

Christoph Schmitz,
Steffen Goldfuß,
Philipp Cielen

ColdFusion MX

Professionelle Anwendungsentwicklung fürs Web



An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

3 Formulare und Form-Variablen

Über Formulare können recht praktisch User-Eingaben an den Server übermittelt werden. Zum einen sind sie relativ einfach zu erstellen und zum anderen müssen keine ellenlangen URLs mit einer Vielzahl von Parametern übergeben werden.

Zuerst integriert man in der aufrufenden Seite ein handelsübliches HTML-Formular. In diesen Formularen können Sie verschiedene Eingabefelder definieren, die der Besucher der Website dann ausfüllen kann. Sie können mit reinem HTML Eingabefelder für einzeilige oder mehrzeilige Texte, Auswahllisten, Checkboxen und vieles mehr definieren. Mit einem Button kann der Anwender dann die Daten zum Server schicken, auf dem sie ihm dann für einen leichten Zugriff zur Verfügung stehen. Einschränkungen der Datengröße oder Probleme mit Sonderzeichen wie bei URL-Parametern gibt es dabei nicht.

Mit Formularen können Sie z.B. Eingabemasken für das Suchen in Datenbanken erstellen oder Login-Masken für Passwortabfragen generieren.

3.1 Aufbau von Formularen

Das Grundgerüst eines Formulars besteht aus einem `<FORM>`-Tag-Paar. Es müssen zwei Attribute für die korrekte Funktion angegeben werden. Mit dem `ACTION`-Attribut benennen Sie die Datei, der die Daten gesendet werden sollen bzw. die den Request bearbeiten soll. Das wird in diesem Fall natürlich eine ColdFusion-Datei sein. Mit `METHOD` spezifizieren Sie die Übertragungsmethode zum Server. Dabei gibt es zwei Möglichkeiten:

- ▶ Mit dem Attributwert `METHOD=»get«` werden die Daten des Formulars als Parameter des URLs zum Server hin übertragen. Das im `ACTION`-Attribut angegebene ColdFusion-Template muss die Variablen dann als URL-Parameter ansprechen.
- ▶ Wenn Sie `METHOD=»post«` verwenden, werden die Daten im HTTP-Header des Browsers zum Server gesendet. Die Einschränkungen, die es bei URL-Parametern gibt, gelten hier nicht. Sie brauchen Sonderzeichen nicht zu kodieren und können unbegrenzte Datenmengen senden. Wollen Sie auf der nachfolgenden Seite FORM-Variablen verwenden, müssen Sie sogar zu `METHOD=»post«` greifen!

Zur Dateneingabe müssen innerhalb der <FORM>-Tags Dateneingabefelder z.B. mit dem <INPUT>-Tag definiert werden. Ein Submit-Button erlaubt es dem Benutzer, das Formular abzuschicken.

Das erste Beispiel-Formular soll eine Login-Maske mit zwei Eingabefeldern darstellen:

```
<form action="bar.cfm" method="post">
  <input type="text" name="user"><br>
  <input type="password" name="pass"><br>
  <input type="submit" value="login">
</form>
```

Das Ergebnis sehen Sie in der Abbildung:

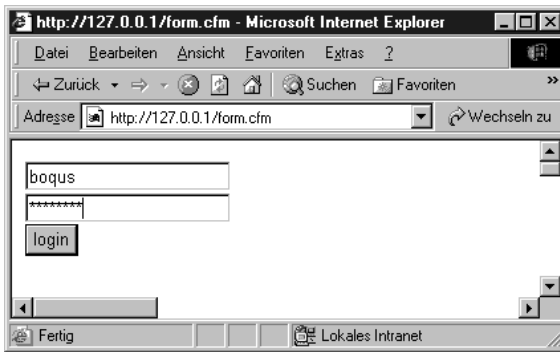


Abbildung 3.1: kleines Login-Formular

In die beiden Eingabefelder können der Benutzername und das Passwort eingegeben werden. Die TYPE-Attribute spezifizieren jeweils einen speziellen Typ von Eingabefeld. Mit TYPE=»text« wird ein normales einzeliliges Eingabefeld erzeugt. Mit TYPE=»password« erzeugen Sie hingegen Eingabefelder, in denen die Eingaben nur als * sichtbar sind.

Das letzte Eingabeelement erzeugt den Submit-Button, mit dem der Anwender das Formular abschicken kann. Mit dem VALUE-Attribut kann eine Beschriftung für den Button festgelegt werden.

Die Eingabeelemente (mit Ausnahme des Submit-Buttons) müssen mit einem NAME-Attribut versehen werden, damit das im ACTION-Attribut angegebene Template die Werte als ColdFusion-Variablen mit dem Präfix FORM ansprechen kann.

Nehmen wir an, der Benutzer füllt unsere Beispiel-Login-Maske aus und schickt die Daten zum Server. ColdFusion startet jetzt das Template *bar.cfm* und erzeugt die beiden Variablen `FORM.user` und `FORM.pass`, die den Inhalt der zugehörigen Formularelemente enthalten. Sie können die Variablen dann wie jede andere Variable auslesen.

Hinweis:

Sie können diesen Variablen jedoch auch neue Werte zuweisen. Wir raten aber dringend davon ab, empfangenen URL- oder FORM-Variablen manuell neue Werte zuzuweisen, da dies im Programmablauf spätestens dann zu Verwirrungen führt, wenn eine Seite debugged (von Fehlern bereinigt) werden soll!! (Siehe Kapitel 6, Fehlermeldungen und Debugging)

Die Seite, die das HTML-Formular enthält, kann durchaus ein reines HTML-Dokument sein, denn das Verschicken der Daten übernimmt der Browser und nicht etwa ColdFusion.

Will man einen Wert übergeben, ohne dass der User ihn sehen soll, kann man ein verstecktes Formelement (`TYPE="hidden"`) benutzen:

```
<CFSET id="233" />
<CFOUTPUT>
  <form action="bar.cfm" method="POST">
    <input type="hidden" name="id" value="#id#"><br>
    <input type="submit" value="go!">
  </form>
</CFOUTPUT>
```

In *bar.cfm* können Sie dann das Ergebnis ganz normal ausgeben:

```
<CFOUTPUT>
  id=#FORM.id#
</CFOUTPUT>
```

Man kann anhand der einzelnen FORM-Variablen in ColdFusion nicht mehr erkennen, von welchem Typ Formelement der Wert stammt.

Ein Wert aus einem Hidden-Feld namens »*pass*« sieht also genauso aus wie ein Wert aus einer Checkbox »*pass*« (vorausgesetzt die Checkbox ist selektiert (angekreuzt) worden und hat denselben Value-Wert wie das Hidden-Feld).

Beachten Sie aber, wenn sie Daten mit Hidden-Feldern in Formularen unterbringen, dass die Werte im HTML-Sourcecode sichtbar sind. Der Benutzer kann also die Werte trotzdem erkennen, wenn er im Browser die Sourcecode-Ansicht wählt (und manche User machen so etwas tatsächlich!). Informationen, die nicht für jedermann bestimmt sind, haben dort also nichts zu suchen!

3.2 Zeichensatz-Konvertierung im Formular

ColdFusion arbeitet seit der Version MX mit dem Unicode-Zeichensatz, d.h. Texte werden intern als Double-Byte pro Zeichen gespeichert. HTML-Formulare arbeiten jedoch

mit der Single-Byte-Darstellung, ein Zeichen besteht hier aus einem Byte. Da ColdFusion nun mit Double-Byte-Darstellung arbeitet, wandelt es jedes Byte, das es durch ein Formular übermittelt bekommt, wieder in eine Double-Byte-Darstellung um. Das wirkt sich so lange nicht aus, solange der User Single-Byte-Zeichen in sein Formular eingibt (z.B. A, B C). Gibt er jedoch Double-Byte-Zeichen (z.B. Ä, _ usw.) ein, sendet das Formular 2 Single-Byte-Zeichen anstatt eines Double-Byte-Zeichens. ColdFusion wandelt die 2 Single-Byte-Zeichen nun fälschlicherweise in 2 Double-Byte-Zeichen und es entstehen 4 Byte. Die ursprünglich eingegebene Information wird damit zerstört und ein seltsamer Zeichensalat entsteht.

Probieren Sie das selbst einmal an diesem kleinen Formular aus. Geben Sie verschiedene Zeichen ein und sehen Sie sich das Ergebnis im Browser an. Achten Sie dabei auch einmal auf die ausgegebene Länge der Variablen `FORM.in`:

```
<cfif isDefined("FORM.in") >
<!-- Ausgabe des Datenmuells --->
<cfoutput>
in:[<b>#FORM.in#</b>]<br>
Len=#len(Trim(FORM.in))#<br>
</cfoutput>
</cfif>

<cfoutput>
<form action="#CGI.script_name#" method="post">
<input name="in" type="text">
<input type="Submit">
</form>
</cfoutput>
```

Listing 3.1: testForm.cfm

Um das zu vermeiden liefert ColdFusion die Funktion `setEncoding()`, die Sie in Verbindung mit Formularen stets verwenden müssen. Mit `setEncoding()` spezifizieren Sie den Zeichensatz des Formulars, d.h. Sie veranlassen ColdFusion, die vom Formular gelieferten Daten anders zu interpretieren. Die Syntax von `setEncoding()` lautet

```
<cfset setEncoding(Scope,Zeichensatz)/>
```

Scope spezifiziert den Variablen-Scope, in dem das Encoding angewandt werden soll, der zweite Parameter spezifiziert den Zeichensatz, der in diesem Scope benutzt werden soll.

In Fall eines Formulars wäre der Scope »FORM« und der Zeichensatz »UTF-8«. Der Befehl würde wie folgt lauten:

```
<cfset SetEncoding("FORM", "UTF-8")/>
```

Das kann man natürlich auch in CFScript schreiben:

```
<cfscript>
SetEncoding("FORM", "UTF-8");
</cfscript>
```

Wenn jetzt ColdFusion die FORM-Variablen in den FORM-Scope schreibt, werden die vom Formular gesendeten Zeichen richtig interpretiert.

3.3 Besonderheiten bei Checkboxes und Radiobuttons

Bei einfachen Input-Feldern wie Textfeldern oder Text-Areas wird immer ein Wert gesendet und die zugehörige FORM-Variable erzeugt. Wird kein Wert eingegeben, wird ein Leer-String (" ") übergeben.

Das ist aber nicht bei allen Eingabeelementen so. Bei Checkboxes und Radiobuttons werden nur die vom Anwender aktivierten Elemente übertragen. Sie enthalten dann die Werte, die Sie im Formular mit der Angabe des VALUE-Attributs spezifiziert haben. Sind sie nicht aktiviert, dann erfolgt auch keine Übertragung dieser Formularelemente! Es können deshalb von ColdFusion auch keine entsprechenden Variablen erzeugt werden.

Damit ergeben sich also ähnliche Probleme wie bei URL-Variablen, wenn auf fehlende Variablen zugegriffen wird.

Auch hier können Sie mit `IsDefined()` vor der Benutzung auf die Existenz einer Variablen hin überprüfen oder mit `<CFPARAM>` einen Defaultwert festlegen.

Das folgende Formular demonstriert die Verwendung einer Checkbox:

```
<form action="save.cfm" method="POST">
  Pfad: <input type="text" name="pfad">
  <BR>
  Speichern: <input type="checkbox" name="save" value="1">
  <P>
  <input type="submit" value="weiter">
</form>
```

Ist in diesem Formular die Checkbox beim Abschicken ausgewählt, ist im Template *bar.cfm* die Variable `FORM.save="1"` vorhanden, anderenfalls fehlt sie.

In *save.cfm* könnte dann z.B. dieser Auswertungscode stehen:

```
<body>
<CFPARAM NAME="FORM.save" DEFAULT="0"/>

<CFIF FORM.save IS "1">
  <!--- speichern --->
```

```
<CFELSE>
  <!-- nicht speichern -->
</CFIF>
</body>
```

Wir definieren mit `<CFPARAM>` einen Defaultwert für die Checkbox, der den Zustand *unchecked* (nicht markiert) kennzeichnen soll.

3.4 Mehrfachselektion mit List- und Checkboxes

Setzen Sie in einer Select-Box das HTML-Attribut *MULTIPLE*, kann der Benutzer dort mehrere Einträge auswählen. Da die Listbox aber nur einen Namen hat, bedeutet das, dass in der zugehörigen FORM-Variablen mehrere Werte übergeben werden müssen.

Benutzen Sie eine Select-Box, in der mehrere Einträge selektierbar sind, werden bei einem *Submit* des Formulars die gewählten Einträge als kommaseparierte Liste übergeben. Ist kein Eintrag ausgewählt worden, existiert die FORM-Variable nicht, Sie sollten also wieder eine Prüfung mit `IsDefined()` einbauen oder einen Defaultwert mit `<CFPARAM>` einbauen.

Im folgenden Beispielprogramm erzeugen wir zuerst ein Formular, das eine Listbox mit Mehrfachauswahlmöglichkeit enthalten soll. Der Benutzer kann hier mehrere Farben auswählen, die dann auf der zweiten Seite angezeigt werden sollen.

```
<html>
<head>
  <title>colors.cfm mit Multi-Select-Box</title>
</head>
<body>
<form action="processListBox.cfm" method="post">
  <select size="4" name="box" multiple>
    <option value="Red">Red
    <option value="Black">Black
    <option value="Aqua">Aqua
    <option value="Fuchsia">Fuchsia
    <option value="Navy">Navy
    <option value="Green">Green
    <option value="Teal">Teal
  </select>
  <input type="submit">
</form>
</body>
```

Listing 3.2: colors.cfm

Beachten Sie, dass im `<SELECT>`-Tag das Attribut *MULTIPLE* gesetzt sein muss!

Das zugehörige Antwort-Template *processListBox.cfm* soll die ausgewählten Elemente gleich in der entsprechenden Farbe anzeigen:

```
<html>
<head>
  <title>processListBox.cfm</title>
</head>
<body>
<CFPARAM NAME="FORM.box" DEFAULT="" />
<CFLOOP INDEX="color" LIST="#FORM.box#" DELIMITERS=", ">
  <CFOUTPUT>
    <font color="#color#">#color#</font><br>
  </CFOUTPUT>
</CFLOOP>
</body>
</html>
```

Listing 3.3: processListBox.cfm

Zuerst definieren wir mit dem `<CFPARAM>`-Tag einen Defaultwert für die Listbox, denn wenn der Benutzer keine Auswahl trifft, wird auch keine FORM-Variable erzeugt!

Da alle ausgewählten Farbwerte als Elemente einer kommaseparierten Liste in der Variablen `FORM.box` übergeben werden, benutzen wir eine Listenschleife (mehr dazu im Kapitel über Schleifen), um an die einzelnen Werte zu gelangen. In der Schleife erzeugen wir einen dynamischen HTML-Code, der nicht nur den Farbwert als Text ausgibt, sondern den Text auch gleich in der entsprechenden Farbe schreibt.

Wenn Sie einer Gruppe von Checkboxes den gleichen Namen geben, wird die getroffene Auswahl auf die gleiche Weise übertragen. Mit dem `VALUE`-Attribut der Checkbox können Sie den Wert festlegen, der übertragen werden soll. Dieser Wert wird auch hier nur übertragen, wenn die Checkbox angekreuzt ist.

Durch die Verwendung eines einheitlichen Namens erzeugt ColdFusion wieder eine Liste mit allen aktiven Checkboxes. Der folgende HTML-Code hat dieselbe Funktionalität wie das oben beschriebene Beispiele bezüglich der Listbox:

```
<html>
<head>
  <title>colors.cfm mit Checkboxes</title>
</head>
<body>
<form action="processListBox.cfm" method="post">
  <input type="Checkbox" name="box" value="Red">Red
  <BR>
  <input type="Checkbox" name="box" value="Black">Black
  <BR>
  <input type="Checkbox" name="box" value="Aqua">Aqua
```

```

<BR>
<input type="Checkbox" name="box" value="Fuchsia">Fuchsia
<BR>
<input type ="Checkbox" name="box" value="Navy">Navy
<BR>
<input type ="Checkbox" name="box" value="Green">Green
<BR>
<input type ="Checkbox" name="box" value="Teal">Teal
<BR>
<input type ="submit">
</form>
</body>
</html>

```

Listing 3.4: colors2.cfm mit Checkboxes

Zur Bearbeitung des Requests können Sie das Template *processListBox.cfm* komplett übernehmen.

3.5 Die Fieldnames-Variable

Bei jedem POST-Aufruf eines ColdFusion-Templates wird durch ColdFusion eine spezielle Variable namens `FORM.fieldnames` angelegt, die die Namen aller im POST übermittelten Variablennamen enthält.

Mit ihr können Sie dynamisch alle Wert in einer Schleife auslesen. Sie könnten z.B. einen Formular-Mailer programmieren, der Ihnen alle Formularwerte per Mail zusendet, ohne dass Sie den Mail-Code ändern müssten, wenn neue Elemente in das Formular integriert werden sollen. Wie das Ganze genau gemacht wird, werden wir Ihnen im Kapitel E-Mail zeigen. Jetzt wollen wir uns damit begnügen, die Namen und Werte in der Seite auszugeben:

```

<html>
<head>
  <title>Form.fieldnames</title>
</head>
<body>
  Folgende Parameter wurden dem Template übermittelt:
  <br><br>
  <!-- Schleife ueber alle FORM-Variablen -->
  <CFLOOP INDEX="idx"
    LIST="#FORM.fieldnames#"
    DELIMITERS=", ">
    <!-- Variablenname ermitteln -->
    <CFSET va="FORM.#{idx}#" />
    <CFOUTPUT>
      <!-- Variablenname ausgeben -->

```

```

        Name=<b>#idx#</b>
        <!-- Inhalt ermitteln und ausgeben --->
        Wert=<b>#evaluate(va)#</b>
        <br>
    </CFOUTPUT>
</CFLOOP>
</body>
</html>

```

Listing 3.5: fieldnames.cfm

Im Script wird die Funktion `evaluate()` benutzt, um den Inhalt einer Variablen auszuwerten.

Die Funktion wertet den Parameter-String (in diesem Fall ein Variablenname als Inhalt von `idx`) aus und gibt das Auswertungsergebnis (hier den Inhalt des übergebenen Variablennamens) zurück.

3.6 Die FORM-Struktur

FORM diente bisher als Präfix für die Variablen eines Formulars. ColdFusionMX bildet alle Scopes als Strukturen ab, deshalb ist FORM gleichzeitig eine Struktur-Variablen mit allen Formular-Variablen. Wir können uns die Struktur mit dem `<CFDUMP>`-Tag ansehen:

```
<CFDUMP var="#FORM#" />
```

Ein weiterer Vorteil ist, dass wir mit einem Collection-Loop alle Variablen dynamisch ausgeben können, genau so, wie wir das soeben mit Hilfe von `FORM.fieldnames` getan haben:

```

<cfloop collection="#FORM#" item="var">
    <cfoutput>
        #var# : #FORM[var]#<br>
    </cfoutput>
</cfloop>

```

Das letzte Beispiel mit der Schleife über die `FORM.fieldnames`-Variable lässt sich dann auch in einer kürzeren Form schreiben:

```

<CFLOOP INDEX="idx"
    LIST="#FORM.fieldnames#"
    DELIMITERS=", ">
    <!-- Variablenname ermitteln --->
    <CFOUTPUT>
        <!-- Variablenname ausgeben --->
        Name=<b>#idx#</b>

```

```

<!-- Inhalt ermitteln und ausgeben -->
Wert=<b>#FORM["#idx#"]#</b>
<br>
</CFOUTPUT>
</CFLOOP>

```

3.7 Formularvalidierung

Unter Formularvalidierung versteht man die Überprüfung eines Formulars auf unbedingt notwendige bzw. gültige Werte hin. Generell kann man zwischen serverseitiger Validierung (ausgeführt durch ColdFusion) und clientseitiger Validierung (geschieht im Browser, z. B. durch JavaScript) unterscheiden.

3.7.1 Serverseitige Formularvalidierung

ColdFusion bietet uns die Möglichkeit, Eingaben in Formularfeldern in mehrerlei Hinsicht zu überprüfen. So ist es z. B. möglich, ein Formularfeld als *required* zu kennzeichnen (es ist dann ein Pflichtfeld, eine Eingabe muss vorgenommen werden) oder den Inhalt auf Zahlenwerte zu beschränken.

Um ein Formularfeld zum Pflichtfeld zu machen, fügen wir ein weiteres Formularelement vom Typ *hidden* hinzu. Der Name dieses Elementes entspricht genau dem des zu überprüfenden Formularfeldes und wird um *"_required"* erweitert. Im VALUE-Attribut des neuen Elementes verwenden wir die Fehlermeldung, die ausgegeben werden soll, wenn die Pflichteingabe nicht erfolgt. Wollen wir also eine Eingabe in das Feld *username* erzwingen, so schreiben wir:

```

<input type="text" name="username" value="">
<input type="hidden" name="username_required"
      value="Bitte geben Sie einen Username ein!">

```

Klickt nun ein User auf *Submit*, ohne etwas in das Feld *username* eingegeben zu haben, wird die Actions-Seite nicht abgearbeitet, sondern es erscheint folgende Fehlermeldung:



Abbildung 3.2: Fehlermeldung nach unvollständiger Eingabe (serverseitige Validierung)

Mit diesem Mechanismus sind in ColdFusion folgende Überprüfungen möglich:

Überprüfung	Verlangte Eingabe
<code>_date</code>	Datum im angloamerikanischen Format: MM/TT/JJ bzw. MM/TT/JJJJ
<code>_eurodate</code>	Datum im »europäischen« Format: DD/MM/JJ bzw. TT/MM/JJJJ
<code>_float</code>	Zahl mit erlaubtem Dezimalpunkt (nicht Komma!)
<code>_integer</code>	Ganzzahl ohne Dezimalpunkt
<code>_range</code>	Bereich für Zahlenwerte. Die eingegebene Zahl muss zwischen den Grenzen liegen, die im VALUE-Attribut mit MIN=x and MAX=y angegeben werden.
<code>_required</code>	Eingabe erforderlich
<code>_time</code>	Zeit in den üblichen Formaten

Tabelle 3.1: Validierungsfunktionen für serverseitige Formularvalidierung

Die angegebenen Möglichkeiten sind natürlich nicht universell anwendbar, so sind die Datumsformate für deutsche Webseiten gänzlich ungeeignet und die Angabe von »`_range`« ermöglicht keine eigene Fehlermeldung, da im VALUE-Attribut des `hidden`-Elementes die Grenzen des erlaubten Bereiches angegeben werden müssen:

```
<input type="hidden" name="anzahl_range" value="MIN=1 and MAX=10">
```

Hinweis:

Überprüfungen wie `_integer` oder `_range` schließen nicht automatisch die `_required`-Überprüfung mit ein!

Die vorgestellte Validierung ist zwar sehr bequem, aber leider weder besonders schön noch benutzerfreundlich. Das Aussehen der Fehlermeldung kann nicht beeinflusst werden und zudem ist ein »Zurück«-Link nicht realisierbar. Besser, aber auch aufwändiger, ist da auf jeden Fall eine »manuelle« Validierung, in der man die Variablen einzeln auf ihre Gültigkeit hin überprüft.

3.7.2 Clientseitige Formularvalidierung mit <CFFORM>

ColdFusion bietet einen recht praktischen Ersatz für das HTML-Tag `<FORM>` an, das Tag `<CFFORM>`. Die Verwendung von `<CFFORM>` ist fast identisch mit der von `<FORM>`, lediglich das Attribut `METHOD=»POST«` braucht nicht angegeben zu werden, da `<CFFORM>` dieses Attribut automatisch setzt.

Selbstverständlich akzeptiert `<CFFORM>` alle Attribute des `<FORM>`-Tags. Innerhalb von `<CFFORM>` können alle normalen HTML-Formularelemente verwendet werden, es können aber auch die zusätzlichen Elemente `<CFINPUT>`, `<CFTEXTINPUT>`, `<CFSELECT>`, `<CFAPPLET>`, `<CFGRID>`, `<CFSLIDER>` und `<CFTREE>` verwendet werden. `<CFAPPLET>`, `<CFGRID>`, `<CFSLIDER>`

und `<CFTREE>` sind Java-Applets, die ein eigenes Java-Applet ins Formular einbinden können, ein Spreadsheet simulieren, in dem die Daten direkt upgedatet werden können, und einen Schieberegler bzw. eine Baumstruktur grafisch darstellen. Damit diese Elemente einwandfrei funktionieren, muss im Browser Java aktiviert sein. Details zu diesen Elementen entnehmen Sie bitte der Online-Dokumentation, wir wollen uns hier nur mit `<CFINPUT>`, `<CFTEXTINPUT>` und `<CFSELECT>` beschäftigen, da sie prinzipiell mit den »herkömmlichen« Formularelementen identisch sind, jedoch einige zusätzliche Funktionen bieten.

`<CFFORM>` wird von ColdFusion automatisch in ein `<FORM>`-Tag umgewandelt, das den Aufruf einer JavaScript-Funktion zum Überprüfen der im Formular enthaltenen Elemente enthält. Aus

```
<CFFORM NAME="myForm" ACTION="seite.cfm">
```

wird so im Browser:

```
<form name="myForm"
      action="seite.cfm"
      method=POST
      onSubmit="return _CF_checkmyForm(this)">
```

Gleichzeitig wird hier die JavaScript-Funktion `_CF_checkmyForm()` erzeugt, die die einzelnen Formularelemente abfragen kann.

`<CFINPUT>`

Anstelle eines `<INPUT>`-Feldes können wir innerhalb von `<CFFORM>` das `<CFINPUT>`-Tag verwenden. `<CFINPUT>` akzeptiert neben den Attributen des HTML-Tags `<INPUT>` folgende zusätzliche Attribute:

- ▶ `MESSAGE=»Text«` - Meldung die ausgegeben wird, falls die Überprüfung des Feldes einen Fehler ergibt.
- ▶ `REQUIRED=»Yes/No«` - Ist das Feld ein Pflichtfeld (`REQUIRED=»Yes«`)?
- ▶ `VALIDATE=»Format«` - Prüft den Feldinhalt auf folgende Kriterien hin.
- ▶ *date* – Datum im US-Format: MM/TT/JJJJ.
- ▶ *eurodate* – Datum im europäischen Format: TT/MM/JJJJ.
- ▶ *time* – Zeit im Format hh:mm:ss.
- ▶ *float* – Fließkommazahl.
- ▶ *integer* – Ganzzahl.
- ▶ *telephone* – Telefonnummer im US-Format ###-###-####. Der Trennstrich kann durch Leerzeichen ersetzt werden. Area code und Exchange müssen mit einer Zahl zwischen 1 und 9 beginnen.

- ▶ *zipcode* – Postleitzahl im US-Format Es kann eine fünf- oder neunstellige US-Postleitzahl im Format #####-#### eingegeben werden. Der Trennstrich kann durch ein Leerzeichen ersetzt werden.
- ▶ *creditcard* – Leerstellen und Striche werden entfernt und die Nummer nach dem MOD10-Algorithmus wird auf Echtheit hin überprüft.
- ▶ *social_security_number* – US-Sozialversicherungsnummer im Format ###-##-####. Die Trennstriche können durch Leerzeichen ersetzt werden.
- ▶ RANGE=>Minimum, Maximum<< – Der eingegebene Wert muss zwischen den angegebenen Minimum- und Maximum-Werten liegen.
- ▶ ONVALIDATE=>JS-Funktion<< – Benennt eine JavaScript-Funktion die die Überprüfung durchführt. Kann alternativ zu *validate* gebraucht werden.
- ▶ ONERROR=>JavaScript-Funktion<< – Benennt eine JavaScript-Funktion, die ausgeführt werden soll, falls die Validierung fehlschlägt.
- ▶ PASTTHROUGH=>HTML-Attribute<< – Weitere HTML-Attribute, die verwendet werden sollen.

Haben wir z. B. ein Eingabefeld *username*, in das der User unbedingt seinen Namen eingeben soll, so würden wir schreiben

```
<CFINPUT TYPE="text"
        NAME="username"
        REQUIRED="Yes"
        MESSAGE="Bitte geben Sie Ihren Usernamen an!" />
```

Gibt der User nun nichts in das Feld ein, so erscheint beim Klick auf den *Submit-Button* eine Fehlermeldung:



Abbildung 3.3: Fehlermeldung nach unvollständiger Eingabe (clientseitige Validierung)

Hinweis:

Bei `<CFINPUT>` muss dem Element immer ein Name gegeben werden, da die JavaScript-Funktion ansonsten nicht funktioniert!

Für europäische bzw. deutsche Anwendungen dürften die Optionen des *validate*-Attributes, die auf US-Formate ausgelegt sind, nicht besonders hilfreich sein. Allerdings sind die Prüfungskriterien *time*, *float* und *integer* schon sehr nützlich. In Kombination mit `required="yes"` und `range="Minimum, Maximum"` können uns diese Attribute im Vergleich zu einem mit `<FORM>` erstellten Formular sehr viel Schreibarbeit sparen. Wer schon einmal eine JavaScript-Funktion zur Überprüfung eines längeren Formulars geschrieben hat, wird wissen, was wir meinen, denn für komplexe Formulare wachsen solche Funktionen schnell auf u. U. einige Dutzend Programmzeilen JavaScript-Code an.

<CFTEXTINPUT>

`<CFTEXTINPUT>` bietet zur Validierung der Eingabe die gleichen Möglichkeiten wie `<CFINPUT>`. Allerdings handelt es sich bei `<CFTEXTINPUT>` um ein Java-Applet, das die Verwendung zusätzlicher Parameter für die Schriftart erlaubt. Um z.B. die Eingabe mit Font *Comic Sans MS* in der Größe 20p, in der Farbe Weiß auf rotem Hintergrund und zudem noch fett und kursiv darzustellen, würde man schreiben:

```
<CFTEXTINPUT NAME="username"
              FONT="Comic Sans MS"
              FONTSIZE="20"
              SIZE="50"
              BGCOLOR="FF0000"
              TEXTCOLOR="White"
              REQUIRED="No"
              BOLD="Yes"
              ITALIC="Yes"/>
```

Das Ergebnis dieser ungewöhnlichen Darstellung sehen Sie in der folgenden Abbildung.



Abbildung 3.4: Verwendung von `<CFTEXTINPUT>`

Hinweis:

Da `<CFTEXTINPUT>` ein Java-Applet verwendet, funktioniert es nur bei Browsern, die Java unterstützen und in denen es auch aktiviert wurde!

<CFSELECT>

`<CFSELECT>` kann innerhalb von `<CFFORM>` das normale `<SELECT>`-Tag ersetzen. Es bietet auch wieder die schon von `<CFINPUT>` bekannten Attribute `REQUIRED` und `MESSAGE` zur Validierung an, hat aber darüber hinaus noch große Vorteile, wenn man eine Select-Box mit den Ergebnissen einer Query füllen will.

Nehmen wir an, wir hätten eine Datenbankabfrage mit dem Namen `qryGetKategorien` durchgeführt, die uns die IDs und die Bezeichnungen von verschiedenen Kategorien zurückgibt, und wollten die Ergebnisse dieser Abfrage nun in einer Select-Box präsentieren. Die ID soll im `VALUE`-Attribut des jeweiligen Option-Tags angegeben, die Bezeichnung soll als Optionstext ausgegeben werden. Zusätzlich soll die Kategorie mit der ID 5 selektiert werden. (Details und Hinweise zu Datenbankabfragen entnehmen Sie bitte dem Kapitel [Datenbankzugriff](#)) Der herkömmliche Code für ein derart aufgebautes Selectfeld wäre dieser:

```
<select name="kategorie">
  <CFOUTPUT query="qryGetKategorien">
    <option value="#qryGetKategorien.ID#"
      <CFIF qryGetKategorien.ID EQ 5> SELECTED</CFIF>
      #qryGetKategorien.Bezeichnung#
  </CFOUTPUT>
</select>
```

Einfacher geht es mit `<CFSELECT>`:

```
<CFSELECT NAME="kategorie"
  QUERY="qryGetKategorien"
  VALUE=" ID"
  DISPLAY="Bezeichnung"
  SELECTED="5">
</CFSELECT>
```

Auch dieses ColdFusion-Tag hilft, komplexen Code zu vereinfachen und den Aufwand zu reduzieren.

3.7.3 Serverseitige Validierung oder doch clientseitig?

Beide Methoden haben ihre Vor- und Nachteile.

Der große Vorteil der serverseitigen Validierung ist der, dass die Überprüfung browserunabhängig ist. Sie funktioniert auch bei Browsern, die kein JavaScript verstehen oder bei denen JavaScript deaktiviert wurde.

Nachteilig wirkt sich aus, dass die Validierung erst nach dem Absenden des Formulars geschieht. Tritt ein Fehler auf, muss der User zurückgehen und alle Eingaben noch einmal vornehmen.

Bei der clientseitigen Validierung ist es genau andersherum: Die Validierung geschieht VOR einem Absenden des Formulars, funktioniert aber nur mit Browsern, die JavaScript verstehen und bei denen es auch aktiviert wurde.

Um ganz sicherzugehen, sollte man also stets beide Möglichkeiten implementieren. Die für den User komfortable clientseitige Validierung und zusätzlich die serverseitige Validierung, falls der User einen ganz exotischen Browser benutzt oder JavaScript deaktiviert hat.