASP.NET-Control:

```
<asp:RequiredFieldValidator
  ControlToValidate="eingabe"
  ErrorMessage="Bitte füllen Sie das Feld aus!"
  runat="server" />
```

Werfen Sie zunächst einen Blick auf das Listing:

Listing 7.1: Ein erstes Beispiel (*beispiel.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  <asp:TextBox id="eingabe" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="eingabe"
    ErrorMessage="Bitte füllen Sie das Feld aus!"
    runat="server" />
</form>
</body>
</html>
```

Laden Sie das Beispiel in Ihren Webbrowser. Sie sehen zunächst das Eingabefeld, die Schaltfläche zum Versenden und dazwischen einen leeren Bereich (siehe Abbildung 7.1).

Abbildung 7.1: Das Formular nach dem Laden

Der Sinn und Zweck des leeren Bereichs offenbart sich, wenn Sie versuchen, das Formular leer abzuschicken. Im leeren Bereich wird eine Fehlermeldung angezeigt, in auffallendem Rot.

Wenn Sie genau hingeschaut haben, werden Sie festgestellt haben, dass diese Fehlermeldung eingeblendet worden ist, ohne dass der Browser eine Verbindung zum Webserver aufgebaut hat. Sie ahnen es vermutlich – hier ist die clientseitige Scriptsprache JavaScript mit im Spiel. Werfen wir einen Blick auf den Quellcode der Seite; auf den HTML-Code der vom Server an den Browser geschickt wird, wohlgemerkt. Da er so umfangreich ist, zeigen wir ihn nur in Auszügen. Beginnen wir mit dem `<form>`-Element:

Abbildung 7.2: Die Fehlermeldung wird angezeigt.

```
<form name="_ctl0" method="post" action="beispiel.aspx"
language="javascript" onsubmit="ValidatorOnSubmit();" id="_ctl0">
```

JavaScript-Kenner wissen sofort Bescheid:

Durch die Anweisung `onsubmit="ValidatorOnSubmit();"` wird beim Versand des Formulars eine JavaScript-Funktion namens `ValidatorOnSubmit()` aufgerufen. Nach etwas Suchen ist auch klar, woher diese Funktion kommt:

```
<script language="javascript" src="/aspnet_client/system_web/
   1_0_3705_0/WebUIValidation.js"></script>
```

Mit diesem Kommando wird die Datei `WebUIValidation.js` eingebunden, die sich auf Ihrem lokalen System befindet und die mit dem .NET-Framework-SDK mitgeliefert wird. Sie enthält eine Reihe von mehr oder minder komplexen Überprüfungsfunktionen.

Woher kommt aber nun die Fehlermeldung? Weiter unten in der Seite befindet sich noch ein `<span>`-Element:

```
<span id="_ctl1" controltovalidate="eingabe" errormessage="Bitte füllen
   Sie das Feld aus!"
   evaluationfunction="RequiredFieldValidatorEvaluateIsValid"
   initialvalue="" style="color:Red;visibility:hidden;">Bitte füllen
   Sie das Feld aus!</span>
```

Die Fehlermeldung ist also schon die ganze Zeit präsent. Durch die Stil-Anweisung `visibility:hidden` wird sie jedoch zunächst nicht angezeigt.

Zu guter Letzt ist auch noch die VERSENDEN-Schaltfläche mit JavaScript gespickt. Wenn Sie auf die Schaltfläche klicken, wird ebenfalls eine Überprüfungsfunktion angestoßen:

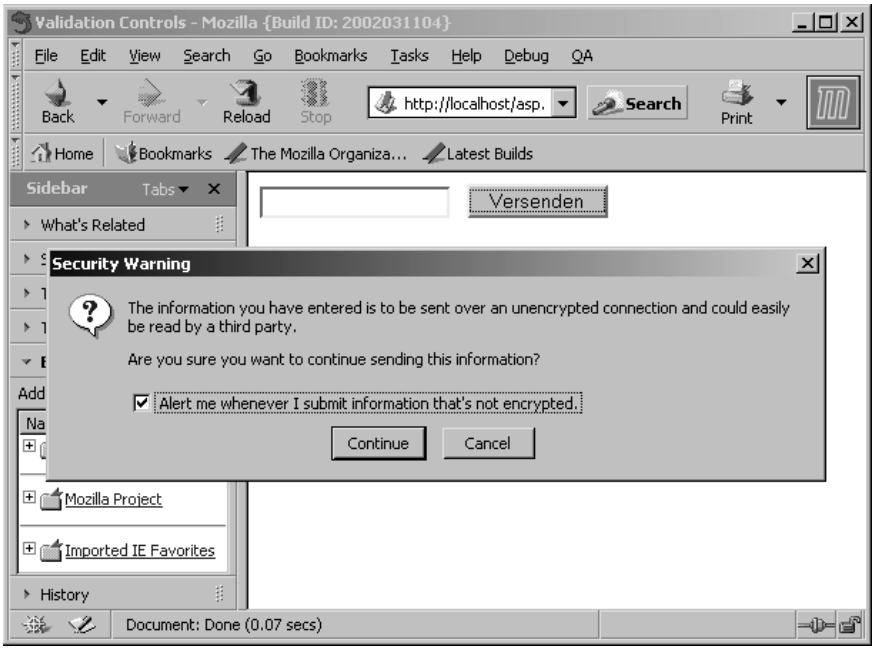
```
<input language="javascript" onclick="if (typeof(Page_ClientValidate)
== 'function') Page_ClientValidate(); " name="_ctl2" type="submit"
value="Versenden" />
```

Der verwendete JavaScript-Code ist ziemlich komplex und sehr auf den Microsoft Internet Explorer zugeschnitten (ein Schelm, der Absicht dahinter vermutet ...). Wenn Sie dieselbe Seite beispielsweise im Mozilla-Browser aufrufen, erhalten Sie einen etwas anderen Code. Betrachten wir das `<form>`-Element:

```
<form name="_ctl0" method="post" action="beispiel.aspx" id="_ctl0">
```

Sie sehen – es enthält kein JavaScript. Das gesamte Dokument enthält nur ganz wenig JavaScript-Code und Sie können das Formular auch leer verschicken (siehe Abbildung 7.3). Nach dem Versand wird das Formular jedoch wieder angezeigt, dieses Mal mit der Fehlermeldung (siehe Abbildung 7.4).

Abbildung 7.3: Das Formular kann leer abgeschickt werden ...



Daran können Sie sehen, dass Sie auch hier auf mehreren verschiedenen Browsern testen sollten. Insbesondere sollten Sie jederzeit damit rechnen, dass die Formulardaten an den Server geschickt und dort erst überprüft werden. Viele Benutzer schalten JavaScript aufgrund in jüngster Zeit aufgetretener Sicherheitslücken aus. Dann kann die Überprüfung nicht clientseitig durchgeführt werden.

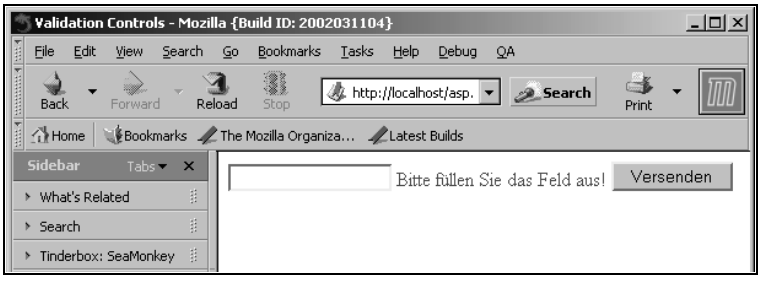


Abbildung 7.4: ... die Freude ist jedoch nur von kurzer Dauer

7.2 Validation Controls

Bevor wir nun die einzelnen Überprüfungsmöglichkeiten vorstellen, sei noch darauf hingewiesen, dass nicht alle Formularfeldtypen und Controls per ASP.NET Validation Controls überprüft werden können. Sie können Tabelle 7.1 die HTML Controls entnehmen, die überprüft werden können; in Tabelle 7.2 finden Sie die validierbaren Web Controls. Andere Controls können Sie nicht überprüfen.

Formularfeldtyp	HTML-Code	Überprüfte Eigenschaft
Textfeld (HtmlInputText)	`<input type="text" runat="server" />`	Value
Passwortfeld (HtmlInputText)	`<input type="password" runat="server" />`	Value
Mehrzeiliges Textfeld (HtmlTextarea)	`<textarea runat="server" />`	Value
Auswahlliste (HtmlSelect)	`<select runat="server">...</select>`	Value
File-Upload (HtmlInputFile)	`<input type="file" runat="server" />`	Value

Tabelle 7.1: Per Validation Controls überprüfbare HTML Controls

Formularfeldtyp	Web Control	Überprüfte Eigenschaft
Textfeld	`<asp:TextBox />`	Text
Passwortfeld	`<asp:TextBox />`	Text
Mehrzeiliges Textfeld	`<asp:TextBox />`	Text

Tabelle 7.2: Per Validation Controls überprüfbare Web Controls

Tabelle 7.2: Per Validation Controls überprüfbare Web Controls (Forts.)

Formularfeldtyp	Web Control	Überprüfte Eigenschaft
Gruppe aus Radiobuttons	`<asp:RadioButtonList />`	`Value` des gewählten Radiobuttons
Auswahlliste	`<asp:DropDownList />` `<asp:ListBox />`	`Value` des (ersten) gewählten Elements

Mit diesem Vorwissen ausgestattet, wollen wir Ihnen nun die verschiedenen Überprüfungsmöglichkeiten einzeln vorstellen.

7.2.1 Pflichtfelder: RequiredFieldValidator

Das einfachste Validation Control (und das, das Sie wohl am häufigsten einsetzen werden), ist `RequiredFieldValidator`. Es werden also *required fields* (Pflichtfelder) überprüft. Das Control sieht folgendermaßen aus:

```
<asp:RequiredFieldValidator runat="server" />
```

Sie können dieses Control mit einer Reihe von Parametern näher spezifizieren. Die Wichtigsten sind die beiden Folgenden:

- `ControlToValidate` – die ID des Formularelements, das überprüft werden soll
- `ErrorMessage` – die Fehlermeldung, die ausgegeben werden soll.

Sie können auch in der Eigenschaft `Text` *eine Fehlermeldung angeben; Details hierzu finden Sie in Abschnitt 7.3.1.*

Der Einsatz ist also ganz einfach: Sie fügen das Validation Control an einer beliebigen Stelle in Ihrem Formular ein. Wenn das Formular abgeschickt werden soll (falls Internet Explorer vorliegt und JavaScript aktiviert ist) oder abgeschickt wurde (andere Browser oder Internet Explorer oder JavaScript), wird die Eingabe im angegebenen Fomularfeld überprüft. Wenn diese Überprüfung fehlschlägt, wird eine Fehlermeldung ausgegeben. Diese Fehlermeldung erscheint exakt an der Stelle, an der das Validation Control in die Seite eingesetzt worden ist.

Nachfolgend ein Beispiel mit drei verschiedenen Formularfeldern: einem Textfeld, einem Passwortfeld und einem mehrzeiligen Textfeld:

Listing 7.2: Pflichtfelder werden überprüft (*requiredfieldvalidator.aspx*).


```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  Login:
  <input type="text" id="Textfeld" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Textfeld"
    ErrorMessage="Geben Sie Ihr Login an!"
    runat="server" />
  <br />
  Passwort:
  <input type="password" id="Passwortfeld"
         runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Passwortfeld"
    ErrorMessage="Haben Sie Ihr Passwort vergessen?"
    runat="server" />
  <br />
  Sonstige Kommentare:
  <textarea id="Mehrzeilig" wrap="virtual"
            runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Mehrzeilig"
    ErrorMessage="Möchten Sie nicht etwas loswerden?"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

In Abbildung 7.5 sehen Sie das Formular nach Anklicken der VERSENDEN-Schaltfläche. Obwohl im Passwortfeld offensichtlich eine Angabe gemacht wurde, wurde eine Fehlermeldung ausgegeben. Was ist passiert? Nun, wir haben in das Passwortfeld drei Leerzeichen eingegeben. Außerdem haben wir im mehrzeiligen Textfeld ein paar Mal die ↵-Taste betätigt. In den vorherigen Kapiteln haben wir denselben Überprüfungseffekt erreicht, indem wir die Eingaben in die jeweiligen Formularfelder mit `Trim()` vorbehandelt und Leerzeichen am Anfang und am Ende entfernt haben. Der `RequiredFieldValidator` geht genauso vor und ignoriert so genannte `Whitespace`-Zeichen am Anfang und am Ende der Eingabe: Leerzeichen und Zeilenwechsel.

Wenn Sie dies nachvollziehen, im Internet Explorer das Formular laden, ein paar Leerzeichen z.B. in das Passwortfeld eingeben und dann ein anderes Feld auswählen (beispielsweise mit der ⇆-Taste), wird Ihnen auffallen, dass die Fehlermeldung sofort dargestellt wird (siehe Abbildung 7.6).

Abbildung 7.5: Drei Fehlermeldungen

Abbildung 7.6: Die Fehlermeldung wird sofort angezeigt.

Sie ahnen sicher, dass hier wieder JavaScript mit im Spiel ist. Und so praktisch die sofortige Anzeige auch ist, sie hat zwei kleine Nachteile:

- Erstens: Wenn jemand ein Formular nicht von oben nach unten, sondern in einer anderen Reihenfolge ausfüllt, wirken die Fehlermeldungen doch ein wenig störend.
- Und zweitens: Diese sofortige Einblendung funktioniert bei Version 1.0 des .NET-Frameworks ausschließlich beim Microsoft Internet Explorer; es ist nicht bekannt, ob sich das bei zukünftigen Versionen ändern wird.

Sie könnten daraus für sich den Wunsch ableiten, die sofortige Einblendung deaktivieren zu wollen. Hierzu gibt es zwei Möglichkeiten:

1. Setzen Sie die Eigenschaft `EnableClientScript` des Validation Controls auf `"False"`.
2. Fügen Sie die folgende Direktive in den Kopf der Seite ein:

   ```
   <%@ Page ClientTarget="downlevel" %>
   ```

   Wenn Sie bereits eine `Page`-Direktive verwenden, dürfen Sie keine zweite Direktive in die Seite einfügen; ergänzen Sie stattdessen in der existierenden Direktive den `ClientTarget`-Parameter:

   ```
   <%@ Page Language="c#" ClientTarget="downlevel" %>
   ```

Letztere Methode (also die `Page`-Direktive) führt allerdings auch dazu, dass die gesamte resultierende HTML-Seite (zumindest alle von ASP.NET serverseitig generierten Elemente) dem HTML 3.2-Standard entspricht. Die aktuelle (und wohl auch finale) Version ist 4.01, woran Sie schon sehen können, dass dann möglicherweise einiges fehlt.

Diese Hinweise gelten auch für alle folgenden Validation Controls.

Auf vielen Websites sind Texteingabefelder vorbelegt, beispielsweise auf `http://login.passport.com/` mit `"<Geben Sie Ihre E-Mail-Adresse ein>"`. Nun wäre es ziemlich ärgerlich, wenn der Benutzer diese Vorausfüllung bestehen lassen würde. Im Parameter `InitialValue` des `RequiredFieldValidator` können Sie den ursprünglichen Text für das Formularelement angeben. Das Validation Control überprüft dann, ob der Benutzer etwas anderes als die Vorausfüllung angegeben hat:

```
<asp:RequiredFieldValidator
  ControlToValidate="Feldname"
  ErrorMessage="Möchten Sie nicht etwas loswerden?"
  InitialValue="&lt;Geben Sie Ihre E-Mail-Adresse ein&gt;"
  runat="server" />
```

7.2.2 Eingaben im Intervall: RangeValidator

Oft müssen Feldeingaben innerhalb gewisser Grenzen liegen: beispielsweise, wenn in ein Textfeld ein Monat eingegeben werden soll. Dieser muss einen Wert zwischen 1 und 12 annehmen. Hierfür können Sie den `RangeValidator` verwenden, der überprüft ob ein Formularfeld einen Wert innerhalb eines definierten Intervalls hat:

```
<asp:RangeValidator runat="server" />
```

Dieser Validator funktioniert nicht bei jedem Wertetyp; folgende Werte sind erlaubt:

- `Currency`
- `Date`
- `Double`
- `Integer`
- `String`

Um nun einen Wert zu überprüfen, müssen Sie die folgenden Parameter des Validation Controls setzen:

- `ControlToValidate` – die ID des zu überprüfenden Formularelements
- `ErrorMessage` – die Fehlermeldung, die ausgegeben werden soll, wenn die Überprüfung fehlschlägt
- `Type` – der beim Überprüfen zu verwendende Datentyp (siehe vorherige Auflistung)
- `MinimumValue` – Untergrenze des gültigen Intervalls
- `MaximumValue` – Obergrenze des gültigen Intervalls

Ober- und Untergrenze, also `MaximumValue` *und* `MinimumValue`*, sind jeweils inklusive.*

Um das Beispielszenario von zuvor aufzugreifen: Folgender `RangeValidator` würde überprüfen, ob in einem Textfeld ein gültiger Monat von 1 bis 12 eingegeben wurde:

```
<asp:RangeValidator
  ControlToValidate="Feldname"
  ErrorMessage="Das ist kein gültiger Monat!"
  Type="Integer"
  MinimumValue="1"
  MaximumValue="12"
  runat="server" />
```

Eine andere Möglichkeit besteht darin, ein komplettes Geburtsdatum zu überprüfen. Als Untergrenze verwenden wir hier den 1. Januar 1900 (wer vor diesem Datum geboren ist, möge uns diese Einschränkung verzeihen), als Obergrenze 18 Jahre vom aktuellen Datum aus. Damit können nur volljährige Personen das Formular »korrekt« ausfüllen. Setzen Sie also `Type` auf `"Date"` und ändern Sie die anderen Parameter entsprechend.

```
<asp:RangeValidator
  ControlToValidate="Feldname"
  ErrorMessage="Sie sind vermutlich nicht volljährig!"
  Type="Date"
  MinimumValue="1.1.1900"
  runat="server" />
```

Beachten Sie, dass wir den Parameter `MaximumValue` nicht gesetzt haben. Das liegt daran, dass sich der Wert dieses Parameters jeden Tag wieder ändert; wir müssen ihn also dynamisch setzen. Dies funktioniert sehr einfach, indem wir in der **Funktion** `Page_Load` diesen Wert anpassen:

```
Feldname.MaximumValue = DateTime.Now.Date.AddYears(-18)
```

Wenn Sie in das Textfeld keinen Wert eingeben, schlägt der Validator keinen Alarm! Sie müssen also gegebenenfalls das `RangeValidator`*-Control mit einem* `RequiredFieldValidator`*-Control kombinieren.*

Nachfolgend nun ein komplettes Listing, in dem wir beide `RangeValidator`-Elemente einsetzen:

Listing 7.3: Zwei `RangeValidator`-Controls (*rangevalidator.vb.aspx*)


```
<%@ Page Language="vb" %>
<script runat="server">
Sub Page_Load
  GeburtVal.MaximumValue = _
    DateTime.Now.Date.AddYears(-18)
End Sub
</script>
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  In welchem Monat sind Sie geboren? (1..12)
  <input type="text" id="Monat"
         size="2" maxlength="2"
         runat="server" />
  <asp:RangeValidator
    ControlToValidate="Monat"
    ErrorMessage="Das ist kein gültiger Monat!"
    Type="Integer"
    MinimumValue="1"
    MaximumValue="12"
    runat="server" />
  <br />
  Ihr komplettes Geburtsdatum bitte: (tt.mm.jjjj)
  <input type="text" id="Geburt"
         size="10" maxlength="10"
         runat="server" />
  <asp:RangeValidator
```

```
    id="GeburtVal"
    ControlToValidate="Geburt"
    ErrorMessage="Sie sind vermutlich nicht volljährig!"
    Type="Date"
    MinimumValue="1.1.1900"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

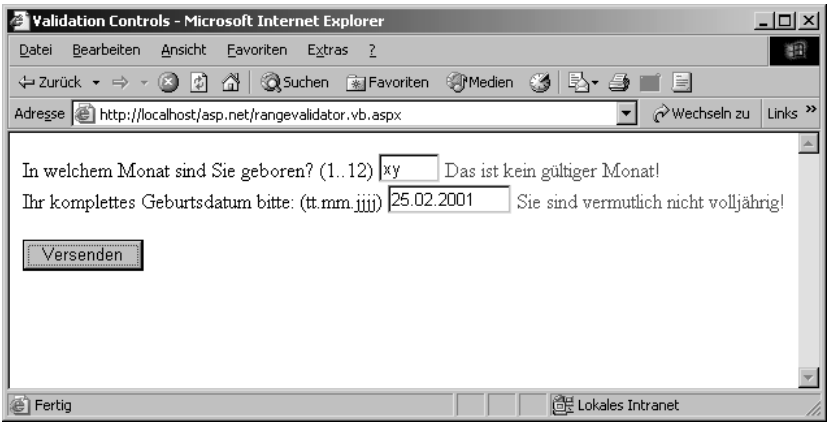


Abbildung 7.7: Ungültige Angaben werden abgefangen.

:-) TIPP

Wie Sie Abbildung 7.7 entnehmen können, wird auch eine Typüberprüfung vorgenommen – die Zeichenkette `"xy"` *wird als nicht-numerischer Wert erkannt. Wenn Sie* `MinimumValue` *und* `MaximumValue` *nicht angeben, können Sie sehr bequem feststellen, ob ein Eingabewert beispielsweise ein Datumswert (*`Date`*) oder ein numerischer Wert ist.*

7.2.3 Werte vergleichen: CompareValidator

Wenn Sie sich bei einem Dienst, welcher Art auch immer, anmelden und ein Passwort wählen, fordern Sie viele Anbieter dazu auf, zwei Passwortfelder auszufüllen. Der Grund: Da bei der Passworteingabe auf dem Bildschirm pro Zeichen nur ein Sternchen, nicht aber die Eingabe selbst angezeigt wird, passieren oft Fehler – insbesondere bei ungeübten Tippern.

Die Eingaben in beiden Feldern müssen also übereinstimmen. Ein Fall für das `CompareValidator`-Control:

```
<asp:CompareValidator runat="server" />
```

Das Control vergleicht die Eingaben in zwei Textfeldern miteinander. Dazu müssen Sie die folgenden Parameter setzen:

- `ControlToValidate` – ID des Formularelements, das überprüft werden soll
- `ControlToCompare` – ID des Formularelements, das den Wert enthält, mit dem verglichen werden soll
- `ErrorMessage` – Fehlermeldung, die ausgegeben werden soll, wenn der Vergleich fehlschlägt.

Bitte achten Sie unbedingt darauf, mit welchen Werten Sie `ControlToValidate` und `ControlToCompare` belegen! Betrachten Sie folgendes Beispiel:

Listing 7.4: Die Felder werden auf Übereinstimmung geprüft (*comparevalidator1.aspx*).


```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  W&auml;hlen Sie Ihr Passwort:
  <input type="password" id="Passwort1"
         runat="server" />
  <br />
  Wiederholen Sie Ihr Passwort:
  <input type="password" id="Passwort2"
         runat="server" />
  <asp:CompareValidator
    ControlToValidate="Passwort1"
    ControlToCompare="Passwort2"
    ErrorMessage="Die Passwörter stimmen nicht überein!"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Wenn Sie nun in das erste **Eingabefeld einen Wert eingeben und die** ⇥-Taste betätigen (oder mit der Maus in das zweite Passwortfeld klicken), wird die Fehlermeldung sofort angezeigt (siehe Abbildung 7.8). Das ist auch nahe liegend, denn gemäß dem obigen Code soll das Feld mit `id="Passwort1"` **validiert** werden. Also findet sofort eine Überprüfung statt, wenn dieses Feld ausgefüllt worden ist. Der einfachste Weg ist, das zweite **Passwortfeld validieren zu lassen**; als Vergleichsbasis dient der Wert des ersten Passwortfelds:

```
<asp:CompareValidator
  ControlToValidate="Passwort2"
  ControlToCompare="Passwort1"
  ErrorMessage="Die Passwörter stimmen nicht überein!"
  runat="server" />
```

Den entsprechend korrigierten Code finden Sie auf der ***Buch-CD-ROM*** *unter dem Dateinamen comparevalidator2.aspx.*

Abbildung 7.8: Die Fehlermeldung wird zu früh ausgegeben.

Wenn wir bisher das Wort »vergleichen« verwendet haben, wurde es immer implizit mit »Gleichheit« gleichgesetzt. Es ist aber mit dem `CompareValidator`-Control ohne weiteres möglich, andere Vergleichsoperatoren zu bedienen. Dazu dient der Parameter `Operator`. Nachfolgend eine Auflistung der möglichen Werte und ihre Bedeutung:

Listing 7.5: Die gültigen Werte für den Parameter `Operator`

Operator	Bedeutung
`DataTypeCheck`	Überprüft den Datentyp der Eingabe (Datentyp kann im Parameter `Type` angegeben werden)
`Equal`	Gleich
`GreaterThan`	Größer als
`GreaterThanEqual`	Größer oder gleich
`LessThan`	Kleiner als
`LessThanEqual`	Kleiner oder gleich
`NotEqual`	Ungleich

Ein ***Hinweis zum Wert*** `DataTypeCheck`*: Damit können Eingaben daraufhin überprüft werden, ob sie eines bestimmten Typs sind. Wie Sie zuvor gesehen haben, funktioniert das allerdings auch mit dem* `RangeValidator`*-Control. So würde eine Überprüfung auf ein Datum mit dem* `CompareValidator`*-Control aussehen:*

```
<asp:CompareValidator
```

```
  ControlToValidate="Feldname"
  Operator="DateTypeCheck"
  Type="Date"
  runat="server" />
```

Mit den Operatorwerten aus Tabelle 7.3 können Sie nun wie gewohnt arbeiten. Im folgenden Beispiel muss der Benutzer bei einer fiktiven virtuellen Partnervermittlung das Wunschalter des Wunschpartners / der Wunschpartnerin angeben, und zwar als Intervall. Da ist es klar, dass die Obergrenze nicht kleiner als die Untergrenze sein darf:

Listing 7.6: Der Operator `GreaterThanEqual` im Einsatz (*comparevalidator3.aspx*)


```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  Mindestalter des Wunschpartners:
  <input type="text" id="Min" runat="server" />
  <br />
  H&ouml;chstalter des Wunschpartners:
  <input type="text" id="Max" runat="server" />
  <asp:CompareValidator
    ControlToValidate="Max"
    ControlToCompare="Min"
    Operator="GreaterThanEqual"
    Type="Integer"
    ErrorMessage="So werden Sie nie fündig!"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Sie müssen für den Vergleich unbedingt `Type="Integer"` *setzen. Andernfalls würde ein String-Vergleich durchgeführt, bei dem beispielsweise der Wert* `"9"` *größer als* `"10"` *wäre (es wird zeichenweise verglichen, und* `"1"` *kommt vor* `"9"`*). Allerdings gibt es vermutlich keine Partnervermittlungen, die Minderjährige anpreisen, noch dazu neunjährige.*

7.2.4 Musterprüfung: RegularExpressionValidator

Die bisherigen Validatoren waren zwar sehr mächtig, es gab aber immer noch Limitierungen. Ein Beispiel sind beispielsweise Postleitzahlen. Eine deutsche Postleitzahl besteht aus fünf Ziffern. Mit den bisherigen Mitteln war es jedoch nicht möglich, dies zu überprüfen. Hier einige Ansätze für dieses Problem – und wieso sie nicht funktionieren würden:

Abbildung 7.9: Dieses Intervall ist nicht gültig.

- Mit einem `RequiredFieldValidator` könnte überprüft werden, ob in dem Feld etwas steht. Dann könnte aber auch eine beliebige Zeichenkette eingegeben werden, ohne dass der Validator Alarm schlägt.
- Mit einem `RangeValidator` oder `CompareValidator` (Option `DateTypeCheck`) könnte überprüft werden, ob der eingegebene Wert ein Integerwert ist. Allerdings ist die Postleitzahl 01234 wohl eine gültige Postleitzahl, aufgrund der führenden Null aber kein Integerwert. Außerdem wäre 100.000 zwar ein Integerwert, aber keine gültige Postleitzahl.
- Zusätzlich zur Typüberprüfung auf Integer könnte man außerdem mit einem `RangeValidator` überprüfen, ob der Wert im Textfeld zwischen `"00000"` und `"99999"` liegt. Dazu würde allerdings ein String-Vergleich angestellt, den auch die Zeichenkette `"1234X"` bestehen würde.

Sie sehen also, die Überprüfung einer Postleitzahl könnte so nicht realisiert werden. Hier kommt das `RegularExpressionValidator`-Control ins Spiel, das Formulareingaben gegen einen regulären Ausdruck überprüft:

```
<asp:RegularExpressionValidator runat="server" />
```

Dieses Control erwartet die bereits bekannten Parameter:

- `ControlToValidate` – die ID des Formularelements, das überprüft werden soll
- `ErrorMessage` – die Fehlermeldung, die ausgegeben werden soll, falls die Überprüfung fehlschlägt
- `ValidationExpression` – der reguläre Ausdruck, gegen den die Formularwerte überprüft werden sollen

Reguläre Ausdrücke

Die Frage ist nun – was sind reguläre Ausdrücke überhaupt? Sollten Sie bereits mit dem entsprechenden Vorwissen ausgestattet sein, können Sie diesen Abschnitt getrost überspringen und etwas später wieder einsteigen, wenn wir das Postleitzahlenproblem lösen. Für alle anderen nun ein Schnellkursus in Sachen reguläre Ausdrücke.

Eine vollständige Einführung in die Materie ist an dieser Stelle leider nicht möglich. Daher beschränken wir uns auf die wichtigsten Sprachelemente, die Sie bei dem `RegularExpressionValidator` *benötigen. Wenn Sie sich für dieses Thema interessieren, können wir Ihnen das inoffizielle Standardwerk über reguläre Ausdrücke ans Herz legen: Jeffrey E. F. Friedl, »Reguläre Ausdrücke«. Das Buch ist aus dem Jahr 1997 und daher in gewissen Bereichen leider nicht mehr up-to-date. Dennoch gibt es kaum bessere Veröffentlichungen zu diesem Thema.*

Zurück zum Thema: Ein regulärer Ausdruck ist zunächst einmal eine Beschreibung eines Musters, nicht mehr und nicht weniger. `ASP` ist beispielsweise ein solches Muster. Sein informativer Inhalt ist: Zuerst ein `A`, gefolgt von einem `S` und einem `P`. Das alleine ist ja noch nichts Besonderes, denn eine Überprüfung auf Zeichenketten funktioniert auch mit den Bordmitteln jeder Programmiersprache und zur Not über die Klasse `System.String`. Innerhalb des Musters können nämlich auch bestimmte Anweisungen und Sonderzeichen enthalten sein.

Zunächst gibt es Zeichen, die angeben, wie oft ein Zeichen in einer Zeichenkette vorkommen darf. `A?` bedeutet beispielsweise, dass das Zeichen `A` entweder gar nicht oder ein Mal vorkommt. `A*` dagegen steht für das Zeichen `A`, das beliebig oft vorkommt: `A`, `AA`, `AAA`, oder auch null Mal.

Diese Sonderzeichen (bisher haben Sie `?` und `*` kennen gelernt) werden auch *Multiplikatoren* genannt. Nachfolgend finden Sie eine Übersicht:

Multiplikator	Bedeutung	Beispiel	Muster passt auf ...
`?`	null- oder ein Mal	`A?B`	`B, AB`
`*`	beliebig oft, auch null Mal	`A*B`	`B, AB, AAB, AAAB, ...`
`+`	beliebig oft, mindestens ein Mal	`A+B`	`AB, AAB, AAAB, ...`
`{x}`	exakt x-mal	`A{2}B`	`AAB`

Tabelle 7.3: Die Multiplikatoren für reguläre Ausdrücke

Tabelle 7.3: Die Multiplikatoren für reguläre Ausdrücke (Forts.)

Multiplikator	Bedeutung	Beispiel	Muster passt auf ...
`{x,y}`	mindestens x-mal, höchstens y-mal	`A{2,4}B`	`AAB, AAAB, AAAAB`
`{x,}`	mindestens x-mal	`A{2,}B`	`AAB, AAAB, AAAAB, ...`

Auch bei Hinzunahme der Multiplikatoren fehlen zum ordentlichen Arbeiten mit regulären Ausdrücken noch weitere Elemente. Beispielsweise bietet der bisher vorgestellte »Sprachschatz« von regulären Ausdrücken noch keine Alternativmöglichkeit , also »entweder A oder B«. Dies kann mit zwei Operatoren behoben werden.

Zunächst werden die eckigen Klammern angegeben, `[]`: Von allen Zeichen innerhalb der eckigen Klammer wird genau eines gewählt. Auf `[ABC]` passen also `A`, `B` und `C`.

:-) TIPP

Durch den Bindestrich können Sie ein »von ... bis« innerhalb von eckigen Klammern realisieren. Anstelle des langen Ausdrucks

```
[ABCDEFGHIJKLMNOPQRSTUVWXYZ]
```

können Sie kürzer schreiben:

```
[A-Z]
```

`|` – Der senkrechte Strich (auch *Pipe* genannt, AltGr+<) trennt zwei Alternativen. Auf das Muster `AB|CD` passen also `AB` und `CD`.

:-) TIPP

Beim senkrechten Strich wird versucht, links und rechts möglichst viele Elemente anzugeben. Auf das Muster `AB|CD` *passen also nicht die Werte* `B` *oder* `C`. *Um den Auswirkungsbereich von* `|` *einzugrenzen, können Sie runde Klammern verwenden. Das Muster* `A(B|C)D` *passt auf* `ABD` *und* `ACD`.

Zu guter Letzt möchten wir noch einige Sonderzeichen vorstellen, die innerhalb eines Musters eine besondere Bedeutung haben. Die meisten dieser Zeichen werden durch einen Backslash eingeleitet. Er hat innerhalb eines regulären Ausdrucks eine besondere Bedeutung und entwertet das darauf folgende Zeichen *oder* gibt dem folgenden Zeichen eine besondere Bedeutung. Insbesondere der Aspekt der Entwertung ist in der Praxis sehr wichtig. Stellen Sie sich vor, Sie benötigten in Ihren Mustern unbedingt runde Klammern; diese haben jedoch wie oben gesehen eine besondere Bedeutung. Um eine solche Klammer zu entwerten, ihr also die besondere Funktion der Abgrenzung bestimmter Abschnitte im regulären Ausdruck zu entziehen, stellen Sie einen Backslash voraus. Auf das Muster `\(A\)` passt also `(A)`.

Auch der Backslash selbst kann durch einen zweiten Backslash entwertet werden; `C:\\` *steht für* `C:\`.

Nachfolgend eine Auflistung der speziellen Sonderzeichen innerhalb eines regulären Ausdrucks, die jeweils durch einen Backslash eingeleitet werden:

Tabelle 7.4: Sonderzeichen bei regulären Ausdrücken

Sonderzeichen	Bedeutung
`\d`	Ziffer (entspricht also `[0-9]`)
`\D`	Keine Ziffer (also alles außer `[0-9]`)
`\w`	Buchstabe, Ziffer, Satz- oder Leerzeichen
`\W`	Weder Buchstabe noch Ziffer noch Satz- noch Leerzeichen
`\s`	Whitespace, also Leerzeichen, Tabulator, Zeilensprung.
`\S`	Kein Whitespace

Ein Ausschluss bestimmter Zeichen kann auch dadurch erreicht werden, dass Sie eckige Klammern verwenden und als erstes Zeichen `^` *verwenden.* `[^ABC]` *steht also weder für* `A` *noch* `B` *noch* `C`.

Wie bereits angekündigt, gibt es noch weitere Sonderzeichen, die nicht von einem Backslash eingeleitet werden:

Tabelle 7.5: Weitere Sonderzeichen bei regulären Ausdrücken

Sonderzeichen	Bedeutung
`.`	Beliebiges Zeichen
`^`	Anfang der Zeichenkette
`$`	Ende der Zeichenkette

Sie sehen jetzt vielleicht auch, wieso wir als eingehendes Beispiel das Muster `ASP` verwendet haben und nicht das thematisch nahe liegendere `ASP.NET`. Der Punkt hat eine besondere Bedeutung, auf das Muster `ASP.NET` würden also beispielsweise `ASPaNET`, `ASPbNET` und so weiter passen. Um tatsächlich `ASP.NET` auszudrücken, muss das Muster `ASP\.NET` lauten.

Und damit beenden wir den Crashkurs über reguläre Ausdrücke!

RegularExpressionControl einsetzen

Wenn Sie den vorherigen Abschnitt übersprungen haben, willkommen zurück! Auf jeden Fall verfügen Sie nun über das benötigte Grundwissen, um diesen Abschnitt zu verstehen. Die ursprüngliche Problemstellung und gleichzeitig Motivation war das Problem, auf gültige deutsche Postleitzahlen zu überprüfen. Wie wir bereits festgestellt hatten, besteht eine deutsche Postleitzahl aus fünf aufeinander folgenden Ziffern (wir verzichten an dieser Stelle darauf des Weiteren zu überprüfen, ob die Postleitzahl tatsächlich existiert). Sie können dieses Muster nun wie folgt in einen regulärem Ausdruck verwenden:

- Entweder Sie verwenden `[0-9]` für eine Ziffer und setzen das fünfmal hintereinander: `[0-9][0-9][0-9][0-9][0-9]`,
- oder Sie verwenden für Ziffern das Sonderzeichen `\d` und erhalten als regulären Ausdruck das etwas kürzere `\d\d\d\d\d`,
- oder Sie verwenden zusätzlich noch einen Multiplikator und erhalten das kompakte `\d{5}`.

Es ist gleichgültig, für welche der Alternativen Sie sich entscheiden. Am Ende müssen Sie den regulären Ausdruck im Parameter `ValidationExpression` angeben. Der Rest läuft wie gewohnt ab; je nach Browser (und gegebenenfalls Ihren Einstellungen) wird die Überprüfung sofort durchgeführt oder erst serverseitig, nach dem Versand.

Listing 7.7: Überprüfung einer Postleitzahl mit einem regulären Ausdruck (*regexp1.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  Postleitzahl:
  <input type="text" id="PLZ" size="5" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="PLZ"
    ErrorMessage="Ungültige PLZ!"
    ValidationExpression="\d{5}"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Abbildung 7.10: Diese Postleitzahl gab es nur vor der Postleitzahlenreform.

Es gibt natürlich noch unzählige weitere Einsatzmöglichkeiten für reguläre Ausdrücke. Das wohl häufigste Anwendungsgebiet ist jedoch die Überprüfung der Gültigkeit einer E-Mail-Adresse.

In dem zuvor schon erwähnten Buch über reguläre Ausdrücke nimmt das Muster für E-Mail-Adressen vier Seiten ein. Es ist also sehr mühsam, eine E-Mail-Adresse auf syntaktische Gültigkeit zu überprüfen. Und selbst wenn Ihnen das gelingt, haben Sie immer noch keine Gewissheit, dass diese Adresse auch existiert und zu der Person gehört, die das Formular ausgefüllt hat. Viele paranoide Gesellen verwenden statt ihrer eigenen Adresse lieber die eines Feindbildes, sei es Bill Gates oder Linus Torvalds.

!! STOP

Der Hauptzweck einer Validierung von E-Mail-Adressen dient dazu, den Ausfüller des Formulars auf Fehler hinzuweisen. Bei dem Onlinedienst AOL beispielsweise ist es so, dass sich die E-Mail-Adressen nach dem Muster `Benutzername@aol.com` zusammensetzen. Innerhalb von AOL kommunizieren die einzelnen Mitglieder jedoch nur über die Benutzernamen, ohne angehängtes `@aol.com`. Es ist also nachvollziehbar, dass ein unbedarfter Benutzer in das Formular nur seinen AOL-Benutzernamen eingibt, aber das Anhängsel vergisst.

Bevor Anwälte hinzugezogen werden: Das kann natürlich nicht nur bei AOL passieren. Aufgefallen ist es den Autoren allerdings bisher ausschließlich bei AOL-Kunden. :-)

!! STOP

Die erste Überlegung besteht darin, festzustellen, welches Muster der vordere Teil einer E-Mail-Adresse erfüllen muss und welchem Muster die E-Mail-Adresse entspricht. Erlaubte Zeichen sind:

- Buchstaben `[a-zA-Z]`
- Punkt (`.`)
- Unterstrich (`_`)
- Bindestrich (`-`)

Der folgende reguläre Ausdruck repräsentiert Zeichenketten, die aus den oben angeführten Zeichen bestehen:

```
[a-zA-Z._\-]
```

Ein regulärer Ausdruck für E-Mail-Adressen sollte folgende Elemente enthalten (lesen Sie von oben nach unten):

Tabelle 7.6: Die einzelnen Bestandteile des regulären Ausdrucks

Element	Entsprechung im regulären Ausdruck
Anfang der Zeichenkette (sonst wäre auch `#email@adresse.de#` gültig, weil ja eine E-Mail-Adresse enthalten ist)	`^`
Zunächst eine beliebige Zeichenfolge,	`[a-zA-Z._\-]+`
dann ein Klammeraffe,	`@`
dann wieder eine beliebige Zeichenfolge, die aus mindestens zwei Zeichen bestehen muss (Domainnamen sind mindestens zwei Zeichen lang, in Deutschland mindestens drei Zeichen lang, es gibt lediglich vier Ausnahmen),	`[a-zA-Z._\-]{2,}`
ein Punkt,	`\.`
danach die Domain-Endung, zuzeit zwischen zwei und vier Zeichen. Das kann sich allerdings ändern, die *TLD* (Top-Level-Domain, die Angabe hinter dem letzten Punkt) `.museum` wurde bereits verabschiedet.	`[a-zA-Z]{2,4}`
Ende der Zeichenkette (siehe oben)	`$`

Insgesamt erhalten wir also folgenden regulären Ausdruck:

```
^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.[a-zA-Z]{2,4}$
```

Er lässt sich analog in ein `RegularExpressionValidator`-Control einbauen:

Listing 7.8: Überprüfung einer E-Mail-Adresse mit einem regulären Ausdruck (*regexp2.aspx*)


```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    ErrorMessage="Komische E-Mail-Adresse ..."
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.
                          [a-zA-Z]{2,4}$"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

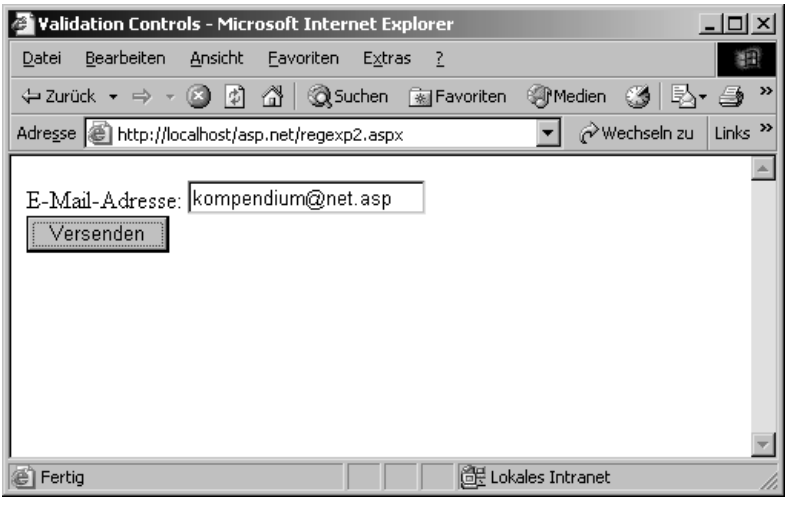


Abbildung 7.11: Die Adresse ist syntaktisch korrekt, nur die `TLD .asp` gibt es nicht.

In Abbildung 7.11 sehen Sie bereits, dass Sie unmöglich alle ungültigen E-Mail-Adressen abfangen ***können; wir haben uns darüber auch schon an*** *anderer Stelle in diesem Kapitel ausgelassen. Was aber viel wichtiger ist: Überprüfen Sie Ihre regulären Ausdrücke auf Herz und Nieren.* ***Wenn Sie*** *einen Fall übersehen und ein Benutzer bei einer korrekten E-Mail-Adresse eine Fehlermeldung* ***erhält, sorgt das sicherlich für Unmut und möglicher****weise einen (potenziellen) Kunden weniger.*

7.2.5 Eigene Funktion: CustomValidator

Ein Validation Control haben wir bereits unterschlagen, `CustomValidator`. Wie der Name schon sagt, können Sie damit maßgeschneiderte Validierungsfunktionen verwenden.

```
<asp:CustomValidator />
```

Das `CustomValidator`-Control übergibt die Formulardaten an eine speziell zugeschnittene Überprüfungsfunktion. Wieso diese Funktion so genau auf Ihre Aufgabenstellung passt? Ganz einfach – Sie müssen sie selbst schreiben!

Daran sehen Sie, dass hier die Überprüfung nur serverseitig stattfinden kann!

Das Control erwartet die folgenden Parameter:

- `ControlToValidate` – die ID des zu überprüfenden Formularelements
- `ErrorMessage` – die Fehlermeldung für den Fall, dass die Überprüfung fehlschlägt
- `OnServerValidate` – der Name der Funktion, die zur Validierung aufgerufen werden soll

Die Standard-Parameter wie beispielsweise `EnableClientScript` *existieren natürlich weiterhin, werden aber nicht explizit aufgeführt.*

Das interessanteste Element ist damit wohl `OnServerValidate`. Genauer gesagt handelt es sich hierbei um einen Event-Handler. Das Ereignis selbst heißt `ServerValidate` und mit `OnServerValidate` geben Sie an, was bei der (serverseitigen) Validierung des Formulars geschehen soll.

Die Funktion, die aufgerufen wird, hat folgendes Muster:

```
Sub XYZ(o As Object, e As ServerValidateEventArgs)
  ' ...
End Sub
```

Wenn Sie C# verwenden, analog:

```
void XYZ(Object o, ServerValidateEventArgs e) {
  // ...
}
```

Beachten Sie die Parameter, die die Validierungsfunktion erhält. Als ersten Parameter immer das aufrufende Objekt, als zweiten Parameter eine Variable

`e` des Typs `ServerValidateEventArgs`. Dieser Parameter ist eine Referenz auf das Formularelement, das im Parameter `ControlToValidate` steht. Sie können also beispielsweise über `e.Value` auf den Wert im Formularfeld zugreifen.

Die Aufgabe der Funktion ist es nun, die Überprüfung vorzunehmen und dann zu entscheiden, was zu tun ist:

- Wenn die Überprüfung fehlschlägt, muss die Eigenschaft `IsValid` des Controls auf `False` (bzw. `false` bei C#) gesetzt werden.
- Falls die Überprüfung zu keinem Fehler führt, kann `IsValid` auf `True` (bzw. `true`) gesetzt werden; da dies jedoch der Standardwert ist, kann er auch ausgelassen werden.

Als Beispiel soll eine ISBN (Internationale Standardbuchnummer) überprüft werden. Eine ISBN ist immer zehnstellig, die ISBN dieses Titels ist beispielsweise 3-8272-6270-4. Die Bindestriche sind eigentlich unerheblich und dienen hier nur der optischen Trennung der einzelnen Bestandteile:

- 3 steht hier für das Land, in dem das Buch erschienen ist: Deutschland. Die ISBNs aller in Deutschland erschienenen Bücher beginnen mit einer 3.
- 8272 ist die Verlagsnummer, hier also Markt+Technik. Addison-Wesley, der andere Imprint von Pearson Education Deutschland, hat beispielsweise die Nummer 8273.
- 6270 ist die Buchnummer. Der Titel »Jetzt lerne ich ASP.NET« hat beispielsweise die Buchnummer 6268. Es liegt also die Vermutung nahe, dass die beiden Titel relativ kurz nacheinander in die internen Planungen aufgenommen wurden.
- 4 schließlich ist die Prüfsumme der ISBN; damit sollen Tippfehler schnell erkannt werden.

Auf der letzten Ziffer, der Prüfsumme, soll der Fokus unserer Überprüfungen liegen. Sie wird berechnet, indem zunächst die ersten neun Ziffern der ISBN mit ihrer Position multipliziert werden und das Ganze dann addiert wird. »Position« bedeutet, dass die erste Ziffer mit 1, die zweite mit 2 usw. multipliziert wird. Für unsere Beispiel-ISBN 3827262704 wäre das dann:

```
1*3 + 2*8 + 3*2 + 4*7 + 5*2 + 6*6 + 7*2 + 8*7 + 9*0
```

Das Ergebnis lautet (wie Sie leicht nachrechnen können) 169. Im nächsten – und letzten – Schritt muss der Elferrest dieser Zahl ermittelt werden. Sie müssen also feststellen, welcher Rest bei der Division durch 11 übrig bleibt. In unserem Beispiel ist das Ergebnis `4`, denn `169 = 15*11 + 4`. Und dieses Ergebnis ist gleichzeitig die letzte Ziffer der ISBN. Die ISBN `3827262704` ist also korrekt.

Wenn eine Zahl den Elferrest 10 hat, wird als letzte »Ziffer« der ISBN ein X verwendet.

Die Validierungsfunktion überprüft nun eine eingegebene ISBN und handelt dementsprechend. Die Berechnung erfolgt gemäß oben gezeigtem Algorithmus. Zunächst einmal wird die ISBN aus dem Parameter an die Funktion ermittelt:

```
Dim isbn As String = e.Value
```

Im nächsten Schritt wird überprüft, ob der übergebene Wert überhaupt aus Ziffern besteht. Dazu wird die VB.NET-Funktion `IsNumeric` verwendet:

```
If Not IsNumeric(isbn) Then
  e.IsValid = False
  Exit Sub
End If
```

Als Nächstes muss die ISBN natürlich exakt zehnstellig sein:

```
If (isbn.Length <> 10) Then
  e.IsValid = False
  Exit Sub
End If
```

Nun wird die Prüfsumme berechnet, wie oben beschrieben. Die Umwandlung der einzelnen Zeichen innerhalb der ISBN wird mittels `Integer.Parse` vorgenommen.

```
Dim summe As integer = 0
Dim i As integer = 0
For i=0 To 8
  summe += (i+1) * Integer.Parse(isbn.Substring(i, 1))
Dim pruefziffer As integer = summe Mod 11
```

Nun muss nur noch überprüft werden, ob die Prüfziffer stimmt. Wenn das Ergebnis der vorherigen Berechnungen 10 war, muss die ISBN auf `X` enden, ansonsten mit der Prüfziffer übereinstimmen:

```
If (isbn.Substring(9, 1) = "X" And pruefziffer = 10) Or _
    (Integer.Parse(isbn.Substring(9, 1)) = pruefziffer) Then
  e.IsValid = True
Else
  e.IsValid = false
End If
```

Wenn Sie C# einsetzen, müssen Sie einiges beachten. Beispielsweise existiert in C# kein Äquivalent zu `IsNumeric`, was auf diversen Entwickler-Mailinglisten auch schon für Unmut gesorgt hat. Stattdessen können Sie jedoch mit

try/catch arbeiten: Wandeln Sie den String in eine Zahl um (Sie benötigen Int64, damit die Daten auch hineinpassen) und überprüfen Sie, ob Sie eine Fehlermeldung erhalten. Da der Rest analog ist, präsentieren wir Ihnen die gesamte Überprüfungsfunktion am Stück:

```
void CheckISBN(Object o, ServerValidateEventArgs e) {
  String isbn = e.Value;

  Int64 zahl;
  try {
    zahl = Int64.Parse(isbn); //numerischer Wert?
  } catch (Exception ex) {
    e.IsValid = false;
    return;
  }

  if (isbn.Length != 10) {
    e.IsValid = false;
    return;
  }

  int summe = 0;
  for (int i=0; i<9; i++) {
    summe += (i+1) * int.Parse(isbn.Substring(i, 1));
  }
  int pruefziffer = summe % 11;

  if ((isbn.Substring(9, 1) == "X" && pruefziffer == 10) ||
      (int.Parse(isbn.Substring(9, 1)) == pruefziffer)) {
    e.IsValid = true;
  } else {
    e.IsValid = false;
  }
}
```

Nachfolgend noch ein komplettes Listing für die VB.NET-Fraktion; die C#-Variante befindet sich auf der CD-ROM, ist aber außer den gezeigten Unterschieden in der Überprüfungsfunktion identisch.

Listing 7.9: Überprüfung einer ISBN (*customvalidator.aspx*)


```
<%@ Page Language="vb" %>
<script runat="server">
Sub CheckISBN(o As Object, e As ServerValidateEventArgs)
  Dim isbn As String = e.Value

  If Not IsNumeric(isbn) Then
    e.IsValid = False
    Exit Sub
  End If
  If (isbn.Length <> 10) Then
    e.IsValid = False
    Exit Sub
  End If
```

```
  Dim summe As integer = 0
  Dim i As integer = 0
  For i=0 To 8
    summe += (i+1) * Integer.Parse(isbn.Substring(i, 1))
  Next
  Dim pruefziffer As integer = summe Mod 11

  If (isbn.Substring(9, 1) = "X" And pruefziffer = 10) Or _
      (Integer.Parse(isbn.Substring(9, 1)) = pruefziffer) Then
    e.IsValid = True
  Else
    e.IsValid = false
  End If
End Sub
</script>
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  ISBN:
  <input type="text" id="ISBN" runat="server" />
  <asp:CustomValidator
    ControlToValidate="ISBN"
    ErrorMessage="Ungültige ISBN!"
    OnServerValidate="CheckISBN"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Abbildung 7.12: Der Tippfehler in der ISBN wird festgestellt.

Die Überprüfung findt konstruktionsbedingt erst statt, nachdem die Daten an den Webserver übermittelt wurden. Sie können jedoch – zusätzlich zur serverseitigen Prüffunktion – eine clientseitige Funktion in JavaScript verfassen. Beginnen wir mit dieser Funktion. Sie überprüft auch die Formulareingabe und wendet den Prüfalgorithmus für ISBNs an.

Zunächst müssen Sie die Signatur (also die erwarteten Parameter) der Funktion beachten. Wie bei dem serverseitigen Pendant wird zunächst das Objekt, das die Überprüfung auslöst (hier: der Validator) und der Parameter (hier: das Formularelement) übergeben. Die Überprüfung selbst läuft analog zum serverseitigen Code, nur dieses Mal in JavaScript umgeschrieben:

```
function CheckISBN(o, e) {
  var isbn = e.Value;
  if (parseInt(isbn) == NaN || isbn.length != 10) {
    e.IsValid = false;
    return true;
  }
  var summe = 0, i = 0;
  for (i=0; i<9; i++)
    summe += (i+1) * parseInt(isbn.charAt(i));
  pruefziffer = summe % 11;
  if ((isbn.charAt(9) == "X" && pruefziffer == 10) ||
      pruefziffer == isbn.charAt(9))
    e.IsValid = true;
  else
    e.IsValid = false;
  return true;
}
```

Um diese Funktion clientseitig aufzurufen, müssen Sie im `ValidationControl`-Element nur noch den Parameter `ClientValidationFunction` auf den Namen der Funktion, hier also `"CheckISBN"` setzen:

```
<asp:CustomValidator
  ControlToValidate="ISBN"
  ErrorMessage="Ungültige ISBN!"
  OnServerValidate="CheckISBN"
  ClientValidationFunction="CheckISBN"
  runat="server" />
```

Hier der vollständige Quellcode:

Listing 7.10: Die ISBN wird nun direkt nach der Eingabe geprüft (*customvalidator-js.vb.aspx*).


```
<%@ Page Language="vb" %>
<script runat="server">
Sub CheckISBN(o As Object, e As ServerValidateEventArgs)
  Dim isbn As String = e.Value
```

```
  If Not IsNumeric(isbn) Then
    e.IsValid = False
    Exit Sub
  End If

  If (isbn.Length <> 10) Then
    e.IsValid = False
    Exit Sub
  End If

  Dim summe As integer = 0
  Dim i As integer = 0
  For i=0 To 8
    summe += (i+1) * Integer.Parse(isbn.Substring(i, 1))
  Next
  Dim pruefziffer As integer = summe Mod 11

  If (isbn.Substring(9, 1) = "X" And pruefziffer = 10) Or _
      (Integer.Parse(isbn.Substring(9, 1)) = pruefziffer) Then
    e.IsValid = True
  Else
    e.IsValid = false
  End If
End Sub
</script>
<html>
<head>
<title>Validation Controls</title>
<script language="JavaScript"><!--
function CheckISBN(o, e) {
  var isbn = e.Value;
  if (parseInt(isbn) == NaN || isbn.length != 10) {
    e.IsValid = false;
    return true;
  }
  var summe = 0, i = 0;
  for (i=0; i<9; i++)
    summe += (i+1) * parseInt(isbn.charAt(i));
  pruefziffer = summe % 11;
  if ((isbn.charAt(9) == "X" && pruefziffer == 10) ||
      pruefziffer == isbn.charAt(9))
    e.IsValid = true;
  else
    e.IsValid = false;
  return true;
}
//--></script>
</head>
<body>
<form runat="server">
  ISBN:
  <input type="text" id="ISBN" runat="server" />
  <asp:CustomValidator
    ControlToValidate="ISBN"
    ErrorMessage="Ungültige ISBN!"
```

```
    OnServerValidate="CheckISBN"
    ClientValidationFunction="CheckISBN"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Mit serverseitigen Überprüfungsfunktionen in Kombination mit dem clientseitigen Pendant verfügen Sie nun über maximale Funktionalität bei der Überprüfung von Formulareingaben.

7.3 Fehlermeldungen ausgeben

Bisher war der Ablauf unserer Beispiele immer derselbe: Zunächst wurden ein oder mehrere Eingabefelder ausgegeben. Wenn Fehler auftraten, wurden die entsprechenden Fehlermeldungen sofort (Internet Explorer mit aktivierter JavaScript-Unterstützung) oder nach dem Versand (andere Browser und Konfigurationen) ausgegeben. Diese Fehlermeldungen erschienen immer an der Stelle, an der auch die Validation Controls positioniert wurden.

7.3.1 Validierungsergebnis: ValidationSummary

Nun ist es unter Umständen eine gute Idee, wenn eine Zusammenfassung aller aufgetretenen Fehler angezeigt werden könnte. Dazu dient das Control `ValidationSummary`. Es ist eigentlich auch ein Validation Control, da es aber selbst keine Felder überprüft, haben wir es im vorherigen Abschnitt außen vor gelassen.

```
<asp:ValidationSummary runat="server" />
```

Fehler zusammenfassen

Die einfachste Variante dieses Controls erhalten Sie, wenn Sie lediglich den Parameter `HeaderText` setzen. Dort geben Sie den Text an, der über den gesammelten Fehlermeldungen angezeigt werden soll.

Sie können den Parameter `HeaderText` *auch* ***nicht*** *setzen. Dann würde die Zusammenfassung der aufgetretenen Fehler allerdings etwas leer im Raum stehen und deswegen empfehlen wir dieses Vorgehen nicht.*

Hier ein einfaches Beispiel mit zwei Textfeldern einschließlich zugehöriger Validation Controls, sowie das neue `ValidationSummary`-Control:

Listing 7.11: Zusammenfassung der Fehlermeldungen (*validationsummary1.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  <asp:ValidationSummary
    HeaderText="Die folgenden Fehler sind aufgetreten:"
    runat="server" />
  <br />
  Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Name fehlt!"
    runat="server" />
  <br />  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    ErrorMessage="E-Mail-Adresse ungültig!"
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.[a-zA-
Z]{2,4}$"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

In Abbildung 7.13 sehen Sie das Ergebnis, wenn Sie das Formular mit ungültigen Angaben verschicken.

Keine einzelnen Fehlermeldungen mehr

Die Zusammenfassung aller Fehler wird ausgegeben, nachdem das Formular verschickt wurde. Nun ist es aber unter Umständen unnötig, alle Fehlermeldungen zweimal anzuzeigen, einmal oben und das zweite Mal dieselbe Meldung an dem entsprechenden Formularelement.

Weiter oben haben Sie bereits einmal den Hinweis erhalten, dass ein Validation Control zusätzlich den Parameter `Text` besitzt. Diese Eigenschaft enthält die »eigentliche« Fehlermeldung des Controls. Wenn sie nicht gesetzt wird, wird der Wert des Parameters `ErrorMessage` verwendet.

Der Parameter `ErrorMessage` enthält die im `ValidationSummary`-Control ausgegebene Fehlermeldung. Aus Bequemlichkeitsgründen wird aber oft nur der Parameter `ErrorMessage` gesetzt.

Abbildung 7.13: Die Fehler werden oben zusammengefasst.

Im nachfolgenden Beispiel finden Sie die beiden **Textfelder aus dem vorhergehenden Listing wieder. Dieses Mal enthalten die einzelnen Validation Controls jedoch eine andere Fehlermeldung (im Parameter** `Text`**) als in** `ValidationSummary` **ausgegeben wird (Parameter** `ErrorMessage`**).**

Listing 7.12: Fehlermeldung und Zusammenfassung sind unterschiedlich (*validationsummary2.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  <asp:ValidationSummary
    HeaderText="Die folgenden Fehler sind aufgetreten:"
    runat="server" />
  <br />
  Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    Text="Hier den Namen eingeben!"
    ErrorMessage="Name fehlt!"
    runat="server" />
  <br />  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    Text="Prüfen Sie die E-Mail-Adresse!"
    ErrorMessage="E-Mail-Adresse ungültig!"
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}
                          \.[a-zA-Z]{2,4}$"
    runat="server" />
  <br />
```

```
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

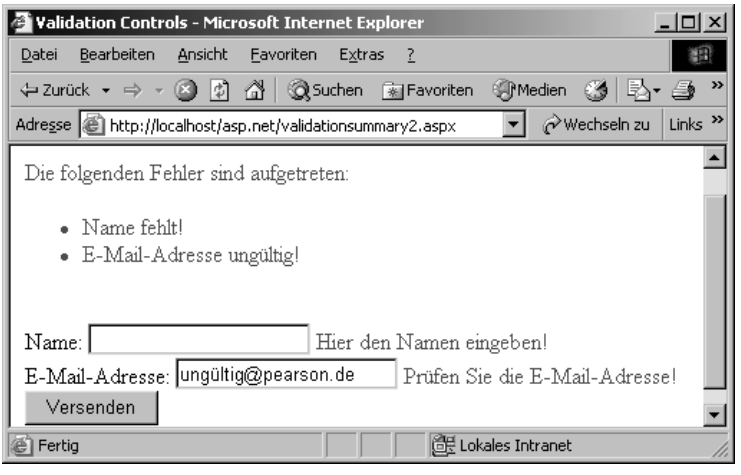


Abbildung 7.14: Der Unterschied zwischen `Text` und `ErrorMessage`

Layout der Zusammenfassung

Die Zusammenfassung ist bis jetzt immer in Form einer Aufzählungsliste dargestellt worden. Mit dem Parameter `DisplayMode` können Sie dieses Aussehen bis zu einem bestimmten Grad bestimmen. Die folgenden drei Werte sind hierbei erlaubt:

Tabelle 7.7: Die verschiedenen Werte für `DisplayMode`

Wert für `DisplayMode`	Beschreibung
`BulletList`	Standard; Aufzählungliste mit grafischen Aufzählungszeichen (»Knödeln«)
`List`	Liste ohne Aufzählungszeichen
`SingleParagraph`	Die einzelnen Fehlermeldungen in einem einzigen Absatz, durch Leerzeichen voneinander getrennt

Folgendes Listing enthält drei `ValidationSummary`-Controls mit jeweils einem der drei Darstellungsmodi. Daran lässt sich der Unterschied sehr schön erkennen.

Listing 7.13: Drei Darstellungsmodi für Fehlerzusammenfassungen (*validationsummary3.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
```

```
</head>
<body>
<form runat="server">
  <asp:ValidationSummary
    DisplayMode="BulletList"
    HeaderText="Fehler (BulletList):"
    runat="server" />
  <br />
  <asp:ValidationSummary
    DisplayMode="List"
    HeaderText="Fehler (List):"
    runat="server" />
  <br />
  <asp:ValidationSummary
    DisplayMode="SingleParagraph"
    HeaderText="Fehler (SingleParagraph):"
    runat="server" />
  <br />
  Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Name fehlt!"
    runat="server" />
  <br />  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    ErrorMessage="E-Mail-Adresse ungültig!"
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.[a-zA-
Z]{2,4}$"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

7.3.2 Dynamische Anzeige

An anderer Stelle in diesem Kapitel haben wir es bereits angesprochen: Wenn Sie ein `RegularExpressionValidator`-Control einsetzen, um einen Formularwert zu überprüfen, schlägt diese Überprüfung bei einem leeren Formularfeld fehl. Sie können das direkt im vorherigen Beispiel ausprobieren: Wenn Sie keine E-Mail-Adresse angeben, erhalten Sie keine Fehlermeldungen.

Ein nahe liegender Ausweg ist, zusätzlich zum `RegularExpressionValidator`-Control einfach ein `RequiredFieldValidator`-Control einzusetzen:

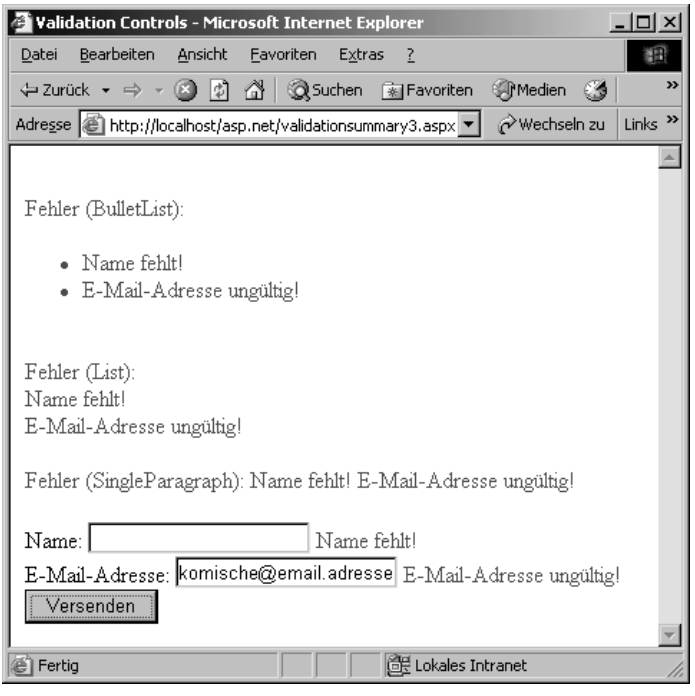


Abbildung 7.15: Die verschiedenen Darstellungsarten

Listing 7.14: Fehlermeldung, auch bei fehlender E-Mail-Adresse (*display1.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    ErrorMessage="E-Mail-Adresse ungültig!"
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}
                          \.[a-zA-Z]{2,4}$"
    runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="mail"
    ErrorMessage="Keine E-Mail-Adresse angegeben!"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Wenn Sie das Script im Browser ausführen und das Formular leer abschicken, erhalten Sie eine Fehlermeldung – allerdings befindet sich ziemlich viel Platz zwischen dem Texteingabefeld und der Meldung (siehe Abbildung 7.16).

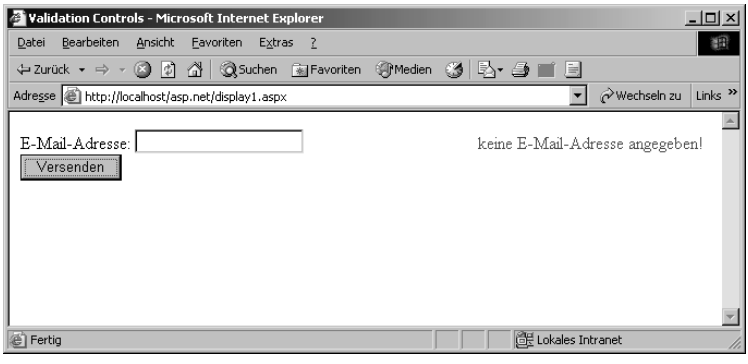


Abbildung 7.16: Die Fehlermeldung erscheint – ziemlich weit rechts

Der Grund: Der leere Platz dazwischen ist für die Fehlermeldung des `RegularExpressionValidator`-Controls reserviert. Aber auch hierfür kennt ASP.NET einen Ausweg. Setzen Sie in beiden beteiligten Validation Controls den Parameter `Display` auf `"Dynamic"`. Die Positionen der Fehlermeldungen werden dann dynamisch bestimmt, es erscheint also kein Leerraum mehr. Beim Internet Explorer erfolgt dies dank kräftiger JavaScript-Unterstützung in der ASP.NET-Seite während der Formularausfüllung, bei allen anderen Browsern nach dem Versand.

Auf der Buch-CD-ROM finden Sie das Script display2.aspx, das im Vergleich zu display1.aspx (vorheriges Listing) zusätzlich lediglich ein `Display="Dynamic"` *in beiden Validation Controls aufweist. Die Auswirkungen dieser Hinzufügung sehen Sie in Abbildung 7.17: Der Leerraum zwischen Fehlermeldung und Formularelement fehlt.*

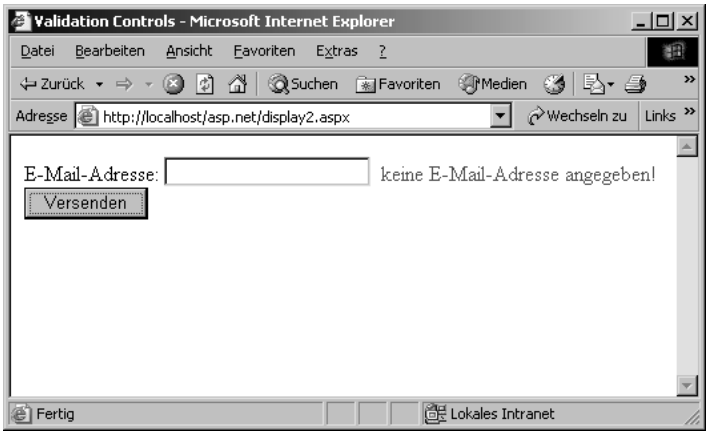


Abbildung 7.17: Die Fehlermeldung erscheint jetzt unmittelbar neben dem Textfeld.

7.3.3 Layout der Fehlermeldungen

Bisher sind die einzelnen Fehlermeldungen immer in Rot und in der Standardschrift des Browsers (meistens Times oder Times New Roman) ausgegeben worden. Wenn es das Corporate Design Ihres Unternehmens und/oder der Website erfordert, sollten Sie dies jedoch anpassen. Am einfachsten geht das, wenn Sie den Parameter `CSSClass` der Validation Controls setzen. Als Wert geben Sie den Namen der CSS-Klasse an, die Sie dem Element zuweisen möchten.

Alles, was Sie nun noch tun müssen, ist eine entsprechende Stil-Klasse im Kopf Ihres Dokuments zu definieren:

```
<style type="text/css"><!--
  .fehler {
    font-family: Verdana;
    font-size: 10pt;
    font-weight: bold;
  }
//--></style>
```

Alle Validation Controls, die den Parameter `CSSClass="fehler"` aufweisen, werden nun in Verdana 10 Punkt und fett dargestellt. Wenn Sie dem Text noch eine andere Farbe geben möchten, können Sie das nicht über CSS-Klassen realisieren. Sie müssen stattdessen den Parameter `ForeColor` setzen; dieser enthält als Wert die gewünschte Farbe.

!! STOP

Mit dem Parameter `BackColor` *können Sie analog die Hintergrundfarbe der Fehlermeldung setzen.*

Im folgenden Listing wird dies demonstriert. Das Stylesheet von oben wird eingebunden, außerdem erscheint die Fehlermeldung in weißer Schrift auf schwarzem Grund.

{} LIST

Listing 7.15: Die Fehlermeldung in anderer Schriftgestaltung (*font.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
<style type="text/css"><!--
  .fehler {
    font-family: Verdana;
    font-size: 10pt;
    font-weight: bold;
  }
//--></style>
</head>
<body>
<form runat="server">
```

```
  Ihr Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Geben Sie Ihren Namen an!"
    CSSClass="fehler"
    ForeColor="white"
    BackColor="black"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

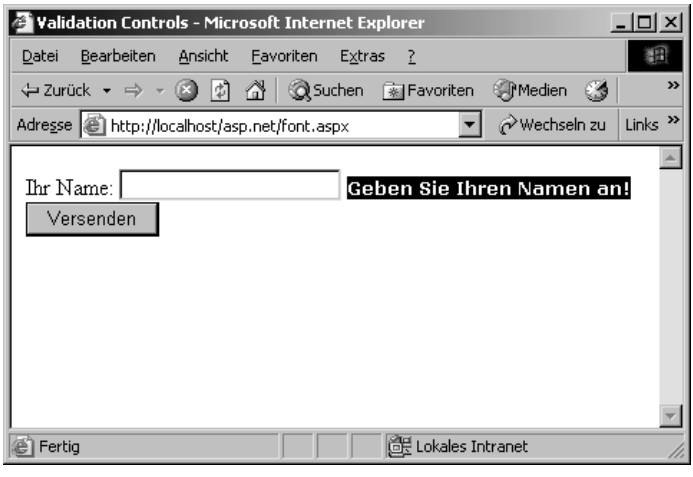


Abbildung 7.18: Die Fehlermeldung in Weiß auf schwarzem Grund

Models Fenster

Zu guter Letzt soll nicht verschwiegen werden, dass die Fehlermeldung auch prominent per Popup-Fenster angezeigt werden kann. Dies funktioniert selbstverständlich nur dann, wenn JavaScript im Browser aktiviert ist. Vom ASP.NET-Interpreter wird nämlich der JavaScript-Befehl `window.alert()` in die resultierende HTML-Seite eingesetzt. Durch dieses Kommando wird dann ein Warnfenster geöffnet, das der Benutzer erst per Mausklick schließen muss (siehe Abbildung 7.19). Ihnen steht hiermit ein Mittel zur Verfügung, Ihren Nutzern die Fehlermeldungen ganz besonders aufdringlich zu präsentieren. Allerdings sollten Sie für sich selbst entscheiden, ob es tatsächlich so aufdringlich sein muss.

Um dies zu erreichen, müssen Sie zunächst ein `ValidationSummary`-Control einfügen. Dort setzen Sie dann den Parameter `ShowMessageBox` auf `"True"`. Hier das Listing, das die Ausgabe in Abbildung 7.19 erzeugt:

Abbildung 7.19: Die Fehlermeldungen werden im Warnfenster angezeigt.

Listing 7.16: Fehlerausgabe per Warnfenster (*validationsummary4.aspx*)

```
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<form runat="server">
  <asp:ValidationSummary
    ShowMessageBox="True"
    runat="server" />
  <br />
  Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Name fehlt!"
    runat="server" />
  <br />  E-Mail-Adresse:
  <input type="text" id="mail" runat="server" />
  <asp:RegularExpressionValidator
    ControlToValidate="mail"
    ErrorMessage="E-Mail-Adresse ungültig!"
    ValidationExpression="^[a-zA-Z._\-]+@[a-zA-Z._\-]{2,}\.
                          [a-zA-Z]{2,4}$"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         runat="server" />
</form>
</body>
</html>
```

Das Aussehen der Fehlermeldungen im Warnfenster kann auch dem zuvor schon vorgestellten Parameter `DisplayMode` *angepasst werden (siehe auch Tabelle 7.7).*

7.4 Formular-Handling

Nachdem Sie nun so viel über Validation Controls erfahren haben, zeigen wir Ihnen zum Abschluss noch, wie Sie die Ergebnisse der Überprüfungen auch von der Scriptseite aus abfragen können.

7.4.1 Formular versenden

Bisher war es immer so: Wenn ein Formular ausgefüllt wurde, zeigte der Internet Explorer Fehler sofort an, bei anderen Browsern erschienen sie erst nach dem Formularversand. Wenn das Formular aber vollständig und korrekt ausgefüllt wurde, möchten Sie ja die Formulardaten auch weiterverarbeiten, beispielsweise in eine Datenbank schreiben. Sie benötigen also einen Mechanismus um festzustellen, ob das Formular komplett ausgefüllt wurde oder nicht.

Die Eigenschaft `IsValid` haben Sie bereits beim `CustomValidator`-Control kennen gelernt. Das `Page`-Objekt besitzt diese Eigenschaft auch. Wenn ein Formular korrekt ausgefüllt wurde – d.h., keines der Validation Controls schlägt Alarm –, hat diese Eigenschaft den Wert `True` (bei C#: `true`), ansonsten `False` (bei C#: `false`). Sie können also eine Versende-Funktion nach folgendem Muster schreiben:

```
Sub Versand(o As Object, e As EventArgs)
  If Page.IsValid Then
    ' Formulardaten verarbeiten ...
    ausgabe.InnerHtml = "<b>Danke!</b>"
    Formular.Visible = False
  End If
End Sub
```

In C# sieht das ähnlich aus:

```
void Versand(Object o, EventArgs e) {
  if (Page.IsValid) {
    // Formulardaten verarbeiten ...
    ausgabe.InnerHtml = "<b>Danke!</b>";
    Formular.Visible = false;
  }
}
```

Nachfolgend ein komplettes Script:

Listing 7.17: Dankesmeldung nach Formularversand (*versand.vb.aspx*)


```
<%@ Page Language="vb" %>
<script runat="server">
Sub Versand(o As Object, e As EventArgs)
```

```
  If Page.IsValid Then
    ' Formulardaten verarbeiten ...
    ausgabe.InnerHtml = "<b>Danke!</b>"
    Formular.Visible = False
  End If
End Sub
</script>
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<p id="ausgabe" runat="server" />
<form id="Formular" runat="server">
  Ihr Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Geben Sie Ihren Namen an!"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         OnServerClick="Versand"
         runat="server" />
</form>
</body>
</html>
```

Vergessen Sie die wichtigen Elemente dieses Scripts nicht:

- *Die Versende-Schaltfläche muss mit* `OnServerClick="Versand"` *versehen werden.*
- *Das Formular muss den Parameter* `id="Formular"` *erhalten, damit es später ausgeblendet werden kann.*
- *Sie benötigen einen Absatz zur Ausgabe der Dankesmeldung, und zwar außerhalb des Formulars:*

  ```
  <p id="ausgabe" runat="server" />
  ```

7.4.2 Überprüfung abbrechen

Bei mehrseitigen Formularen gibt es oft eine Möglichkeit, auf die vorhergehende Formularseite zurückzugehen. Andere Applikationen ermöglichen es dem Benutzer, die Formularausfüllung abzubrechen.

Normalerweise würden Sie dazu eine Schaltfläche verwenden und dann serverseitig eine Funktion aufrufen, die den Benutzer auf die Startseite oder die vorhergehende Seite umleitet. Bei den bisher vorgestellten Scripten geht das leider nicht, zumindest nicht wenn Sie den Internet Explorer mit aktiviertem

JavaScript einsetzen. Dort kann das Formular nämlich nicht verschickt werden, wenn die Angaben nicht vollständig sind.

Einen Ausweg gibt es jedoch. Setzen Sie in der Versende-Schaltfläche (also `<input type="button" />` oder `<asp:Button />`) den Parameter `CausesValidation` auf `"False"`. Dann wird das Formular nicht validiert, wenn Sie auf die Schaltfläche klicken, indes verschickt auch der Internet Explorer die Formulardaten an den Webserver und ermöglicht so die Ausführung der Abbruchfunktion.

Hier ein komplettes Beispiel auf der Basis des vorhergehenden Listings:

Listing 7.18: Die Formularausfüllung kann abgebrochen werden (*abbruch.vb.aspx*).


```
<%@ Page Language="vb" %>
<script runat="server">
Sub Versand(o As Object, e As EventArgs)
  If Page.IsValid Then
    ' Formulardaten verarbeiten ...
    ausgabe.InnerHtml = "<b>Danke!</b>"
    Formular.Visible = False
  End If
End Sub
Sub Abbruch(o As Object, e As EventArgs)
  Response.Redirect("seite.aspx")
End Sub
</script>
<html>
<head>
<title>Validation Controls</title>
</head>
<body>
<p id="ausgabe" runat="server" />
<form id="Formular" runat="server">
  Ihr Name:
  <input type="text" id="Name" runat="server" />
  <asp:RequiredFieldValidator
    ControlToValidate="Name"
    ErrorMessage="Geben Sie Ihren Namen an!"
    runat="server" />
  <br />
  <input type="submit" value="Versenden"
         OnServerClick="Versand"
         runat="server" />
  <input type="submit" value="Abbrechen"
         OnServerClick="Abbruch"
         CausesValidation="False"
         runat="server" />
</form>
</body>
</html>
```

!!
STOP

Das Beispiel versucht, auf die Seite seite.aspx weiterzuleiten. Ersetzen Sie diese Angabe durch eine eigene URL oder erstellen Sie eine entsprechende Datei seite.aspx.

7.5 Fazit

In diesem Kapitel haben Sie eine ganze Reihe von Möglichkeiten kennen gelernt, Formulardaten zu überprüfen. Dabei haben wir nicht nur Vollständigkeitsüberprüfungen vorgenommen, sondern mehrere weitere Überprüfungen mit verschiedenen Komplexitätsgraden.

Im Allgemeinen gilt: Je kürzer ein Formular gehalten ist, desto wahrscheinlicher werden Ihre Nutzer es ausfüllen. Analog: Je weniger Pflichtfelder Sie verwenden, desto wahrscheinlicher werden Ihre Benutzer persönliche Daten preisgeben. Überlegen Sie sich also gut, welche Daten tatsächlich wichtig sind und welche nur optionales Beiwerk. Bereiten Sie zudem eine entsprechende Datenschutzerklärung für Ihre Website vor, in der Sie erklären, welche Daten Sie wo und zu welchem Zweck speichern. Die Einhaltung des Datenschutzes wird insbesondere in Deutschland sehr genau überwacht.

Bei allen Vorteilen, die der Einsatz von Validation Controls bietet, sollten Sie jedoch immer im Hinterkopf behalten, dass Sie damit einen Großteil der Kontrolle über das Aussehen und Verhalten Ihrer Website aus der Hand geben. Es ist also oftmals vorteilhafter (wenngleich auch aufwändiger), die Überprüfung selbst zu programmieren, dann aber genau zu wissen, was passieren wird.

Unser persönliches Fazit: Validation Controls sind ein großartiges Werkzeug, sie sollten aber mit Bedacht eingesetzt werden.