Handguide

C# und ASP.NET 2.0

Autor: Dominik Hug

Inhaltsverzeichnis

1 Visual Studio Designer Grundlagen

1.1 Datei-Endungen

1.1.1 aspx

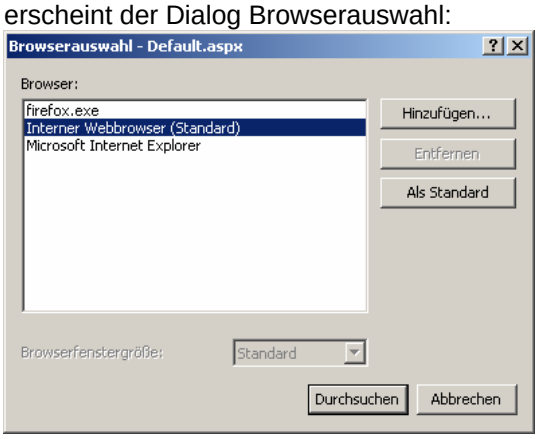
Enthält eine Mischung aus XHTML und XML

1.1.2 aspx.cs

Enthalten den C#-Code (von System.Web.UI.Page abgeleitete Klasse), der ausgeführt wird wenn die Seite von einem Browser angefordert wird.

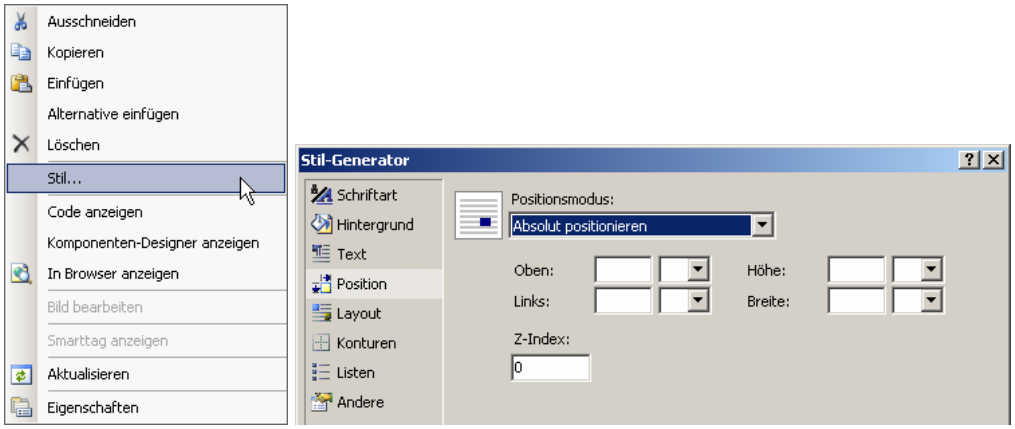
1.2 Visual Studio Browserauswahl

Zuerst im Projektmappen-Explorer das Projekt markieren, dann *Datei* | *Browserauswahl…* und es erscheint der Dialog Browserauswahl:



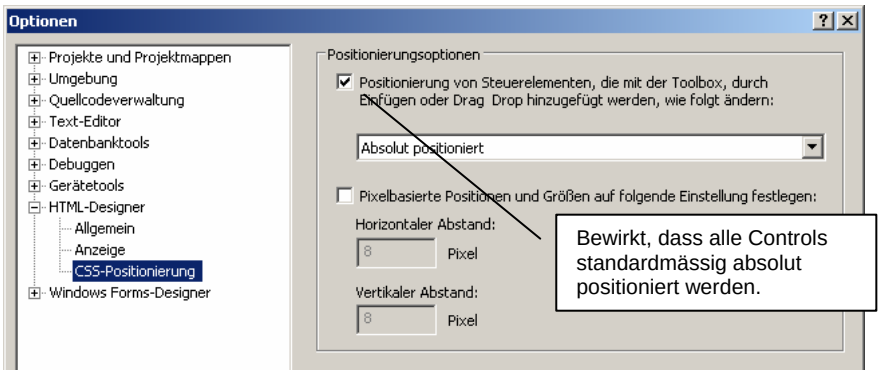
1.3 Controls absolut positionieren

1.3.1 Einzelnes Control absolut positionieren



1.3.2 Sämtliche Controls eines Projekts absolut positionieren

In der Entwurfsansicht einer Seite im Menü *Layout* | *Position* | *Optionen für automatische Positionierung* wählen.



2 Allgemeines

2.1 Automatische Postbacks

Existieren auf einer Seite ein oder mehrere Controls deren `AutoPostback` Property auf `true` gesetzt ist, platziert ASP.NET eine JavaScript Funktion namens `__doPostBack()` sowie zwei versteckte Felder (__EVENTTARGET für die ID des Controls und __EVENTARGUMENT für zusätzliche Informationen über das Event). Die Funktion `__doPostBack()` wird mit dem `onclick` oder `onchange` Event des Controls verknüpft und sendet das Formular ab.

Beispiel: Form mit einer CheckBox deren AutoPostback auf true ist

Die beiden versteckten Felder für Control-ID und Event-Argumente

```
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
...
```

Die JavaScript-Funktion welche die beiden versteckten Felder initialisiert und das Form absendet

```
var theForm = document.forms['form1'];
if (!theForm) {
    theForm = document.form1;
}
function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}
...
```

Das Control mit dem Aufruf von __doPostBack() im onclick-Event

```
<input id="CheckBox1" type="checkbox" name="CheckBox1"
onclick="javascript:setTimeout('__doPostBack(\'CheckBox1\',\'\')', 0)" />
```

Auf dem Server wird das Event `CheckBox1.CheckedChanged` gefeuert und ein allfällig damit verknüpfter EventHandler ausgeführt.

2.2 View State

Der View State Mechanismus löst ein fundamentales Problem der Zustandslosigkeit von HTTP. Vor einem Postback wird der Seiteninhalt normalerweise etwas geändert (z.B. der Text oder die Hintergrundfarbe eines Labels angepasst). Diese Änderungen reflektieren sich natürlich nicht im ASPX-Code.
Wurde der Code für eine Seite ausgeführt (kurz bevor das HTML gerendert und an den Client gesendet wird) untersucht ASP.NET die Properties aller Controls und prüft ob sie sich gegenüber dem Initialzustand (dem ASPX-Code) geändert haben. Falls ja wird dies in einer Name/Value Collection gespeichert. Diese Information wird in einen Base64[1] String serialisiert und in einem versteckten Feld namens `__VIEWSTATE` gespeichert.

Bei einem Postback geschieht folgendes:

1. ASP.NET kreiert die Controls basierend auf den Standardwerten im ASPX-Code.

[1] Die Base64-Codierung verhindert das Auftreten von Sonderzeichen die in HTML ungültig sind.

2. ASP.NET deserialisiert die View State Information und nimmt an den Controls die notwendigen Änderungen vor.
3. ASP.NET passt die Controls an die Formulardaten an, die durch den Postback übermittelt wurden (z.B. falls der User Text in eine Textbox geschrieben hat oder in einer ListBox ein anderes Item ausgewählt hat).
4. Es werden die Events ausgelöst, damit der Ereignisbehandlungscode zur Ausführung kommt.

Dieses Feature hat aber den Nachteil, dass die Seiten grösser werden. Der View State wird bei jedem Request / Response vom Client zum Server und wieder zurück übertragen.

Der View State kann auf den Ebenen Anwendung, Page und Control aktiviert / deaktiviert werden. Der View State kann für jedes Control individuell aktiviert / deaktiviert werden, indem man dessen `EnableViewState` Property auf `true` oder `false` setzt.

Der View State ermöglicht dem Programmierer auch eigene Werte darin zu speichern und somit den Session State zu entlasten.

Beispiel: Anzahl Seitenaufrufe im View State speichern

```
protected void Page_Load(object sender, EventArgs e) {
    if (ViewState["Anzahl"] == null) {
        ViewState["Anzahl"] = 1;
    } else {
        int anzahl = (int)ViewState["Anzahl"];
        ViewState["Anzahl"] = ++anzahl;
    }
}
```

2.2.1 View State Chunking

Die Grösse von versteckten Feldern ist unlimitiert, allerdings gibt es Proxies und Firewalls welche die Grösse der View State Felder begrenzen. Im Web.Config kann man die Grösse der versteckten Felder begrenzen, was dazu führt dass falls notwendig mehrere View State Felder verwendet werden.

```
<system.web>
    <!-- Grösse der ViewState Felder auf 1024 Bytes beschränken -->
    <pages maxPageStateFieldLength="1024" />
</system.web>
```

2.3 Session State

Klassen die nicht von `System.Web.UI.Page` oder `System.Web.UI.UserControl` usw. abgeleitet sind verfügen nicht über das Session Property. Um in eigenen Klassen trotzdem auf die Session zugreifen zu können, muss man wie folgt vorgehen:
`HttpContext.Current.Session[…] = …`
Die Klasse `System.Web.HttpContext` hat ein statisches Property `Current`, welche die Instanz von `HttpContext` zurückgibt, welche die Informationen für die aktuelle Webanforderung enthält.

2.4 Seitenübergänge

- Ein clientseitiger Seitenübergang (**Client Redirect**) wird vom Browser initiiert, entweder klickt der Anwender auf einen Link / Schaltfläche oder der HTML-Code definiert eine Weiterleitung via Meta-Tags.
- Bei einem serverseitigen Seitenübergang (**Server Redirect**) initiiert der Server, dass nun eine andere Seite angezeigt werden soll.

In ASP.NET Anwendungen sind clientseitige Seitenübergänge eher die Ausnahme. Der Vorteil von serverseitigen Übergängen ist, dass alle Seiten ihre Angelegenheiten (z.B. Datenbankzugriffe) selbst regeln, indem zuerst die aktuelle Seite erneut aufgerufen wird, die dort definierten Ereignisbehandlungen ausgeführt werden und erst danach der Aufruf der nächsten Seite erfolgt. Dies wird auch als **Postback-Architektur** bezeichnet.

2.4.1 Serverseitiger Übergang mit Response.Redirect()

Diese Methode leitet den Browser mit dem Statuscode 302 (Found) zu einer anderen Seite.
WICHTIG: diese Anweisung muss ausgeführt werden, bevor der HTTP-Header (mit diesem wird der Statuscode übertragen) gesendet wird. Man kann dies verhindern, indem man die Pufferung mit `Response.Buffer = true` aktiviert.

`Response.Redirect()` übernimmt als ersten Parameter die URL zu welcher umgeleitet werden soll. Dies kann auch eine klassische ASP- oder eine statische HTML-Seite sein.
Setzt man den zweiten (optionalen) Parameter auf `false`, wird die aktuelle Seite noch bis zum Ende ausgeführt. Ohne diesen Parameter ruft Redirect() intern `Response.End()` auf.

2.4.2 Serverseitiger Übergang mit Server.Transfer()

Im Gegensatz zu `Response.Redirect()` bekommt der Browser bei `Server.Tranfer()` nichts vom Seitenwechsel mit. Da nur die Programmausführung in einer anderen Datei fortgesetzt wird, ändert sich für den Browser nicht einmal die URL (er erhält den Inhalt der neuen Seite unter der ursprünglich abgefragen URL). Dies führt zum Nachteil, dass der User keinen Link auf die neue Seite setzen kann (resp. erst nachdem diese ihren ersten Postback gemacht hat).

Beim Seitenübergang mit `Server.Transfer()` gibt es neben der URL-Parameterliste und dem Session-Objekt eine dritte Methode für die Datenübergabe zwischen den Seiten. Mit dem Property `Page.PreviousPage` kann man auf die Seite zugreifen, welche den Aufruf initiiert hat. Die Methode `Page.FindControl(String id)` kann auch auf die Controls der aufrufenden Seite zugreifen. Eleganter ist jedoch, der aufrufenden Seite öffentliche Properties zu geben. Da die Methode FindControl() ein Objekt vom Typ `System.Web.UI.Page` zurück gibt, muss man vor dem Zugriff auf Properties noch eine Konvertierung vornehmen. **WICHTIG**: Dabei nicht den Klassennamen der Code-behind-Datei verwenden, sondern der Name der vom Framwork generierten Klasse im Namensraum ASP. Für eine Seite namens Anmeldung.aspx ist dies `ASP.anmeldung_aspx`.

2.5 HTML-Elemente auf dem Server ansprechen

Die HTML-Elemente im ASPX-Code wie <body> oder können auf dem Server angesprochen werden, indem man ihnen die Attribute `id` und `runat="server"` gibt. Dann erzeugt Visual Studio eine Seitenvariable welche im C# angesprochen werden kann.

Beispiel:
```
<body runat="server" id="Body">
    <form id="form1" runat="server">
    <div runat="server" id="div1">

    </div>
    </form>
</body>
```

2.5.1 Attribute von HTML-Elementen zur Laufzeit bearbeiten

Haben wir HTML-Elemente wie oben gezeigt auf dem Server verfügbar gemacht, können wir über deren Property `Attributes`, Attribute hinzufügen oder entfernen.

Beispiel
```
protected void Page_Load(object sender, EventArgs e) {
    div1.Attributes.Add("align", "center"); // Attribut hinzufügen
    Body.Attributes.Remove("onload"); // Attribut entfernen
}
```
Auch die Serversteuerelemente verfügen über die Collection Attributes, mit welcher man deren Attribute zur Laufzeit bearbeiten kann.

3 Die Klasse Page (System.Web.UI)

3.1 Wichtige Member von System.Web.UI.Page

`public event EventHandler` **`Load`**
Tritt ein wenn eine Seite geladen wird (d.h. von einem Browser angefordert wurde).

`public bool` **`IsPostBack`** `{ get; }`
Gibt true zurück, falls die Seite aufgrund der AutoPostBack zu sich selbst zurückgekehrt ist.

`public virtual void` **`Validate`**`()`
Weist alle Validierungssteuerelemente auf der Seite an, die zugewiesenen Informationen zu überprüfen.

`public bool` **`IsValid`** `{ get; }`

Ruft einen Wert ab, der angibt, ob die Seitenvalidierung erfolgreich war (d.h. die IsValid-Eigenschaft aller Validators true ist).

3.2 Smart Navigation

SmartNavigation existiert in ASP.NET 2.0 nicht mehr. Die Scroll-Position im Browser kann über PostBack's hinweg erhalten werden, indem man folgendes Property von Page auf `true` setzt:

```
[BrowsableAttribute(false)]
public bool MaintainScrollPositionOnPostBack { get; set; }
```

4 Standard-Controls

4.1 TextBox

VORSICHT! Setzt man `TextMode` auf `MultiLine`, funktioniert das Property `MaxLength` nicht mehr. D.h. der User kann mehr Zeichen in die TextBox eingeben als in MaxLength spezifiziert.

4.2 RadioButton

Damit von mehreren RadioButtons nur einer `Checked == true` haben kann, muss man bei all diesen RadioButtons die Eigenschaft `GroupName` auf den gleichen Wert setzen.

4.3 Panel

Ermöglicht unter anderem das ein- und ausblenden mehrerer Controls, indem man `Panel.Visible` auf `true` oder `false` setzt (wirkt sich auf alle enthaltenen Controls aus).

VORSICHT mit der Eigenschaft Height
Zieht man ein Panel auf eine Page, wird automatisch die Eigenschaft `Height` initialisiert. Enthält das Panel aber Steuerelemente mit veränderlicher Höhe (wie z.B. GridView), überlappt sich das Panel mit den nachfolgenden Steuerelementen. Dies kann verhindert werden, indem man das `Height` Property nicht setzt resp. löscht.

4.4 TreeView

Das TreeView Steuerelement zeigt hierarchisch strukturierte Daten an. Die angezeigte Baumstruktur darf mehrere Wurzelknoten aufweisen. Jeder Knoten ist eine Instanz von TreeNode (System.Web.UI.WebControls).

4.4.1 TreeNodes

Die Knotenstruktur kann direkt mit verschachtelten `<asp:TreeNode>`-Elementen aufgebaut werden.

Beispiel: Eine TreeView im ASPX-Code initialisieren:

```
<asp:TreeView ID="TreeView1" runat="server">
    <Nodes>
        <asp:TreeNode Text="InfoLex" Value="0" NavigateUrl="~/Default.aspx">
            <asp:TreeNode Text="Programmierung" Value="1">
                <asp:TreeNode Text=".NET" Value="2"></asp:TreeNode>
                <asp:TreeNode Text="Java" Value="3"></asp:TreeNode>
            </asp:TreeNode>
        </asp:TreeNode>
    </Nodes>
</asp:TreeView>
```

Der Baum wird mit seinen Knoten im Designer angezeigt. Ist das `NavigateUrl` Property eines Knotens gesetzt, bewirkt ein Klick einen Seitenwechsel. Ist das Property nicht gesetzt, kommt es zu einem Postback und das Ereignis `SelectedNodeChanged` des TreeView Steuerelements wird ausgelöst.

Mit der Eigenschaft `ImageUrl` kann man dem Knoten ein Icon hinzufügen. Versuche, diese Bilder aus einer Ressourcendatei zu holen schlugen bisher fehl, platziert man die Bilder aber als normale Dateien im Projekt, funktioniert es.

4.4.2 Formatierung

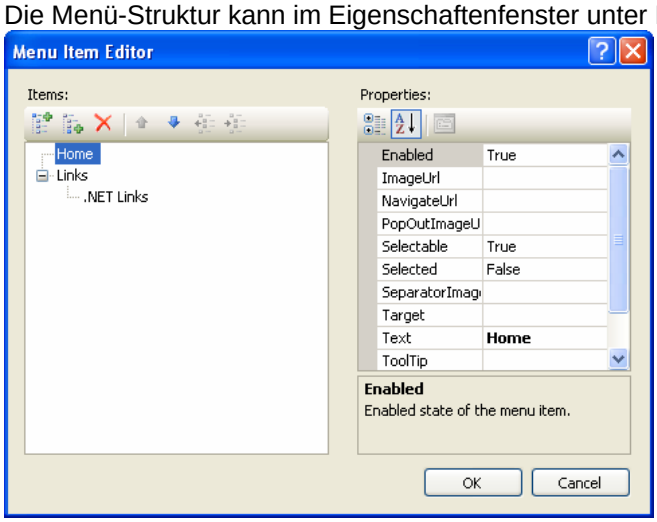
Mit den Eigenschaften

- `NodeStyle`
- `RootNodeStyle`
- `ParentNodeStyle`
- `LeafNodeStyle`
- `SelectedNodeStyle`

von TreeView können wir den Knoten gewünschte Formatierungen geben.

4.5 Menu

Die Menü-Struktur kann im Eigenschaftenfenster unter Items bearbeitet werden.

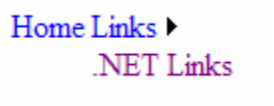


Im ASPX-Code werden die MenuItems wie bei der TreeView als verschachtelte XML-Elemente dargestellt.

```
<asp:Menu ID="Menu1" runat="server" OnMenuItemClick="Menu1_MenuItemClick">
    <Items>
        <asp:MenuItem Text="Home" Value="Home"></asp:MenuItem>
        <asp:MenuItem Text="Links" ToolTip="Hier sind ganz spezielle Links" Value="Links">
            <asp:MenuItem Text=".NET Links" Value=".NET Links" NavigateUrl="DotNet.aspx">
            </asp:MenuItem>
        </asp:MenuItem>
    </Items>
</asp:Menu>
```

Ist bei einem MenuItem die Eigenschaft `NavigateUrl` gesetzt, bewirkt ein Klick die Navigation auf diese Seite. Ist `NavigateUrl` nicht gesetzt, kommt es zu einem Postback, und es wird das Event `MenuItemClick` ausgelöst.
Die Eigenschaft `Orientation` kann auf `Horizontal` oder `Vertical` gesetzt werden.

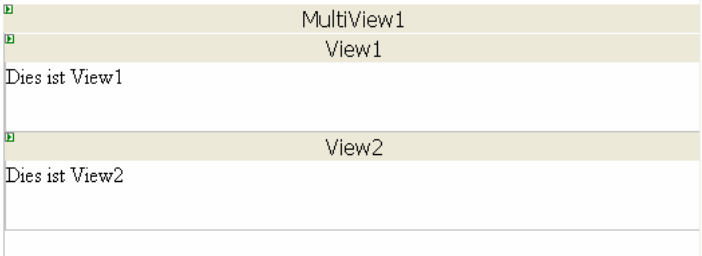
Horizontal



Vertical

4.6 MultiView und View

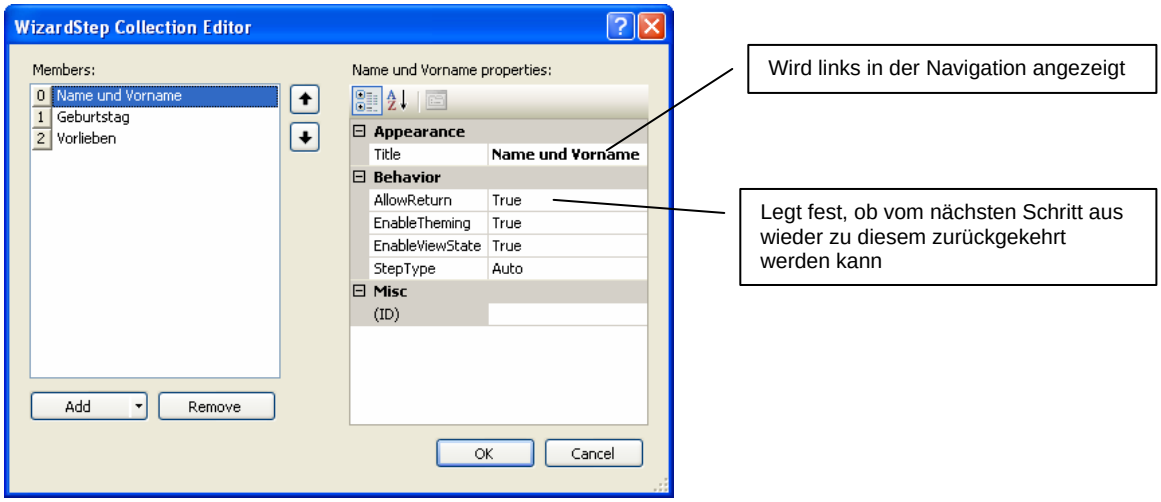
Ein `MultiView` ist ein Container für mehrere `Views`, wobei immer nur eine View gleichzeitig angezeigt wird. Eine View kann nur innerhalb eines MultiView Controls platziert werden.



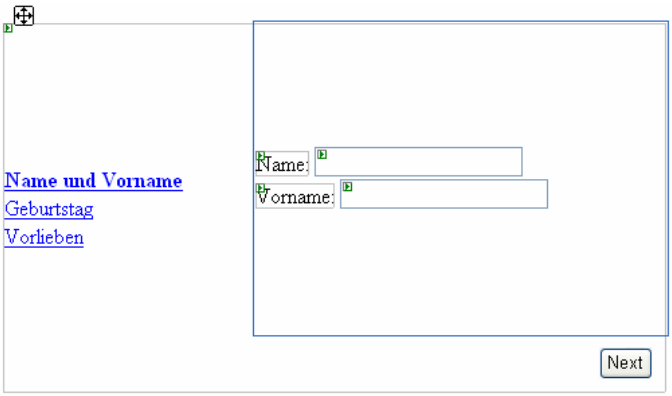
Mit dem Property `MultiView.ActiveViewIndex` oder der Methode `MultiView.SetActiveView(View)` kann man die dargestellte View auswählen.
Der View State der einzelnen Views bleibt über Postbacks erhalten, auch wenn sie nicht sichtbar sind.

4.7 Wizard

Das Wizard Control ermöglicht, Daten in mehreren Schritten zu erfassen ("Erfassungsassistent"). Die Collection `WizardSteps` enthält Instanzen von `WizardStepBase`. Die Eigenschaft WizardSteps ermöglicht, im Eigenschaftenfenster den Dialog "Wizard Step Collection Editor" zu öffnen, in welchem die verschiedenen Schritte definiert werden können.



Hat man die Schritte im WizardStep Collection Editor definiert, erscheinen sie in der Design-Ansicht rechts.



Jeder WizardStep hat einen `StepType`. Es gibt folgende Werte mit folgender Bedeutung:

StepType	Beschreibung
`Auto`	Die gerenderten Navigationsschaltflächen sind durch die Reihenfolge gesteuert. Der erste Step hat nur einen "Weiter" Button, die mittleren einen "Weiter" und "Zurück" Button, der letzte einen "Zurück" und "Fertig stellen" Button.
`Complete`	Es werden keine Navigationsschaltflächen dargestellt. Dieser Wert macht nur für den letzten Schritt Sinn. Er sollte dann eingesetzt werden, wenn man im letzten Schritt die eingegebenen Informationen zusammenfassend darstellt.
`Finish`	Eignet sich für den letzten Schritt bei welchem noch Daten eingegeben werden. Es erscheinen die Schaltflächen "Zurück" und "Fertig stellen".
`Start`	Lediglich die Schaltfläche "Weiter" wird dargstellt.
`Step`	Ein beliebiger Schritt zwischen dem ersten und letzten. Es werden die Schaltflächen "Zurück" und "Weiter" dargestellt.

4.7.1 Wichtige Eigenschaften von Wizard

`HeaderText`
Erscheint bei allen Schritten als Header (Ausnahme: Schritte vom Typ `Complete`)

4.7.2 Wichtige Events von Wizard

`ActiveStepChanged`

Wird ausgelöst, wenn eine Navigation erfolgte und auch beim Darstellen es ersten Schrittes.

`NextButtonClick`

Je nach Szenario will man abhängig von den Benutzereingaben einen Schritt überspringen. Dazu kann man im NextButtonClick-EventHandler `ActiveStepIndex` oder `ActiveStep` auf den gewünschten Schritt setzen. Klickt der User danach auf den "Zurück" Button, wird der zuvor angezeigte Schritt angezeigt, nicht jener der in der WizardStep-Collection dem aktuellen vorgelagert ist.

4.7.3 Datenübernahme

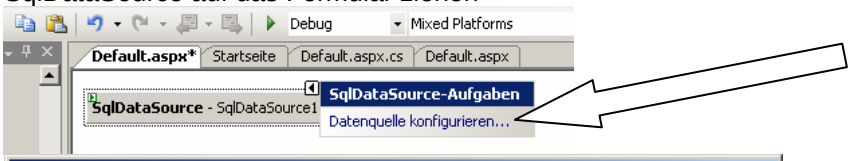
Die Werte von TextBoxen usw. aller Schritte werden im View State gespeichert. Hat man im ersten Schritt eine TextBox mit der ID `txtName` kann man z.B. im `FinishButtonClick` Event mit folgendem Code auf dessen Text-Eigenschaft zugreifen.

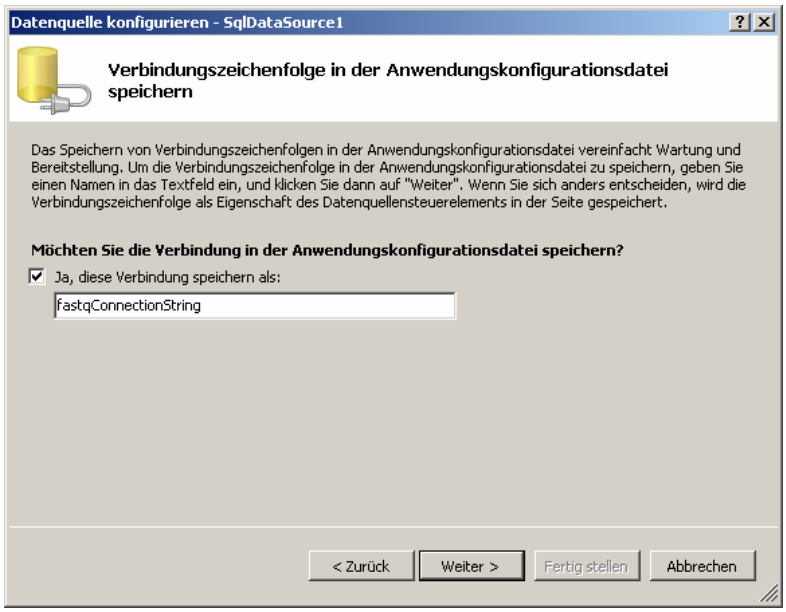
```
protected void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e) {
    TextBox txtBox = (TextBox)Wizard1.WizardSteps[0].FindControl("txtName");
    DBManager.Save(txtBox.Text);
}
```

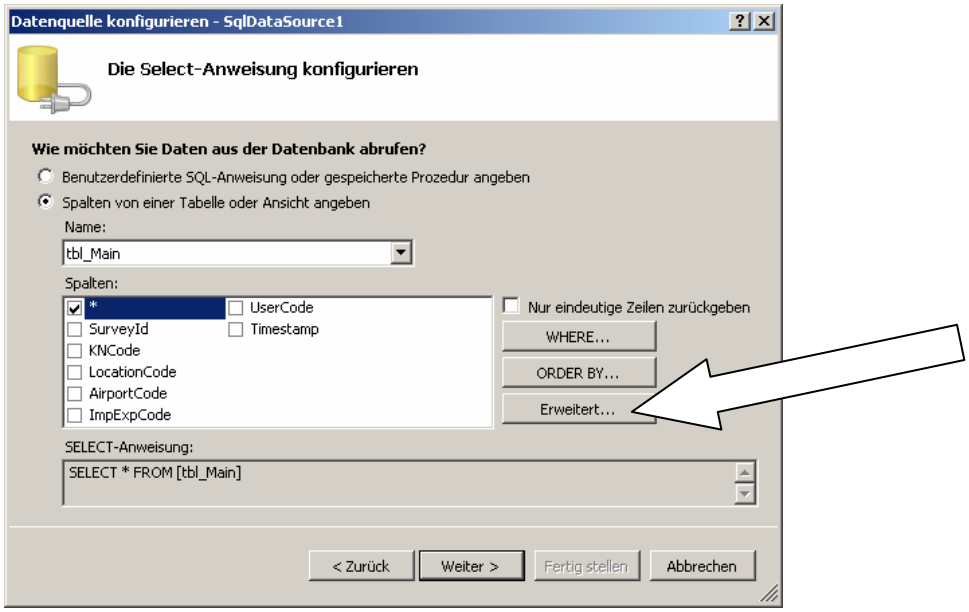
5 Standard-DatenControls

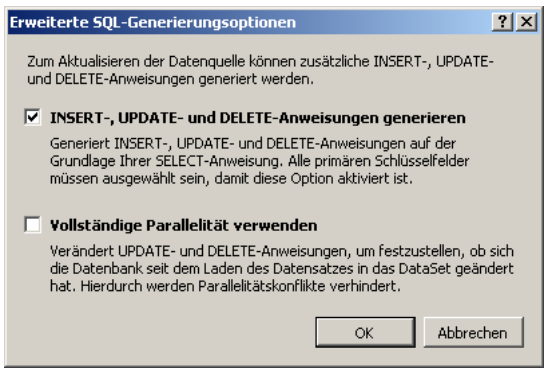
5.1 SqlDataSource und DetailsView verwenden

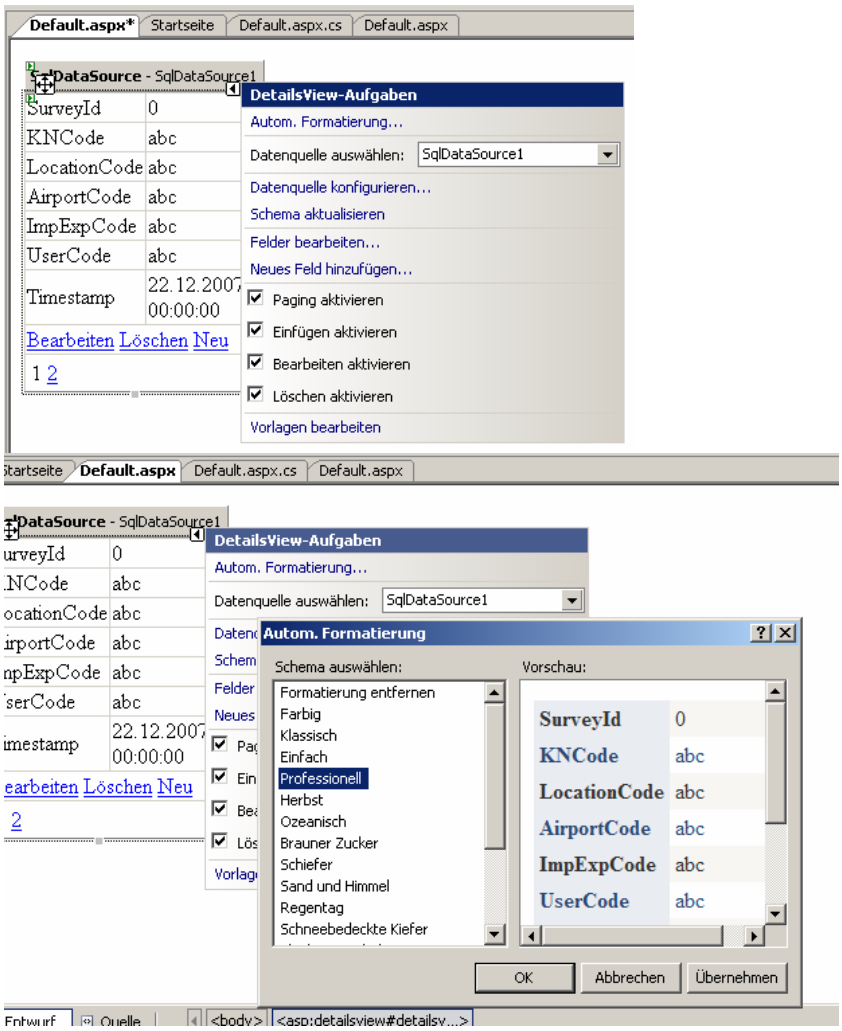
SqlDataSource auf das Formular ziehen




Datenquelle konfigurieren - SqlDataSource1
Verbindungszeichenfolge in der Anwendungskonfigurationsdatei speichern
Das Speichern von Verbindungszeichenfolgen in der Anwendungskonfigurationsdatei vereinfacht Wartung und Bereitstellung. Um die Verbindungszeichenfolge in der Anwendungskonfigurationsdatei zu speichern, geben Sie einen Namen in das Textfeld ein, und klicken Sie dann auf "Weiter". Wenn Sie sich anders entscheiden, wird die Verbindungszeichenfolge als Eigenschaft des Datenquellensteuerelements in der Seite gespeichert.
Möchten Sie die Verbindung in der Anwendungskonfigurationsdatei speichern?
Ja, diese Verbindung speichern als:
fastqConnectionString
< Zurück
Weiter >
Fertig stellen
Abbrechen


Datenquelle konfigurieren - SqlDataSource1
Die Select-Anweisung konfigurieren
Wie möchten Sie Daten aus der Datenbank abrufen?
Benutzerdefinierte SQL-Anweisung oder gespeicherte Prozedur angeben
Spalten von einer Tabelle oder Ansicht angeben
Name:
tbl_Main
Spalten:
*
SurveyId
KNCode
LocationCode
AirportCode
ImpExpCode
UserCode
Timestamp
Nur eindeutige Zeilen zurückgeben
WHERE...
ORDER BY...
Erweitert...
SELECT-Anweisung:
SELECT * FROM [tbl_Main]
< Zurück
Weiter >
Fertig stellen
Abbrechen


Erweiterte SQL-Generierungsoptionen
Zum Aktualisieren der Datenquelle können zusätzliche INSERT-, UPDATE- und DELETE-Anweisungen generiert werden.
INSERT-, UPDATE- und DELETE-Anweisungen generieren
Generiert INSERT-, UPDATE- und DELETE-Anweisungen auf der Grundlage Ihrer SELECT-Anweisung. Alle primären Schlüsselfelder müssen ausgewählt sein, damit diese Option aktiviert ist.
Vollständige Parallelität verwenden
Verändert UPDATE- und DELETE-Anweisungen, um festzustellen, ob sich die Datenbank seit dem Laden des Datensatzes in das DataSet geändert hat. Hierdurch werden Parallelitätskonflikte verhindert.
OK
Abbrechen



ReadOnly-Modus	Bearbeitungs-Modus	Einfüge-Modus
SurveyId 1 KNCode Hello01 LocationCode Loc-02 AirportCode LAX ImpExpCode ImpExpCode14 UserCode user2 Timestamp 22.12.2007 00:00:00 Bearbeiten Löschen Neu 1 2 3 4 5 6 7	SurveyId 1 KNCode Hello01 LocationCode Loc-02 AirportCode LAX ImpExpCode ImpExpCode14 UserCode user2 Timestamp 22.12.2007 00:00:00 Aktualisieren Abbrechen 1 2 3 4 5 6 7	KNCode LocationCode AirportCode ImpExpCode UserCode Timestamp Einfügen Abbrechen

Texte der Buttons anpassen

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="True" AutoGenerateRows="False"
   CellPadding="4" DataKeyNames="SurveyId" DataSourceID="SqlDataSource1" ForeColor="#333333"
   GridLines="None" Height="50px" Width="125px" Font-Names="Verdana">
   <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
   <CommandRowStyle BackColor="#D1DDF1" Font-Bold="True" />
   <EditRowStyle BackColor="#2461BF" />
   <RowStyle BackColor="#EFF3FB" />
   <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
   <Fields>
      <asp:BoundField DataField="SurveyId" HeaderText="SurveyId" InsertVisible="False"
         ReadOnly="True" SortExpression="SurveyId" />
      <asp:BoundField DataField="KNCode" HeaderText="KNCode" SortExpression="KNCode" />
      <asp:BoundField DataField="LocationCode" HeaderText="LocationCode" SortExpression="LocationCode" />
      <asp:BoundField DataField="AirportCode" HeaderText="AirportCode" SortExpression="AirportCode" />
      <asp:BoundField DataField="ImpExpCode" HeaderText="ImpExpCode" SortExpression="ImpExpCode" />
      <asp:BoundField DataField="UserCode" HeaderText="UserCode" SortExpression="UserCode" />
      <asp:BoundField DataField="Timestamp" HeaderText="Timestamp" SortExpression="Timestamp" />
      <asp:CommandField
           ShowEditButton="True"
           CancelText="Cancel"
           UpdateText="Update"
           EditText="Edit"
           InsertText="Insert"
           NewText = "New"
           DeleteText="Delete"
           ShowDeleteButton="True"
           ShowInsertButton="True"/>
   </Fields>
   <FieldHeaderStyle BackColor="#DEE8F5" Font-Bold="True" />
   <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
   <AlternatingRowStyle BackColor="White" />
   <PagerSettings Position="Top" />
</asp:DetailsView>
```

Anpassen der Hyperlink-Texte

5.2 DetailsView

Mit `Rows[rowId].Cells[cellId].Text = "<new Value>"` kann man Werte von Feldern setzen.

SurveyId	1
KNCode	Hello01
LocationCode	Loc-02
AirportCode	LAX
ImpExpCode	ImpExpCode14
UserCode	user2
Timestamp	22.12.2007 00:00:00
Bearbeiten Löschen Neu	
1 2 3 4 5 6 7	

5.3 GridView

Entspricht dem DataGridView aus der Windows-Forms Programmierung.

5.3.1 Wichtige Member von GridView

`public virtual int` **`EditIndex`** `{ get; set; }`

Der Index der Zeile, die sich momentan im Bearbeitungsmodus befindet. -1 falls keine Zeile bearbeitet wird.

5.3.2 Anpassen der Texte für die Bearbeitung:

```
<asp:GridView ID="GridView1" runat="server" AllowSorting="True" AutoGenerateColumns="False"
   CellPadding="4" DataKeyNames="SurveyId" DataSourceID="SqlDataSource1" ForeColor="#333333"
   GridLines="None">
   <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
   <Columns>
      <asp:CommandField ShowDeleteButton="True" ShowEditButton="True" ShowSelectButton="True"
       DeleteText="Delete"  SelectText="Select" CancelText="Cancel" EditText="Edit" UpdateText="Update" />
      <asp:BoundField DataField="SurveyId" HeaderText="SurveyId" InsertVisible="False"
         ReadOnly="True" SortExpression="SurveyId" />
```

	SurveyId	KNCode	LocationCode	AirportC
Edit Delete Select	1	FA-Q	Loc-02	LAX
Edit Delete Select	3	KN-03		
Edit Delete Select	4	KN-03	Loc04	PRS
Edit Delete Select	6	KN15		
Edit Delete Select	7	KN16		

Das Problem von Texten in der falschen Sprache lässt sich auch durch das Setzen der `Culture` und der `UICulture` in der Page-Direktive lösen:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Culture.aspx.cs" Inherits="Culture"
Culture="en-US" UICulture="en"%>
```

5.3.3 Zeilen aufgrund eines Spaltenwerts formatieren

Bevor ein GridView gerendert werden kann, muss für jede Zeile ein `GridViewRow`-Objekt erzeugt werden. Nach jedem Erstellen einer GridViewRow-Instanz wird das `RowCreated`-Ereignis gefeuert. Die EventArgs sind vom Typ `GridViewRowEventArgs`, dessen Property `Row` Zugriff auf die erzeugte Instanz bietet.
Die Klasse GridViewRow hat eine Eigenschaft RowType, mit welcher ermittelt werden kann, ob es sich um eine datengebundene Zeile handelt (Header, Footer, Pager usw. werden auch mit GridViewRow-Instanzen gerendert).

Beispiel: BackgroundColor aufgrund der CategoryID setzen

```
protected void GridView1_RowCreated(object sender, GridViewRowEventArgs e) {
    // e.Row.BackColor nur setzen, wenn es sich um eine datenbebundene Zeile handelt
    if (e.Row.RowType == DataControlRowType.DataRow) {
        int category = (int)DataBinder.Eval(e.Row.DataItem, "CategoryID");
        switch (category) {
            case 1:
                e.Row.BackColor = Color.LightBlue;
                break;
            case 2:
                e.Row.BackColor = Color.LightCoral;
                break;
            case 3:
                e.Row.BackColor = Color.LightCyan;
                break;
            …
        }
    }
}
```

ProductID	ProductName	CategoryID	UnitPrice
1	Chai	1	18.0000
2	Chang	1	19.0000
3	Aniseed Syrup	2	10.0000
4	Chef Antons Cajun Seasoning	2	22.0000
6	Grandma's Boysenberry Spread	2	25.0000
7	Uncle Bob's Organic Dried Pears	7	30.0000
8	Northwoods Cranberry Sauce	2	40.0000
9	Mishi Kobe Niku	6	97.0000

5.3.4 Ausgewählte Zeile nach Sortierung erhalten

Selektiert der User die dritte Zeile, wird `GridView.SelectedIndex` auf 2 gesetzt. Nimmt der User danach aber eine Neusortierung vor, ist an dritter Position meist eine andere Zeile. D.h. es ist nicht mehr die Zeile mit dem gleichen Schlüssel selektiert.

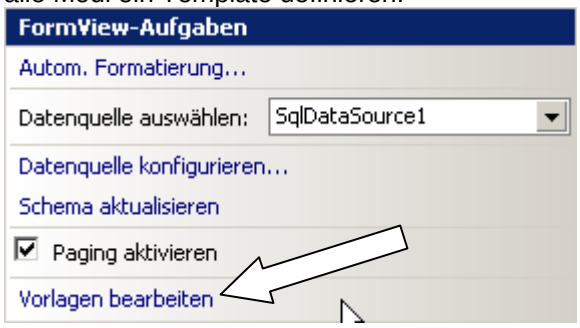
Lösung

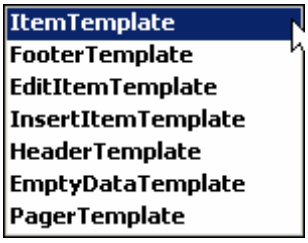
```
/// <summary>Nachdem der User eine Zeile Selektiert hat,
/// deren Schlüssel im Session-State speichern.</summary>
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e) {
    if (GridView1.SelectedIndex != -1) {
        // GridView.SelectedValue enthält den Schlüsselwert der selektierten Zeile
        // Welche Spalte die Schlüsselwerte enthält wird durch
        // GridView.DataKeyNames festgelegt
        Session["SelectedValue"] = GridView1.SelectedValue.ToString();
    }
}

/// <summary>Nachdem die Datenbindung beim GridView erfolgte, aufgrund des
/// Wertes im Session-State die richtige Zeile suchen und SelectedIndex setzen</summary>
protected void GridView1_DataBound(object sender, EventArgs e) {
    if (Session["SelectedValue"] != null) {
        string selectedValue = (string)Session["SelectedValue"];
        foreach (GridViewRow row in GridView1.Rows) {
            // die Auflistung GridView.DataKeys[] enthält für jede Zeile den Schlüsselwert
            if (GridView1.DataKeys[row.RowIndex].Value.ToString() == selectedValue) {
                GridView1.SelectedIndex = row.RowIndex;
                return;
            }
        }
    }
}
```

5.4 FormView

Ähnlich wie die DetailsView, stellt das FormView nur einen Datensatz gleichzeitig dar. Man kann für alle Modi ein Template definieren.





5.4.1 Wichtige Member von FormView

`public void` **`ChangeMode`**`(FormViewMode newMode)`
Versetzt das Control in einen der drei Zustände der Enumeration FormViewMode (Insert, Edit und ReadOnly).

`public FormViewMode` **`CurrentMode`** `{ get; }`

5.4.2 Pager-Template in den Bearbeitungs-Modi ausblenden

Mit `FormView1.PagerSettings.Visible = true|false` kann das Pager-Template programmatisch ein- und ausgeblendet werden.

5.5 Weitere Tipps zu GridView und FormView

5.5.1 Zeit ohne Datum in Access speichern

Die von der SqlDataSource automatisch generierten DateTime Parameter in Parameter vom Typ String ändern.
Beispiel:

```
<UpdateParameters>
   <asp:SessionParameter Name="SurveyId" SessionField="SurveyId" Type="Int32" />
   <asp:Parameter Name="Zone" Type="String" />
   <asp:Parameter Name="DaySort" />
   <asp:Parameter Name="Day" Type="String" />
   <asp:Parameter Name="TourNumber" Type="String" />
   <asp:Parameter Name="LeavingTime" Type="DateTime" />
   <asp:Parameter Name="DeliveryTime" Type="DateTime" />
```

Hier "String" statt "DateTime" verwenden

```
   <asp:Parameter Name="ID" Type="Int32" />
</UpdateParameters>
```

5.6 DataList

DataList ist ein Template-Control, damit es etwas anzeigt muss zumindest ein ItemTemplate definiert werden.
Mit der Eigenschaft `RepeatLayout` lässt sich festlegen, ob die Items als Zellen einer Tabelle (RepeatLayout = Table) oder mit dem `<div>` Tag (RepeatLayout = Flow) gerendert werden.
Mit der Eigenschaft `RepeatDirection` legt man fest, ob die Items untereinander oder nebeneinander angeordnet werden.

5.6.1 Selektion von Items

Ein SelectedItemTemplate macht nur Sinn, wenn der User auch ein Item selektieren kann. Um dies zu ermöglichen fügt man dem ItemTemplate einen Button oder einen HyperlinkButton hinzu. Auf das Klicken auf einen solchen Button kann man reagieren, indem man das ItemCommand-Event des DataList Controls behandelt. Falls man dem Template mehrere Buttons hinzufügt, sollte man deren CommandName-Properties benutzen um sie im EventHandler voneinander unterscheiden zu können.

Beispiel:

```
<asp:DataList ID="DataList1" runat="server" OnItemCommand="DataList1_ItemCommand">
    <ItemTemplate>
        <asp:LinkButton ID="LinkButton1" runat="server" CommandName="Select"
        Width="100px">Select</asp:LinkButton>
        <asp:Label ID="Label1" runat="server" Text="<%# Container.DataItem %>"
        Width="200px"></asp:Label>
    </ItemTemplate>
…

protected void DataList1_ItemCommand(object source, DataListCommandEventArgs e) {
    if (e.CommandName == "Select") {
        this.DataList1.SelectedIndex = e.Item.ItemIndex;
        this.DataList1.DataBind();
    }
}
```

5.7 SqlDataSource

Eine neue Zeile in die Datenbank einfügen

```
protected void btnInsert_Click(object sender, EventArgs e) {
       SqlDataSource1.InsertParameters["KNCode"].DefaultValue = "KN-New";
       SqlDataSource1.InsertParameters["LocationCode"].DefaultValue = "Loc-Code";
       SqlDataSource1.InsertParameters["AirportCode"].DefaultValue = "Air-C";
       SqlDataSource1.InsertParameters["ImpExpCode"].DefaultValue = "ImpExpCode";
       SqlDataSource1.InsertParameters["UserCode"].DefaultValue = "UserCode";
       SqlDataSource1.InsertParameters["Timestamp"].DefaultValue = DateTime.Now.ToString();
       SqlDataSource1.Insert();
}
```

5.8 ObjectDataSource

5.8.1 Allgemeines

Die ObjectDataSource benötigt ein Objekt für die CRUD-Operationen (**C**reate, **R**etrieve, **U**pdate und **D**elete). Der Typ dieses Objekts (auch **Data Mapper Object** genannt) kann mit dem `TypeName`-Property gesetzt werden.
Jedes Mal wenn die ObjectDataSource eine CRUD-Operation vornehmen muss, sucht es via Reklektion die entsprechende Methode und ruft sie auf. Diese Methoden können mit den Eigenschaften `SelectMethod`, `InsertMethod`, `UpdateMethod` und `DeleteMethode` spezifiziert werden (falls man Daten nur darstellen will, muss man nur `SelectMethod` setzen). Zusätzlich sollte das Property `SelectCountMethod` auf eine Methode gesetzt werden, welche die Anzahl Records zurückgibt. Handelt es sich dabei um nicht statische Methoden, so erzeugt die ObjectDataSource jedes mal eine Instanz dieser Klasse und zerstört sie danach. Um dies zu verhindern, kann man die Events `Creating` und `Disposing` behandeln.

Die ObjectDataSource kennt zwei Modi für die Übergabe von Parametern an DataMapper-Methoden:

- **Simple Types**: Die DataMapper-Methoden haben einen Parameter für jeden gebundenen Wert (weitere Parameter sind möglich).
- **Custom Objects**: Die Parameter für die DataMapper-Methoden übernehmen ein Objekt mit einem Property für jeden gebunden Wert, optional können weitere Parameter definiert werden. ObjectDataSource hat ein Property `DataObjectTypeName`, mit welchem man die Klasse für diese Parameter angeben kann.

Einige mögliche Rückgabetypen für die Select-Methode sind: DataView, DataSet, DataTable, IEnumerable.

5.8.2 Verwendung mit Custom Objects und GridView

Das Custom Object

```
public class Product {

    /// <summary>WICHTIG: Diese Klasse muss über einen
    /// Defaultkonstruktur (parameterlos) verfügen</summary>
    public Product() { }

    private int _productID;
    public int ProductID {
        get { return _productID; }
        set { _productID = value; }
    }

    private string _productName;
    public string ProductName {
        get { return _productName; }
        set { _productName = value; }
    }

    private decimal _price;
    public decimal Price {
        get { return _price; }
        set { _price = value; }
    }
}
```

Das DataMapper-Objekt

```
public class ProductDB {
    /// <summary>Instanzmethode um Paging effizient zu unterstützen</summary>
    public List<Product> SelectAll() {
        List<Product> lstProducts = new List<Product>();
        SqlConnection cnn = new SqlConnection();
        cnn.ConnectionString =
           ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
        SqlCommand cmd = cnn.CreateCommand();
        cmd.CommandText = "SELECT ProductID, ProductName, UnitPrice FROM Products";
        cnn.Open();
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read()) {
            Product product = new Product();
            product.ProductID = (int)reader["ProductID"];
            product.ProductName = (string)reader["ProductName"];
            product.Price = (decimal)reader["UnitPrice"];
            lstProducts.Add(product);
        }
        reader.Close();
        cnn.Close();
        return lstProducts;
    }
}
```

Die ObjectDataSource

```
<asp:ObjectDataSource
    ID="ObjectDataSource1" runat="server"
    TypeName="ProductDB"
    DataObjectTypeName="Product"
    SelectMethod="SelectAll"
</asp:ObjectDataSource>
```

Mit dem SmartTag des GridViews kann man das GridView sehr bequem an die ObjectDataSource binden und auch eine Auswahlfunktionalität hinzufügen.

Verwendet man auch noch eine automatische Formatierung, wird der ausgewählte Datensatz visuell hervorgehoben.

5.8.3 Update Funktionalität hinzufügen

Die Update-Methode

```
public class ProductDB {
    ...

    /// <summary>Diese Methode sollte aus Performancegründen statisch sein.</summary>
    public static void Update(Product product) {
        SqlConnection cnn = new SqlConnection();
        cnn.ConnectionString =
            ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
        SqlCommand cmd = cnn.CreateCommand();
        cmd.CommandText = "UPDATE Products SET ProductName = @ProductName, UnitPrice =
@UnitPrice WHERE ProductID = @ProductID";
        cmd.Parameters.Add(new SqlParameter("@ProductName", product.ProductName));
        cmd.Parameters.Add(new SqlParameter("@UnitPrice", product.Price));
        cmd.Parameters.Add(new SqlParameter("@ProductID", product.ProductID));
        cnn.Open();
        cmd.ExecuteNonQuery();
        cnn.Close();
    }
}
```

Je nach CurrentCulture kann es beim Update-Vorgang aufgrund der Formatierung der Werte Probleme geben. Dies kann man mit dem Hack in der CodeBehind-Datei gelöst werden.

```
protected void Page_Load(object sender, EventArgs e) {
    System.Threading.Thread.CurrentThread.CurrentCulture =
       System.Globalization.CultureInfo.InvariantCulture;
}
```

Die Delete-Methode

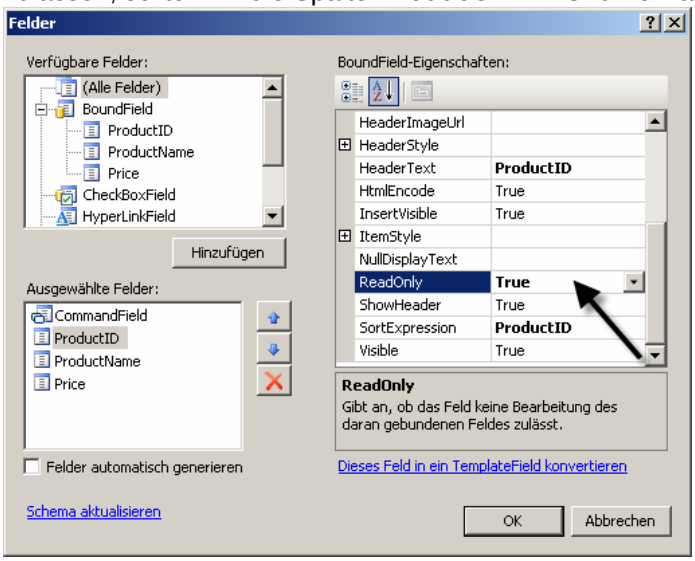
```
public class ProductDB {
    …
    /// <summary>Diese Methode sollte aus Performancegründen statisch sein.</summary>
    public static void Delete(Product product) {
        SqlConnection cnn = new SqlConnection();
        cnn.ConnectionString =
            ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
        SqlCommand cmd = cnn.CreateCommand();
        cmd.CommandText = "DELETE FROM Products WHERE ProductID = @ProductID";
        cmd.Parameters.Add(new SqlParameter("@ProductID", product.ProductID));
        cnn.Open();
        cmd.ExecuteNonQuery();
        cnn.Close();
    }
}
```

HINWEIS:
Damit dieses Beispiel mit der Northwind-Datenbank funktioniert, muss man zuerst die Foreign-Key-Constraint `FK_Order_Details_Products` aus der Datenbank löschen.

Bei der ObjectDataSource müssen wir nun noch die Properties `UpdateMethod` und `DeleteMethod` auf unsere beiden Methoden setzen.

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    TypeName="ProductDB"
    DataObjectTypeName="Product"
    SelectMethod="SelectAll"
    UpdateMethod="Update"
    DeleteMethod="Delete">
</asp:ObjectDataSource>
```

Das Löschen funktioniert nur, wenn wir die Eigenschaft `DataKeyNames` auf den Primary Key der Tabelle (ProductID) setzen. Da wir in der `Update()` Methode keine Änderung des Primärschlüssels zulassen, sollten wir die Spalte `ProductID` im GridView auf ReadOnly setzen.




```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="ProductID"
    DataSourceID="ObjectDataSource1">
    <Columns>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
            ShowSelectButton="True" />
        <asp:BoundField DataField="ProductID" HeaderText="ProductID"
            SortExpression="ProductID" ReadOnly="True"/>
        <asp:BoundField DataField="ProductName" HeaderText="ProductName"
            SortExpression="ProductName" />
        <asp:BoundField DataField="Price" HeaderText="Price" SortExpression="Price" />
    </Columns>
</asp:GridView>
```

5.8.4 Sortieren ermöglichen

Um Sortieren zu ermöglichen, müssen wir die `SelectAll()` Methode ändern und sie mit einem Parameter für den Sort-Expression ausstatten.

Neue Version von SelectAll (Änderungen fett hervorgehoben)

```
public List<Product> SelectAll(string sortedBy) {
    List<Product> lstProducts = new List<Product>();
    SqlConnection cnn = new SqlConnection();
    cnn.ConnectionString =
       ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
    SqlCommand cmd = cnn.CreateCommand();
    cmd.CommandText = "SELECT ProductID, ProductName, UnitPrice FROM Products";
    if (sortedBy != null && sortedBy != string.Empty) {
        cmd.CommandText += " ORDER BY " + sortedBy;
    }
    cnn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read()) {
        Product product = new Product();
        product.ProductID = (int)reader["ProductID"];
        product.ProductName = (string)reader["ProductName"];
        product.Price = (decimal)reader["UnitPrice"];
        lstProducts.Add(product);
    }
    reader.Close();
    cnn.Close();
    return lstProducts;
```

```
}
```

Sortieren in GridView ermöglichen

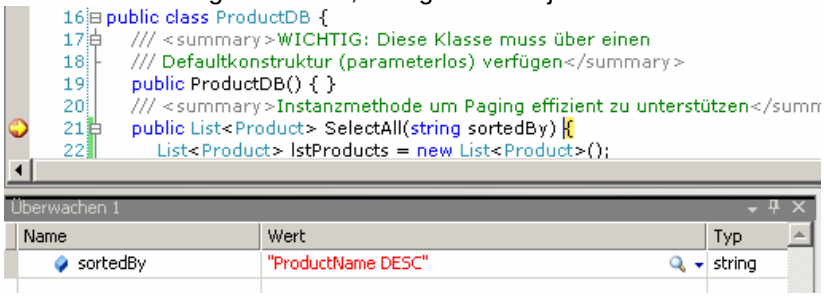
```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="ProductID"
    DataSourceID="ObjectDataSource1"
    AllowSorting="True"
    AllowPaging="True">
    <Columns>
        <asp:CommandField ShowDeleteButton="True" ShowEditButton="True"
            ShowSelectButton="True" />
        <asp:BoundField DataField="ProductID" HeaderText="ProductID"
            SortExpression="ProductID" />
        <asp:BoundField DataField="ProductName" HeaderText="ProductName"
            SortExpression="ProductName" />
        <asp:BoundField DataField="Price" HeaderText="Price" SortExpression="UnitPrice" />
    </Columns>
</asp:GridView>
```

Da die Eigenschaft `Price` der Klasse `Product` nicht mit dem Namen der in der DB zugrundeliegenden Spalte übereinstimmt, müssen wir `SortExpression` dieser Spalte in der GridView auf `"UnitPrice"` ändern.

Damit die ObjectDataSource weiss, welcher Parameter der `SelectAll()` Methode die Sortierungskriterien enthält, müssen wir deren `SortParameterName` Eigenschaft auf den Namen des Parameters (in unserem Fall `sortedBy`) setzten.

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    TypeName="ProductDB"
    DataObjectTypeName="Product"
    SelectMethod="SelectAll"
    UpdateMethod="Update"
    DeleteMethod="Delete"
    SortParameterName="sortedBy">
</asp:ObjectDataSource>
```

1. Beim ersten Aufruf der Seite (wenn vom User noch keine Sortierung ausgewählt wurde), übergibt die ObjectDataSource der `SelectAll(string sortedBy)` Methode als Argument `String.Empty`.
2. Wählt der User eine Sortierung nach ProductName, indem er auf diesen SpaltenHeader klickt, übergibt die ObjectDataSource der `SelectAll()` Methode als Argument `"ProductName"`.
3. Klickt der User erneut auf den SpaltenHeader von ProductName um eine absteigende Sortierung zu wählen, übergibt die ObjectDataSource `"ProductName DESC"`.



5.9 ConnectionStrings

Das Verzeichnis `App_Data` ist für .mdb oder .mdf (SQL-Server Dateien) vorgesehen. Im Web.config kann man dieses Verzeichnis wie folgt ansprechen:

```
<connectionStrings>
  <add
    name="fastqConnectionString"
    connectionString="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|fastq.mdb"
    providerName="System.Data.OleDb" />
</connectionStrings>
```

5.9.1 Zugriff auf den Connection-String im ASPX-Code

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:fastqConnectionString %>"
   DeleteCommand="DELETE FROM [tbl_Main] WHERE [SurveyId] = ?"
   InsertCommand="INSERT INTO [tbl_Main] ([SurveyId], [KNCode], [LocationCode], [
   ProviderName="<%$ ConnectionStrings:fastqConnectionString.ProviderName %>"
   SelectCommand="SELECT * FROM [tbl_Main]"
   UpdateCommand="UPDATE [tbl_Main] SET [KNCode] = ?, [LocationCode] = ?, [Airpo
```

5.9.2 Zugriff auf Connection-Strings in C#-Code

```
cnn.ConnectionString = ConfigurationManager.ConnectionStrings["<name>"].ConnectionString;
```

6 Validierung

6.1 CompareValidator

Vorgehen allgemein:

1. `ControlToValidate` setzen
2. `ControlToCompare` setzen
3. `Operator` setzen
4. `Type` setzen
5. `ValueToCompare` auf die zu vergleichende Eigenschaft setzen (meist `Text`)
6. `CausesValidation` des `ControlToValidate` auf `true` setzen

Validierung von Zeiten

Die Eigenschaft `Type` kann auf `Date` gesetzt werden. Dann funktioniert aber nur der Vergleich von Datumswerten. Für den Vergleich von Zeiten kann der Typ auf `String` gesetzt werden, dies funktioniert da "7:59" < "8:00" usw.

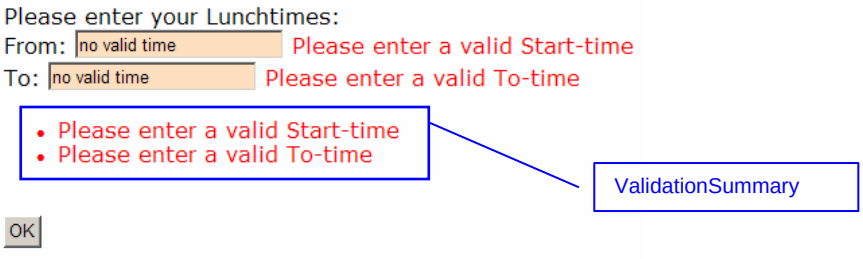
6.2 CustomValidator

Vorgehensweise:

1. Die Eigenschaft `ErrorMessage` setzen
2. Die Eigenschaft `ControlToValidate` auf eine TextBox setzen
3. Einen Event-Handler an das Event `ServerValidate` hängen
4. Im Falle eines **PostBacks** im `Page.Load`-Event `Validate()` aufrufen
5. Im Event-Handler mit `args.Value` den zu validierenden Wert holen und prüfen. Falls er gültig ist → `args.IsValid` auf `true` setzen, sonst auf `false`.

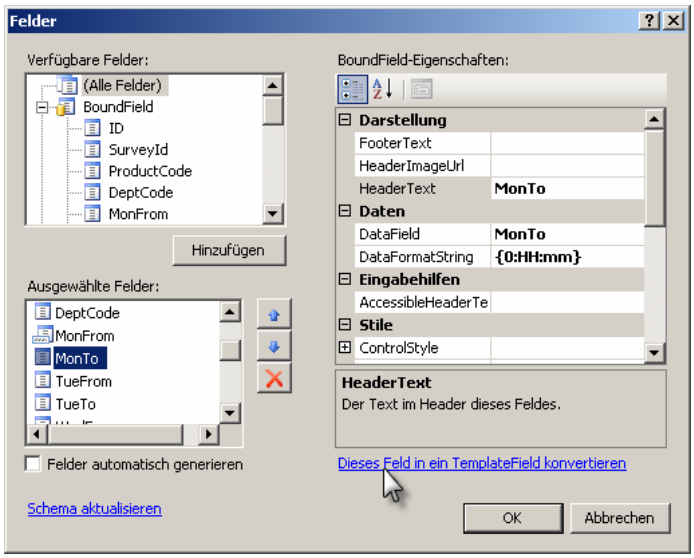
6.3 ValidationSummary

Stellt die ErrorTexte aller Validators deren `IsValid`-Eigenschaft = `false` ist, dar.



6.4 Validierung mit dem GridView

Dazu muss man zuerst das BoundField in der Columns-Auflistung des GridViews in ein TemplateField konvertieren (hierzu hat gibt es unten einen Hyperlink).



Vorher:

```
<asp:BoundField DataField="MonTo" HeaderText="MonTo" SortExpression="MonTo" HtmlEncode="False"
DataFormatString="{0:HH:mm}" ApplyFormatInEditMode="True" />
```

Nachher:

```
<asp:TemplateField HeaderText="MonTo" SortExpression="MonTo">
   <EditItemTemplate>
      <asp:TextBox
         ID="TextBox2" runat="server" Text='<%# Bind("MonTo", "{0:HH:mm}")%>'></asp:TextBox>
   </EditItemTemplate>
   <ItemTemplate>
      <asp:Label
          ID="Label2" runat="server" Text='<%# Bind("MonTo", "{0:HH:mm}") %>'></asp:Label>
   </ItemTemplate>
</asp:TemplateField>
```

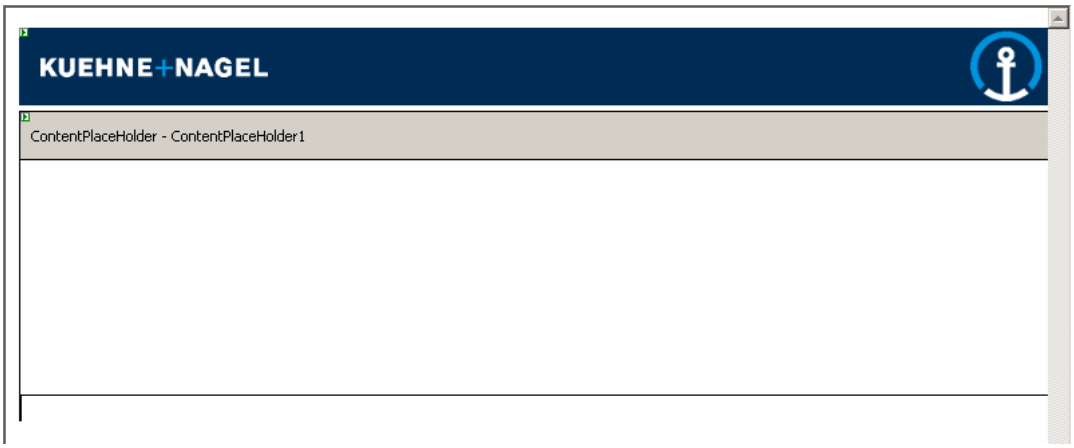
Jetzt kann man dem EditItemTemplate einen Validator hinzufügen:

```
<asp:TemplateField HeaderText="MonTo" SortExpression="MonTo">
   <EditItemTemplate>
      <asp:TextBox
         ID="TextBox2" runat="server" Text='<%# Bind("MonTo", "{0:HH:mm}") %>'></asp:TextBox>
      <asp:CustomValidator ID="CustomValidator2" runat="server" ControlToValidate="TextBox2"
         ErrorMessage="hh:mm" OnServerValidate="Time_ServerValidate"></asp:CustomValidator>
   </EditItemTemplate>
   <ItemTemplate>
      <asp:Label
         ID="Label2" runat="server" Text='<%# Bind("MonTo", "{0:HH:mm}") %>'></asp:Label>
   </ItemTemplate>
</asp:TemplateField>
```

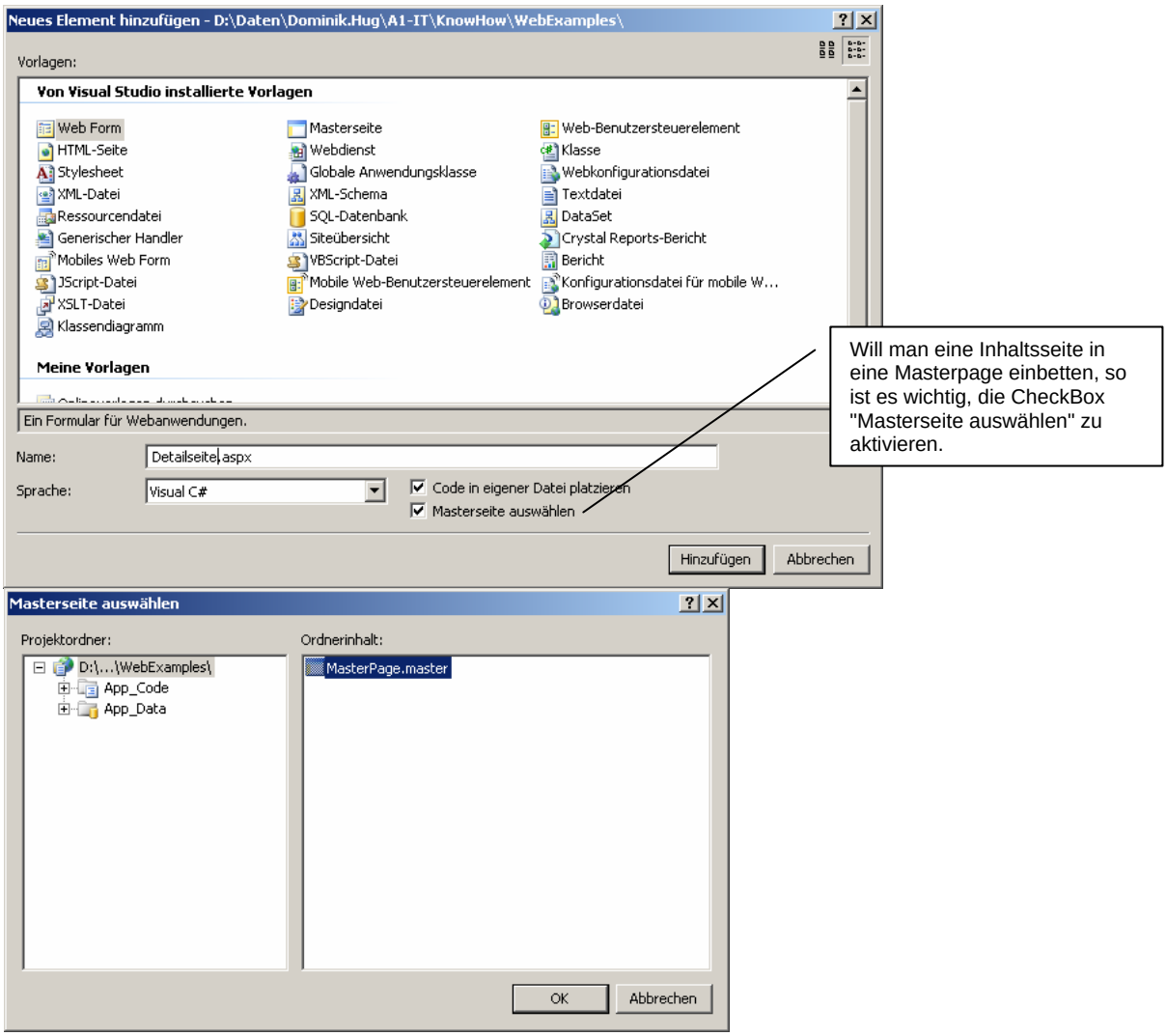
7 Masterpages

7.1 Masterpage anlegen

Um eine Masterpage (Template) zu erstellen, wählt man im Dialog "Neues Element hinzufügen" den Eintrag Masterseite.
Eine Masterpage enthält automatisch einen `ContentPlaceHolder` für die Inhaltsseiten.



7.2 Masterpage mit einer Inhaltseite benutzen



Um eine ASPX-Seite nächträglich mit einer Masterpage zu versehen, muss man in der Page-Direktive das Attribut MasterPageFile setzen.

```
MasterPageFile="~/MasterPage.master"
```

Anschliessend muss der ganze Code ausserhalb des Form-Elements entfernt werden. Denn `<html>`, `<head>`, `<body>` usw. sind ja bereits im Master vorhanden.

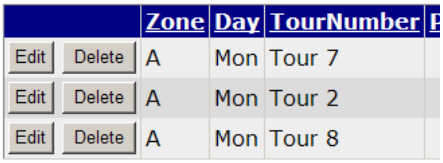
Setzt man im Eigenschaftenfenster einen Titel für eine Seite die eine Masterpage verwendet, wird der Titel nicht im `<title>`-Element gespeichert (dieses steht in nur in der Masterseite zur Verfügung) sondern im Title-Attribut der Page-Direktive.

```
Title="Detailseite"
```

8 Datenbindung

8.1 Formatierung

Dieses Beispiel ermöglicht die folgende Darstellung des Feldes `TourNumber`, welches vom Typ int ist:



	Zone	Day	TourNumber
Edit Delete	A	Mon	Tour 7
Edit Delete	A	Mon	Tour 2
Edit Delete	A	Mon	Tour 8

```
<ItemTemplate>
   <asp:Label ID="Label4" runat="server" Text='<%# Bind("TourNumber", "Tour {0}") %>'>
   </asp:Label>
</ItemTemplate>
```

8.2 Datum und Uhrzeit formatieren

8.2.1 FormView

Lässt man sich mit dem Assistenten Templates für FormView generieren, wird im Datenbindungsausdruck (innerhalb von <%# und %>) die `Bind()` Methode verwendet. Sie nimmt als zweiten (optionalen) Parameter einen Format-String entgegen.

```
<asp:TextBox ID="Q04_N_MonFromTextBox" runat="server" Text='<%# Bind("Q04_N_MonFrom",
"{0:HH:mm}")%>'>
```

8.2.2 GridView

Der `DataFormatString` scheint beim Typ `DateTime` nur zu wirken, wenn `HtmlEncode` auf `false` gesetzt ist.

```
<asp:BoundField
   DataField="MonFrom"
   HeaderText="MonFrom"
   SortExpression="MonFrom"
   HtmlEncode="false"
   DataFormatString="{0:HH:mm}"
   ApplyFormatInEditMode="true"/>
```

9 Security

9.1 Geschützte Verzeichnisse

Subdirectory für geschützte Seiten erstellen
Im Web.config der Applikation:

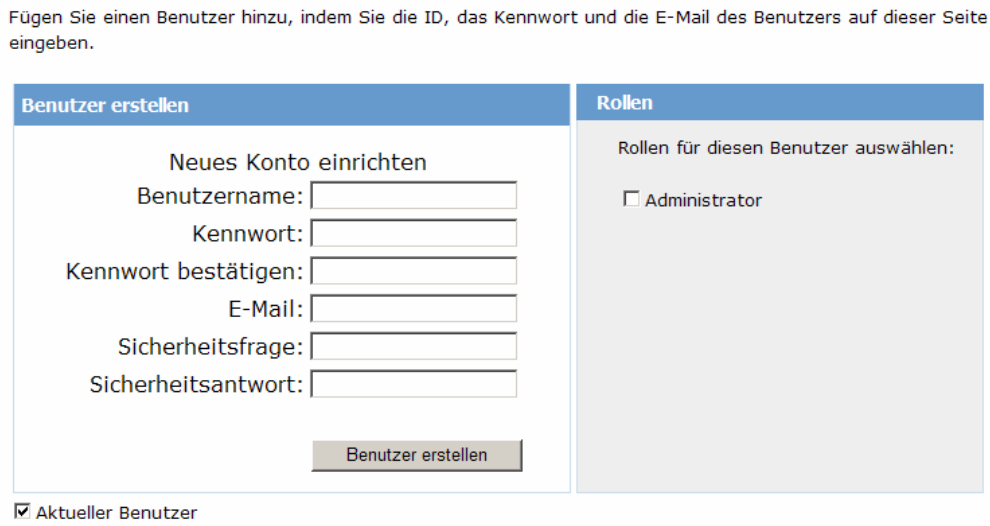
```
<authentication mode="Forms">
    <forms loginUrl="Admin/Login.aspx" />
</authentication>

<authorization>
    <allow users="?" />
</authorization>
```

Im Web.config des geschützten Verzeichnisses:

```
<authorization>
    <deny users="?" />
    <allow roles="Administrator" />
</authorization>
```

Unter Webseite | ASP.NET-Konfiguration Benutzer erstellen und einer Rolle zuordnen:



Ein Login-Control auf der Login-Seite platzieren.

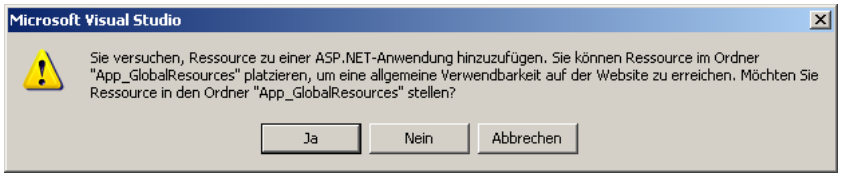
10 Ressourcen und Lokalisierbarkeit

10.1 Ressourcen in .NET Applikationen

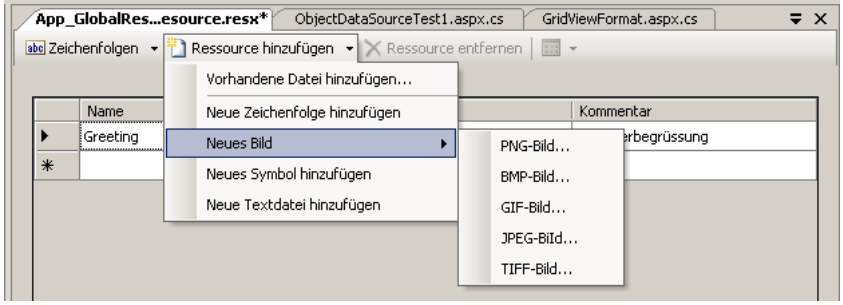
Ressourcendateien ermöglichen das Platzieren von Bildern und Strings in separaten Dateien, was deren Änderung bedeutend erleichtert, da man nicht den gesamten Quellcode durchkämmen muss. Das Hauptanwendungsgebiet von Ressourcen ist natürlich die Lokalisierung.
Ressourcen werden in .RESX-Dateien gespeichert, es handelt sich dabei um XML Dateien welche die Strings oder Links zu externen Dateien enthalten.

10.1.1 Ressourcen zu einer Web Applikation hinzufügen

Rechtsklick auf das Projekt im Projektmappen-Explorer, *Neues Element hinzufügen…* | *Ressourcendatei*.
Visual Studio fragt, ob es die Ressourcendatei im Ordner `App_GlobalRessources` platzieren soll. Es empfiehlt sich, diese Frage mit Ja zu beantworten.



Die Ressourcendatei erscheint in Visual Studio folgendermassen:



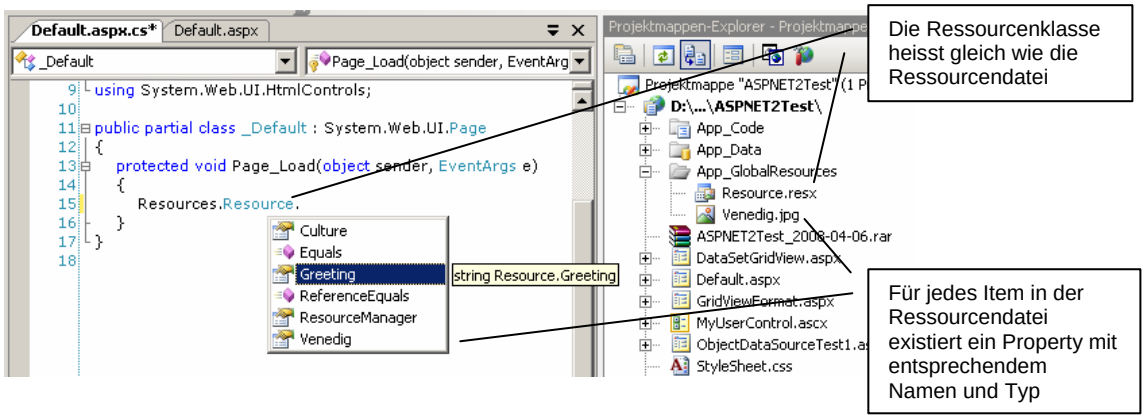
Durch *Ressource hinzufügen* | *Vorhandene Datei hinzufügen…* können beliebige externe Dateien zum Projekt hinzugefügt werden. Diese Dateien werden automatisch einem der Ressourcen-Ordner hinzugefügt und als eingebettete Ressource in die Binärdateien kompiliert.

Visual Studio unterscheidet zwischen Globalen und Lokalen Ressourcen. Auf **Globale Ressourcen** kann von jeder Seite der Applikation zugegriffen werden, sie liegen im Verzeichnis `/App_GlobalResources`. Auf **Lokale Ressourcen** kann nur von der Seite aus zugegriffen werden

für welche sie definiert wurden, sie liegen im Verzeichnis `/App_LocalResources` und tragen den Namen der Seite für die sie definiert wurden mit der Endung `.resx`.

10.1.2 Zugriff auf Ressourcen im Code

Visual Studio erzeugt beim Kompilieren für jede Ressourcendatei eine Klasse im Namespace `Resources`. Diese Klasse heisst gleich wie die Ressourcendatei und hat für jede darin enthaltene Ressource ein Property mit deren Name und deren Typ.



10.1.3 Zugriff auf Ressourcen im ASPX-Code

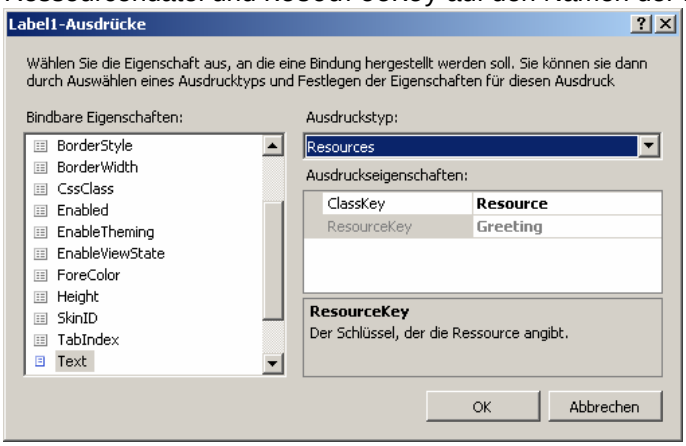
```
<asp:Label ID="Label1" runat="server" Text="<%$ Resources:Resource, Greeting %>" />
```

Die Syntax ist:

`$ Resources:`<Name der Resourcendatei>`,` <Name der Ressource>

Dies lässt sich auch visuell bewerkstelligen

Im Eigenschaftenfenster auf `(Expressions)` klicken und im erscheinenden Dialog die zu bindende Eigenschaft auswählen, `Ausdruckstyp` auf Resources setzen, `ClassKey` auf den Namen der Ressourcendatei und `ResourceKey` auf den Namen der Ressource setzen.



10.2 Lokalisierung von Web-Applikationen

Beim Lokalisieren einer Web-Applikation genügt es meistens nicht, nur die Strings anzupassen, es müssen auch die Formatierung von Währungsangaben, Zeitangaben usw. angepasst werden. Werden entsprechende User Eingaben entgegengenommen kommt auch deren Validierung hinzu.

10.2.1 Kulturen und Kulturcodes

Ressourcen werden normalerweise für alle unterstützten Kulturen erzeugt. Die oben erzeugten Ressourcen beziehen sich auf die Standard-Kultur, da wir dem Namen der Ressourcendatei keine kulturspezifische Information hinzugefügt haben. Wollen wir unsere Ressourcendatei für die Sprache

Englisch und das Land Amerika lokalisieren, müssen wir ihren Namen von `Resource.resx` in `Resource.en-US.resx` ändern.

Die Kulturcodes wie `en-US` bestehen aus einem Sprachcode (in Kleinbuchstaben) und optional einem Landes / Regionscode (in Grossbuchstaben). Diese Codes sind in RFC 1766 spezifiziert.

Sämtliche definierten Kulturcodes erhält man z.B. mit folgendem Code:

```
foreach (CultureInfo ci in CultureInfo.GetCultures(CultureTypes.AllCultures)) {
    Debug.WriteLine(ci.Name + " " + ci.EnglishName);
}
```

Ein Ausschnitt der Ausgabe:

```
cs-CZ Czech (Czech Republic)
da-DK Danish (Denmark)
de-DE German (Germany)
el-GR Greek (Greece)
en-US English (United States)
```

Man unterscheidet:

- **Invariante Kulturen**: Sind sprachlich dem Englisch zugeordnet, verfügen aber über kein Land oder Region.
- **Neutrale Kulturen**: Sind einer Sprache aber keinem Land / Region zugewiesen.
- **Spezifische Kulturen**: Verfügen über eine Sprache und ein Land / Region.

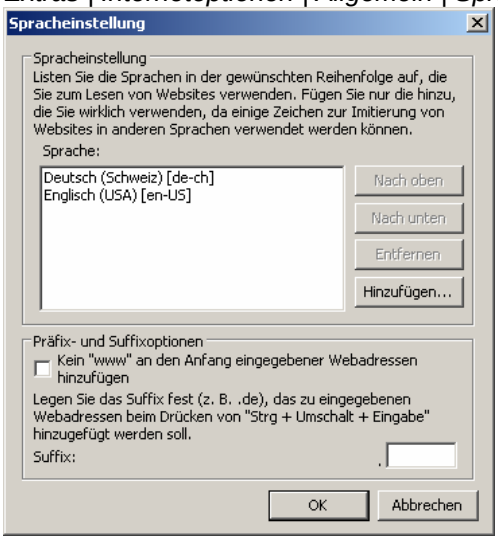
10.2.2 Die Kultur einer Page festlegen

Die Klasse `System.Web.UI.Page` hat die geschützte Methode `InitializeCulture()`, sie wird im Lebenszyklus der Seite sehr früh aufgerufen und kann zur Initialisierung von `Thread.CurrentThread.CurrentCulture` resp. `CurrentUICulture` verwendet werden.

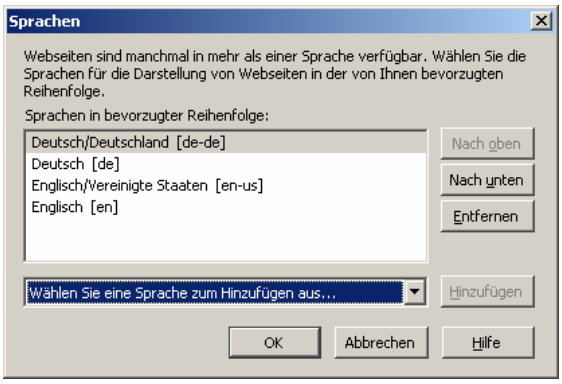
10.2.3 Spracheinstellungen des Browsers verwenden

Internet Explorer, Firefox und andere Browser ermöglichen dem User, eine resp. mehrere bevorzugte Sprachen einzustellen. Diese werden bei Requests als Header dem Server übermittelt.

Internet Explorer
Extras | *Internetoptionen* | *Allgemein* | *Sprachen*



Firefox
Extras | *Einstellungen* | *Erweitert* | *Sprachen*



Auf diese Spracheinstellungen kann man mit `Request.UserLanguages` zugreifen, es handelt sich um einen String-Array.

In der Seitendirektive kann man mit den Attributen `Culture` für Formatierungen und `UICulture` für Ressourcen die Sprache steuern.

- Spracheinstellung des Browsers verwenden:
  `<%@ Page … Culture="auto" UICulture="auto" %>`
- Spracheinstellung des Browsers verwenden, falls nicht unterstützt Deutsch verwenden:
  `<%@ Page … Culture="auto:de-CH" UICulture="auto:de-CH" %>`
- Sprache fest auf US-Englisch setzen:
  `<%@ Page … Culture="en-US" UICulture="en-US" %>`

Diese Einstellungen bewirken, dass `Thread.CurrentThread.CurrentCulture` (für Formatierungen) und `Thread.CurrentThread.CurrentCulture` (für Ressourcen) entsprechend gesetzt werden.
Diese Einstellungen sind auch im web.config möglich:

```
<system.web>
      <globalization culture="auto" uiCulture="auto" />
</system.web>
```

10.2.4 Eine Seite mit Visual Studio lokalisieren

Mit *Extras* | *Lokale Ressource generieren* bringt man Visual Studio dazu, für die aktuelle Seite in `\App_LocalResources` eine invariante Ressourcendatei (lokale Ressource) zu generieren.
Den vorhandenen Steuerelementen auf der Seite wird das Attribut `meta:resourcekey="…"` hinzugefügt. In der erzeugten Ressourcendatei werden für alle Steuerelemente gewisse Name-Value Paare erzeugt. `meta:resourcekey` bewirkt, dass die Eigenschaften der Steuerelemente mit Werten aus der Ressourcendatei initialisiert werden.

	Name	Wert	Kommentar
▶	lblNameResource1.Text	Name:	
	lblNameResource1.ToolTip		
	lblSalaryResource1.Text	Salary:	
	lblSalaryResource1.ToolTip		
	PageResource1.Title	Welcome	
	txtNameResource1.Text		
	txtNameResource1.ToolTip		
	txtSalaryResource1.Text		
	txtSalaryResource1.ToolTip		
*			

Um Eigenschaften von Controls mit Werten aus der Ressourcendatei zu initialisieren müssen die Namen der Ressourcen folgender Konvention entsprechen. Wert des `meta:resourcekey` Attributs gefolgt von einem Punkt, gefolgt vom Namen der Eigenschaft.
Um Ressourcendateien für die verschiedenen Sprachen/Regionen zu erstellen muss man die Ressourcendatei kopieren und ihrem Namen vor der Dateiendung den entsprechenden Kulturcode hinzufügen.
Beispiel:
Erstellt man für `Localized.aspx` mit *Extras* | *Lokale Ressource generieren* eine Ressourcendatei, heisst diese `Localized.aspx.resx`. Um nun eine Ressourcendatei für Deutsch/Schweiz bereit zu stellen muss diese `Localized.aspx.de-CH.resx` heissen.

11 Selbstdefinierte Steuerelemente

Selbstdefinierte Steuerelemente können in Webseiten, aber auch innerhalb von anderen Steuerelementen eingebunden werden. Sie haben eine gewisse Ähnlichkeit zu Tag Libraries von Java 2 Enterprise Edition (J2EE).

ASP.NET kenn zwei Arten von selbstdefinierten Steuerelementen. User Controls und Custom Controls.

User Controls (in der deutschen Dokumentation "Webbenutzersteuerelemente" genannt) bestehen aus einer HTML-/XML-Quellcodedatei mit optionaler Hintergrundcode-Datei. Sie haben die Dateiendung `.ascx` und liegen innerhalb einer Webanwendung. Sie sind entweder von `System.Web.UI.UserControl` oder von `System.Web.UI.MobileControls.MobileUserControl` abgeleitet.

Custom Controls werden rein codebasiert entwickelt (keine HTML-/XML-Codedatei). Sie liegen in eigenen Assemblies im /bin Verzeichnis oder im GAC. Der auszugebende Code wird über ein `HtmlTextWriter`-Objekt geliefert.
Von Custom Controls gibt es drei Unterarten:

- **Inheritance Control**: von einem bestehenden Steuerelement abgeleitet
- **Composite Control**: von `System.Web.UI.Control` abgeleitet
- **Direct Control**: von `System.Web.UI.WebControls.WebControl` abgeleitet

11.1 Einbindung selbstdefinierter Steuerelemente

Registrierung eines User Controls	Registrierung eines Custom Controls
`<%@ Register TagPrefix="DH" TagName="MyUserControl" Src="../_Controls/MyUserControl.ascx" %>`[2]	`<%@ Register TagPrefix="DH" Namespace="ch.dominikhug.CustomControls" %>`

Nutzung eines User Controls	Nutzung eines Custom Controls
`<DH:MyUserControl ID="MyUserControl1" Text="Hello" Runat="Server" />`	`<DH:MyCustomControl ID="MyCustomControl1" Runat="Server" Text="Hello"> </DH:MyCustomCtontrol>`

Neu in ASP.NET 2.0 ist die Möglichkeit, selbstdefinierte Steuerelemente global im Web.config zu registrieren. Damit entfällt die Registrierung in jeder Seite welche das Control benutzt.

User Control in web.config registrieren	Custom Control in web.config registrieren
`<pages> <controls> <add src="~/Controls/MyUserControl.aspx" tagName="MyUserControl" tagPrefix="DH" /> </controls> </pages>`	`<pages> <controls> <add tagPrefix="DH" Namespace="ch.dominikhug.CustomControls" </controls> </pages>`

11.2 User Controls

Unterschiede zu einem Webform

- Dateiendung `ascx` statt `aspx`
- `@Control` Direktive statt @Page Direktive
  Beispiel:
  ```
  <%@ Control Language="C#" Codefile="MyUserControl.ascx.cs"
  Inherits="MyUserControl.ascx" %>
  ```
- Die Codebehind-Klasse erbt von System.Web.UI.UserControl statt System.Web.UI.Page
- User Controls können nicht direkt im Browser angezeigt werden, man muss sie in eine Seite einbinden

User Controls bieten die Möglichkeit, öffentliche Methoden, Ereignisse und Properties (welche im ASPX-Code gesetzt werden können, falls es keine komplexen Typen sind) zu deklarieren. Alle diese Member können im Code des Containers verwendet werden wie die Member jedes anderen Objekts auch.

[2] Der Pfad kann auch mit "~" relativ zur Wurzel der Webanwendung angegeben werden

Im Gegensatz zu den Custom Controls können User Controls nicht in die Toolbox aufgenommen werden. Es ist jedoch möglich, ein User Control aus dem Projektmappen-Explorer auf ein Form zu ziehen, die `@Register`-Direktive wird von Visual Studio 2005 automatisch erzeugt.

Alternativ dazu kann man User Controls mit `Page.LoadControl()` zur Laufzeit dynamisch laden. Am besten verwendet man für die Positionierung innerhalb der Seite ein `PlaceHolder` Steuerelement und fügt das geladene Control der Controls-Collection der PlaceHolder-Instanz hinzu.

Die Steuerelemente die im User Control enthalten sind, sind für die Seite nicht sichtbar. Um sie für die Page zugreifbar zu machen, muss man das User Control mit entsprechenden öffentlichen Properties ausstatten.

11.2.1 Ereignisabfolge bei der Verwendung von User Controls

1. Die Seite wird vom Browser angefordert.
2. Das Control wird instanziiert. Standardwerte von Variablen werden gesetzt und der Konstruktor-Code ausgeführt.
3. Falls im Tag (im ASPX-Code) Eigenschaften gesetzt wurden, werden diese jetzt angewendet.
4. Das Page.Load-Event der Seite wird gefeuert.
5. Das Page.Load-Event des User Controls wird gefeuert.

Man sollte also keine Eigenschaften im Page.Load-Event initialisieren, da dies die als Attribute im Tag gesetzten Werte überschreiben würde.

11.2.2 Eine Page in ein User Control konvertieren

Im Grossen und Ganzen eine Paste & Copy Operation. Man sollte folgende Punkte beachten:

- Alle `<html>`, `<body>` und `<form>`-Tags aus der Seite entfernen
- Die `@Page`-Direktive durch die `@Control`-Direktive ersetzen und alle Attribute entfernen, die von der `@Control-Direktive` nicht unterstützt werden
- Die Dateiendung von `.aspx` zu `.ascx` ändern

12 AJAX

12.1 Allgemeines zu AJAX

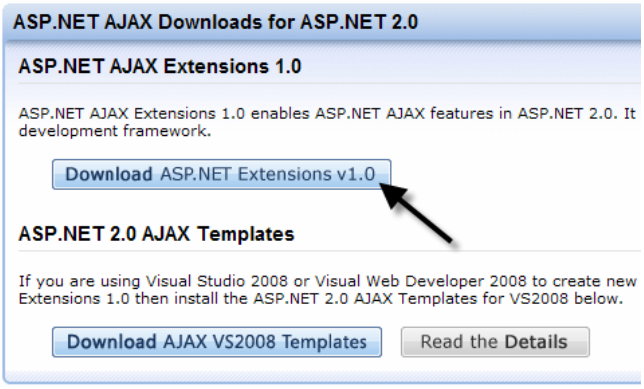
AJAX ist die Abkürzung für **A**synchronous **J**avaScript **A**nd **X**ML. Es ist eine Technologie, welche Datentransfer zwischen Browser und Server und somit ein Update der Webseite ermöglicht, ohne dass die ganze Seite neu geladen wird.
Eine Ajax-Anwendung basiert auf folgenden Web-Technologien:

- HTML oder XHTML
- Document Object Model (DOM) zur Repräsentation der Inhalte
- JavaScript zur Manipulation von DOM und als Schnittstelle zwischen den einzelnen Komponenten
- XMLHttpRequest-Objekt welches von vielen Browsern unterstützt wird und asynchronen Datenaustausch mit dem Webserver (häufig via WebServices) ermöglicht.

Es werden nicht immer Daten im XML-Format ausgetauscht. **JSON** (**J**ava**S**cript **O**bject **N**otation) ist ein weiteres Datenformat um Daten zwischen Server und Client hin- und her zu schicken.

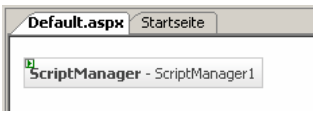
12.2 Vorbereitungen für die Benutzung von AJAX mit .NET

Um AJAX zu benutzen muss man zuerst `ASPAJAXExtSetup.msi` von http://ajax.asp.net herunterladen und installieren.



Nach der Installation können wir ein neues AJAX-Projekt mit dem Template "ASP.NET AJAX-Enabled Web Site" eröffnen.

Jede ASPX-Seite welche AJAX-Funktionalität unterstützen soll, muss das `ScriptManager` Serversteuerelement enthalten.



12.3 XMLHttpRequest

Die Grundlage von AJAX ist das `XMLHttpRequest` Objekt, mit welchem man einen HTTP Request absetzen und die Response empfangen kann, ohne einen vollen Page-Postback durchführen zu müssen.
Die Erzeugung eines XMLHttpRequest Objekts lässt sich nicht in jedem Browser auf die gleiche Art erzeugen. Die folgende Funktion sollte aber in beinahe jedem Browser funktionieren:

Browserunabhängig ein XMLHttpRequest Objekt instanziieren

```
function getXMLHttp() {
   var XMLHttp = null;
   if (window.ActiveXObject) {
      try {
         XMLHttp = new ActiveXObject("Msxml2.XMLHTTP");
      } catch (e) {
         try {
            XMLHttp = new ActiveXObject("Microsoft.XMLHTTP");
         } catch (e) { }
      }
   } else if (window.XMLHttpRequest) {
      try {
         XMLHttp = new XMLHttpRequest() {
      } catch (e) { }
   }
   return XMLHttp;
}
```

Ein XMLHttpRequest-Objekt verfügt über Porperties und Methoden für das Senden von HTTP-Requests und das Empfangen der Responses.

Um einen HTTP-Request zu erzeugen und die Response auszuwerten geht man in folgenden vier Schritten vor:

1. Ein XMLHttpRequest-Objekt erzeugen
2. Dessen Methode `open()` aufrufen und den Request vorzubereiten.
   Die `open()`-Methode übernimmt fünf Parameter, wobei meist die ersten beiden reichen. Der Typ des Requests (GET oder POST) und die URL (absolut oder relativ). Der dritte Parameter gibt an ob der Request asynchron durchgeführt werden soll, der Standardwert ist true. Falls

der HTTP-Request eine Authentifizierung benötigt, kann man als vierten und fünften Parameter Username und Passwort übergeben.
3. Dem Property `onreadystatechanged` eine Referenz auf eine Callback-Methode zuweisen. Diese Methode wird aufgerufen, wenn der Server den Request beantwortet hat.
4. Den HTTP-Request mit der `send()`-Methode absenden. Benutzt man asynchrone Kommunikation, wird das Script weiter ausgeführt und der User kann weiter mit der Seite interagieren.

Das `readyState` Property des XMLHttpRequest Objekts informiert über den aktuellen Zustand und kann 5 Werte annehmen:

0 Objekt ist nicht initialisiert
1 Request wird geladen
2 Request ist vollständig geladen
3 Request wartet auf User-Interaktion
4 Request ist komplett

Jedesmal wenn sich der Wert von readystate ändert, wird die dem `onreadystatechange` Property zugewiesene Methode aufgerufen. Deswegen prüft man in dieser Methode typischerweise ob readystate den Wert 4 hat, was bedeutet, dass die Antwort vom Server angekommen ist.

12.3.1 Anonyme JavaScript Funktion für onreadystatechange

Anonyme Funktionen sind Funktionen ohne Namen, welche als Teil eines Ausdrucks deklariert werden.

Beispiel:

```
var XMLHttp = getXMLHttp();
if (XMLHttp != null) {
   XMLHttp.open("GET", "ajax.aspx?sendData=ok");
   XMLHttp.onreadystatechange = function() {
      if (XMLHttp.readyState == 4 && XMLHttp.status = 200) {
         window.alert(XMLHttp.responseText);
      }
   }
}
```

12.3.2 Wichtige Member von XMLHttpRequest

`responseText`
Gibt die Antwort des Servers als String zurück.

`responseXML`
Gibt die Antwort des Servers als XMLDocument zurück. Um dieses Property zu benutzen, muss der Server well-formed XML zurückgeben.

12.4 XMLDocument Objekt

Falls der Server XML als Antwort zurücksendet, kann man mit dem responseXML Property des XMLHttpRequest Objekts auf eine Instanz von XMLDocument zugreifen. Es ermöglicht leichten Zugriff auf die XML Daten da es vollen DOM Support bietet.

Empfängt man XML Daten, ist es wichtig, dass der Content-type Header vom Server auf `"text/xml"` gesetzt wird, da gewisse Browser sonst `XMLHttpRequest.responseXML` auf null setzten. Dies lässt sich in Page_Load() durch `Response.ContentType = "text/xml";` erreichen.

12.5 Den Inhalt der Seite aktualisieren

Nachdem man mit XMLHttpRequest Daten empfangen hat, will man diese auf der Seite anzeigen, resp. den Inhalt der Webseite aktualisieren. `document.write()` ist nicht die Methode der Wahl, da dies bei einer bereits geladenen Seite bei den meisten Browsern die aktuelle Seite löscht.

Um den Inhalt der Seite zu aktualisieren gibt es zwei Methoden:
1. Den Inhalt bestehender HTML-Elemente aktualisieren
2. Neue HTML-Elemente erzeugen

13 Weiteres

13.1 Emails versenden

Beispiel

```
System.Net.Mail.MailMessage mm = new System.Net.Mail.MailMessage("FASTQ@FASTQ.com",
"dominik.hug@a1-it.ch");
mm.Subject = "FASTQ: Unhandled Exception";
mm.Body = String.Format("Unhandled exception occured:{0}Message: {1}{0}{0} Stack
Trace:{0}{2}",
System.Environment.NewLine, ex.Message, ex.StackTrace);
mm.IsBodyHtml = false;

System.Net.Mail.SmtpClient smtp = new System.Net.Mail.SmtpClient();
smtp.Host = "193.135.144.195";
// falls SMTP-Server Authentifizierung verlangt
smtp.Credentials = new System.Net.NetworkCredential("<UserName>", "<Password>");
smtp.Send(mm);
```

13.2 HTML-Tags als Formulardaten empfangen

Platziert man `ValidateRequest="false"` in der @Page-Direktive, kann man eine Exception vermeiden:
(Ein möglicherweise gefährlicher Request.Form-Wert wurde vom Client (ctl00$ContentPlaceHolder1$txtProcess="<a>Hello") entdeckt.)

14 Index

R

S

T

U

V

W

X