Chapter 14/2
Web Development and ASP.NET
Some WebForm Controls

Inhaltsverzeichnis

WebForm Controls

1. Understanding the Benefits of WebForm Controls

You have the ability to assemble the user interface of your Web pages using the GUI types defined in the `System.Web.UI.WebControls` namespace. These controls are extremely helpful in that they automatically ***generate the necessary HTML tags*** required by the browser. All you need is utilising the design time template and intrinsic **WebForm controls**. Here is an example:

```
<form id="Form1" method="post" runat="server">
  <asp:Button ID="btnGetDate" runat="server" Text="Get Date"
              OnClick="btnReset_Click" /><br />
</form>
```

When the ASP.NET runtime encounters widgets with this attribute, the correct HTML is inserted into the *response stream* automatically. Creating and editing a *Button*-widget on your UI you have to perform the following steps. Click on Button in your *WebForm-ToolBox*-Folder and place it at the appropriate position in your UI. You can access and change its properties by *right-mouseclick* on the Button and select the *Properties*-Field or by manipulating the *Design code*.

The final core benefit of using WebForm controls – rather than raw HTML controls – is the fact that ASP.NET provides a whole set of controls to validate the user-supplied data. You do not need to generate client-side *JavaScript* routines to validate the data.

2. Working with WebForm Controls

Understanding that each server control can be configured using either the **Property Window** of the VS.NET IDE or the **Source File** in VS.NET – see Document. If you have a textbox control – named the ID as txtEMail – you will find the choices shown in Fig. 1.

Fig. 1: The *Properties* Window

Now you can configure a given WebControl using the **Property Window**, your changes are written directly to the *.aspx file. These controls are defined by an opening element tag of the form `<asp:controlType runat = "server">` and a corresponding closing tag.

Each control is represented by the form:

```
<asp:Button id="txtEMail" style="Z-INDEX: 102; LEFT: 283px;
  POSITION: absolute; TOP: 39px" runat="server" Text="Übernehmen"
  Width="177px" Height="46px" BorderColor="#FF8000" Back
  Color="#FF8000">
</asp:Button>
```

The **`runat="server"`** attribute marks this item as a server-side control and informs the ASP.NET runtime that this item needs to be processed before returning the response stream to the browser, to generate the necessary HTML. Opening the ***CodeFile*** class you will find the names of these control variables:

```
public class WebForm1 : System.Web.UI.Page
```

```
{
      protected System.Web.UI.WebControls.Button txtEMail;
...
```

In this way, you can programmatically manipulate your items using C# code in the *.aspx file or in your custom-defined routines in the Page-derived class. The derivation of the WebForm Controls:

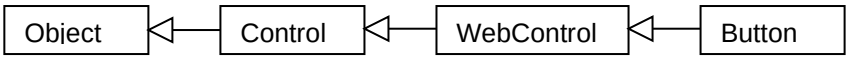


Figure 3: GUI widget Hierarchy.

Control and **WebControl** are the so called **base classes**. *Control* has some very important *properties*:

`ID:` Gets or sets the identifier for the control.
`Visible:` Indicates whether a control should be rendered on the page.
`BackColor:` Gets or sets the background colour of the Web Control.
`Font:` Gets font information for the Web Control.
`Height, Width:` Gets or sets the height and width of the Web Control.
`TabIndex:` Gets or sets the tab index of the Web Control.
`ToolTip:` Tool tip for the Web Control to be displayed when the cursor is over the control.

These following types are basically .NET components that have a direct HTML widget counterpart. There are many of them, i.e.

WebForm Intrinsic Control	Meaning
`Button`	Various button types.
`CheckBox`	A basic check box or a list box containing a set of check boxes.
`DropDownList`	These types allow you to construct standard list box items.
`Image`	These types represent containers for static text and images.
`RadioButton`	A basic radio button type or a list box containing a set of radio buttons.
`TextBox`	Text box for user input. May be configured as a single-line or multi-line text.

2.1. Creating a ListBox (LBX), DropDownList (DDL) and a TextBox

Use the `DropDownList/ListBox` control to create a single-selection *drop-down list/list* control. To specify the items that you want to appear in the control, place a `ListItem` object for each entry between the opening and closing tags of the list control. The `ListItemCollection` is a collection type that is list endowed with all main functions provided by the interface `IList`.

We want to solve the following task:

- Fill a `ListBox` (or equivalently a `DropDownList`) with student names (Fig. 5).
- Create a *ListBox-Event* `onSelectedIndex(…)` that displays the *Student Name* and *ID* in a `TextBox` as shown in Fig. 9. The data are provided in a data base table Fig 4.

StudentID	KatNo	FirstName	LastName	TeacherID
1	3	Tomy	Hofer	1
2	7	Anna	Luber	1
3	6	Leo	Keiner	2
4	5	Fritz	Karner	1
5	1	Hans	Ascher	1
6	2	Burgi	Ebner	2
7	4	Karl	Hart	1

Figure 4: Data base table containing Student's data.

We want to consider 3 types of populating a **ListBox** with data:

1. Add the data source (table) in the *Design View*:

- Pull a *ListBox*- as well as a *TextBox*-Component into your *Default.aspx*. Click the *ListBox* Tasks (Fig. 5) *Enable AutoPostBack* and click *Choose Data Source*. Follow the wizard in the steps:

  Choose Data Source <new data source> → *Choose a Data Source Type* <database> → *Choose Your Data Connection* <new connection (to your database: server name + database name)> → *Save the Connection String* → *Configure the Select Statement* → *Test Query* → *Select data field to display* (LastName) → *Select data field for calculation*. (StudentID) → Run the application (Fig. 6).

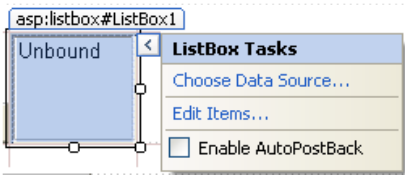


Figure 5: ListBox Task

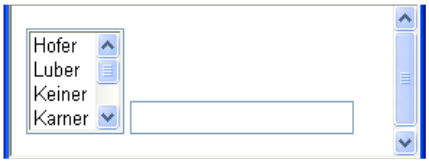


Figure 6: Result

- The *Poperties*-window (Fig. 7, Fig. 8) allows you to design your component just as you like. Two properties are very important: The *ID*-property, which is the object identity of the component and the ***AutoPostBack*-property, which provides the control with automatic server response if an event is fired.**

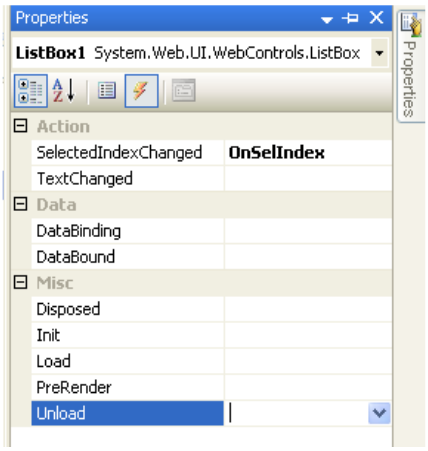


Figure 7: ListBox Task

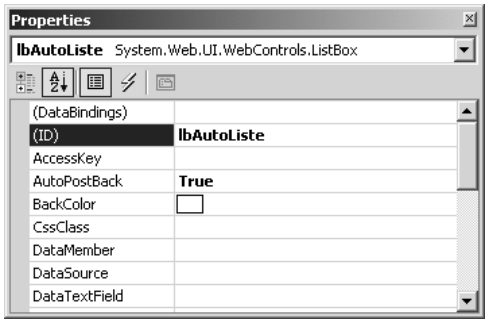


Figure 8: Result

- Implement a "Selected Index Changed" event that displays the text and the value of the selected list item in the text box:

```
protected void OnSelectedIndex(object sender, EventArgs e)
{
    // Get the text and the value of the list box
    String selName = ((ListBox)sender).SelectedItem.Text;
    String selValue = ((ListBox)sender).SelectedItem.Value;
    // Output to text box
    tbxStudent.Text = String.Format("{0} - {1}", selName, selValue);
}
```

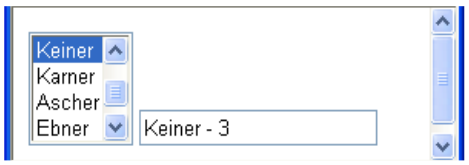


Figure 9: Result of the "Selected Index Chanced"

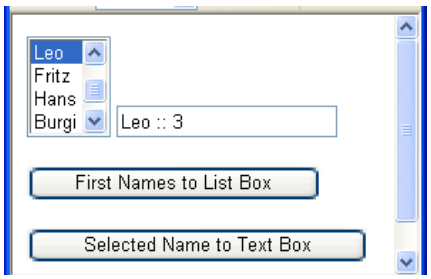


Fig. 10: Result of Programmatic Implem.

2. Programmatic Implementation applying **`DataBind()`**:

In this case a complete Collection is automatically bound to the Control (LBX or DDL)Here any command has to be implemented by hand. Two buttons were used in this case:

- *First Names to List Box*: Preceding this event we must provide the data table "Student" in a Session object. Further we have to specify, which Column of the data table should be displayed `DataTextField`) and which column shoult be stored as an additional information (`DataValueField`). Here we decide "FirstName" as the column to be displayed in the control and "StudentID" is to be stored as additional information. Then bind the table to the textbox:

```
lbxStudent.DataSourceID = ""
lbxStudent.DataSource = (DataTable)Session["Student"];
lbxStudent.DataTextField = "FirstName";
lbxStudent.DataValueField = "StudentID";
lbxStudent.DataBind();
```

- *Selected Name To Text Box*: Get the Value and Text of the selected item and display it in the text box.

```
try
{
    String value = lbxStudent.SelectedItem.Value;
    String text = lbxStudent.SelectedItem.Text;
    tbxStudent.Text = String.Format("{0} :: {1}", text, value);
}
catch (Exception ex) { }
```

As you can see, the `ListBox` control provides you with the `DataSource` attribute and the `DataBind()` function to allow you to render the contents of a given `ListBox`. However, *WebForm* controls allow you to bind other sources of data to a given widget. With this technique of binding you can attach an array to a GUI type.

Data binding is essential for displaying data in server controls. Such as listboxes, datagrids, etc. can be bound to any object that exposes an `IListSource`, etc. interface, which include `DataTable` objects.

3. Programmatic Implementation by **`elementwise adding items`**:

In this case you can add any *key/value*-pairs (*text/value*-pairs) to your *DDL/LBX*. We have to explicitely create `ListItem`s that store text and value as pairs:

```
ListItem lit = new ListItem(text: string, value: string);
```

This object hast to be added to the control *LBX/DDL*. As in the example above the text/value-pair is to be implemented as:

```
string lN = r["LastName"].ToString();
string id = r["StudentID"].ToString();
ListItem lit = new ListItem(lN, id);
lbxID.Items.Add(lit);
```

2.2. Creating a GridView

A **Gridview** is a control in **asp**.net, displays the values of a data source (sql server database) in a tabular form where each column represents a field and each row represents a record. The **GridView** control also, enables you to select, sort, and edit these items. Multiple data fields for the hyperlink columns.

Nachname	Vorname
Huber	Hans
Berger	Anna
Gigerl	Bernd
Weber	Sepp

Fig. 11: Calendar widget

1. Adding a GridView to the aspx-file:

```
<asp:GridView ID="grvID" runat="server" AutoGenerateColumns="False"
        CellPadding="4" ShowHeaderWhenEmpty="True" ForeColor="#333333"
        GridLines="None" Width="103px">
    <EmptyDataTemplate>No Record Available</EmptyDataTemplate>
    <RowStyle BackColor="#EFF3FB" />
    <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Lft" />
    <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#33" />
    <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
    <EditRowStyle BackColor="#2461BF" />
    <AlternatingRowStyle BackColor="White" />
</asp:GridView>
```

2. Add the DataSource to the GridView: To bind a DataSource to the GridView when you want to display all rows/columns from the DataTable:

```
grvID.DataSource = (DataTable)Session["Stud"];
grvID.DataBind();
```

3. Display just a selected number of colums from a DataSource: Sometimes it is necessary to filter the data to be displayed. In this case we have to specify the columns to be displayed. In this case the columns of the DataTable are named *LastName* and *FirstName*. But we want to rename the Columns for the GridView to *Nachname* and *Vorname*:

```
grvID.DataSource = (DataTable)Session["Stud"];
BoundField bf = new BoundField();
bf.DataField = "LastName";
bf.HeaderText = "Nachname";
grvID.Columns.Add(bf);
bf = new BoundField();
bf.DataField = "FirstName";
bf.HeaderText = "Vorname";
grvID.Columns.Add(bf);
grvID.DataBind();
```

4. Display a GridView with no data: Sometimes there may be no data available, because the DataTable might be empty or so. Then you have to force the GridView to display just the column-names with some text, that explains the situation. Here we have to add the properties

```
ShowHeaderWhenEmpty="True"
<EmptyDataTemplate>No Record Available</EmptyDataTemplate>
```

in the GridView-definition as given above.

2.3. Some Properties

`DataSource`: Gets or sets the object from which the data-bound control retrieves its list of data items.

`DataMember`: Gets or sets the name of the list of data that the data-bound control binds to, in cases where the data source contains more than one distinct list of data items.

`DataSourceID`: Gets or sets the ID of the control from which the data-bound control retrieves its list of data items.

`DataTextField`: Gets or sets the field of the data source that provides the text content of the list items.

`DataValueField`: Gets or sets the field of the data source that provides the value of each list item.

`Font`: Gets the font properties associated with the Web server control.

`Items`: Gets the collection of items in the list control.

`Rows`: Gets or sets the number of rows displayed in the `ListBox` control.
`SelectedIndex`: Gets or sets the lowest ordinal index of the selected items in the list.
`SelectedItem`: Gets the selected item with the lowest index in the list control.
`SelectedValue`: Gets the value of the selected item in the list control, or selects the item in the list control that contains the specified value.
`SelectionMode`: Gets or sets the selection mode **(single** or **multiple** item selection) of the `ListBox` control.
`Text`: Gets or sets the `SelectedValue` property of the `ListControl` control.
`Width`: Gets or sets the width of the Web server control.

Some Events and Methods

`ClearSelection`: Clears out the list selection and sets the `Selected` property of all items to false.
`Focus`: Sets input focus to a control.
`DataBind`: Bind data source to the control.
`ToString`: Returns a String that represents the current Object.
`SelectedIndexChanged`: Occurs when the selection from the list control changes between posts to the server.
`TextChanged`: Occurs when the `Text` and `SelectedValue` properties change.

3. Working with Rich Controls

Rich controls are also widgets that emit HTML to the HTTP response stream. The difference between these types and the set of intrinsic controls is that they have no direct HTML counterpart.

3.1. The Calendar Control

You'll find that a huge amount of HTML code has been generated automatically! To test things, place a `Calendar` type on your design time template. Like the Windows Forms counterpart, the server-side Calendar control is highly customisable. The members of interest are `SelectionMode`, `SelectedDate`. Investigate these controls writing a code on a `SelectionChanged`-Event that yields the following results.

You can specify whether the `Calendar` control allows you to *select* a single *day*, *week*, or entire *month* by setting the `SelectionMode` property.

By default, the control displays the *days of the month*, *day headings* for the days of the week, a *title* with the month name and year, links for selecting individual days of the month, and links for moving to the next and previous month. You can customize the appearance of the Calendar control by setting the properties that control the style for different parts of the control. The following table lists the properties that specify the style for the different parts of the control.

Fig. 12: Calendar widget

Some Properties

`Caption`: Gets or sets a text value that is rendered as a caption for the calendar.
`FirstDayOfWeek`: Gets or sets the day of the week to display in the first day column of the `Calendar` control.
`Height`: Gets or sets the height of the Web server control.
`ID`: Gets or sets the programmatic identifier assigned to the server control.
`SelectedDate`: Gets or sets the selected date.
`SelectedDates`: Gets a collection of System.DateTime objects that represent the selected dates on the `Calendar` control.
`SelectionMode`: Gets or sets the date selection mode on the `Calendar` control that specifies whether the user can select a single day, a week, or an entire month.
`TodaysDate`: Gets or sets the value for today's date.

Some Events

`DataBinding`: Occurs when the server control binds to a data source.
`SelectionChanged`: Occurs when the user selects a day, a week, or an entire month by clicking the date selector controls.

3.2. Working with the AdRotator Control

The purpose of this widget is to randomly display a given advertisement at some position in the browser. When you place a server-side *AdRotator* widget on your design time template, the display is a simple placeholder. Functionally, this control cannot do its magic until you set the *AdvertisementFile* property to point to the XML file that describes each ad.

Figure 13: Placeholder for AdRotator

The format of the advertisement file is quite simple. For each ad you wish to show, create a unique `<Ad>` element. At Minimum, each `<Ad>` element specifies the image to display (`ImageUrl`), the URL to navigate to if the image is selected (`TargetUrl`), mouseover text (`AlternateText`) and the weighing for the ad (`Impression`). For example, assume you have a file ads.xml that defines two possible ads, as shown here:

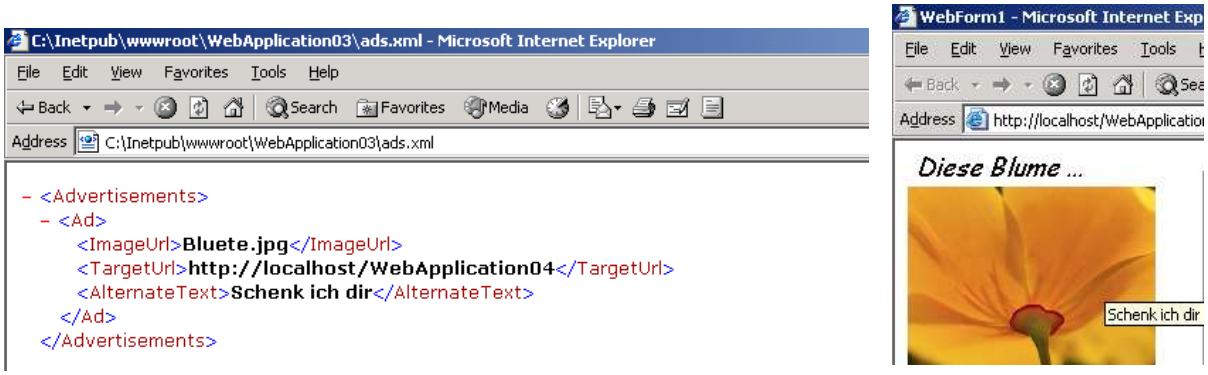


Figure 14: XML-File and result in the Browser

Be aware that the Height and the Width of the `AdRotator` are used to establish the size of your ads. If your ads are larger or smaller than the `AdRotator's` size, you will find skewed images.

3.3. Validation Controls

The final conceptual set of WebForm controls are termed *validation controls*. Like their Windows Forms equivalents, these types are used to ensure that the data submitted by the user is well formatted based on your application logic. Table 1 gives a rundown of the core validation controls.

WebForm Validation Control	Meaning
`CompareValidator`	Validates that the value of an input control is equal to a given value of another input control.
`CustomValidator`	Allows you to build a custom validation function that validates a given control.
`RangeValidator`	Determines that a given value is in a predetermined range.
`RequiredFieldValidator`	Ensures that a given input control contains a value (and is thus not empty).
`ValidationSummary`	Displays a summary of all validation errors of a page in a list, bulleted list, or single paragraph format. The errors can be displayed inline and/or in a popup message box.

Table 1

4. Examples

WFC01: Try to implement the following Calendar widget and get the results as shown in the multi line text box.

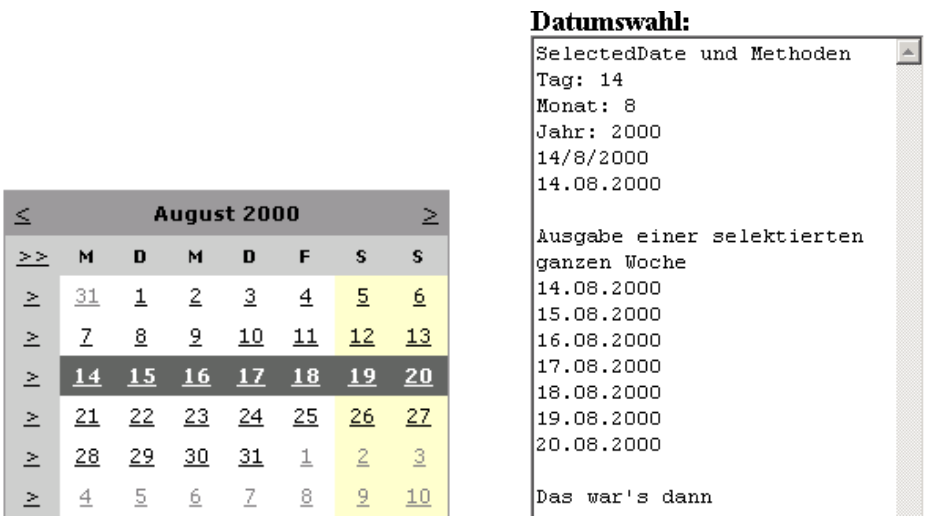


Figure 15: Calendar-Object with results in a multiline TextBox.

WFC03: Try to implement the an *AdRotator* example of your own choice.

WFC04: *Validation Control*

To illustrate the basics with validation controls, let's create a new C# Web Application project workspace named `ValidateWebApp`. Change the name of your *.aspx file to default.aspx and then open the design time template. Now create the simple UI shown in Figure 12 using standard drag-and-drop techniques.

Next let's examine how to use the WebForm validation controls. To illustrate, assume that you wish to ensure that the `tbEmail` text box contains information (before you submit the form to the Web server). At design time, you can simply place a `RequiredFieldValidator` widget on your form.

Using the Properties window, you can set the `ErrorMessge` property to a given value. This widget is in charge of validating using the `ControlToValidate` property (Figure 13).

Figure 16: Simple Web UI

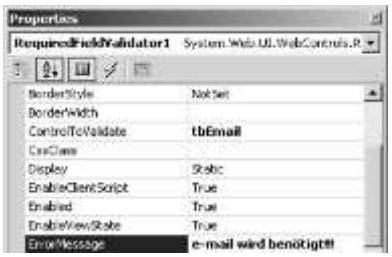


Figure 17: Configuring data validation.

F. 18: Autozeitschrift.aspx

If the button click "versenden" is successful, change to a new page "Autozeitschrift.aspx". To successfully browse to this page implement the line Response.Redirect("Autozeitschrift. aspx"). Design this according to Fig. 14.

Functionality of page "Autozeitschrift.aspx":

- ✓ When opening fill the data grid view as well the list box "Autoliste".
- ✓ Select a car in the listbox.
- ✓ From the selected car retrieve the "Einzelpreis" as well as the "Autonamen" (Niki Laude, Juan Fangio in this example). "Autoname" unhappily means the "Weltmeister" in Formula 1 that actually are driving or drove the selected car.
- ✓ Use 2 DB tables: Auto(Marke, PS, Preis), Weltmeister(First name, last name).
- ✓ Use a web service

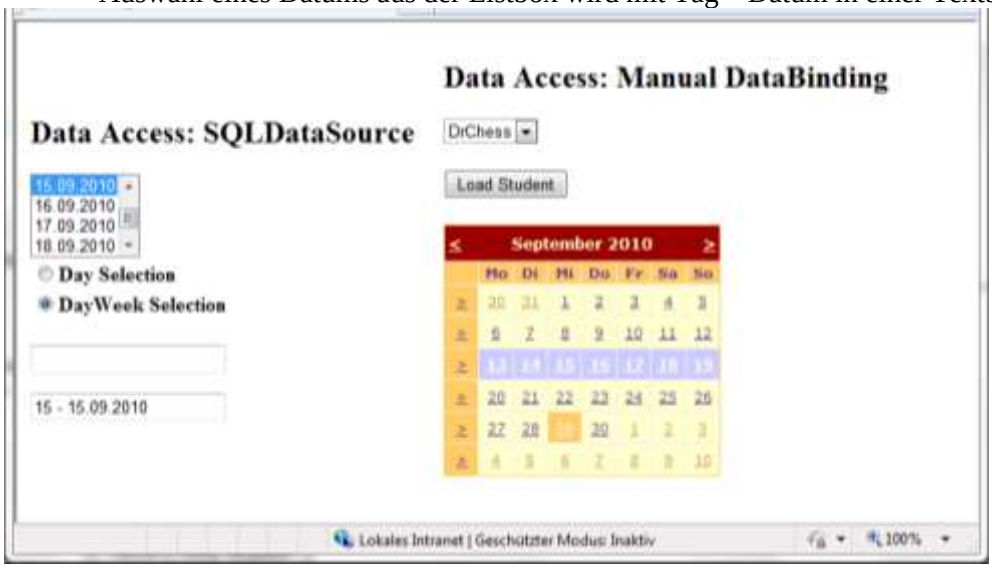
Codebeispiel zur Calendar

```
private void cdArtikel_SelectionChanged(object sender, System.EventArgs e)
{
      tbMultiLine.Rows = 20;
      string sTag = cdArtikel.SelectedDate.Day.ToString();
      string sMonat = cdArtikel.SelectedDate.Month.ToString();
      string sJahr = cdArtikel.SelectedDate.Year.ToString();
      DateTime dT = cdArtikel.SelectedDate;
      SelectedDatesCollection selDts = cdArtikel.SelectedDates;
      tbMultiLine.Text = "SelectedDate und Methoden";
      tbMultiLine.Text += "\nTag: " + sTag;
        tbMultiLine.Text += "\nMonat: " + sMonat;
      tbMultiLine.Text += "\nJahr: " + sJahr;
      tbMultiLine.Text += "\n" + sTag + "/" + sMonat + "/" + sJahr;
      tbMultiLine.Text += "\n" + dT.ToShortDateString();
      tbMultiLine.Text += "\n\nAusgabe einer selektierten ganzen Woche";
      foreach(DateTime d in selDts)
      {
      tbMultiLine.Text += "\n" + d.ToShortDateString();
      }
      tbMultiLine.Text += "\n\nDas war's dann";
```

```
}
```

Beispiel mit Schülern:

1. Erstelle eine DB mit den Tabellen Schüler(ID, LehrerID, FirstName, LastName); Lehrer(ID, FN, LN)
2. Arbeiten mit SqlDataSource
   - ✓ Verbinde eine ListBox mit der SQL-DataSource. Zeige die Lehrerdaten (LN=Text und ID=value) an.
   - ✓ Konfiguriere die SqlDataSource mit der Where-Klausel und binde eine TextBox an
3. Arbeiten mit manuellem Anbinden an eine DAL-DataSource
   - ✓ Erstelle eine DAL und verbinde zur DB.
   - ✓ Erstelle eine Methode, die die Lehrer in eine DDL listet (text = LN, value = ID).
   - ✓ Selektiere einen Lehrer und ermittle alle Schüler und zeige diese in der ListBox an.
4. Arbeiten mit Calendar
   - ✓ Erstelle eine Radiobuttongroup für Tages- und Tages-Wochen-Ansicht. Und lege den Mode fest.
   - ✓ Die selektierten Datumsdaten sind in die ListBox zu stellen.
   - ✓ Auswahl eines Datums aus der Listbox wird mit Tag – Datum in einer Textbox angezeigt.




```
public partial class _Default : System.Web.UI.Page
{
    // Field
    protected BL bl;
    // Post back method
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            // Specify the radio button group
            ListItem item = new ListItem("Day Selection");
            rbgStudent.Items.Add(item);
            rbgStudent.Items.Add("DayWeek Selection");
        }

    }
    protected void onLoadTeacher(object sender, EventArgs e)
    {
        try
        {
            bl = new BL();
```

```
            // Bind a data source
            ddlTeacher.DataSource = bl.getTeacherTable();
            ddlTeacher.DataTextField = "LastName";
            ddlTeacher.DataValueField = "TeacherID";
            ddlTeacher.DataBind();
        }
        catch (Exception exe)
        {
            tbxTeacher.Text = exe.Message;
        }
    }
    protected void onSelectedDateChanged(object sender, EventArgs e)
    {
        SelectedDatesCollection dates = calTermine.SelectedDates;
        // Print in listbox
        lbxTeacher.Items.Clear();
        for (int i = 0; i < dates.Count; i++)
        {
            ListItem lIt = new ListItem(dates[i].ToShortDateString(),
                                        dates[i].Day.ToString());
            lbxTeacher.Items.Add(lIt);
        }
    }
    protected void onCalendarModeChanged(object sender, EventArgs e)
    {
        if (rbgStudent.Items[0].Selected)
            calTermine.SelectionMode = CalendarSelectionMode.Day;
        else
            calTermine.SelectionMode = CalendarSelectionMode.DayWeek;
    }
    protected void onSelectedItemChanged(object sender, EventArgs e)
    {
        String value = lbxTeacher.SelectedItem.Value;
        String text = lbxTeacher.SelectedItem.Text;
        String date = String.Format("{0} - {1}", value, text);
        tbxDate.Text = date;
    }
}
//---------------------------------------------------------------------------
public class BL
{
    // Fields
    private DataSet dSet;
    private SqlConnection con;
    // Constructor
      public BL()
      {
        String conStr = @"Data Source=MIIM03\SQLEXPRESS;Initial " +
                        "Catalog=StudTeach;Integrated
Security=True;Pooling=False";
        con = new SqlConnection(conStr);
        dSet = new DataSet();
        String selStr = "SELECT * FROM [Teacher]";
        SqlDataAdapter dAdapt = new SqlDataAdapter(selStr, con);
        dAdapt.Fill(dSet, "Teacher");
      }
    // Return the data soruce
    public DataTable getTeacherTable()
    {
        return this.dSet.Tables[0];
    }
}
```