

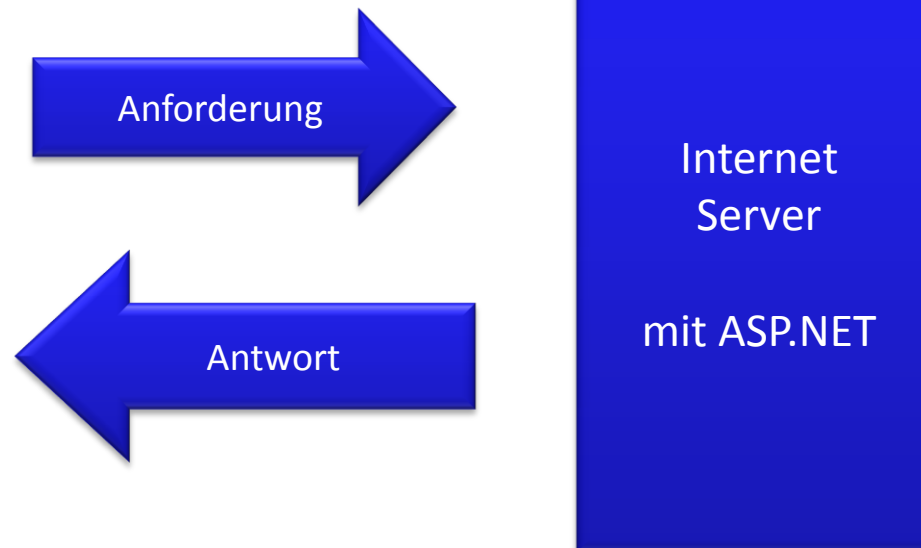
.NET Training

Web-Forms mit ASP.NET

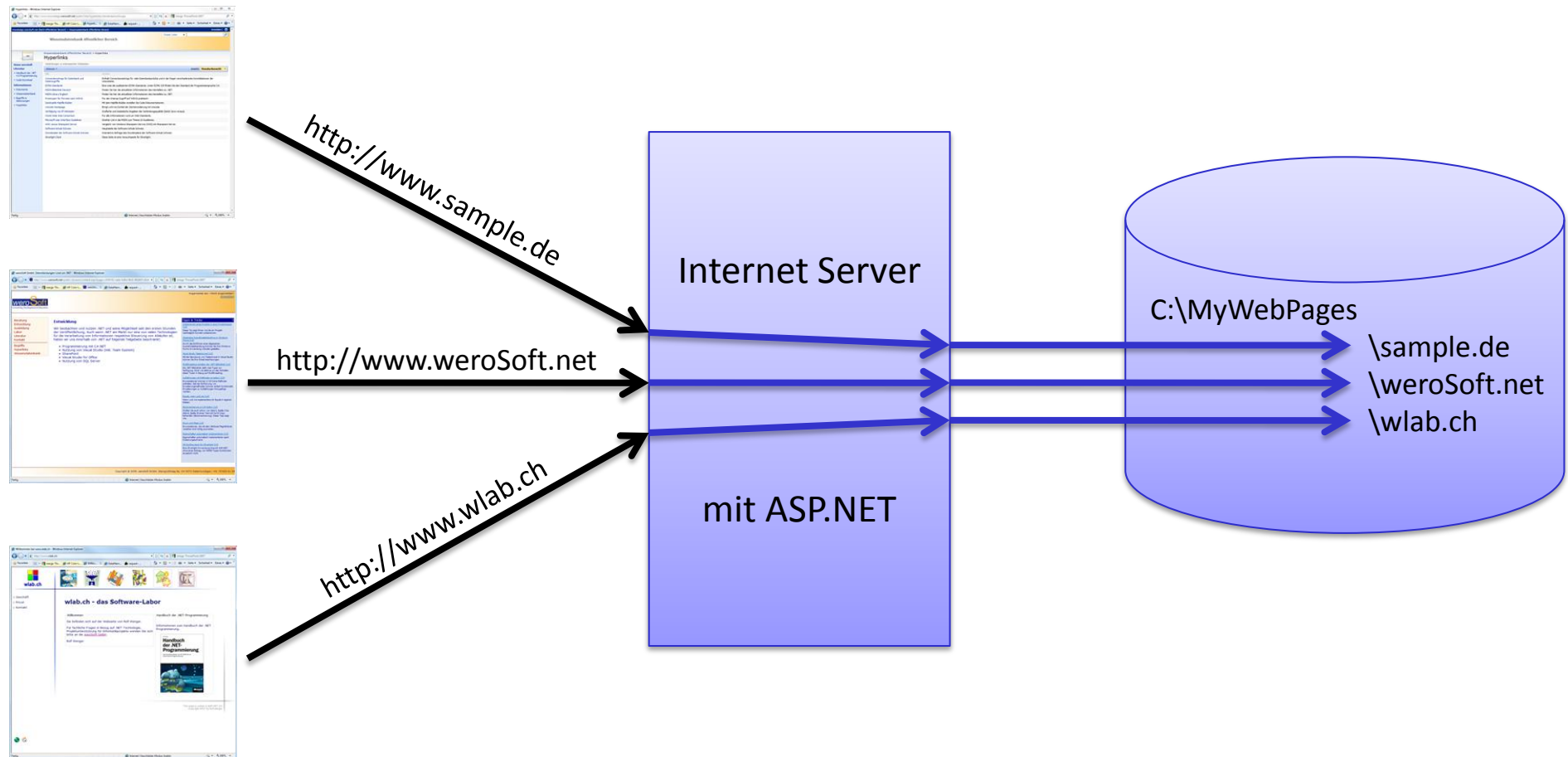
weroSoft GmbH
Rolf Wenger

Version 4.0

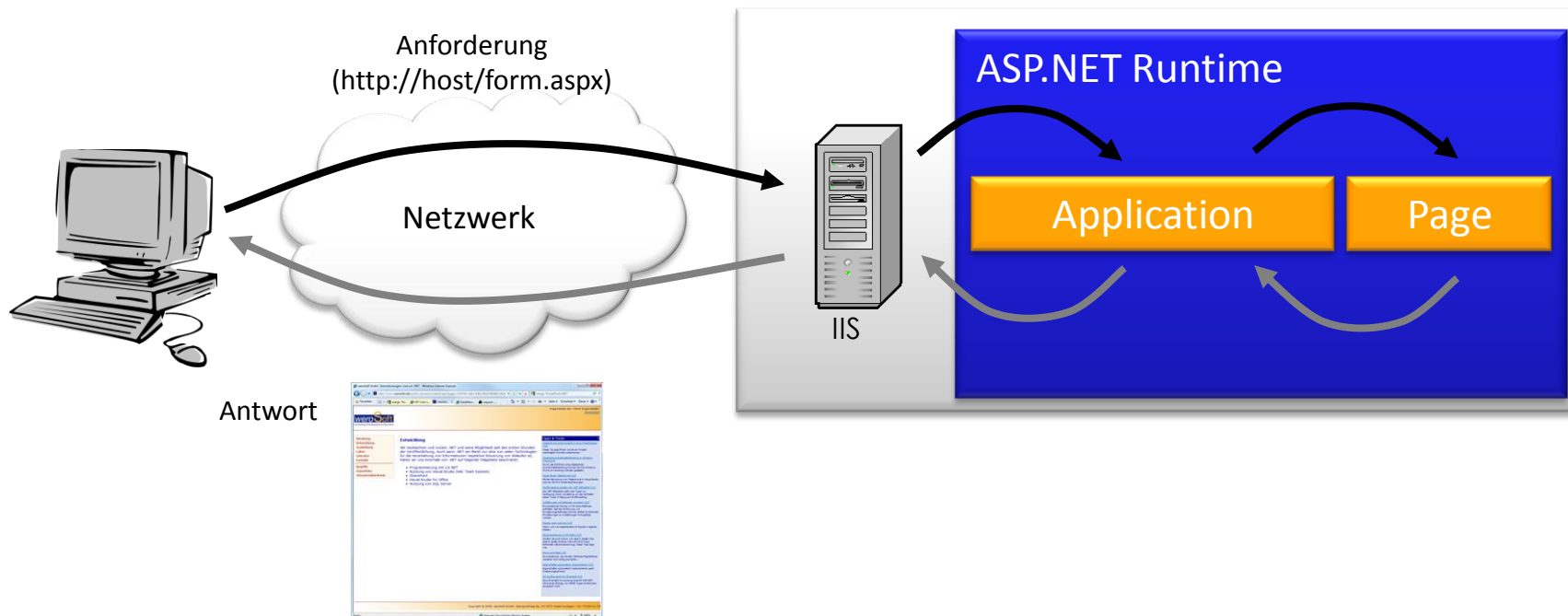
- ASP.NET ist die .NET-Technik für die Herstellung von Web-basierten Anwendungen
- Die Technik teilt sich in zwei Teilbereiche auf:
 - Web-Server mit IIS und ASP.NET verarbeitet Anforderungen des Clients und erstellt eine HTML-Seite
 - Web-Browser, der die Anforderung an den Server stellt und die Resultatseite in Form einer HTML Seite mit etwaig eingelagerten JavaScript verarbeitet



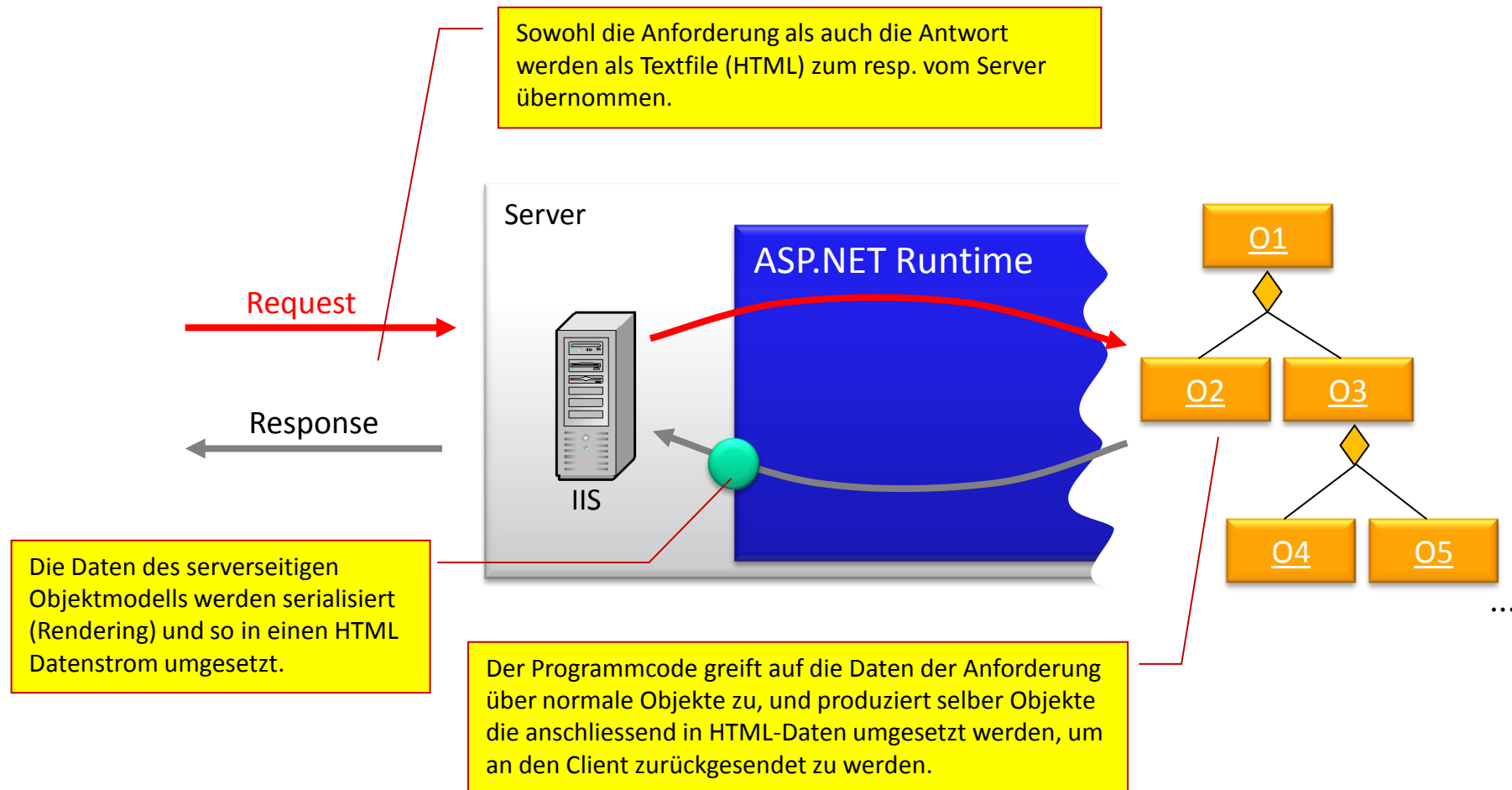
- Die Anforderung wird mit einer URL formuliert und auf dem Server in eine Diskstruktur übersetzt



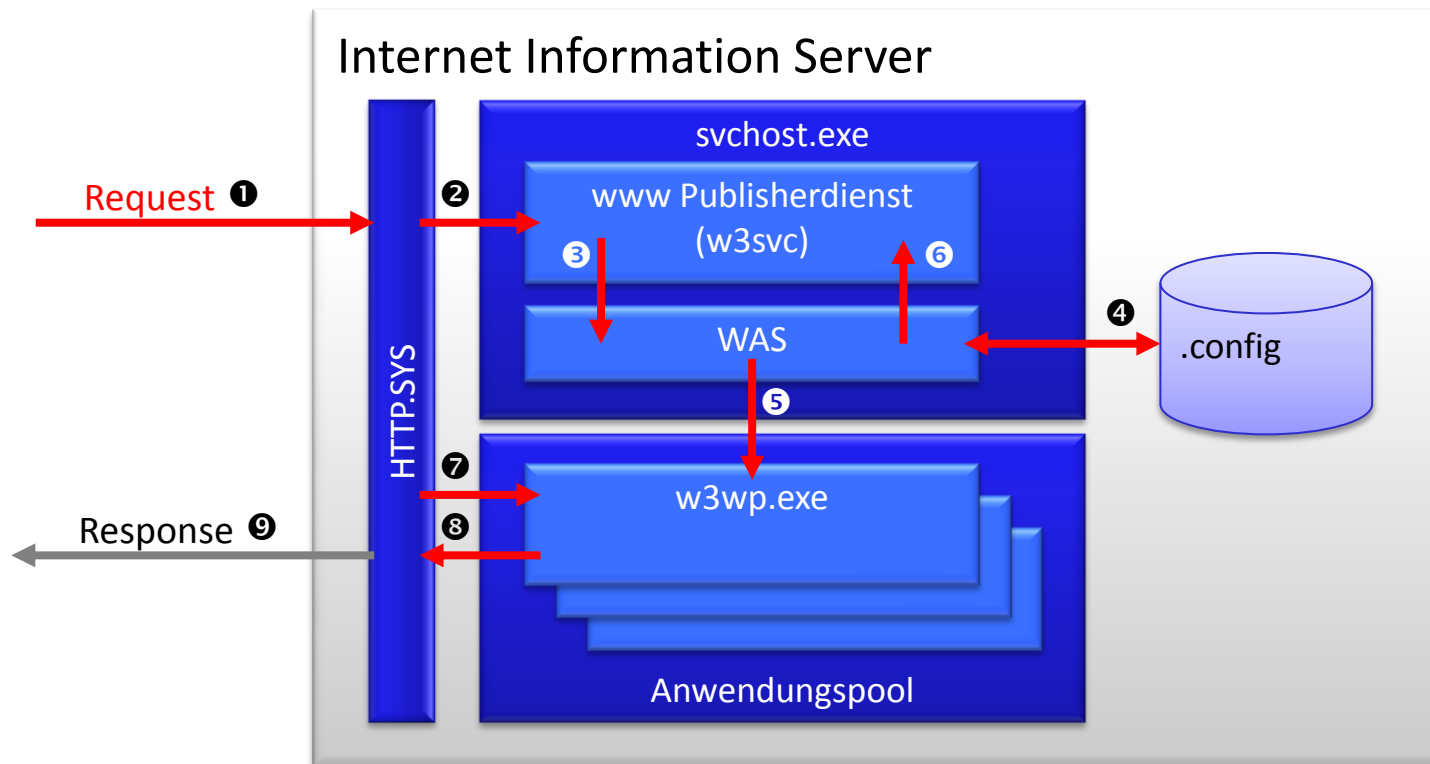
- Mittels Anforderung löst der Client die Lieferung von Informationen beim Server an.
- Die Anforderung wird vom Internetserver aufgenommen und anschliessend auf die ASP.NET Maschine umgesetzt.
- Die ASP.NET Maschine erzeugt das Resultat in Form einer HTML Seite, gibt diese an den Internetserver zurück.
- Der Internetserver ist für die Übermittlung an den Client bemüht.



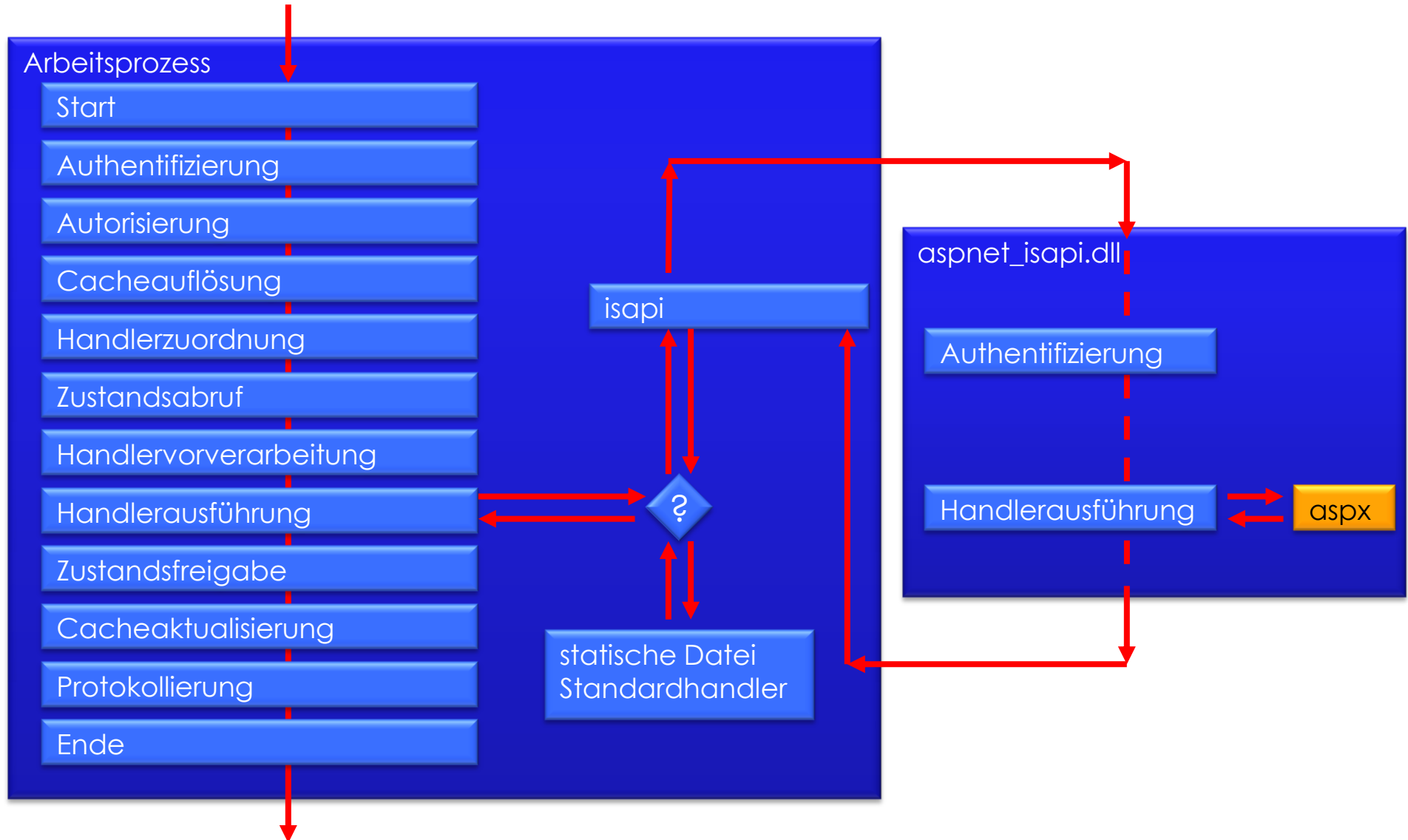
- Nach dem Eintreffen der Anforderung vom Client, setzt der ASP.NET Server die Daten der Anforderung mit Hilfe der angesprochenen Seite in ein Objektmodell um.
- Nach Abschluss der Arbeiten serialisiert der ASP.NET Server die Daten zurück zum Server.



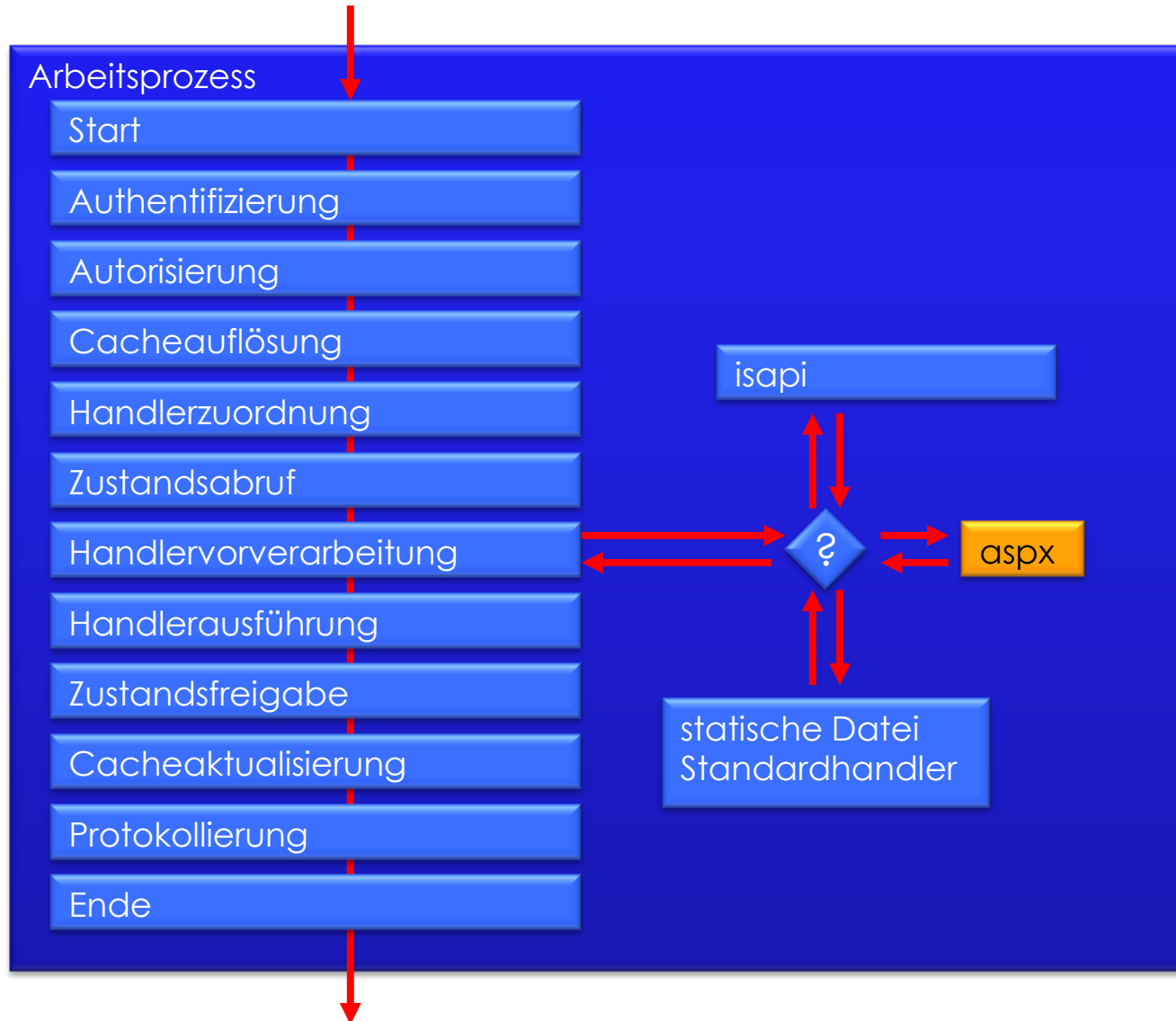
- Der interne Ablauf ist geprägt durch eine vielschichtige Verarbeitung der verschiedenen internen Komponenten des IIS
- HTTP.SYS ist zuständig für die Kommunikation
- Die Dienste w3svc und WAS sind zuständig für die Konfiguration und die Vorbereitung der eigentlichen Verarbeitung
- Im Anwendungspool geschieht die Verarbeitung



- Die Detailverarbeitung geschieht nach dem stets selben Muster



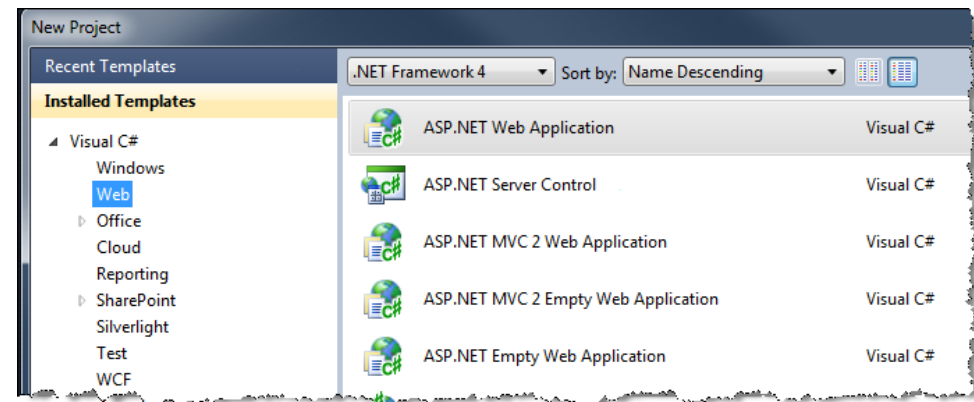
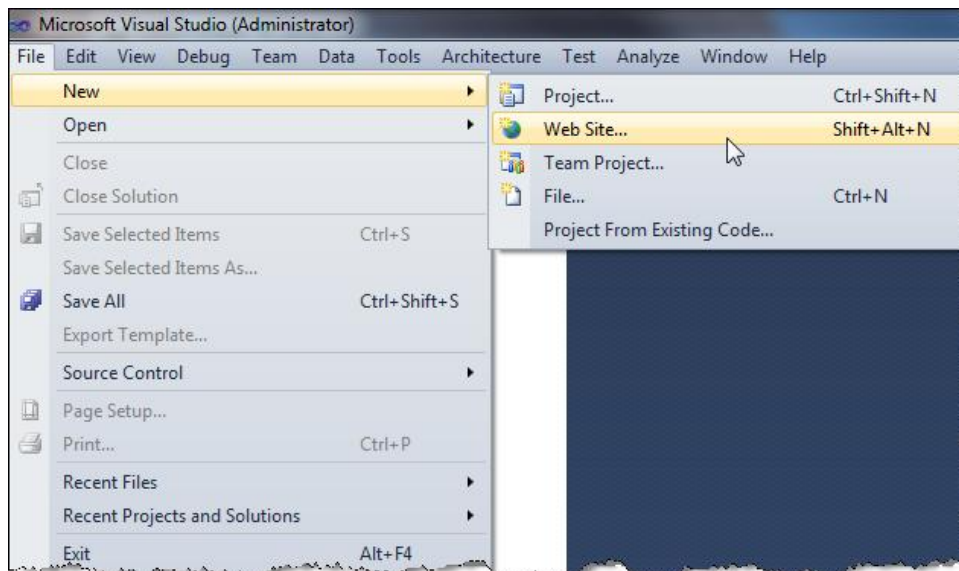
- Die Detailverarbeitung geschieht nach dem stets selben Muster





Eine einfache Seite aufbauen

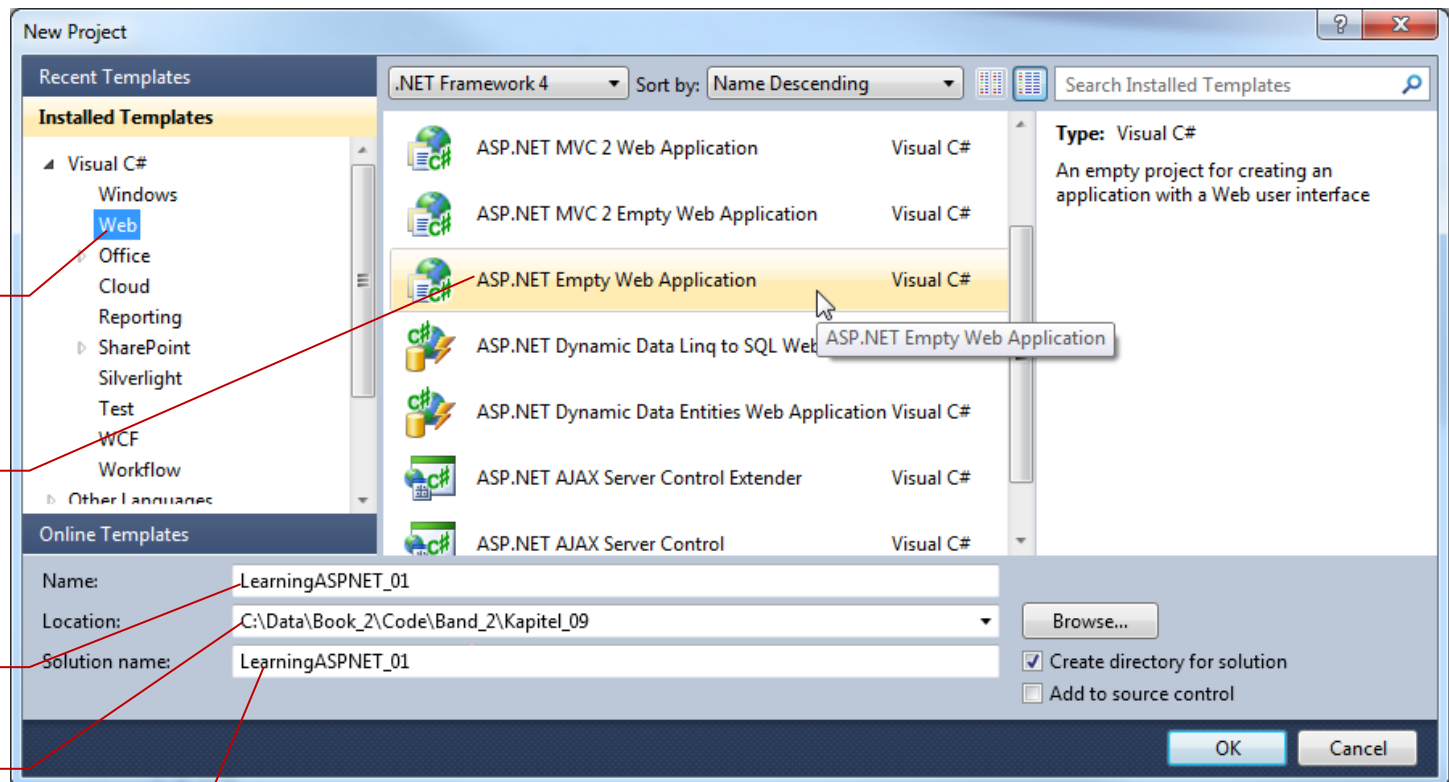
- Visual Studio wartet mit einer ganzen Palette von Web-Projektvorlagen auf
- Dabei werden grundsätzlich zwischen zwei verschiedenen Arten unterschieden
 - Web Site Projekte
 - Web Anwendungen
- Web Site Projekte werden über das Hauptmenü direkt erstellt (Abbildung links)
- Web Anwendungsprojekte werden über den normalen Projektassistenten ausgewählt (Abbildung rechts)



- Wählen Sie eine Web-Anwendungsprojekte
 - Normale Entwicklung und Verwendung von CVS
 - Seiten die ein Staging durchlaufen
 - Seiten, die Projektabhängigkeiten aufweisen
- Wählen Sie Web Site Projekte
 - Sie wollen einzelne Seiten auf dem Server austauschen

Bereich	Web Anwendung	Web Site
Projektstruktur	Normale Visual Studio Projektstruktur mit entsprechenden .sln und .csproj Dateien	Keine Visual Studio Steuerdateien. Alle enthaltenen Dateien gehören automatisch zum Projekt.
Kompilierung	Explizite Kompilierung. Codedateien erzeugen eine Assembly.	Dynamische Kompilierung auf dem Server, Manuelle Vorkompilation möglich. Standardmässig werden mehrere Assemblys erzeugt.
Namensraum	Normale Namensräume	Keine Namensräume. Sie können aber manuell ergänzt werden
Verteilung	Nur der kompilierte Code wird auf dem Server installiert	Normalerweise wird der Quellcode auf dem Server verteilt. Es ist allerdings möglich den Code vorher zu kompilieren und auch nur das Resultat zu verteilen.

Ein Web-Anwendungsprojekt erstellen



Suche auf das Gebiet
Web beschränken

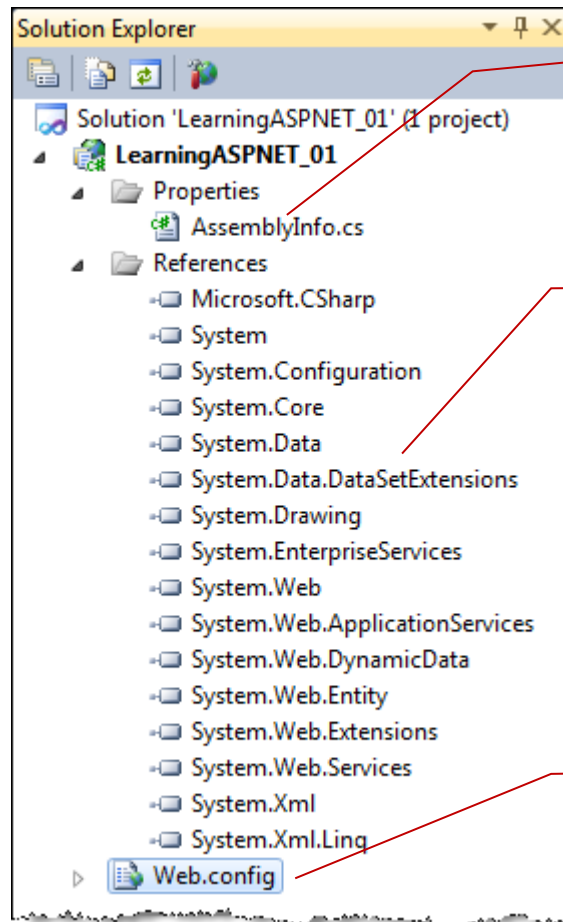
Vorlage auswählen

Projektname definieren

Speicherort auf der
Entwicklermaschine
definieren

Name des Verzeichnis in dem
die Solution abgelegt wird.
Dieser Name ist standardmässig
gleich wie der Projektname

- Das Resultat des Assistenten für eine leere Webanwendung ist ein beinahe leeres Projekt

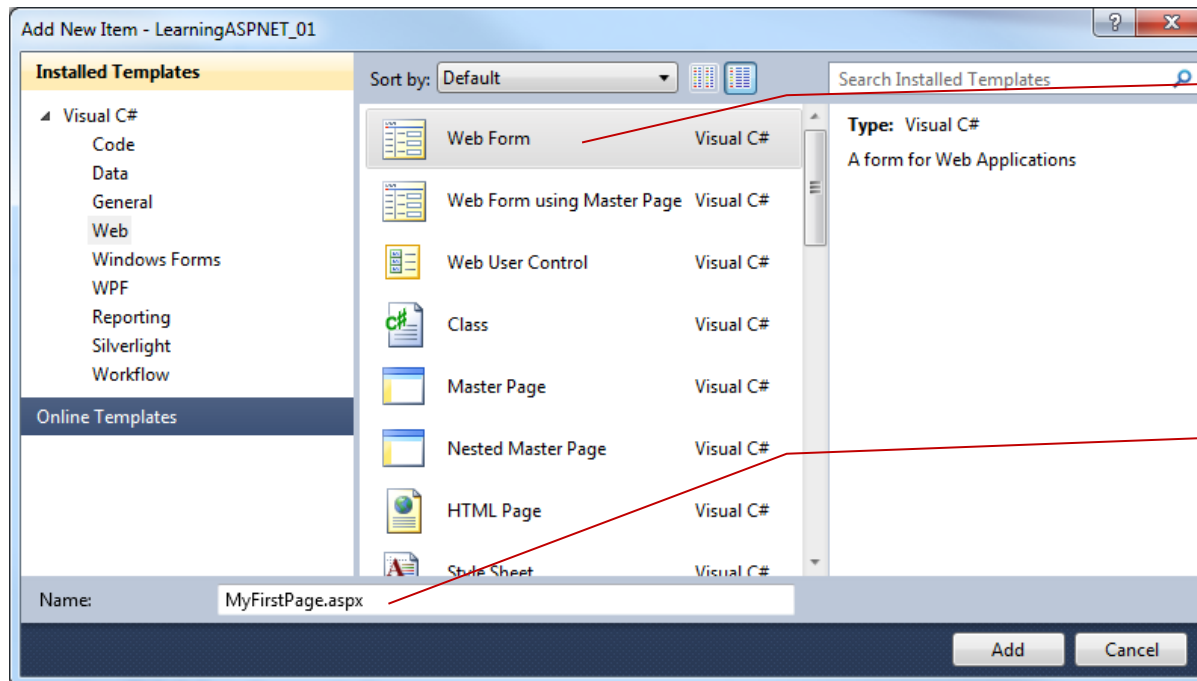


Standarddatei (wird über die Projekteigenschaften verändert).

Automatisch generierte Referenzierungen der .NET Assemblys

.NET-Anwendungskonfigurationsdatei einer Web Anwendung (gilt auch für Web Site Projekte). Verwenden Sie die Konfigurationsdatei wie jede andere Anwendungskonfigurationsdatei.

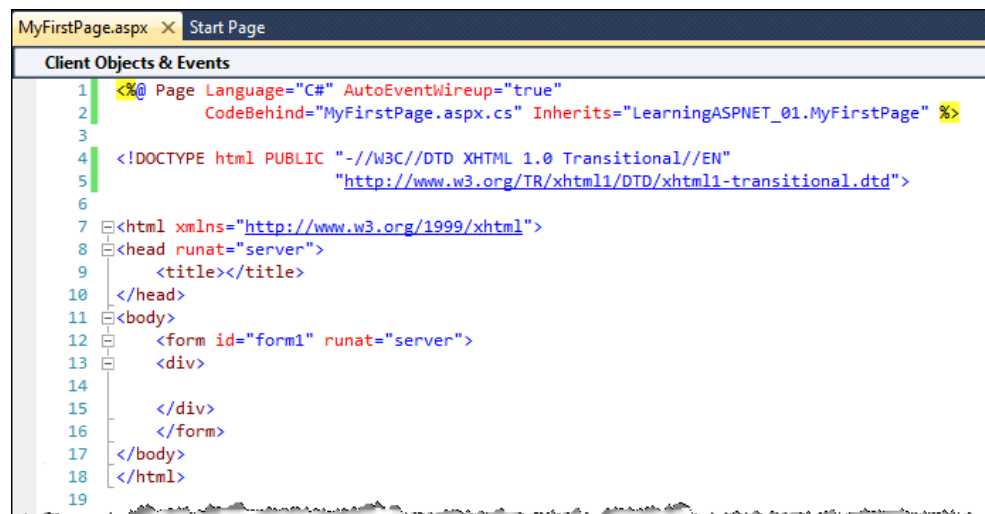
- Nach dem Erstellen des Projekts benötigen wir eine erste, vom Browser ansprechbare Inhaltsseite.
- Diese Seite wird über den Assistenten für Projektelemente erstellt
 - Wählen Sie im Solution Explorer den Projektknoten aus
 - Benutzen Sie den Menübefehl "Add/New Item" aus dem Kontextmenü



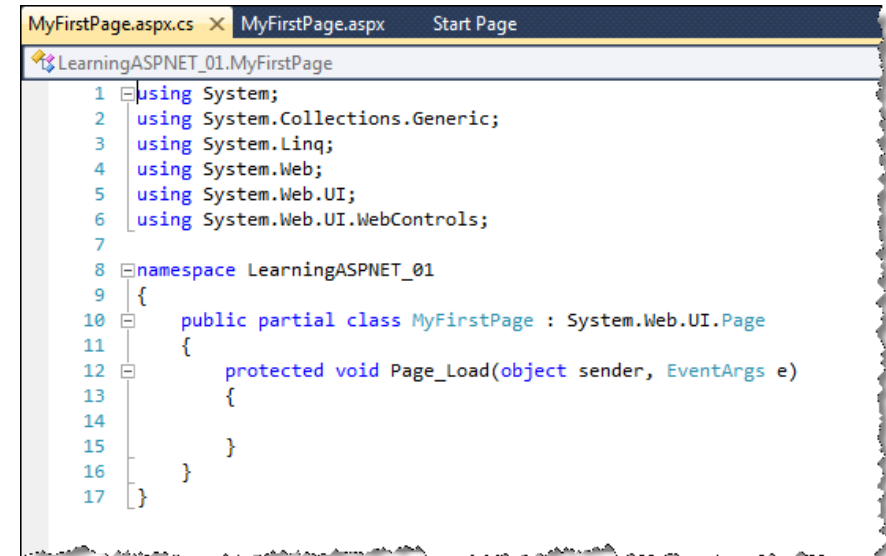
Web Form erstellt eine gewöhnliche Seite

Definieren Sie den Dateinamen. Belassen Sie die Namensendung aspx.

- Das Resultat eine generierten Web Form Seite sind zwei Dateien
 - MyFirstPage.aspx
 - MyFirstpage.aspx.cs
- Die Datei MyFirstPage.aspx nennen wir Designerdatei. Diese Datei besteht aus XHTML Code und beschreibt das Layout der Seite (unten links).
- Die Datei MyFirstPage.aspx.cs nennen wir Codebehind-Datei. Diese Datei besteht aus C# Code und definiert die Funktionalität der Seite (unten rechts).
- Wenn Sie genau hinschauen finden sie auch noch eine Datei mit Namen MyFirstPage.aspx.designer.cs. Hierbei handelt es sich um eine generierte Datei von Visual Studio mit den funktionellen Extrakten aus der Designer-Datei.

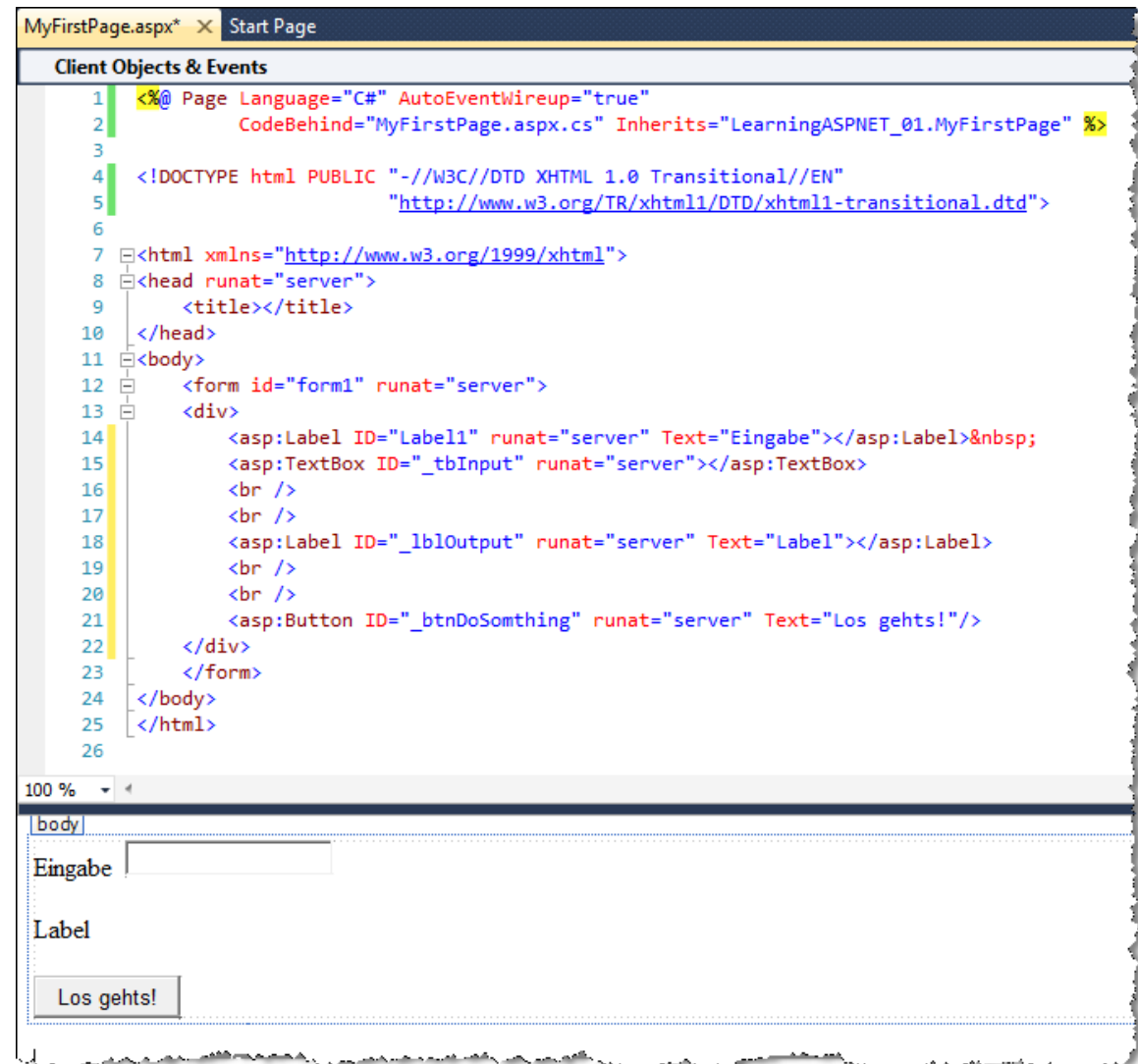


```
1 <%@ Page Language="C#" AutoEventWireup="true"
2     CodeBehind="MyFirstPage.aspx.cs" Inherits="LearningASPNET_01.MyFirstPage" %>
3
4 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
5     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6
7 <html xmlns="http://www.w3.org/1999/xhtml">
8 <head runat="server">
9 <title></title>
10 </head>
11 <body>
12 <form id="form1" runat="server">
13 <div>
14
15 </div>
16 </form>
17 </body>
18 </html>
19
```



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.UI;
6 using System.Web.UI.WebControls;
7
8 namespace LearningASPNET_01
9 {
10     public partial class MyFirstPage : System.Web.UI.Page
11     {
12         protected void Page_Load(object sender, EventArgs e)
13         {
14
15         }
16     }
17 }
```

- Die generierte Seite kann zwar aufgerufen werden, ist aber noch funktionslos.
- Wir ergänzen die Seite mit einer ersten Funktionalität. Öffnen Sie die Seite MyFirstpage.aspx im Editor und stellen Sie diesen in den Split-Modus (unten am Schirm).
- Ergänzen Sie den Code entsprechend dem nebenstehenden Bild.
- Der Editor zeigt im unteren Teil eine ungefähre Abbildung des resultierenden Bildes im Browser.



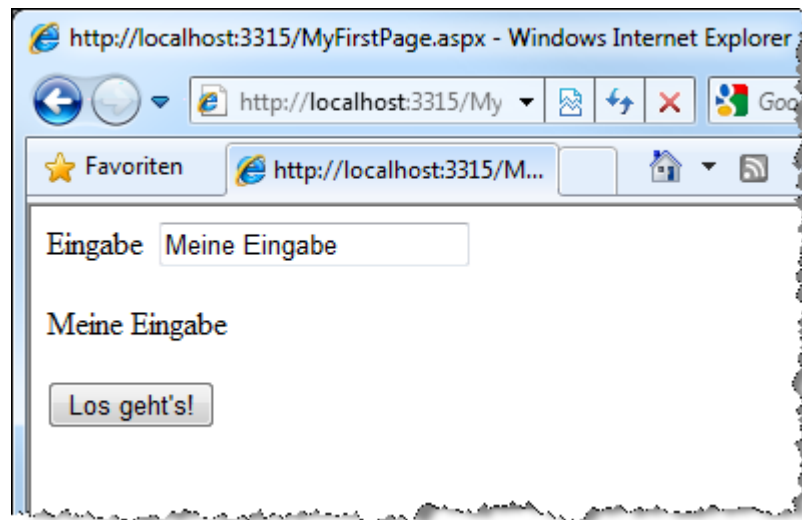
- Durch einen Doppelklick auf die Schaltfläche "Los geht's!" bringen wir die hier notwendige Funktionalität ein.
- Das Resultat ist ein zusätzliches Attribut im Element Button (onclick) ...

```
17 <br />
18 <asp:Label ID="_lblOutput" runat="server" Text="Label"></asp:Label>
19 <br />
20 <br />
21 <asp:Button ID="_btnDoSomething" runat="server" Text="Los gehts!"
22     onclick="_btnDoSomething_Click"/>
23 </div>
24 </form>
25 </body>
26 </html>
```

- ... und eine Methode im Codebehind, die es noch nach untenstehender Vorgabe anzupassen gilt

```
8 namespace LearningASPNET_01
9 {
10     public partial class MyFirstPage : System.Web.UI.Page
11     {
12         protected void Page_Load(object sender, EventArgs e)
13         {
14         }
15     }
16
17     protected void _btnDoSomething_Click(object sender, EventArgs e)
18     {
19         _lblOutput.Text = _tbInput.Text;
20     }
21 }
22 }
```

- Starten Sie nun die Anwendung, indem Sie über das Kontextmenü der Datei MyFirstPage.aspx im Solution Explorer den Menübefehl "View in Browser" benutzen.
- Visual Studio startet aufgrund dieses Befehls den integrierten ASP.NET Entwicklungsserver, der sich im Infobereich des Taskbar manifestiert.
- Dem Browser wird eine entsprechende URL für das Projekt übermittelt um die Seite darzustellen.
- Durch Eingabe eines Textes mit anschließendem Betätigen der Schaltfläche "Los geht's!" wird die programmierte Funktionalität ausgeführt.



Sie können den selben Effekt auch durch Debugmodus herbeiführen. Setzen Sie auf die entsprechende C# Zeile einen Haltpunkt und starten Sie die Anwendung mit der Taste F5.

- In der Einleitung habe ich auf das serverseitige Objektmodell verwiesen. Mit der ersten Seite können wir dieses Objektmodell visualisieren:

Unser Seitenobjekt

Expression: `((System.Web.UI.Control)(this)).Controls`

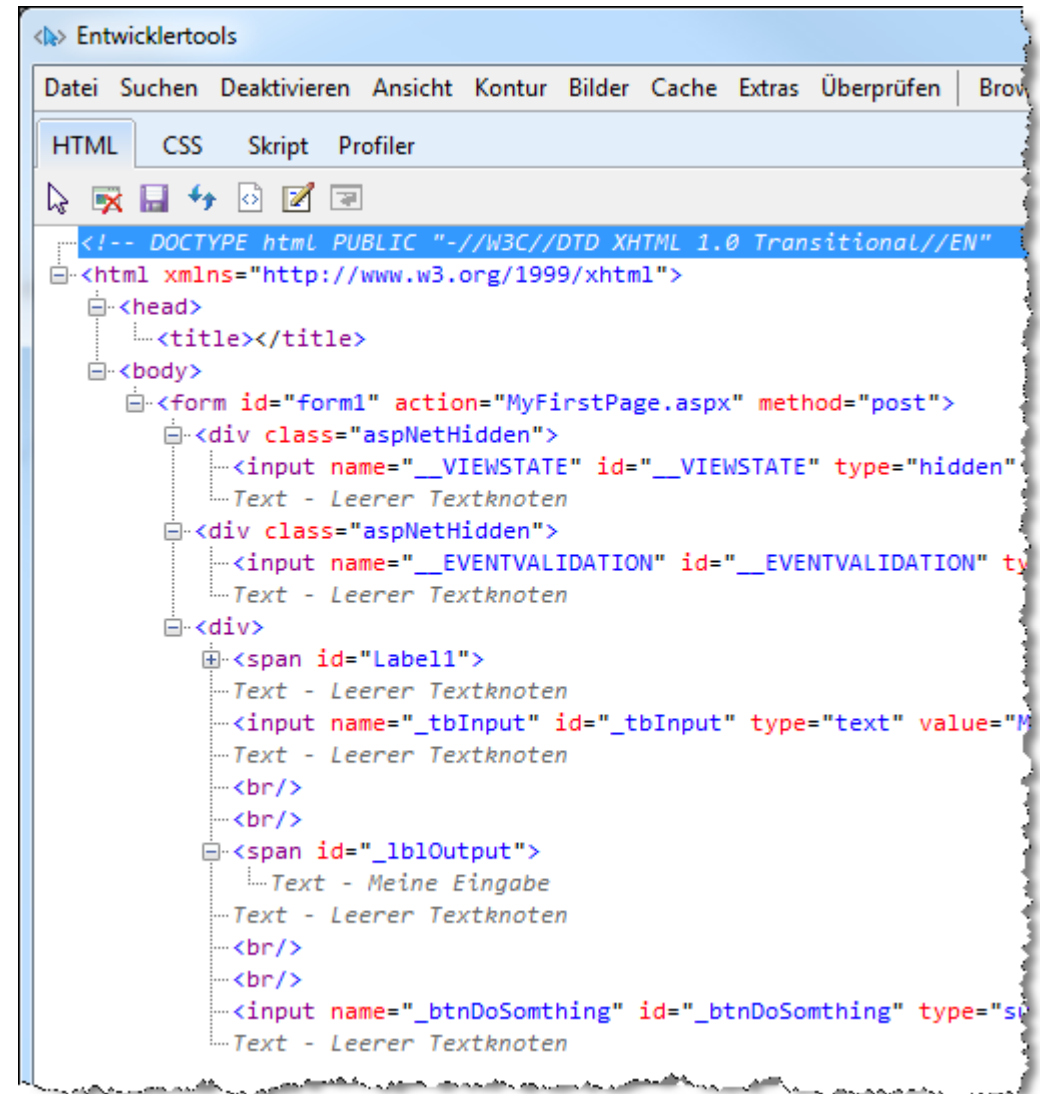
Value:

Name	Value	Type
this	{ASP.myfirstpage_aspx}	LearningASPNET_01.MyFirstPage {ASP.myfirstpage_aspx}
[ASP.myfirstpage_aspx]	{ASP.myfirstpage_aspx}	ASP.myfirstpage_aspx
base	{ASP.myfirstpage_aspx}	LearningASPNET_01.MyFirstPage {ASP.myfirstpage_aspx}
base	{ASP.myfirstpage_aspx}	System.Web.UI.Page {ASP.myfirstpage_aspx}
base	{ASP.myfirstpage_aspx}	System.Web.UI.TemplateControl {ASP.myfirstpage_aspx}
base	{ASP.myfirstpage_aspx}	System.Web.UI.Control {ASP.myfirstpage_aspx}
Adapter	null	System.Web.UI.Adapters.ControlAdapter
AppRelativeTemplateSourceDirectory	"~/	string
BindingContainer	null	System.Web.UI.Control
ChildControlsCreated	true	bool
ClientID	"_Page"	string
ClientIDMode	Inherit	System.Web.UI.ClientIDMode
ClientIDSeparator	95 '_'	char
Context	{System.Web.HttpContext}	System.Web.HttpContext
Controls	{System.Web.UI.ControlCollection}	System.Web.UI.ControlCollection
Count	5	int
IsReadOnly	false	bool
IsSynchronized	false	bool
SyncRoot	{System.Web.UI.ControlCollection}	object {System.Web.UI.ControlCollection}
Non-Public members		
_controls	{System.Web.UI.Control[5]}	System.Web.UI.Control[]
[0]	{System.Web.UI.LiteralControl}	System.Web.UI.Control {System.Web.UI.LiteralControl}
[1]	{InnerText = '((System.Web.UI.HtmlControls.H	System.Web.UI.Control {System.Web.UI.HtmlControls.HtmlHead}
[2]	{System.Web.UI.LiteralControl}	System.Web.UI.Control {System.Web.UI.LiteralControl}
[3]	{System.Web.UI.HtmlControls.HtmlForm}	System.Web.UI.Control {System.Web.UI.HtmlControls.HtmlForm}
[4]	{System.Web.UI.LiteralControl}	System.Web.UI.Control {System.Web.UI.LiteralControl}
_defaultCapacity	5	int

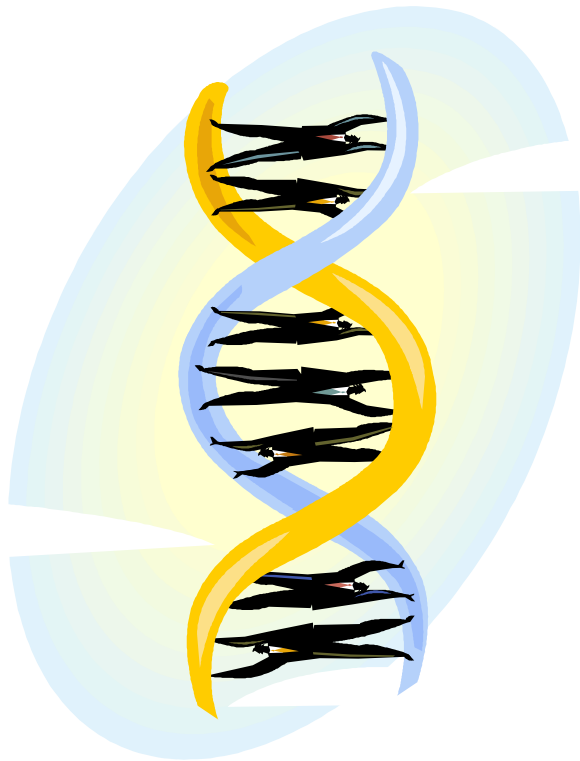
Container für enthaltene Bedienelemente

Liste der enthaltenen Bedienelemente

- Der Browser unterstützt ein clientseitiges Entwicklungstool, mit dem Sie das clientseitige Objektmodell visualisieren können.
- Beobachten Sie, dass beim anwählen der einzelnen Elemente im Entwicklungstool, die Elemente im Browser mit umrahmt werden.
- Sie können auch Clientseitig die Werte im Objektmodell ändern!



```
<!-- DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
<body>
<form id="form1" action="MyFirstPage.aspx" method="post">
  <div class="aspNetHidden">
    <input name="__VIEWSTATE" id="__VIEWSTATE" type="hidden"
    Text - Leerer Textknoten
  <div class="aspNetHidden">
    <input name="__EVENTVALIDATION" id="__EVENTVALIDATION" ty
    Text - Leerer Textknoten
  <div>
    <span id="Label1">
      Text - Leerer Textknoten
      <input name="_tbInput" id="_tbInput" type="text" value="M
      Text - Leerer Textknoten
      <br/>
      <br/>
    <span id="_lblOutput">
      Text - Meine Eingabe
      Text - Leerer Textknoten
      <br/>
      <br/>
      <input name="_btnDoSomthing" id="_btnDoSomthing" type="st
      Text - Leerer Textknoten
```



Anatomie der Web-Anwendung

- Eine Anwendung besteht aus einer Vielzahl von Elementen. Die Elemente definieren der Gestaltung, der Funktionalität oder schlicht der Organisation der Seite.
- Funktionale Elemente
 - 1 Anwendungsklasse
 - n Klassen für Seiten (Code Behind)
 - 1 bis n Anwendungskonfigurationen
 - Zustandsverwaltung
 - Bedienelemente
 - Eigene HTTP-Handler
- Gestalterische Elemente
 - n Klassen für Seiten (Designercode)
 - Themen, Skins und CSS
 - Bitmaps
- Organisatorische Elemente
 - Verzeichnisse
 - Erweiterungen von Dateinamen
 - Referenzen und Benutzen von eigenen oder fremden Assemblys

- Die Organisation der Seite wird mit Verzeichnissen hergestellt
- Sie sind grundsätzlich in der Benennung der Verzeichnisse für ihre Gliederung frei
- Ausnahmen bilden folgende Namen, die zum Teil von ASP.NET vorgegeben sind

Verzeichnis	Verwendung
App_Browsers	Speichert .browser Dateien, die ASP.NET benutzt um Browser zu identifizieren und deren Möglichkeiten zu bestimmen.
App_Code	Verzeichnis für die Aufnahme von beliebigen allgemeinen Quellcodedateien. Die in diesem Verzeichnis enthaltenen Dateien werden automatisch in eine Assembly kompiliert und von den andern Assemblies referenziert.
App_Data	Verzeichnis für die Aufnahme von Daten (lokale Datenbanken, XML-Dateien usw.).
App_GlobalResources	Nimmt globale Ressourcen auf (.resx und .resources). Globale Ressourcen sind streng typisiert und können programmatisch verwendet werden.
App_LocalResources	Nimmt Ressourceinformationen auf (.resx und .resources), die mit einer bestimmten Seite verbunden sind.
App_Themes	Nimmt alle Dateien auf, die das Erscheinen der Seiten beeinflussen (themenorientierte Farbzusammenstellungen oder Cascaded Stylesheets).
App_WebReferences	Nimmt .wsdl, XML Schemadefinitionen oder andere Definitionsdaten für den Zugriff auf fremde Dienste auf.
Bin	Nimmt Assemblys von Controls oder andern Bibliotheken auf, die in der Webseite verwendet werden sollen. Alle in diesem Verzeichnis untergebrachten Daten werden im Projekt automatisch referenziert.

- Ausnahmen bilden auch die sich durch Marktgewohnheiten etablierten Verzeichnisse:

Verzeichnis	Verwendung
Scripts	Verzeichnis zur Aufnahme von JavaScript-Dateien
Styles	Verzeichnis zur Aufnahme von .css-Dateien. Beachten Sie dass .css-Dateien auch im Verzeichnis App-Themes abgelegt werden
Images	Verzeichnis zur Aufnahme von statischen Bilddateien, die vom Client her auch abgefragt werden können

- Auch primär ein organisatorisches Element sind die Erweiterungen der Dateinamen
- Mit Hilfe von diesen werden zur Laufzeit typen erkannt und zur Entwicklungszeit Editoren gesteuert
- Es lohnt sich generell nicht, die Erweiterungen von Dateinamen anzupassen, obschon dies selbstverständlich möglich wäre

Verzeichnis	Verwendung
.aspx	Layout einer Webseite mit oder ohne Code-Behind.
.ascx	Layout eines eigenen Steuerelements.
.asax	Codedatei für die Handhabung von Ereignissen auf Stufe der Anwendung.
.browser	XML-Datei mit Definitionen über Browser, die für die Steuerung des Rendering verwendet werden.
.config	XML-Datei mit Konfigurationsangaben des Bereichs (web.config).
.cs	Codedatei mit C# Code.
.css	Cascaded Stylesheet.
.htm .html	Layoutdatei einer Webseite im XHTML-Format.
.js	JavaScript Datei.
.master	Layout einer Master-Webseite.
.resx	Datenfile mit Ressourcen im XML-Format.
.sitemap	Datenfile mit einer Beschreibung im XML Format für das Steuerelement Sitemap.
.skin	Layoutangaben für die Steuerung des Aussehens von Seiten.

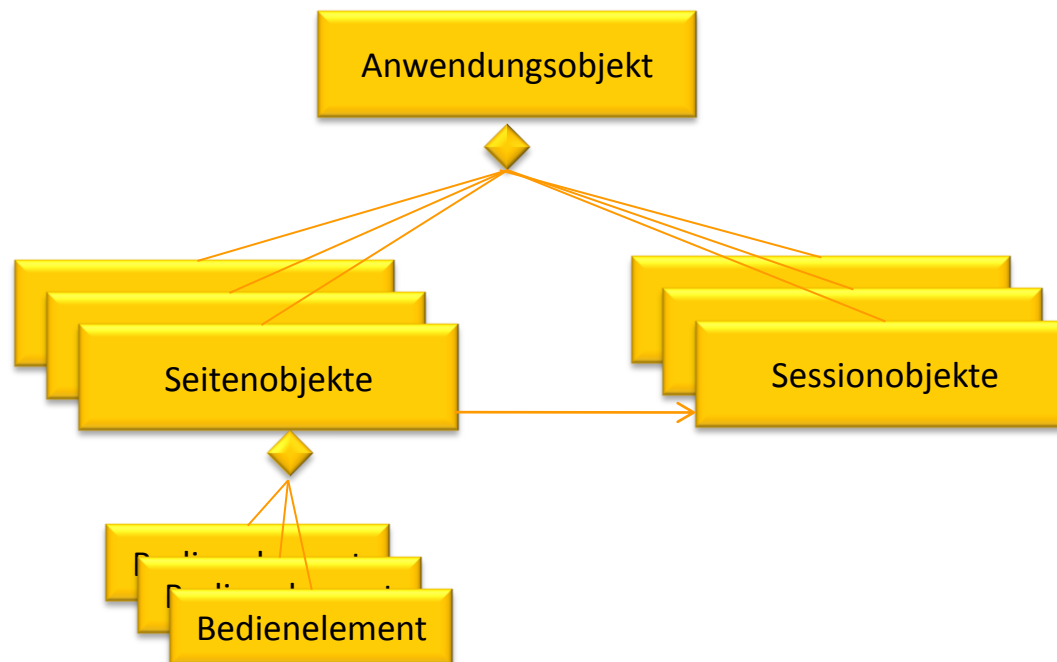
- Im Verzeichnis Referenzen sind die referenzierten Assemblys definiert. Dieses Verzeichnis gilt nur für den Projekttyp Web-Anwendung. Bei einem Web Site Projekt sind die Referenzen über die Konfigurationsdatei des Systems definiert, respektive können über die Anwendungskonfiguration eingebracht werden.
- Folgende Assemblys sind einem Web Site Projekt standardmässig referenziert:

```
<assemblies>
  <add assembly="mscorlib" />
  <add assembly="Microsoft.CSharp, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Configuration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Data, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web.Services, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Xml, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.EnterpriseServices, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.IdentityModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Runtime.Serialization, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel.Activation, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ServiceModel.Activities, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.WorkflowServices, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Core, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web.Extensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Data.DataSetExtensions, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Xml.Linq, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.ComponentModel.DataAnnotations, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web.DynamicData, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="System.Web.ApplicationServices, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <add assembly="*" />
</assemblies>
```

- Für die vereinfachte Verwendung sind in Web Site Projekten folgende Namensräume bereits eingeschaltet:

```
<?xml
<namespaces>
  <add namespace="System" />
  <add namespace="System.Collections" />
  <add namespace="System.Collections.Generic" />
  <add namespace="System.Collections.Specialized" />
  <add namespace="System.ComponentModel.DataAnnotations" />
  <add namespace="System.Configuration" />
  <add namespace="System.Linq" />
  <add namespace="System.Text" />
  <add namespace="System.Text.RegularExpressions" />
  <add namespace="System.Web" />
  <add namespace="System.Web.Caching" />
  <add namespace="System.Web.DynamicData" />
  <add namespace="System.Web.SessionState" />
  <add namespace="System.Web.Security" />
  <add namespace="System.Web.Profile" />
  <add namespace="System.Web.UI" />
  <add namespace="System.Web.UI.WebControls" />
  <add namespace="System.Web.UI.WebControls.WebParts" />
  <add namespace="System.Web.UI.HtmlControls" />
  <add namespace="System.Xml.Linq" />
</namespaces>
```

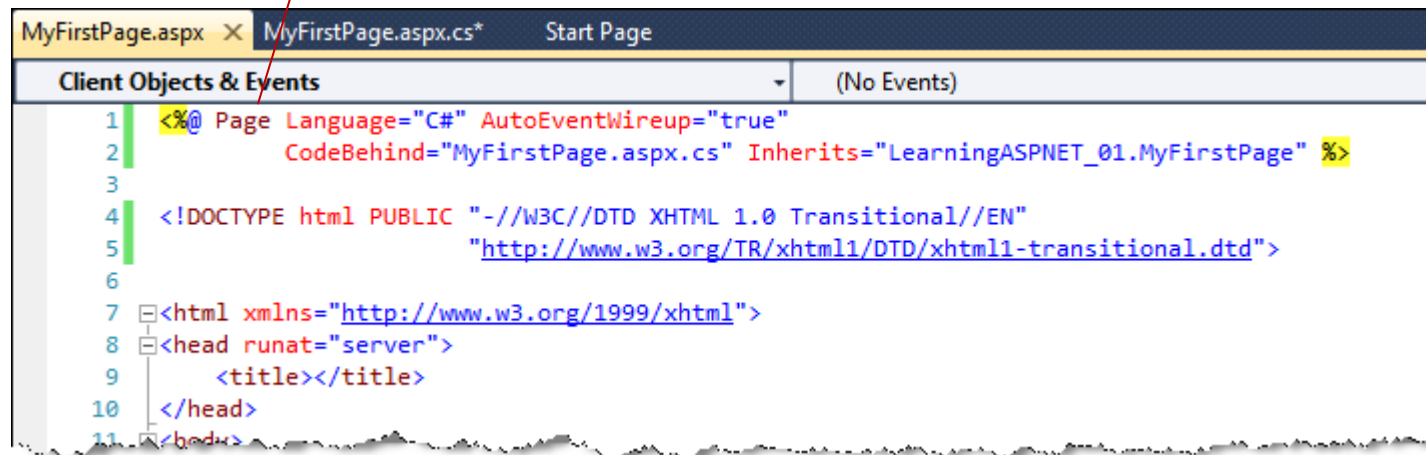
- Eine Web-Anwendung besteht zur Laufzeit aus genau einem Anwendungsobjekt und pro Browserverbindung einem Sessionobjekt. Ferner wird für jede Anforderung kurzzeitig ein Seitenobjekt mit seinem enthaltenen Objektbaum zum Leben erweckt.



- Ein Seitenobjekt ist mit seinem zugehörigen Sessionobjekt verbunden
- Die Erstellung und Zerstörung der Objekte obliegt ASP.NET (siehe auch Lebenszyklen der Objekte einer Anwendung)

- Die Klassen in den Web Projekten werden für die Verarbeitung von ASP.NET in verschiedene Arten eingeteilt. Die Arten dienen ASP.NET und Visual Studio während der Entwicklung für die Unterstützung der Editoren respektive während der Laufzeit für die Steuerung des Verhaltens.
- Die Art des Codes wird mittels Direktiven im Designercode definiert.
- Das Beispiel unserer ersten Seite zeigt in der ersten Zeile die Direktive Page.

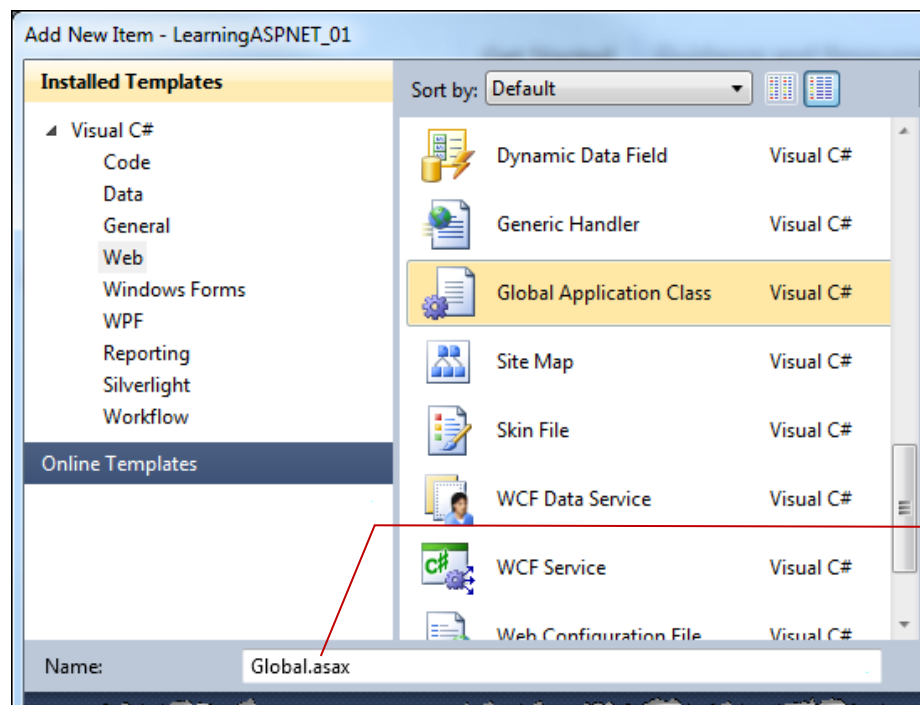
Die Direktive Page definiert die Verarbeitungsart und dient den Editoren für die Erkennung der notwendigen Funktionalität.



```
1 <%@ Page Language="C#" AutoEventWireup="true"
2     CodeBehind="MyFirstPage.aspx.cs" Inherits="LearningASPNET_01.MyFirstPage" %>
3
4 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
5     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
6
7 <html xmlns="http://www.w3.org/1999/xhtml">
8 <head runat="server">
9     <title></title>
10 </head>
11 <body>
```

Direktive	Verwendung
<i>@Page</i>	Definiert seitenspezifische Attribute für den Compiler und die Ausführung der Seite. Diese Direktive kann nur bei .aspx-Dateien angewendet werden.
<i>@Master</i>	Definiert eine Masterseite.
<i>@Assembly</i>	Linkt die aktuelle Seite mit dem definierten Assembly und macht alle darin vorhandenen Typen für die Seite zugänglich.
<i>@Control</i>	Definiert Attribute für den Compiler und die Ausführung der Seite in Bezug auf ein bestimmtes Steuerelement. Diese Direktive kann nur bei .ascx-Dateien verwendet werden.
<i>@Implements</i>	Mit dieser Direktive kann deklarativ angezeigt werden, dass die Seite ein bestimmtes Interface implementiert.
<i>@Import</i>	Importiert explizit einen Namensraum in eine Seite oder ein eigenes Control.
<i>@MasterType</i>	Erlaubt eine strenge Prüfung des Typs der Masterseite.
<i>@OutputCache</i>	Definiert das Verhalten des Ausgabespeichers in Bezug auf die Seite.
<i>@PreviousPageType</i>	Erlaubt die strenge Prüfung gegenüber der vorangehenden Seite. Diese Direktive kann nur bei einer .aspx-Seite angewendet werden.
<i>@Reference</i>	Erlaubt die Angabe einer externen Datei, die dynamisch mit der aktuellen Datei zusammengelinkt werden soll.
<i>@Register</i>	Definiert ein neues Präfix für spezifische Tags in der XML Datei für die Verwendung von Steuerelementen. Diese Direktive kann in normalen Seiten in Masterseiten und in Steuerelementen verwendet werden.
<i>@Application</i>	Definiert eine globale Anwendungsdatei (siehe 8.6.1.2).

- Die globale Anwendungsklasse dient als toplevel Container zur Verwaltung der diversen Objekte, die im Verlauf des Lebenszyklus der Anwendung erstellt werden
- Die globale Anwendungsklasse wird vom Typ `HttpApplication` instanziiert und kann in einem konkreten Projekte als Vererbung spezialisiert werden
- Die Spezialisierung kann dazu verwendet werden, vor dem Verbinden des ersten Clients Daten aufzubereiten und oder Verbindungen zur Anwendung zu kontrollieren
- Eine Spezialisierung der Anwendungsklasse wird mittels Menübefehle "Add/New Item" und Auswahl des Elements "Global Application Class" erstellt.



Der übliche Name "Global.asax" wird bereits vorgeschlagen und muss so belassen werden!

- Die globale Anwendungsklasse definiert diverse Eventhandler, die Sie nach eigenen Anforderungen nutzen können.
- Die Reihenfolge der Aufrufe der Eventhandler entspricht der Ordnung unten.
- Beachten Sie, dass nicht bei jeder Verarbeitung einer Anforderung jeder Eventhandler aufgerufen wird

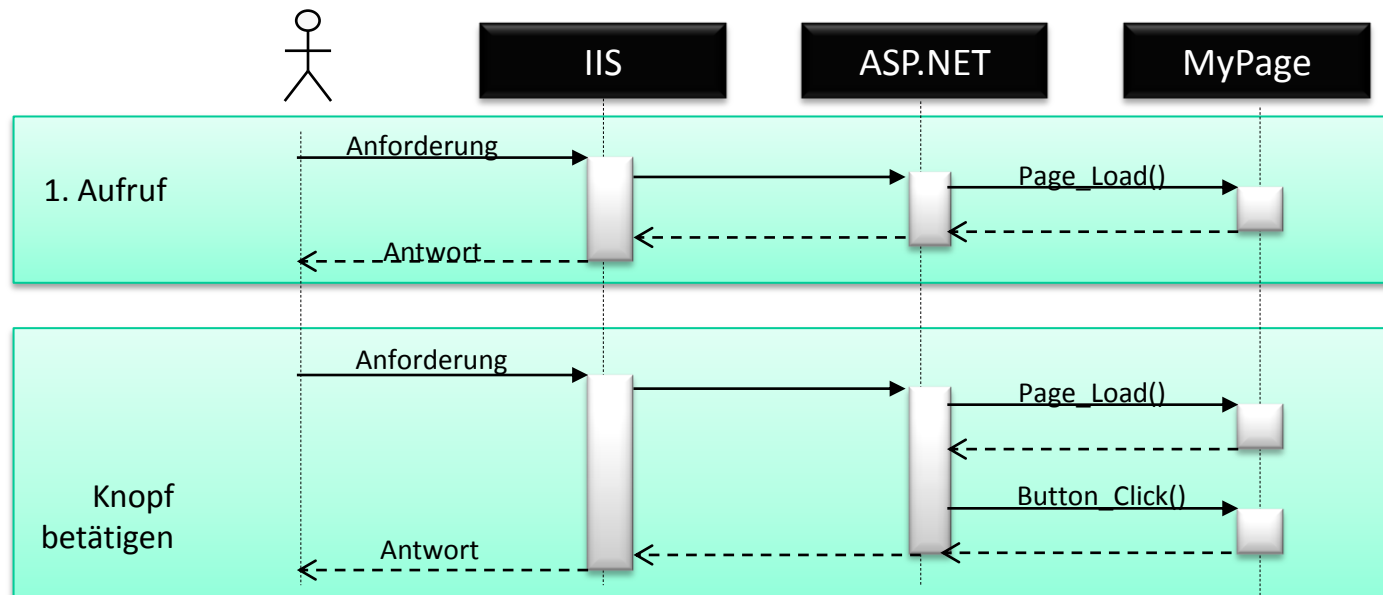
<code>public class Global : System.Web.HttpApplication</code>	
<code>{</code>	
<code>protected void Application_Start(object sender, EventArgs e) {}</code>	Einmal pro Anwendungsobjekt
<code>protected void Application_BeginRequest(object sender, EventArgs e) {}</code>	Einmal pro Anforderung
<code>protected void Application_AuthenticateRequest(object sender, EventArgs e) {}</code>	Einmal pro Anforderung
<code>protected void Session_Start(object sender, EventArgs e) {}</code>	Einmal pro Session
<code>protected void Session_End(object sender, EventArgs e) {}</code>	Einmal pro Session
<code>protected void Application_End(object sender, EventArgs e) {}</code>	Einmal pro Anwendungsobjekt
<code>protected void Application_Error(object sender, EventArgs e) {}</code>	Einmal pro unbehandeltem Fehler
<code>}</code>	

- Für die Anpassungen der globalen Anwendungsklasse eignen sich die Ereignishandler wie folgt:
 - Application_Start für einmalige Initialisierungen der Anwendung
 - Session_Start für einmalige Initialisierungen der Session eines Browsers
 - Application_Error für allgemeine Fehlerbehandlung
- Die restlichen Ereignishandler werden eher selten verwendet
- Im Ereignis Application_Error kann es besonders nützlich sein, den letzten Fehler

```
protected void Application_Error(object sender, EventArgs e) {  
    // Try to get the last exception  
    Exception excObject = Server.GetLastError();  
  
    // Create a mail message  
    ...  
  
    // Check error has been recovered  
    if (excObject != null) {  
        ...  
    } else {  
        ...  
    }  
  
    // Send mail message  
    ...  
}
```

- Studieren Sie auch die Klasse `HttpApplication` in der Onlinedokumentation. Beachten Sie, dass viele Ereignishändler der Klasse nur im IIS 7.0 integrierten Pipelinemodus erlaubt sind.

- Wie wir bereits wissen wird eine Seite als Objektmodell zum Leben erweckt
- Auslöser ist die Anforderung des Browsers, Resultat ist die Antwort zum Browser
- Wir unterscheiden bei der Verarbeitung von Seiten zwei Hauptzustände
 - Erster Aufruf
 - Wiederholter Aufruf (Postback)
- Die Unterscheidung der beiden Zustände ist dahingehend wichtig, da beim Erstaufruf sehr oft Initialisierungen vorgenommen werden, die eben nur beim ersten Aufruf stattfinden



- Der Postback wird normalerweise im Load-Ereignis verarbeitet

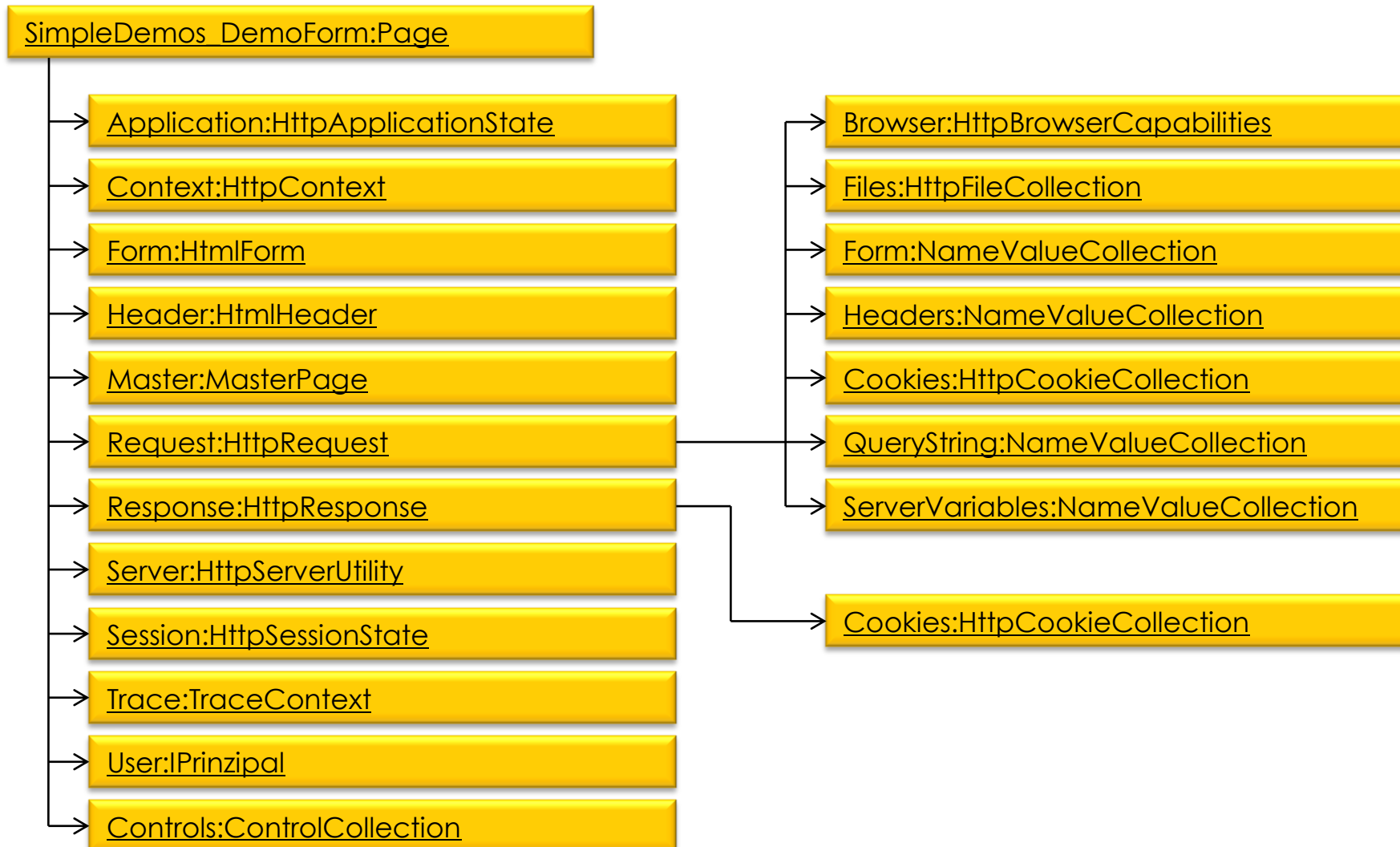
```
/// <summary>
/// Ereignisbehandlung Seite laden
/// </summary>
protected void Page_Load(object sender, EventArgs e) {

    // Prüfe erstmaliger Aufruf
    if (!this.IsPostBack)
    {
        // Code für erstmaligen Aufruf
        ...
    }
    else
    {
        // Code für wiederholten Aufruf
        ...
    }
}
```

- Das Objektmodell der Seite durchläuft einen definierten Lebenszyklus
- Der Lebenszyklus kann mittels Ereignishandler kontrolliert werden.

Ereignis	Verwendung
PreInit	Dieses Ereignis dient zu folgenden Zwecken: Erstellen dynamischer Bedienelemente, Masterseite dynamisch definieren Themen dynamisch definieren, Lesen oder definieren von Profildaten
Init	Die Initialisierung der Elemente erfolgt vor der Initialisierung der Seite. Nutzen Sie dieses Ereignis um anwendungsspezifische, dynamische Initialisierungen von Eigenschaften der Steuerelemente vorzunehmen
InitComplete	Nutzen Sie dieses Ereignis um den ViewState zu beeinflussen
PreLoad	Nutzen Sie diesen Ereignis um Vorverarbeitungen für das Loadereignis vorzunehmen (selten)
Load	Unterschieden Sie den Erstaufwurf vom wiederholten Aufruf und nehmen Sie hier die Hauptinitialisierung der Seite vor
Control Ereignisse	Hier werden alle Ereignisbehandlungen der eigenen Steuerelemente vorgenommen. Bei einem Postback werden die Ereignisse für Validierung vorgängig gehandhabt.
LoadComplete	Abschlussereignis der eigentlichen Verarbeitung. Hier nehmen Sie Einfluss auf das gesamte fertig verarbeitete Objektmodell der Seite
PreRender	Vorverarbeitung der Umsetzung in einen HTML-Datenstrom. Nutzen Sie dieses Ereignis um finale Änderungen der Staus der Steuerelemente vorzunehmen
PreRenderComplete	Etwaig zu verwenden für die dynamische Datenbindung
SaveStateComplete	Wird aufgerufen nachdem der Viewstate und der Controlstate definiert sind
Render	Render ist eigentlich kein Status, sondern ein Verarbeitungsschritt, der für das gesamte Objektmodell durchgeführt wird
Unload	Wird für jedes Element der Seite und die Seite selber aufgerufen um die Seite zu entladen

- Ohne dass wir eigene Steuerelemente in die Seite einbringen, verfügt die Seite bereits über ein umfangreiches Objektmodell



Objekt	Verwendung
Application	Bietet Zugriff auf das globale Anwendungszustandsobjekt und erlaubt den Austausch von Informationen zwischen einzelnen Anforderungen der Anwendung.
Context	Erlaubt den Zugriff auf die Daten des gesamten Kontextes der Anforderung. Darin sind wiederum die meisten Objekte des Modells erreichbar.
Form	Erlaubt den programmatischen Zugriff zum HTML-Element <form>.
MasterPage	Erlaubt den Zugriff auf die Masterseite der Seite.
Request	Erlaubt den Zugriff auf die Daten der Anforderung.
Request.Browser	Erlaubt den Zugriff die Detailinformationen des Browsers, der die Anforderung auslöste
Request.QueryString	Erlaubt den Zugriff auf die Daten der URL der Anforderung
Request.Cookies	Erlaubt den Zugriff auf die Cookies der Anforderung
Controls	Die Liste enthält alle unterstellten Steuerelemente der Seite in einer hierarchischen Abbildung. Der Inhalt ist entsprechend seitenspezifisch.
Server	Erlaubt den Zugriff auf Informationen und Funktionen des aktuellen Serverkontexts
Session	Erlaubt den Zugriff auf die Informationen der aktuellen Browsersession
User	Erlaubt Zugriff auf die Benutzerdaten, die via Anforderung übermittelt wurden

- Internetserver sind für die Bearbeitung von hunderten oder tausenden von Requests von Browsern ausgelegt
- Entsprechend dieser Auslegung wird es verständlich, dass die Objekte einer Seite nach deren Verarbeitung wieder abgebaut werden
- Mit dem ständigen Auf- und wieder Abbau der Seiten geht aber das Problem einher, dass der Server zwischen zwei Anforderungen des selben Browsers auf die selbe Seite keine Daten rettet, und somit beim wiederholten Aufruf grundsätzlich das selbe Objektmodell mit den gleichen Daten erstellt.
- Daraus folgt, dass Zustände zwischen zwei Aufrufen nicht gerettet werden
- Für die Lösung des oben geschilderten Problems bietet ASP.NET nicht nur einen Lösungsansatz sondern eine ganze Palette von Lösungsmöglichkeiten

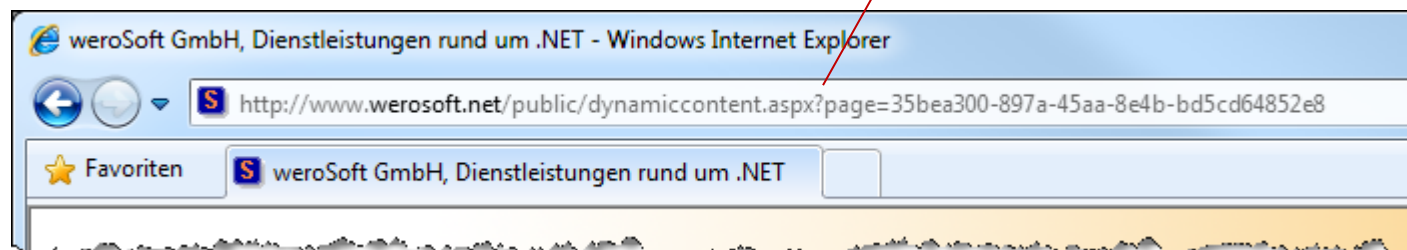
Name der Technik	Kategorie	Browser-abhängig	Gültigkeitsbereich	Sicherheit	Grösse	Automatisch gelöscht
Viewstate	Client	Nein	Anforderung	Mittel	Klein	Ja
Controlstate	Client	Nein	Anforderung	Mittel	Klein	Ja
Hidden Fields	Client	Nein	Anforderung	Niedrig	Klein	Ja
QueryString	Client	Nein	Anforderung	Niedrig	<1024	Ja
Cookies	Client	Ja	Diverse	Niedrig	<4096	Ja/Nein
Applicationstate	Server	Nein	Anwendung	Hoch	Gross	Ja
Cache	Server	Nein	Anwendung	Hoch	Gross	Ja/Nein
Sessionstate	Server	Nein	Sitzung	Hoch	Mittel	Ja/Nein
Profiling	Server	Nein	Benutzer	Hoch	Mittel	Nein

Die Beurteilung der Sicherheit geht davon aus, dass serverseitige Mechanismen grundsätzlich sicher sind, während dem clientseitige grundsätzlich weniger sicher sind, obschon mittels Verschlüsselung von übermittelten Daten heute auch clientseitige Daten relativ sicher gestaltet werden können.

Bei der Beurteilung der Grösse ist darauf zu achten, dass vor allem der Multiplikator der Menge eine Auswirkung auf die gesamthafte benötigte Grösse des Speichers hat. Ein Objekt kommt dabei besser weg, als 1 Objekt pro Benutzer.

- Der Querystring definiert die Übertragung von Informationen vom Client zum Server, indem der URL Argumente übergeben werden
- Der Querystring ist nicht ASP.NET spezifisch
- Die Argumente werden dabei mit einem '?'-Zeichen getrennt, der URL angehängt. Sind mehrere Argumente vorhanden, sind diese mit einem '&'-Zeichen voneinander zu trennen. Der Querystring überträgt die Parameter mit einem normalen HTTP-GET Befehl.
- Auf der Seite des Servers werden die übertragenen Parameter über die Eigenschaft Querystring der Basisklasse von Page ermittelt und im Programm verwendet
- Daten im Querystring können vom Benutzer eingesehen werden
- Querystring können sehr einfach gefälscht werden. Daraus folgt, dass diese Eingabedaten überprüft werden müssen
- Für die Verwendung in der Zustandsverwaltung müssen Querystrings dynamisch auf den entsprechenden Elementen definiert werden

Die Parameter werden der URL angehängt



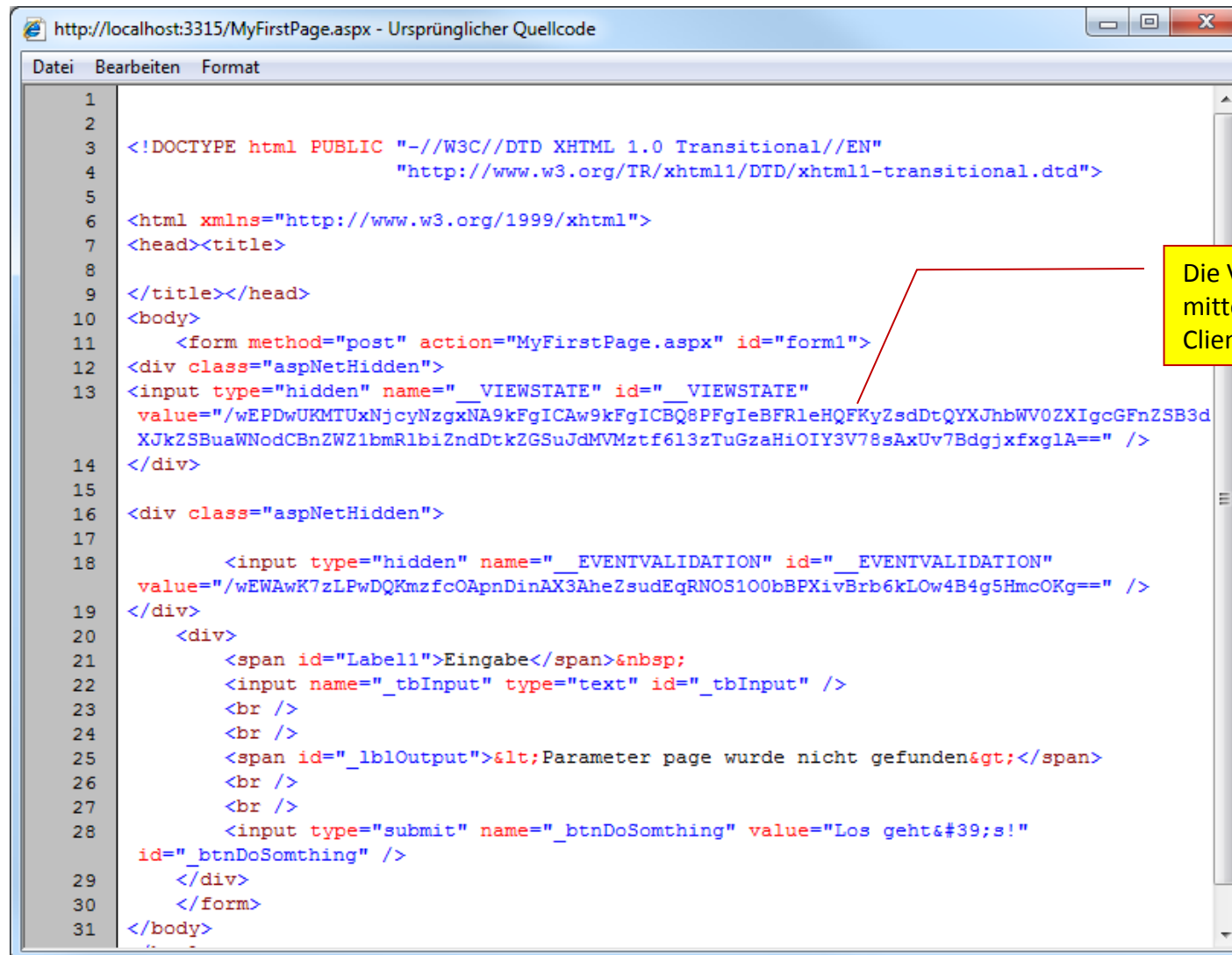
- Das serverseitige Abfragen eines Querystrings geschieht direkt über die entsprechende Eigenschaft des Request-Objekts.
- Beachten Sie, dass in HTML bestimmte Zeichen codiert werden müssen, da sie in HTML eine spezielle Bedeutung haben

```
protected void Page_Load(object sender, EventArgs e)
{
    string strData = Request.QueryString["page"];
    if (string.IsNullOrEmpty(strData))
    {
        _lblOutput.Text = Server.HtmlEncode("<Parameter page wurde nicht gefunden>");
    }
    else
    {
        strData = Server.HtmlDecode(strData);
        _lblOutput.Text = strData;
    }
}
```

- Das hidden Field stammt aus dem HTML Standard 3.2 von Anfang 1997. Es ist somit ein bereits seit längerer Zeit vorhandenes Element, das insbesondere nicht ASP.NET spezifisch ist.
- Das Feld ist vom W3C Konsortium für die sitzungslose Zustandsverwaltung vorgesehen worden
- Das Hidden Field wird in Form eines Steuerelements in die Seite eingelagert
- Gemäß Spezifikation von HTML darf das Feld oder sein Inhalt nicht vom Browser angezeigt werden, und das gesamte Feld muss bei einem HTTP-POST wieder zurück an den Server gesendet werden
- Hidden Fields können über die Ansicht des Quellcodes auf der Clientseite betrachtet werden
- Der Inhalt von Hidden Fields wird in der Regel nicht verschlüsselt (Aufwand). Die Handhabung obliegt aber voll und ganz dem Anwendungsentwickler
- Proxy und Firewalls können über Beschränkungen für die Übermittlung von grossen Dateninhalten in unsichtbaren Feldern verfügen

Mehr Details zu Hidden Field siehe Thema Steuerelemente

- Der Viewstate ist der Standardmechanismus der Klasse *Page* für die persistente Gestaltung der Daten in einer Seite zwischen zwei Anforderungen und ist somit ASP.NET spezifisch
- Die Klasse *Page* speichert darin automatisch notwendige Daten. Zusätzlich können Sie eigene Werte für die Berechnungen oder Bearbeitung der Daten im Viewstate speichern. Der Viewstate unterstützt dabei folgende Datentypen:
 - Fundamentale Datentypen (*String*, *Integer*, *Boolean* ...)
 - Arrays
 - Objekte der Klasse *ArrayList*
 - Objekte der Klasse *HashTables*
 - Beliebige eigene Typen die nach XML serialisierbar sind
- Der Viewstate ist einmal mehr ein Container, der sich in Schlüssel / Wert Paaren organisiert. Die Daten des Containers werden beim Schreiben in die Seite nach XML serialisiert und anschließend in einen String zur Basis 64 umgewandelt. Dieser String wird letztendlich in einem oder mehreren Hidden Fields in der Antwort an den Client verpackt.
- Die Nutzung von Viewstates ist standardmäßig eingeschaltet.
- Viewstates erzeugen ca. 33% Overhead in der Serialisierung, das kann zu großen Datenmengen führen.
- Proxy und Firewalls können über Beschränkungen für die Übermittlung von großen Dateninhalten in unsichtbaren Feldern verfügen.
- Die Viewstate Daten können auf mehrere hidden Fields aufgeteilt werden (siehe Klasse *Page*, Eigenschaft *MaxPageState-FieldLength*).
- Die Größe der Viewstates kann die Performance der Anwendung schwächen (Leitungsgeschwindigkeit zum Client beachten).



```
1
2
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
4     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5
6 <html xmlns="http://www.w3.org/1999/xhtml">
7 <head><title>
8
9 </title></head>
10 <body>
11     <form method="post" action="MyFirstPage.aspx" id="form1">
12 <div class="aspNetHidden">
13 <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
14     value="/wEPDwUKMTUxNjcyNzg5NA9kFgICA9kFgICBQ8PFgIeBFRleHQFKyZsdDtQYXJhbWV0ZXIgcGFnZSB3d
15     XJkZSBuaWNodCBnZWZlbnRlbiZndDtkZGSuJdMVMztf6l3zTuGzaHiOIY3V78sAxUv7BdgjxfxglA==" />
16 </div>
17
18 <div class="aspNetHidden">
19     <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
20     value="/wEWAwK7zLPwDQKmfzfcOApnDinAX3AheZsudEqRNOS100bBPXivBrb6kLOW4B4g5HmcOKg==" />
21 </div>
22     <div>
23         <span id="Label1">Eingabe</span>&nbsp;
24         <input name="_tbInput" type="text" id="_tbInput" />
25         <br />
26         <br />
27         <span id="_lblOutput">&lt;Parameter page wurde nicht gefunden&gt;</span>
28         <br />
29         <br />
30         <input type="submit" name="_btnDoSomething" value="Los geht's!"
31         id="_btnDoSomething" />
32     </div>
33 </form>
34 </body>
```

Die Viewstate wird standardmässig codiert und mittels einem Hidden Field zwischen Server und Client resp. umgekehrt übertragen

- Cookies sind kleine Datenobjekte, die clientseitig gespeichert werden. Um dies zu erreichen, werden die Cookies serverseitig zur Antwort an den Client hinzugefügt und somit automatisch an den Clientbrowser übermittelt, der diese nach Erhalt aber nur bei entsprechender Konfiguration speichert.
- Beim Auslösen einer Anforderung auf Seite des Browsers, werden automatisch alle auf dem Client vorhandenen Cookies zu der betreffenden Webseite zusammengesucht und mit der Anforderung an den Server gesendet.
- Cookies sind bei den meisten Browsern auf eine Grösse von 4096 Bytes beschränkt.
- Browser können die Menge von Cookie pro Web Seite limitieren. Bei vielen Browser ist ein Limit von 20 Cookies pro Seite definiert.
- Sie sollten nie davon ausgehen, dass ein eigenes an den Server übermitteltes Cookie korrekte Daten beinhaltet. Daraus folgt, dass Sie Cookiedaten unbedingt auf ihre Richtigkeit überprüfen müssen.
- Die Verwendung von Cookie kann auf Seite des Client durch entsprechende Browserkonfiguration verboten werden.
- Cookie werden zwischen dem Client und dem Server als Text übertragen. Folglich sollten keine vertraulichen Informationen in einem Cookie unverschlüsselt bleiben.

- Cookies werden mittels der Eigenschaft Cookies des Responseobjekts verwaltet

```
protected void _btnCreateCookie_Click(object sender, EventArgs e) {  
    // Cookie direkt erstellen (7 Tage gültig)  
    Response.Cookies["UserNameDirect"].Value = this.User.Identity.Name;  
    Response.Cookies["UserNameDirect"].Expires = DateTime.Now.AddDays(7);  
  
    // Cookie indirekt erstellen (3 Tage gültig)  
    HttpCookie objCookie = new HttpCookie("UserNameIndirect");  
    objCookie.Value = this.User.Identity.Name;  
    objCookie.Expires = DateTime.Now.AddDays(3);  
    Response.Cookies.Add(objCookie);  
  
    // Cookie mit mehreren Schlüsseln (direkt)  
    Response.Cookies["UserInfoDirect"]["Name"] = this.User.Identity.Name;  
    Response.Cookies["UserInfoDirect"]["LastVisit"] = DateTime.Now.ToString("u");  
    Response.Cookies["UserInfoDirect"]["Language"] = this.Request.UserLanguages[0];  
    Response.Cookies["UserInfoDirect"].Expires = DateTime.Now.AddDays(7);  
  
    // Cookie mit mehreren Schlüsseln (indirekt)  
    objCookie = new HttpCookie("UserInfoIndirect");  
    objCookie.Values["Name"] = this.User.Identity.Name;  
    objCookie.Values["LastVisit"] = DateTime.Now.ToString("u");  
    objCookie.Values["Language"] = this.Request.UserLanguages[0];  
    objCookie.Expires = DateTime.Now.AddDays(3);  
    Response.Cookies.Add(objCookie);  
    Response.Redirect("Cookie.aspx");  
}
```

- Eine ASP.NET Anwendung ist die Summe aller Dateien, Seiten, Handler und Modulen, die im Rahmen eines virtuellen Verzeichnisses und seinen Unterverzeichnissen angesprochen werden können
- Wenn ein Benutzer eine URL in dem virtuellen Verzeichnis anspricht, wird ein einzelnes und für diese Anwendung global gültiges Anwendungsobjekt hergestellt
- Mit Hilfe dieses Objektes können verschiedene Anforderungen, untereinander Daten austauschen, ohne dass diese aufwändig in eine Datenbank oder in ein File geschrieben werden müssen.
- Die Daten werden in das Anwendungsobjekt nach dem Schlüssel / Wertprinzip eingefügt oder von diesem gelesen
- Das Anwendungsobjekt verwaltet diese Informationen im Speicher des Servers. Das hat selbstverständlich die Konsequenz, dass allfällig gehaltene Daten ohne weitergehende Maßnahmen zur Unterstützung der Persistenz bei einem Neustart des Servers verloren gehen.
- Die dazu notwendige Funktionalität finden wir in der Klasse *HttpApplicationState*.
- **Der Zugriff auf das Anwendungsobjekt ist threadsicher**
- **Verwechseln Sie nicht das Anwendungsobjekt (Klasse *HttpApplication*) und das Objekt für die Zustandsverwaltung auf Anwendungsstufe (Klasse *HttpApplicationState*)**

- Anwendungszustände werden mittels der Eigenschaft Application der Seite verwaltet

```
protected void Page_Load(object sender, EventArgs e) {  
    // Anzahl Zugriffe der Sitzung erhöhen  
    if (Application[Session.SessionID] == null)  
    {  
        this.Application[Session.SessionID] = 1;  
    } else {  
        this.Application[Session.SessionID] =  
            ((int)this.Application[Session.SessionID]) + 1;  
    }  
    // Prüfen, ob bereits eine Information definiert wurde  
    if (Application["AspObjectApplication"] == null) {  
        // Anwendungsobjekt sperren  
        this.Application.Lock();  
        // Information definieren  
        this.Application.Set("AspObjectApplication", "Objekt von Session '" +  
            this.Session.SessionID + "'");  
        // Anwendungsobjekt entsperren  
        this.Application.UnLock();  
        // Aktion ausgeben  
        _lblDone.Text = "Sitzungs-ID in das Anwendungsobjekt übertragen." +  
            " Starten Sie eine zweiten Browser um das Resultat zu sehen.";  
    } else {  
        // Daten auslesen  
        _lblDone.Text = Application["AspObjectApplication"].ToString();  
        _lblCount.Text = string.Format( "Anzahl Zugriffe der Sitzung '{0}' ist: {1}",  
            Session.SessionID, Application[Session.SessionID].ToString());  
        _lblSessions.Text = "Anzahl Sitzungen: " + (Application.Count - 1);  
    }  
}
```

- Der sitzungsbasierte Zustand (Sessionstate) erlaubt die Verwaltung von Daten pro Sitzung. Das impliziert, dass vom gleichen Client mit demselben Konto und zwei verschiedenen Browserinstanzen, auf dem Server zwei verschiedene Sitzungen kontrolliert werden.
- Die Zustandsverwaltung für Sitzungen basiert, wie die Verwaltung auf Anwendungsstufe nach dem Schlüssel / Wertprinzip.
- Der Zugriff auf die Daten erfolgt über ein Objekt der Klasse *Http-SessionState*.
- Dieses Objekt ist im eigenen Objekt der Seite, bedingt durch die Vererbung von der Klasse *Page*, verfügbar.
- Die gespeicherten Daten sind nur solange vorhanden, bis die nicht mehr benützte Sitzung ein Timeout erfährt. Bei einer normalen Installation mit dem IIS 7.0 sind das gerade mal 20 Minuten.
- Ebenfalls hat das Neustarten der Maschine oder auch nur des Internetserverns die gleiche Wirkung.

- Sessionbasierte zustände werden mittels der Eigenschaft Session der Seite verwaltet

```
protected void _btnRefresh_Click(object sender, EventArgs e)
{
    // Sequenznummer erhöhen
    Session["SequenceNumber"] = ((int)Session["SequenceNumber"]) + 1;

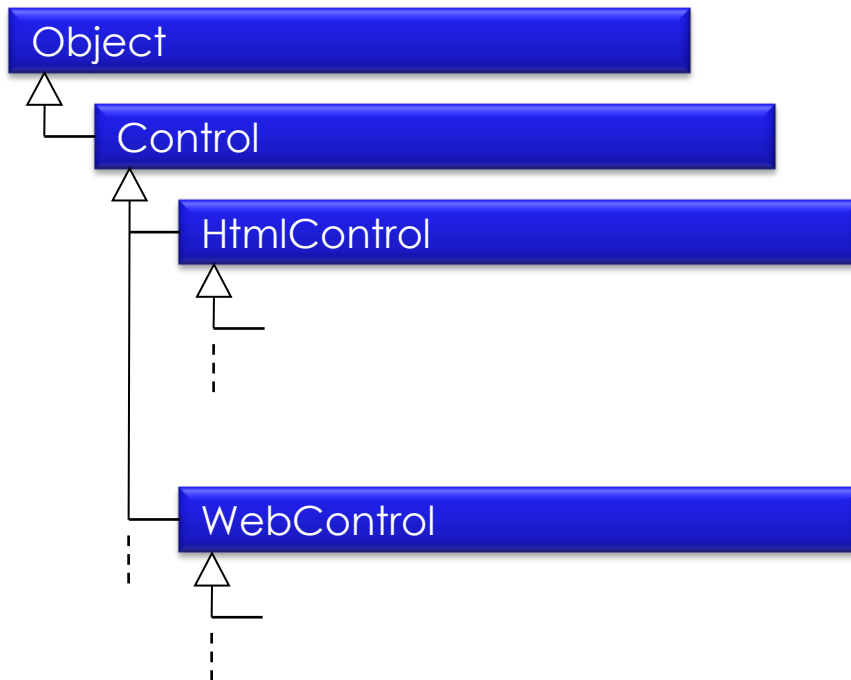
    // Einen neuen Eintrag erstellen
    Session[string.Format("Item_{0:000}",
        (int)Session["SequenceNumber"])] = DateTime.Now.ToString("HH:mm:ss.fff");

    // Liste ausgeben
    RefreshView();
}
```

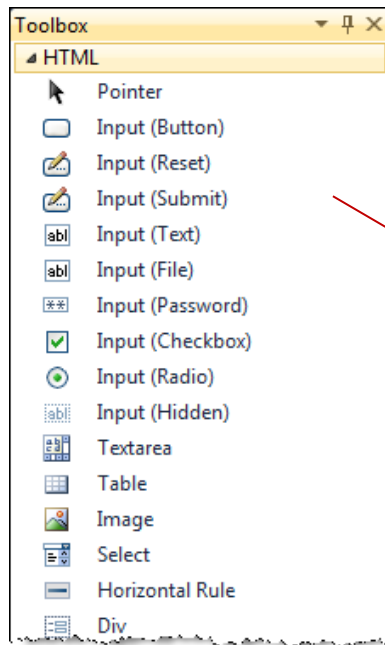
- Das Profiling ist die neuste Technik in ASP.NET um Zustandsinformationen zu speichern. Die Technik erlaubt die datenbankgestützte und userbezogene Ablage von Informationen beliebigen Typs und Größe.
- Profiling setzt eine Datenbank voraus (oder Sie machen eine eigene Implementierung).
- Mit Profiling können beliebige Datentypen die serialisierbar sind gespeichert werden.
- Profiling kann für anonyme und authentifizierte Benutzer verwendet werden.
- Profiling bildet die Basis für den Einsatz von Webpart- und Portal-Erweiterungen von .NET.
- Durch die serverseitige Implementierung ist ein hoher Grad an Sicherheit vorhanden.
- Zentrale Lösungen, insbesondere datenbankgestützte Lösungen bedürfen einer gewissen Wartung und Supportstruktur.
- Für die Nutzung des Profilings sind folgende Schritte notwendig:
 - Definition der Profildaten in der Datei *web.config* vornehmen.
 - Definition des Providers für Profile in der Datei *web.config* vornehmen.
 - Definition des Datenbankzugriffes in der Datei *web.config* vornehmen.
 - Einrichtung der Datenbank mit ***aspnet_regsql.exe***
 - Erstellen des Codes der die entsprechenden Daten nutzt
 - Testen der Zugriffe

- Um die Dialoge mit Inhalten zu füllen, brauchen wir vorgefertigte Steuerelemente. Im Gegensatz zu anderen Windows Techniken, müssen die Elemente der ASP.NET-Technik immer wieder umgeformt werden, denn die Visualisierung derselben geschieht ausnahmslos in XHTML.
- ASP.NET ist dadurch gezwungen, die Standards des Internets einzuhalten, und muss zu diesem Zweck auch die verrücktesten Steuerelemente in XHTML ausdrücken.
- Die zur Verfügung gestellten Steuerelemente werden in der Klassenbibliothek in zwei verschiedene Gruppen unterteilt:
 - HTML-Steuerelemente sind Klassen, die den HTML Elementen gemäß W3C entsprechen. Diese Elemente können serverseitig programmatisch angesprochen werden.
 - Web-Steuerelemente sind Klassen, die eine reine Funktionalität zur Verfügung stellen oder als HTML Element umgesetzt werden. In jedem Fall benutzen sie ein Objektmodell. Viele der Web-Steuerelemente sind den Steuerelementen, wie Sie dies aus der Forms-Technik bereits kennen, sehr ähnlich.
- Die Web-Steuerelemente haben ein reiches Objektmodell und bieten mehr Möglichkeiten. Tendenziell hat das zur Folge, dass sowieso diese Serie an Steuerelementen eingesetzt werden müssen, um gewisse Funktionen erreichen zu können. Auf der anderen Seite sind die HTML-Steuerelemente sehr schlank und einfach in der Umsetzung, also für einfache Aufgaben durchaus geeignet.

- Die Steuerelemente in ASP.NET erben entweder von der Basisklasse HtmlControl (Standard HTML-Elemente nach W3C) oder von der Klasse WebControl (eigentliche ASP.NET Elemente).
- Die gemeinsame Basis bildet die Klasse Control, die für beide Elementare Eigenschaften und Funktionalitäten definiert



- HTML Steuerelemente haben von sich aus keine direkte Auswirkung in das Programmiermodell
- Sie dienen somit zur Ordnung, einfachen Visualisierung oder als Elemente für die clientseitige Codierung mit Scriptsprachen
- Die Eigenschaften der eingefügten Objekte können wahlweise im Designercode direkt oder im Eigenschaftsdialog editiert werden



Die Gruppe HTML in der Toolbox definiert die HTML Steuerelemente nach W3C.

- Durch die Markierung mit dem Attribut `runat="server"` können die HTML Steuerelemente in die serverseitige Verarbeitung eingebunden werden, und im Code Behind direkt angesprochen werden
- Die verwendeten Klassen der HTML Steuerelemente sind jeweils mit dem String `Html` präfixiert

Client Objects & Events (No Events)

```
9 <body>
10 <form id="form1" runat="server">
11 <div>
12 <table style="width: 100%;">
13 <tr>
14 <td>Name</td>
15 <td><input id="_tbName" type="text" value="<Wert eingaben>" /></td>
16 </tr>
17 <tr>
18 <td>Vorname</td>
19 <td><input id="_tbFirstName" type="text" runat="server" /></td>
20 </tr>
21 </table>
22 </div>
23 </form>
24 </body>
```

Die Elemente verwenden die Attribute gemäss W3C Standardisierung.

Das Attribut `runat` bildet eine Ausnahme und ist ASP.NET spezifisch für die Steuerung des Editors. Durch diese Definition wird das Element im Code Behind codierbar.

100 %

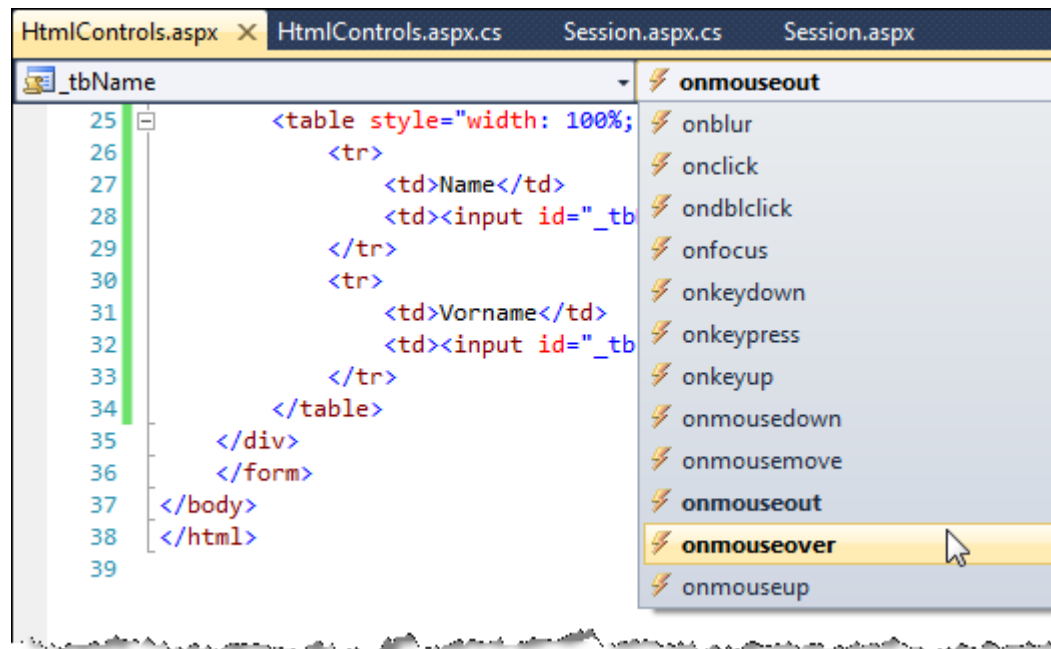
Name

Vorname

input#_tbFirstName

runat
size
style
tabindex
title
type
value
visible
xml:lang

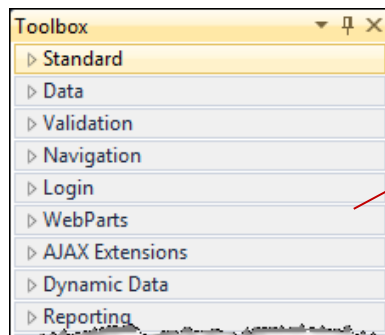
- Visual Studio unterstützt die clientseitige Programmierung von Scripts für HTML-Elemente
- Ein clientseitiger Eventhandler wird wie folgt eingebracht
 - Im Editor links oben das Element auswählen, für das eine clientseitige Scriptmethode eingebracht werden soll
 - Im Editor rechts oben das Ereignis auswählen für das eine Scriptmethode eingebracht werden soll



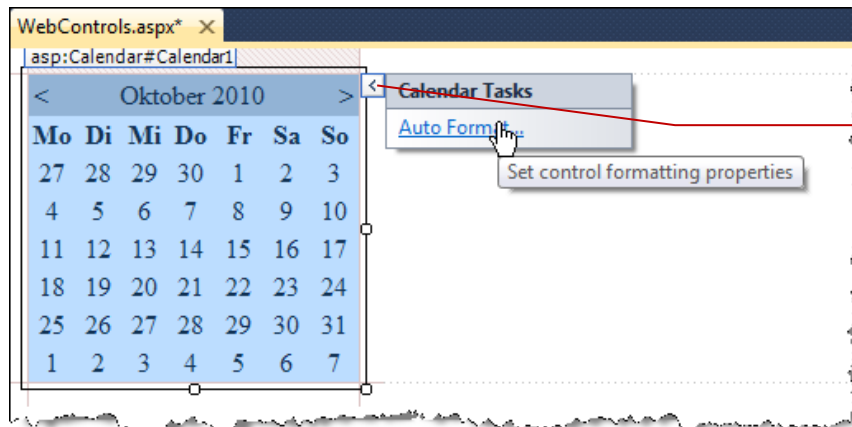
- Zum Debuggen von JavaScript muss das Debuggen in den Einstellungen des Browser eingeschaltet werden.

```
6 <!-- runat="server">
7 <title></title>
8 <script language="javascript" type="text/javascript">
9 // 
10 function _tbName_onmouseover() {
11     var objElement = document.getElementById("_tbName");
12     objElement.style.backgroundColor = 0xFF7777;
13 }
14 function _tbName_onmouseout() {
15     var objElement = document.getElementById("_tbName");
16     objElement.style.backgroundColor = 0xFFFFFF;
17 }
18
19 // ]]&gt;
20 &lt;/script&gt;
21 &lt;/head&gt;
22 &lt;body&gt;
23 &lt;form id="form1" runat="server"&gt;
24 &lt;div&gt;
25 &lt;table style="width: 100%;"&gt;
26 &lt;tr&gt;
27 &lt;td&gt;Name&lt;/td&gt;
28 &lt;td&gt;&lt;input id="_tbName" type="text" value="&lt;Wert eingaben&gt;"
29     onmouseover="return _tbName_onmouseover()"
30     onmouseout="return _tbName_onmouseout()" /&gt;&lt;/td&gt;
31 &lt;/tr&gt;
32</pre></div><div data-bbox="940 954 967 975" data-label="Page-Footer"><p>58</p></div>
```

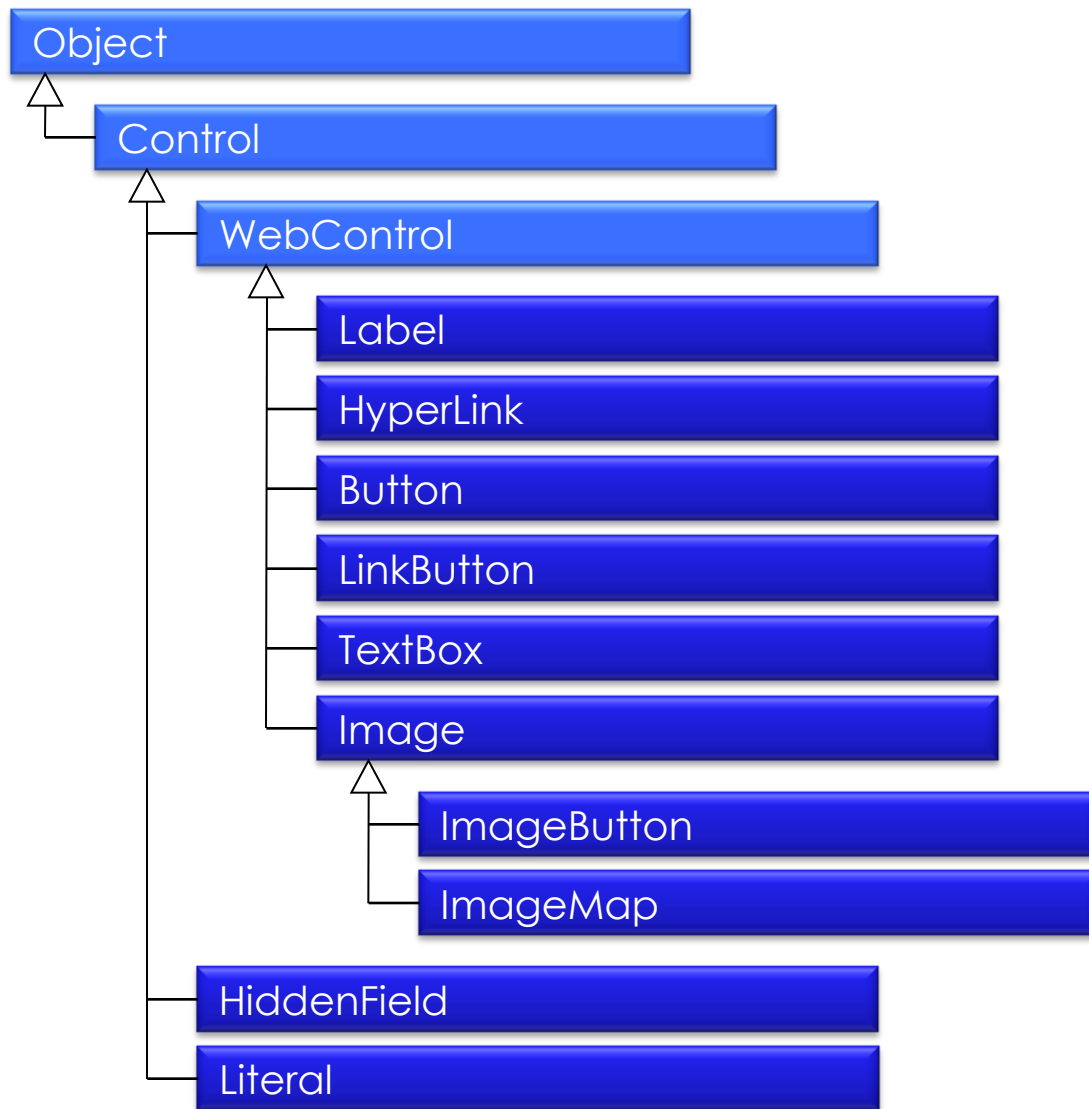
- Web Steuerelemente partizipieren automatisch am Objektmodell von ASP.NET. Sie sind als standardmässig mit dem Attribut `runat="server"` definiert.
- Web-Steuerelemente verwenden einige Eigenschaften gleich wie deren verwandte Elemente in der Forms-Technik.
- Viele Web-Steuerelemente verfügen in der Designansicht im Editor über Smarttags, mit denen oft benötigte Einstellungen vereinfacht vorgenommen werden können.

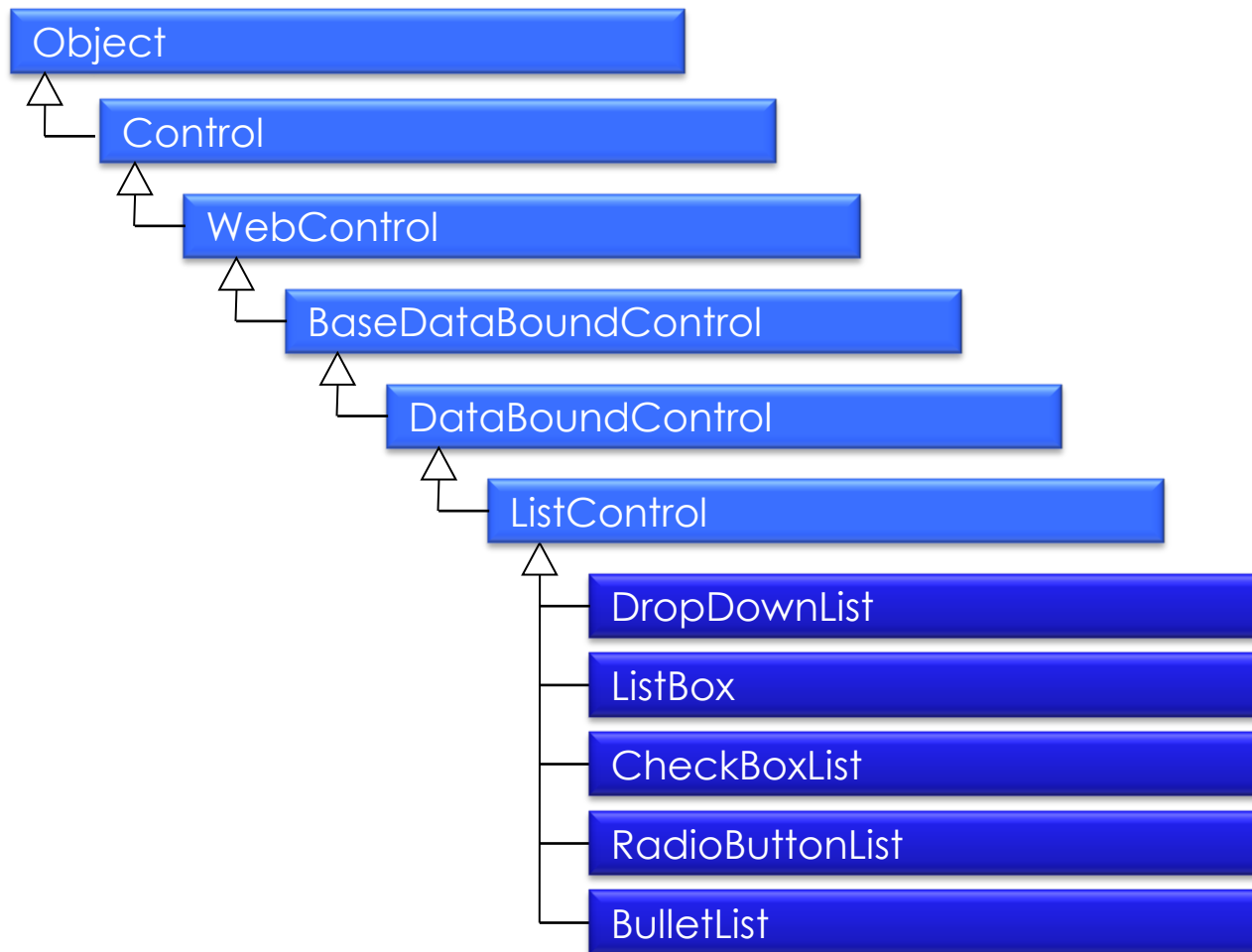


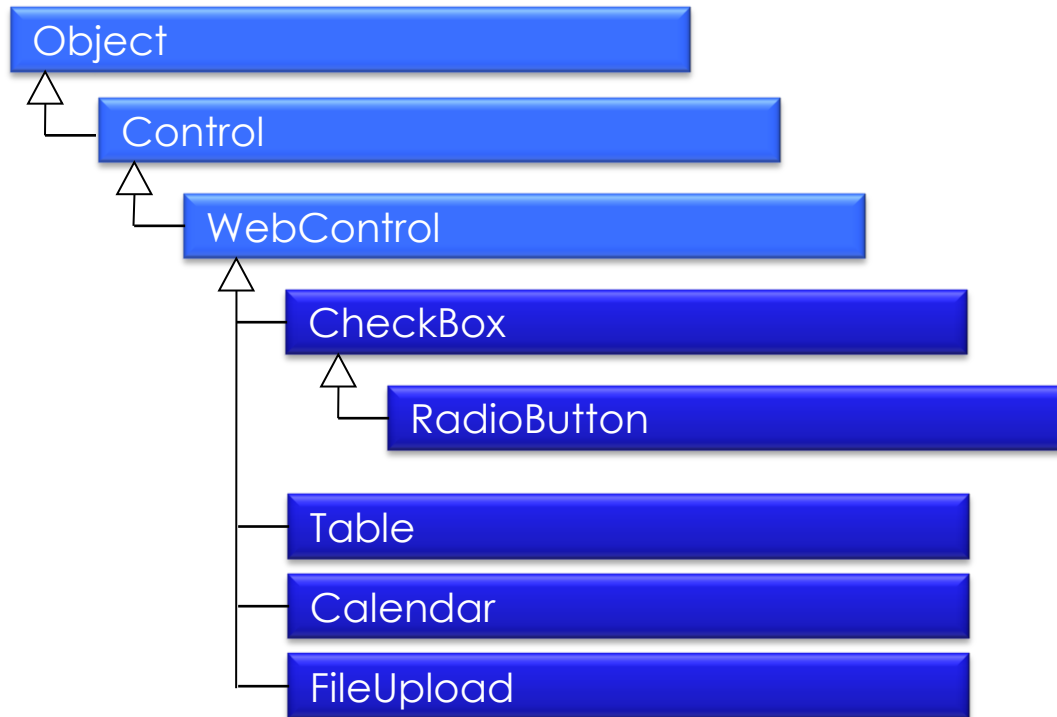
Die Anzahl der Web-Steuerelemente ist einiges grösser als diejenige der HTML-Elemente. Entsprechend sind die Web-Steuerelemente in der Toolbox gruppiert.

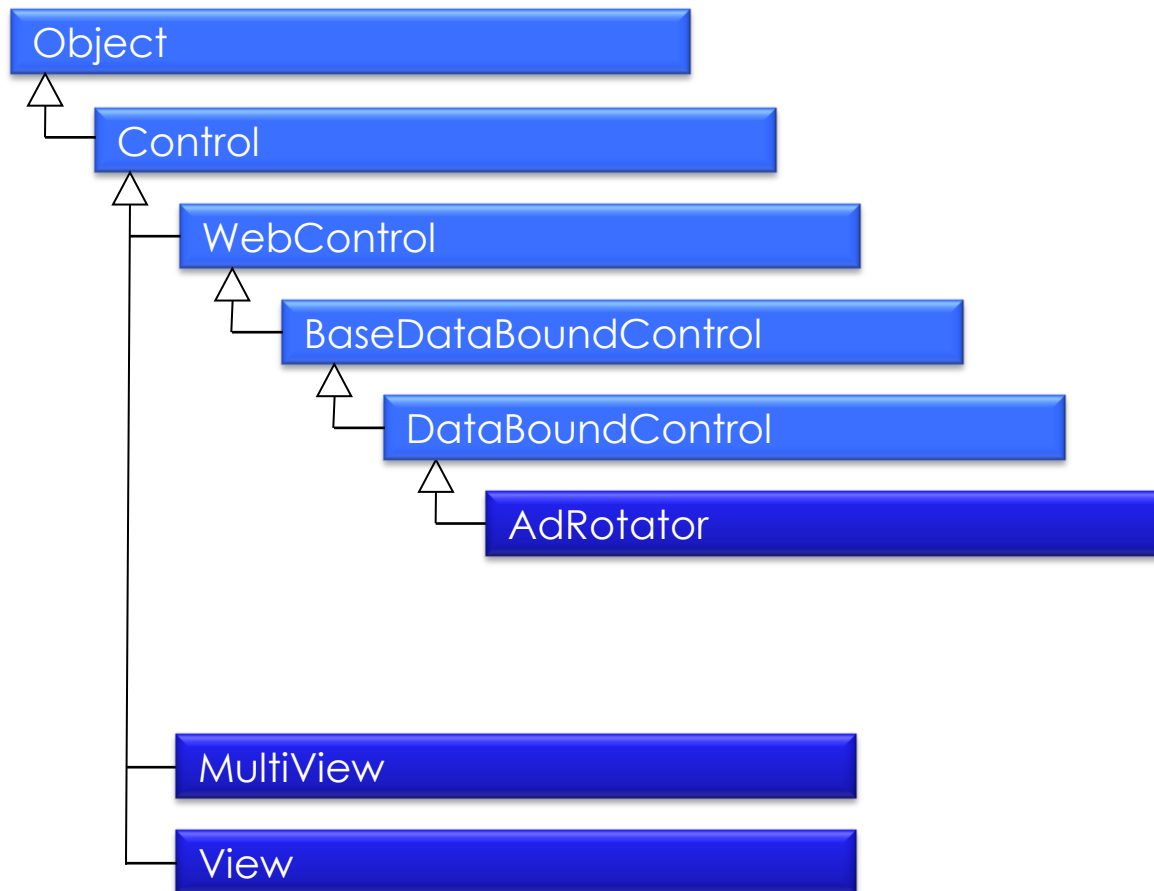


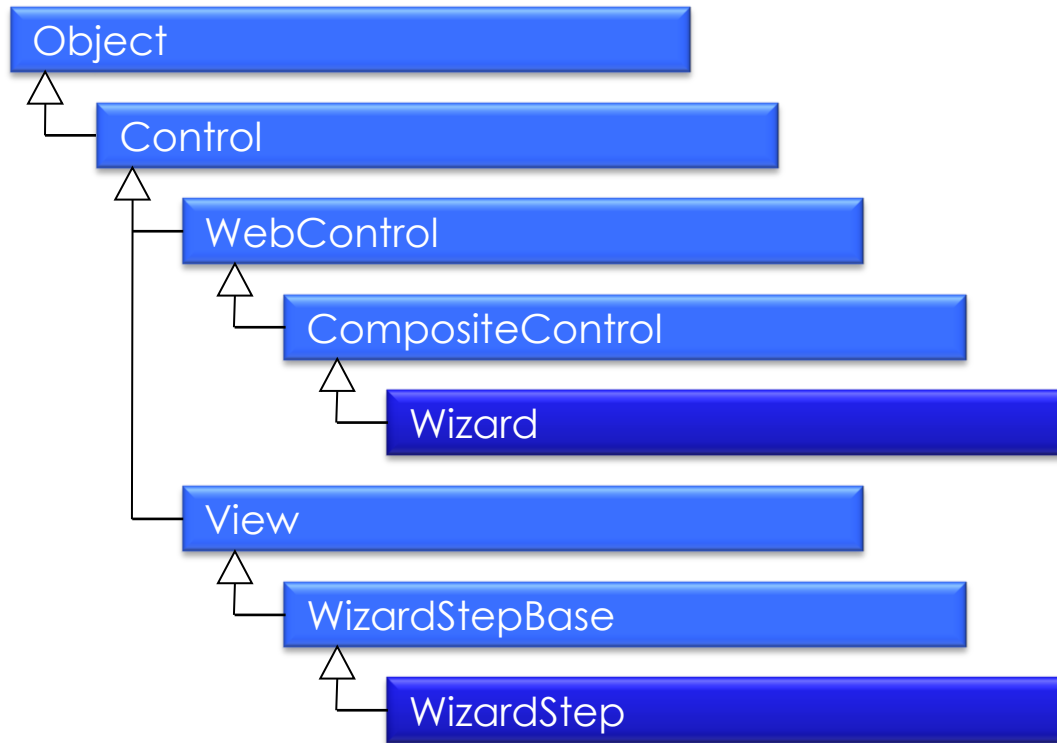
Mit dem Smarttag steht für die vom Element abhängigen und wichtigsten Einstellungen eine Alternative zur Bedienung mittels dem Eigenschaftsdialog zur Verfügung.

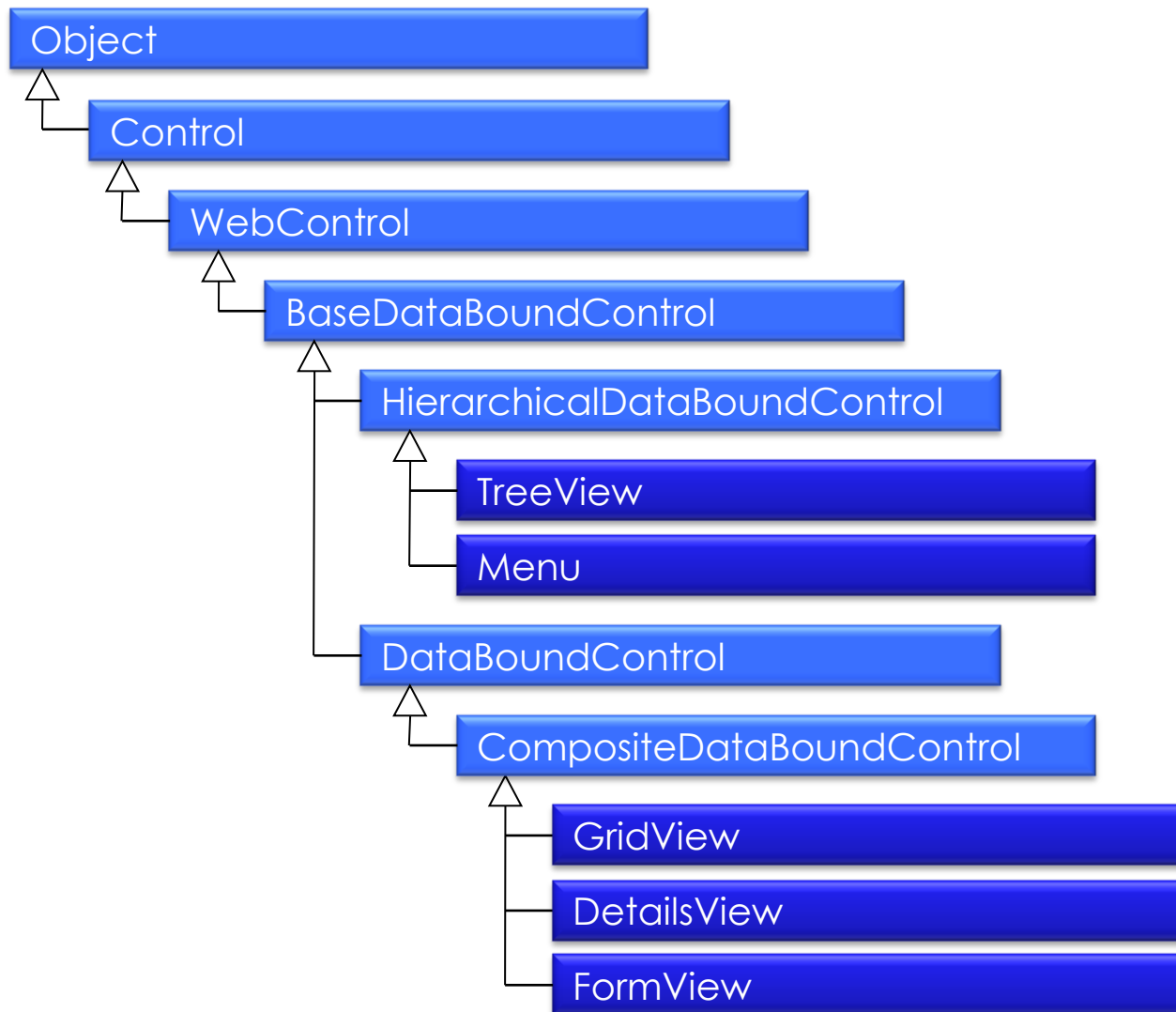


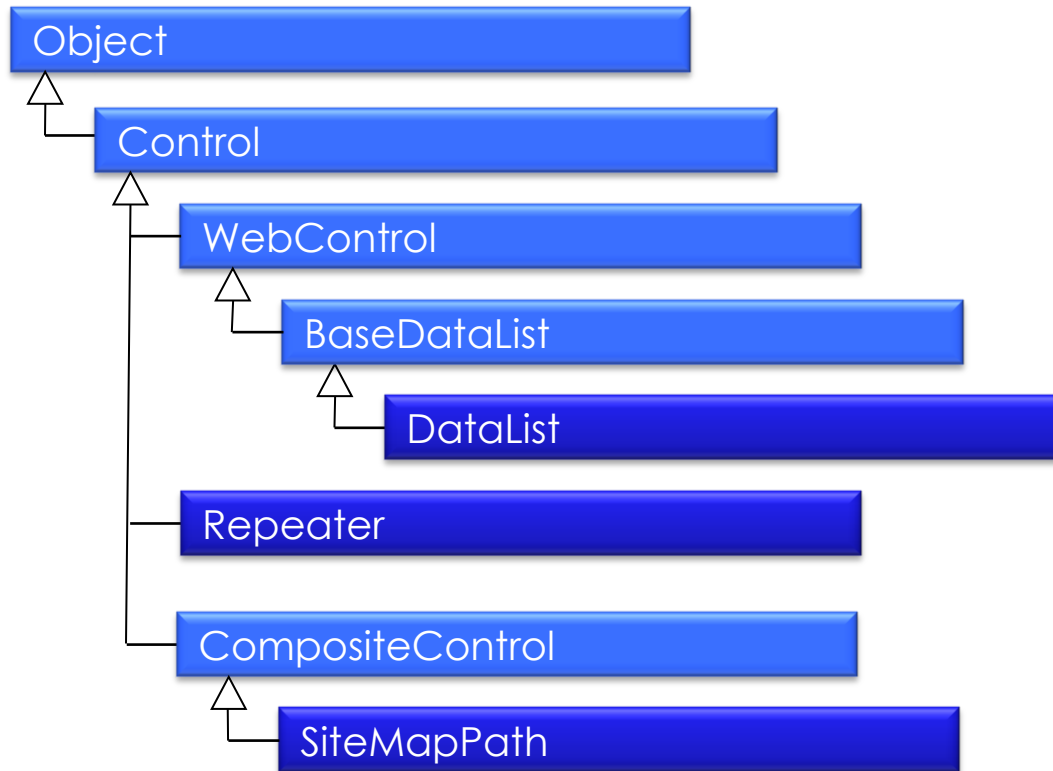


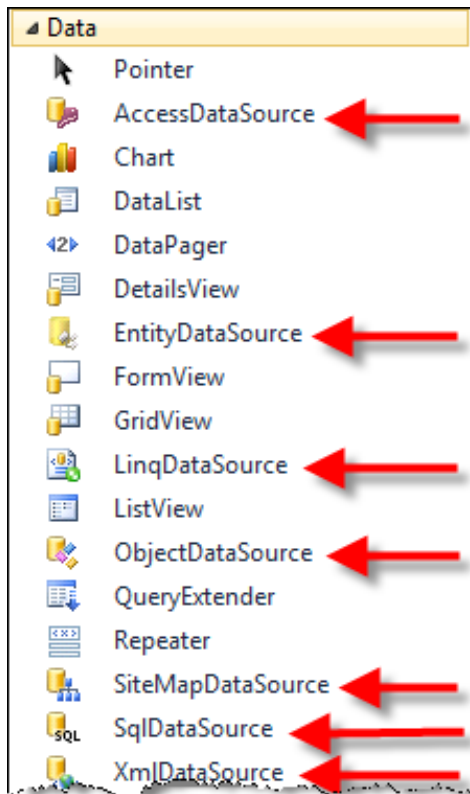




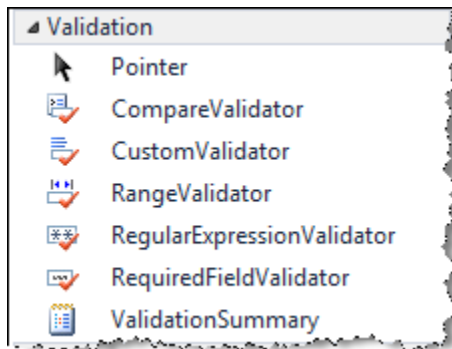








- Datenquellen sind Steuerelemente ohne Repräsentation in der Benutzerschnittstelle.
- Sie werden im Entwurfsmodus eingeblendet und können dort über die Eigenschaften oder über die Smarttags konfiguriert werden.
- Getreu dem Konzept der Kapselung aus der objektorientierten Technik, greifen die Steuerelemente von .NET zunehmend nicht mehr selber auf die externen Datenressourcen zu, sondern delegieren diese Aufgabe Datenquellen.
- Um den verschiedenen Organisationsformen der Datenquellen besser zu entsprechen und trotzdem die vereinheitlichten Zugriffe auf die Quellen zu ermöglichen, wird wiederum das bereits bewährte Konzept von Vererbungen und Implementierung von Interfaces zurückgegriffen.



- Mit Validatoren können Inhalte von Steuerelementen kontrolliert werden und automatisch vorbereitete Fehlermeldungen angezeigt werden
- Je nach vorzunehmender Prüfung können fertige Validatoren genutzt werden, oder für eigene Regelprüfungen der RegEx -oder CustomValidator eingesetzt werden (siehe links)
- Mit der Klasse ValidationSummary können die Fehlermeldungen diverser Validatoren an einer Stelle gezeigt werden
- Validatoren können gruppiert werden, so dass verschiedene Callbacks verschiedene Validierungen durchführen

- Die direkte Validierung erfordert lediglich die Definition der Steuerelemente
- Die wohl wichtigste Angabe besteht darin, dass dem Validator-Steuerelement definiert werden muss, welches Steuerelement validiert werden soll

DirectValidation.aspx X

Client Objects & Events (No Events)

```
39 </td>
40 <asp:TextBox ID="_tbEmail" runat="server"></asp:TextBox>
41 </td>
42 <td style="margin-left: 40px">
43 <asp:RequiredFieldValidator ID="_valEmailMandatory" runat="server"
44 ControlToValidate="_tbEmail" Display="Dynamic" ForeColor="Red">Die Emailadresse muss angegeben
45 <asp:RegularExpressionValidator ID="_valEmail" runat="server"
46 ControlToValidate="_tbEmail"
47 ValidationExpression="\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*"
48 Display="Dynamic" ForeColor="Red">Die Emailadresse ist nicht korrekt definiert.</asp:Regular
49 </td>
50 </tr>
```

100 %

Name Der Name muss definiert werden.

Vorname

Email Die Emailadresse muss angegeben werden. Die Emailadresse ist nicht korrekt definiert.

OK

Definiert, dass die Position am Standort dynamisch berechnet werden soll.

Text der im Fehlerfall angezeigt werden soll.

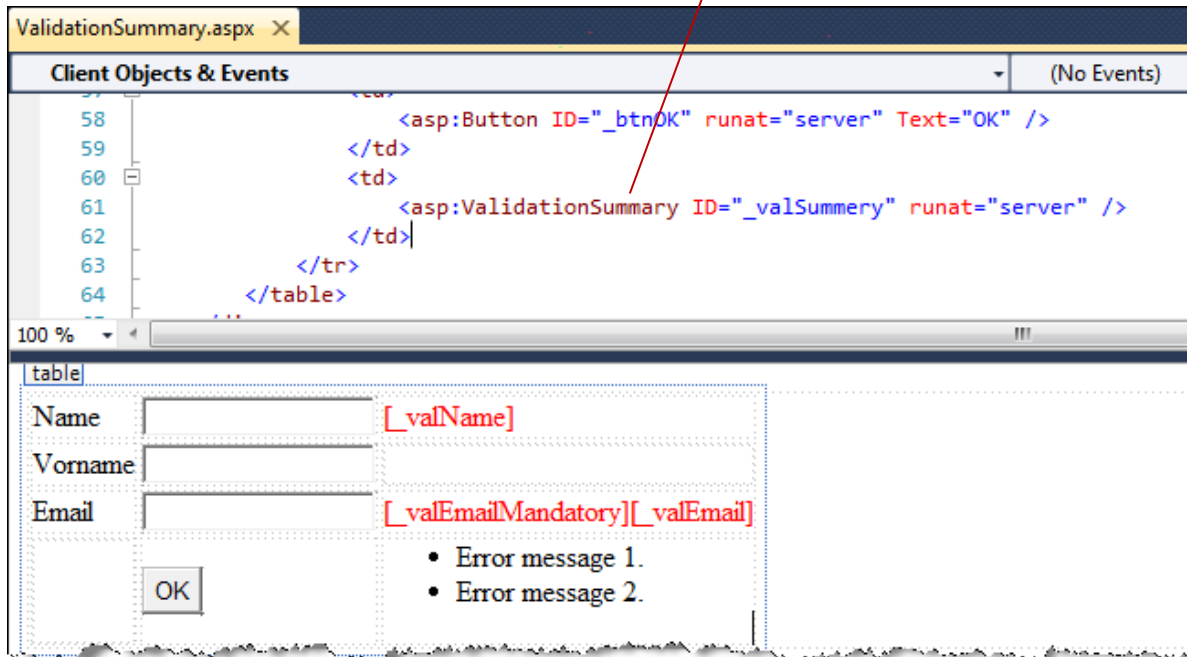
Steuerelement, das kontrolliert werden soll.

Validierungsgruppe um verschiedene Validierungen auf der gleichen Seite zu unterscheiden (Eine Gruppe ist durch einen Namen definiert).

Properties Window:

Property	Value
Display	Dynamic
ErrorMessage	
Font	
ForeColor	Red
Text	Die Emailadresse ist nicht ko
Behavior	
ClientIDMode	Inherit
ControlToValidate	_tbEmail
EnableClientScript	True
Enabled	True
EnableTheming	True
EnableViewState	True
SetFocusOnError	False
SkinID	
ToolTip	
ValidationExpression	\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*
ValidationGroup	
ViewStateMode	Inherit

- Das Summary sammelt die Texte automatisch
- Sehen Sie dazu genügend Platz vor
- Die einzelnen Validatoren müssen anders konfiguriert werden



Das Steuerelement ValidationSummary

Definition eines Validators
(nicht des Summarys):

Display	None
ErrorMessage	Die Emailadresse muss ange...
Font	
ForeColor	Red
Text	
Behavior	
ClientIDMode	Inherit
ControlToValidate	_tbEmail
EnableClientScript	True
Enabled	True
EnableTheming	True

Definiert, dass am Ort des Validators kein Text angezeigt wird.

Text der im ValidationSummary angezeigt werden soll.

Kein Text für die lokale anzeige.

- AJAX erlaubt einer ASP.NET-Anwendung asynchrone partielle Veränderung einer Seite. Dadurch wirkt die Benutzeroberfläche ruhiger.
- Es werden verschiedene clientseitige Möglichkeiten zur Nutzung von AJAX unterstützt:
 - Auslösen eines Callbacks durch eine Benutzerinteraktion.
 - Abbrechen eines ausgelösten asynchronen Callbacks zum Server
 - Auslösen eines Callbacks durch einen Timer.
- Die Nutzung von AJAX basiert auf folgenden Klassen:
 - ScriptManager
Der Script Manager ist zuständig für die Erstellung von Scripts, das Rendering von Scripts, die Erzeugung von Proxy-Objekten bei Zugriff auf Web Services und weitere Funktionalitäten in den Zusammenhang.
 - UpdatePanel
Erstellt einen Bereich in der Webseite der für partielle Veränderung vorgesehen ist.
 - ProgressPanel
Stellt die Möglichkeit eines visuellen Feedbacks während dem Callback zur Verfügung.
 - Timer
Ermöglicht das zyklische, automatische Aufrufen von Callbacks.

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="AjaxCompare.aspx.cs" Inherits="wlab_ch.Samples.WebProject.AjaxCompare" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AJAX Vergleich mit und ohne Ajax</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="_lblUpdateWithoutAjax" runat="server" Text="???"></asp:Label>
            <br />
            <asp:Button ID="_btnUpdateWithoutAjax"
                runat="server"
                Text="Update ohne AJAX"
                OnClick="_btnUpdateWithoutAjax_Click" />
            <br /><br /><br />

            <asp:ScriptManager ID="_objScriptManager" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="_pnlUpdateWithAjax" runat="server">
                <ContentTemplate>
                    <asp:Label ID="_lblUpdateWithAjax" runat="server" Text="???"></asp:Label>
                    <br />
                    <asp:Button ID="_btnUpdateWithAjax"
                        runat="server"
                        Text="Update mit AJAX"
                        OnClick="_btnUpdateWithAjax_Click" />
                </ContentTemplate>
            </asp:UpdatePanel>
        </div>
    </form>
</body>
</html>
```

Löst einen normalen Callback aus.

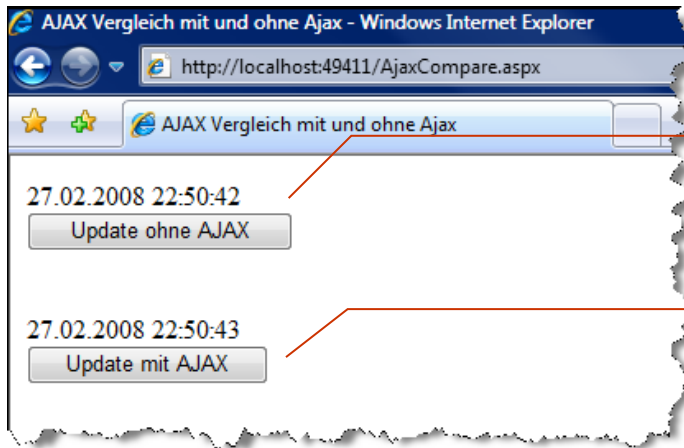
Das Element ScriptManager ist zur Laufzeit nicht sichtbar.

Das Element UpdatePanel bildet den Container aller Elemente die partiell verändert werden können. Eine Seite kann mehrere Update-Panel enthalten.

Code Behind

```
namespace wlab_ch.Samples.WebProject {  
  
    public partial class AjaxCompare : System.Web.UI.Page {  
  
        protected void Page_Load(object sender, EventArgs e) {  
        }  
  
        protected void _btnUpdateWithoutAjax_Click(object sender, EventArgs e) {  
            _lblUpdateWithoutAjax.Text = DateTime.Now.ToString("G");  
        }  
  
        protected void _btnUpdateWithAjax_Click(object sender, EventArgs e) {  
            _lblUpdateWithAjax.Text = DateTime.Now.ToString("G");  
        }  
    }  
}
```

Resultat



Kein partieller Update - Die Seite wird gelöscht und wieder neu aufgebaut -> Wirkt unruhig.

Partieller Update - Kein "Refresh" der ganzen Seite.

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ScriptManager ID="_objScriptManager" runat="server">
      </asp:ScriptManager>

      <script type="text/javascript">
        var objPageRequestManager = Sys.WebForms.PageRequestManager.get
        objPageRequestManager.add_initializeRequest(InitializeRequest);
        function InitializeRequest(sender, args) {
          if (objPageRequestManager.get_isInAsyncPostBack()) {
            args.set_cancel(true);
          }
        }
        function AbortPostBack() {
          if (objPageRequestManager.get_isInAsyncPostBack()) {
            objPageRequestManager.abortPostBack();
          }
        }
      </script>

      <asp:UpdatePanel ID="_objUpdatePanel" runat="server">
        <ContentTemplate>
          <asp:Label ID="_lblUpdatePanel" runat="server" Text="???"></asp:Label>
          <asp:Button ID="_btnUpdatePanel" runat="server" Text="Update" OnClick="_btnUpdatePanel_Click" />
          <asp:UpdateProgress ID="_objUpdateProgress" AssociatedUpdatePanelID="_objUpdatePanel" runat="server">
            <ProgressTemplate>
              <input type="button" value="Abbrechen" onclick="AbortPostBack()" />
            </ProgressTemplate>
          </asp:UpdateProgress>
        </ContentTemplate>
      </asp:UpdatePanel>
    </div>
  </form>
</body>
```

Aus Platzgründen ist nur der HTML- Body abgebildet.

ScriptManager

Mit dem PageRequestManager können Begin und End Events des Callbacks verarbeitet werden

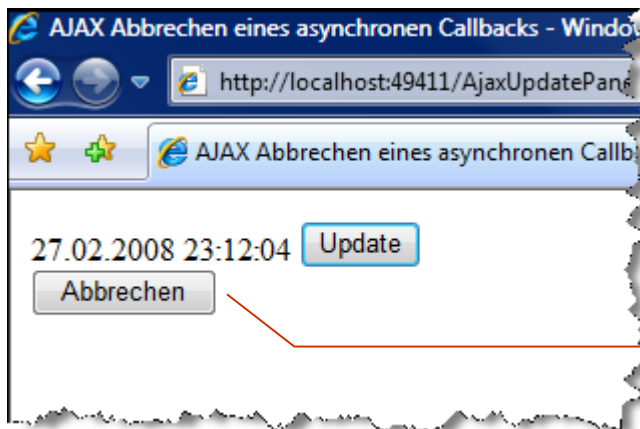
Notwendiges JavaScript zum Abbrechen des Callback

Element UpdateProgress. Der Inhalt dieses Elements wird nur während dem eigentlichen Callback angezeigt.

Code Behind

```
namespace wlab_ch.Samples.WebProject {  
  
    public partial class AjaxUpdatePanel : System.Web.UI.Page {  
  
        protected void Page_Load(object sender, EventArgs e) {  
        }  
  
        protected void _btnUpdatePanel_Click(object sender, EventArgs e) {  
  
            // Simulation einer länger dauernden Operation  
            System.Threading.Thread.Sleep(5000);  
            _lblUpdatePanel.Text = DateTime.Now.ToString("G");  
        }  
    }  
}
```

Resultat



Der Knopf Abbrechen ist nur während dem Callback sichtbar.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="AjaxTimerPanel.aspx.cs"
    Inherits="wlab_ch.Samples.WebProject.AjaxTimerPanel" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>AJAX Zyklische Callbacks</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            </div>
            <asp:ScriptManager ID="_objScriptManager" runat="server"/>
            <asp:Timer ID="_timerMain"
                OnTick="_timerMain_Tick"
                runat="server"
                Interval="1000" />

            <asp:UpdatePanel ID="_pnlUpdate" runat="server">
                <Triggers>
                    <asp:AsyncPostBackTrigger ControlID="_timerMain" />
                </Triggers>
                <ContentTemplate>
                    <asp:Label ID="_lblUpdate" runat="server" Text="???"></asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>

        </form>
    </body>
</html>
```

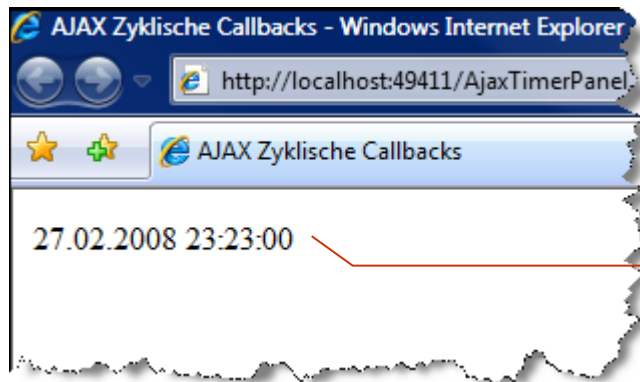
Timerobjekt. Die Zykluszeit wird durch die Eigenschaft Intervall in Millisekunden bestimmt.

In dieser Variante wird der Timer als Trigger zum Auslösen des Callback verwendet. Es ist auch möglich den Timer direkt im Updatepanel zu definieren.

Code Behind

```
namespace wlab_ch.Samples.WebProject {  
  
    public partial class AjaxTimerPanel : System.Web.UI.Page {  
  
        protected void Page_Load(object sender, EventArgs e) {  
        }  
  
        protected void _timerMain_Tick(object sender, EventArgs e) {  
            _lblUpdate.Text = DateTime.Now.ToString("G");  
        }  
    }  
}
```

Resultat



Die Zeit wird gemäss Zykluszeit angezeigt.

Achtung zu klein gewählte Zykluszeiten können erheblichen Verkehr im Netzwerk oder Belastungen des Servers zur Folge haben!!



Layout

- ASP.NET unterstützt verschiedene Design-Techniken
- Mit Design-Techniken meine ich in diesem Abschnitt die Unterstützung der grafischen Designs
- Konkret werden folgende Möglichkeiten unterstützt
 - Statische Einstellungen direkt auf den Steuerelementen (imperativ und deklarativ)
 - Verwendung von Cascaded Style Sheets (css) nach W3C
 - ASP.NET spezifische Technik genannt Theming
 - Unterstützungen mit JavaScript-Bibliotheken wie JQuery
 - MasterPage-Konzept
 - Kombinationen obenstehender Möglichkeiten
- Die im Projekt verwendeten Möglichkeiten sind vom Projekttyp abhängig
 - Web-Forms verwenden tendenziell Masterpages, Theming und css
 - MVC-Anwendungen Masterpages und css
- Achten Sie beim Design unbedingt darauf, dass die Browser Zoomfunktionen unterstützen (Testen Sie das unbedingt, um nicht böse Kundenreaktionen zu erzeugen). Seien Sie aber auch kompromissbereit → Kosten!!

- ASP.NET unterstützt die Verwendung von Stylesheets (.css-Dateien) nach dem W3C-Standard
- Stylesheets können mit Visual Studio erstellt und editiert werden
- Für die Verbindung von Web-Steuerelementen mit einer Stylesheet-Klasse verwenden wir das Attribut `CssClass`.
- Stylesheets werden in der Seite im HTML-Header mit folgender Anweisung registriert (die Registration kann mit Drag&Drop aus dem Projektmappen-Explorer erfolgen).

```
<head runat="server">  
    <link href="../../../Css/Normal.css" rel="stylesheet" type="text/css" />  
    ...
```

- Ziehen Sie die Definition in einem Stylesheet der lokalen Definition vor (einfachere Änderungen)
- Durch direkte Manipulation der Steuerelemente im Designtitor generieren Sie leicht lokale Stildefinitionen, die mühsam zu löschen sind

- Je näher eine Design-Definition am Steuerelement vorgenommen wird, desto grösser ist deren Priorität!

```
StyleSheetDemo.css x StyleSheetDemo.aspx*
1  body
2  {
3      font-family: Verdana;
4      color: #0000FF;
5  }
6
7  p
8  {
9      font-family: 'Comic Sans MS';
10     color: #FF0000;
11 }
12
13 .MyParagraph
14 {
15     color: #008080;
16 }
17
```

```
StyleSheetDemo.aspx* x StyleSheetDemo.css
Client Objects & Events
13 <div>
14     Kein expliziter Stil.</div>
15 <p>
16     Stil von Absatz</p>
17 <p class="MyParagraph">
18     Stil von Absatz mit css-Class</p>
19 <p style="color: Black">
20     Lokaler Stil definiert</p>
21 </div>
22 </form>
23 </body>
```

100 %

body

Kein expliziter Stil.

Stil von Absatz

Stil von Absatz mit css-Class

Lokaler Stil definiert

- ASP.NET verfügt neben der Anwendung von Cascading Stylesheets mit den so genannten Themen über eine weitere Möglichkeit der Gestaltung von Benutzeroberflächen
- Die Themen Technik ist speziell für die optische Gestaltung von Web-Steuerelementen ausgelegt und arbeitet nahtlos mit der Technik von Cascading Stylesheets zusammen
- Der Unterschied zwischen Themen und Cascaded Stylesheets besteht darin, dass mit Themen die erweiterten Attribute der Web-Controls direkt definiert werden können und je nach Steuerelement auch weitergehende Daten definiert werden können (zum Beispiel die verwendeten Bilder in einem Treeview-Steuerelement).
- In einer Anwendung ist es möglich, mehrere Themen gleichzeitig zu definieren, und jeweils eines davon auch zu verwenden
- Themen werden in der Anwendung innerhalb des dafür speziell vorgesehenen Verzeichnisses *App_Themes* angelegt
- Innerhalb dieses Verzeichnisses, kann eine beliebige Anzahl von Themen angelegt werden.
- Jedes Verzeichnis verkörpert ein vollständiges Thema, das für die Anzeige der Seiten verwendet werden kann. Innerhalb eines spezifischen Themas werden die dafür notwendigen Definitionen für die Web-Steuerelemente in einer .skin-Datei abgelegt.
- Die gleichzeitige Verwendung eines Cascading Stylesheets für alle andern Elemente wird erreicht, indem im selben Verzeichnis neben der .skin-Datei eine entsprechende .css-Datei angelegt wird

- Ein Thema wird über den Assistenten für Projektitems eingebracht (wählen Sie Skin File)
- In der .skin-Datei bringen Sie die Elemente analog der Designer Datei ein. Das Attribut runat ist zwingend zu schreiben, das Attribut ID ist zwingend weg zu lassen. Alle andern Attribute verwenden Sie nach Bedarf.

```
1 <asp:TextBox runat="server"  
2     BorderColor="Blue"  
3     BorderWidth="10px"  
4     Background="#FFFFFF"  
5     BorderStyle="Solid"/>  
6  
7 <asp:TextBox runat="server"  
8     SkinId="Mandatory"  
9     BorderWidth="1px"  
10    BorderColor="Red"  
11    Background="#FF99FF"  
12    BorderStyle="Solid"/>  
13
```

Standardeintrag

Benannter Eintrag mit SkinId

- Unterscheiden Sie zwischen den Standardeinträgen und den benannten Einträgen. Benannte Einträge verfügen über eine Definition einer SkinId. Pro Typ kann ein Standardeintrag aber beliebig viele benannte Einträge mit jeweils eindeutiger Benennung eingebracht werden.

- Themen können pro Seite oder für das ganze Projekt eingeschaltet werden
- Beachten Sie, dass zwei verschiedene Verwendungsmodi möglich sind
 - Theme-Modus
 - StylesheetTheme-Modus
- Der gewünschte Modus definiert die Adaption des Themas im Fall von Mehrfachdefinitionen. Im Theme-Modus werden die lokalen Definitionen von den Definitionen im Thema überschrieben. Der StylesheetTheme-Modus ist im Verhalten kompatibel zu den Stylesheets nach W3C. Das heisst lokale Definitionen überschreiben die Definitionen des Themas.
- Die Definition des Themas kann pro Seite festgelegt werden

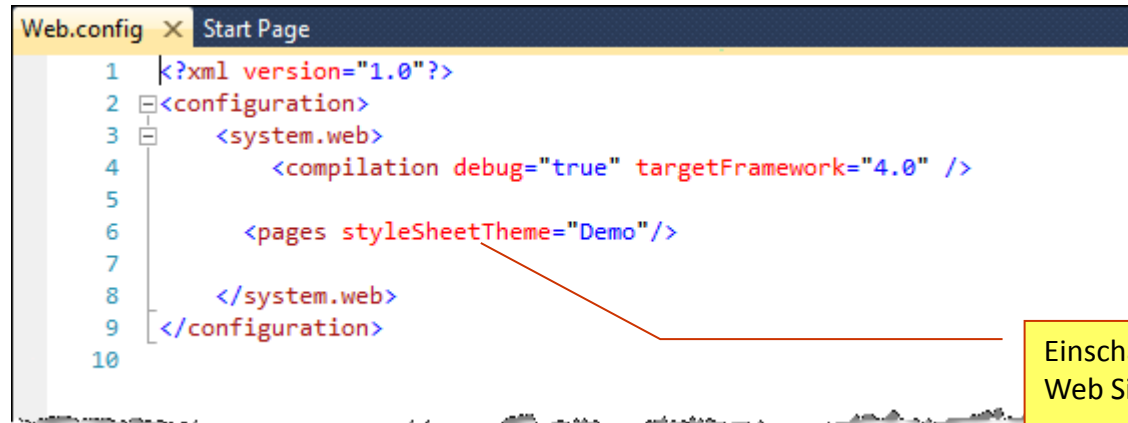
```
1 <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="ThemesDemo.aspx.cs" Inherits="LearningASPNET_
2     StyleSheetTheme="Demo" %>
3
4 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7     <title></title>
8 </head>
9 <body>
10     <form id="form1" runat="server">
11     <div>
12         <asp:TextBox ID="_tbThemesDemo1" runat="server"></asp:TextBox><br /><br />
13         <asp:TextBox ID="_tbThemesDemo2" runat="server" SkinID="Mandatory"></asp:TextBox><br /><br />
14         <asp:TextBox ID="_tbThemesDemo3" runat="server" BorderStyle="Solid" BackColor="#FFFFBB"
15             BorderColor="Green" BorderWidth="4px"></asp:TextBox>
16     </div>
```

Einschalten des StylesheetTheme-Modus für die Seite

Verwendung des Standardthemas

Verwendung des benannten Themas

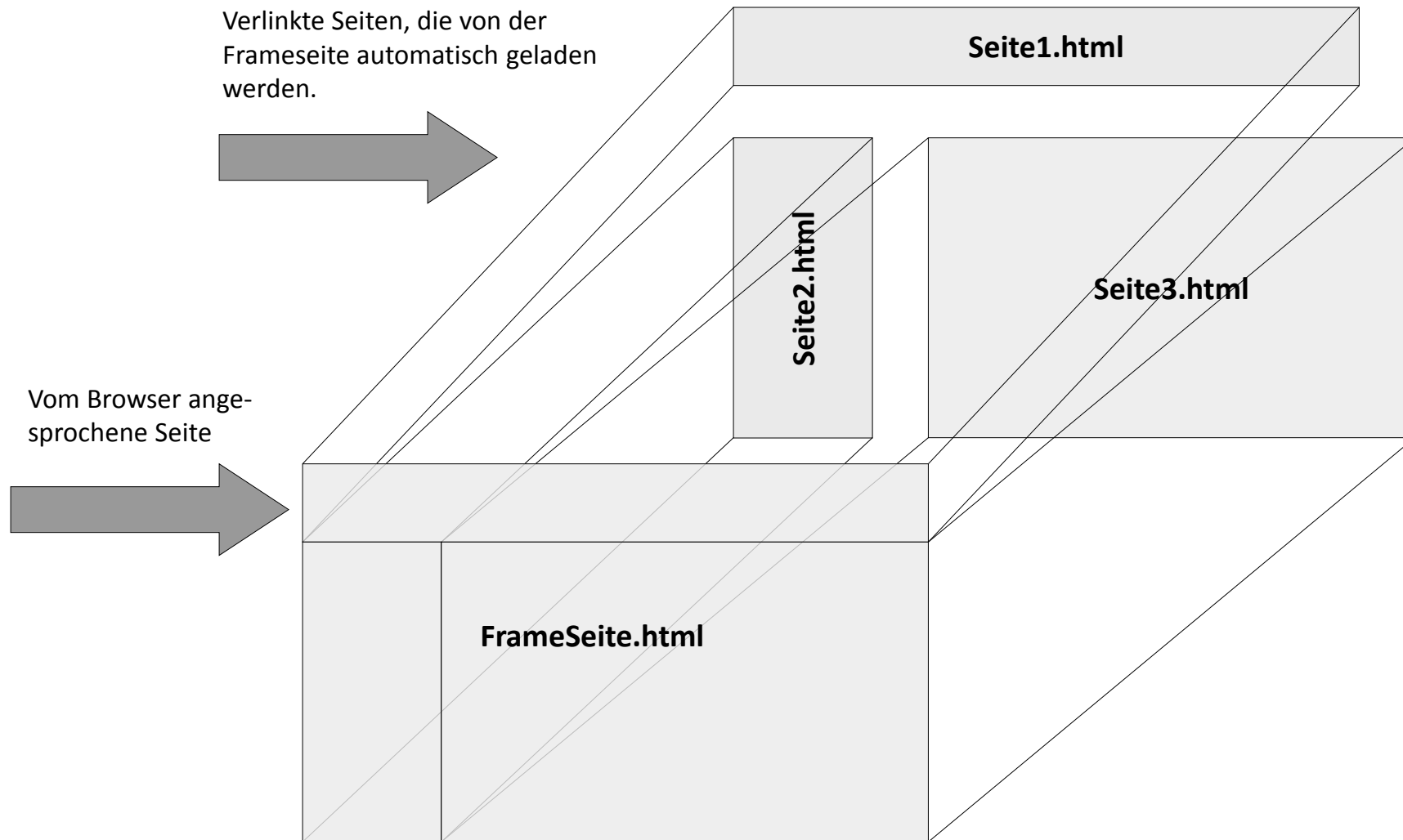
- Alternativ kann das Thema (beide Modi) in der Datei web.config eingeschaltet werden



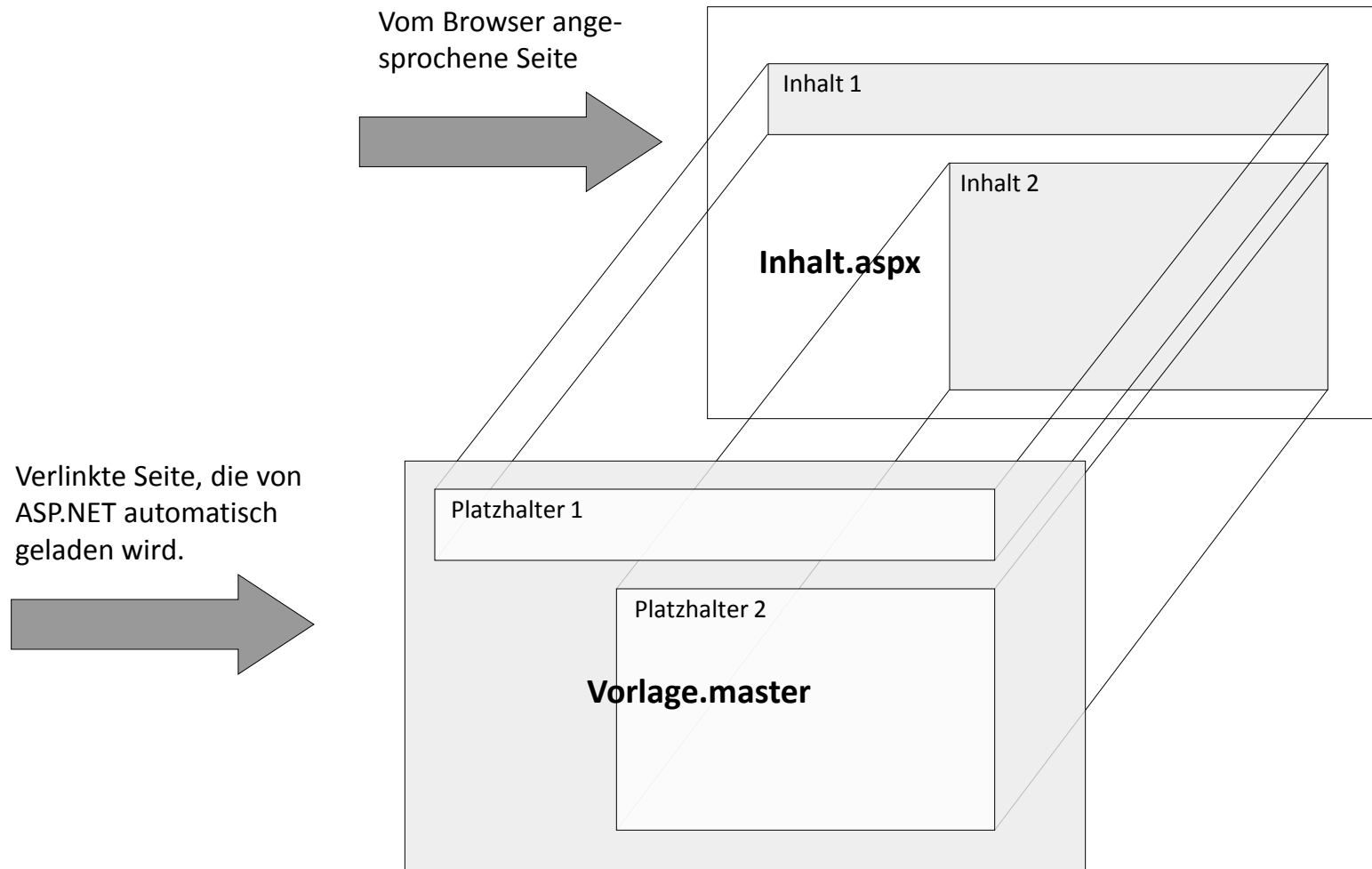
```
1 <?xml version="1.0"?>
2 <configuration>
3   <system.web>
4     <compilation debug="true" targetFramework="4.0" />
5
6     <pages stylesheetTheme="Demo" />
7
8   </system.web>
9 </configuration>
10
```

Einschalten des StylesheetTheme-Modus für die gesamte Web Site.

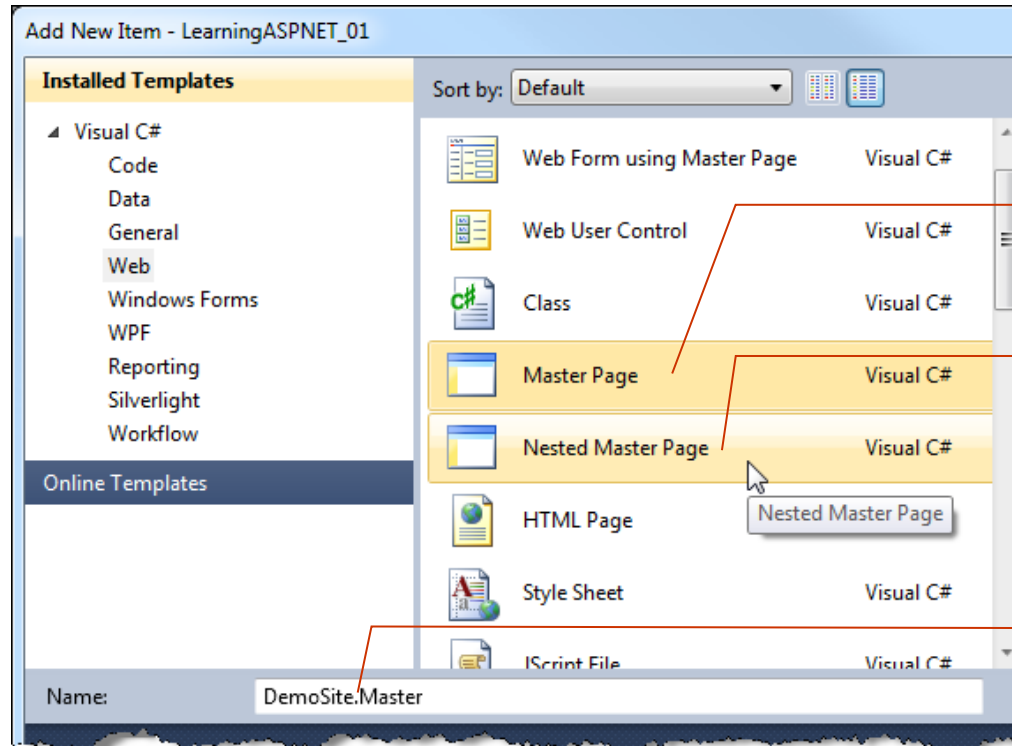
- Das Konzept der Masterseiten verfolgt die Zielsetzung, dass die Teile der Webseite, die nur selten ändern, nicht in jeder Seite redundant implementiert werden, sondern in einer Vorlagenseite einmalig definiert werden können, und anschließend mit den wechselnden Inhalten der einzelnen Seiten kombiniert werden.
- Eine ähnliche Zielsetzung verfolgt das Konzept der Frames eines Browsers. Die Frames sind jedoch Bestandteil des HTML Standards und haben mit der Programmierung der Seite genauso wenig (oder genauso viel) zu tun, wie alle andern HTML Elemente.
- Die Unterstützung der Frames durch ASP.NET beschränkt sich denn auch auf die mögliche Angabe des Ziels in einem Link (die Eigenschaft *Target* ist bei den diversen Klassen beschrieben).
- Die Frame Technik stellt in einer hierarchischen Art und Weise eine HTML-Frameseite mit einer referenzierten Web Seite pro Frame zusammen. Der Benutzer surft die Frameseite an, und erhält als Resultat die verschiedenen referenzierten Webseiten der Frames im Browser zu sehen. Es ist nicht möglich, mit einem Link den Inhalt in mehr als einem andern Fenster anzusprechen. Die Sichtbarkeit der Frames kann durch den Entwickler definiert werden. Ist eine Framegrenze sichtbar, kann Sie durch den Benutzer während dem surfen nicht verschoben werden. Inhalte in einem Frame werden vom Browser bei der Framegrenze gnadenlos abgeschnitten. Verschachtelung von Frame Seiten ist möglich.



- Auch das Konzept der Masterseiten ist eine hierarchische Abbildung von Seiten.
- Anstelle der Frameseite verwendet das Konzept eine Masterseite, welche jedoch nicht wie in der Frame Technik auf Inhaltsseiten referenziert.
- Die Masterseite ist eine Vorlage, welche von Inhaltsseiten benutzt wird. Auch hier können Vorlagenseiten verschachtelt werden. Da die Verbindung der beiden Teile nicht von der Vorlage zum Inhalt, sonder umgekehrt definiert ist, ist auch logisch, dass der Benutzer vom Browser aus die Inhaltsseite direkt anspricht.
- Die ASP Maschine kombiniert den angesprochenen Inhalt mit der Vorlage zu einer HTML-Seite zusammen und sendet das Resultat in das angegebene Frame des Browsers.
- Da das Resultat innerhalb eines Browserfensters eine eigenständige Seite darstellt, können sich nun auch Teile aus der Vorlage mit dem Inhalt überschneiden, was den Einsatz von pulldown Menütechnik oder andern DHTML Anwendungen ermöglicht.



- Masterseiten werden über den Assistenten in Visual Studio erstellt.



Verwenden Sie die Vorlage Master Page für einen toplevel Master.

Verwenden Sie die Vorlage Nested Master Page für einen Master innerhalb eines andern Masters.

Definieren Sie den Dateinamen, lassen Sie aber die Namensweiterung als "Master" stehen.

- Das Resultat ist eine ASP.NET Seite, welche anstelle einer @Page-Direktive eine @Master-Direktive enthält.
- Anstelle eines Inhaltes ist das einzige Element ein Platzhalter (der später durch eine Inhaltsseite ersetzt wird)

- Nach dem generieren gestalten Sie die Masterseite nach ihren Anforderungen
- Das oder die ContentPlaceHolder-Steuerelemente definieren die Fläche der später dynamisch eingesetzten Inhaltsseiten

```
1 <%@ Master Language="C#" AutoEventWireup="true" CodeBehind="DemoSite.master.cs"
2
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.
4
5 <html xmlns="http://www.w3.org/1999/xhtml">
6 <head runat="server">
7   <title></title>
8   <asp:ContentPlaceHolder ID="head" runat="server">
9   </asp:ContentPlaceHolder>
10 </head>
11 <body>
12   <form id="_formMain" runat="server">
13   <div>
14     <asp:ContentPlaceHolder ID="_cntMainPlaceholder" runat="server">
15     </asp:ContentPlaceHolder>
16   </div>
17 </form>
18 </body>
19 </html>
20
```

Die Masterseite verwendet die Direktive @Master.

Definieren Sie einen Standardtitel für alle Seiten. Der Titel kann in einzelnen Inhaltsseiten angepasst werden.

Definieren Sie einen Standardnamen für das Formular.

Definieren Sie einen Standardnamen für die Aufnahme des späteren Inhaltes.

- Nach dem Erstellen einer Masterseite können zu dieser Masterseite mit Hilfe des Assistenten für Projektitems beliebig viele Inhaltsseiten erstellt werden
- Wählen Sie die Vorlage "Web Form using Master Page"
- Diese Vorlage erlaubt die Auswahl des zu verwendenden Masters
- Als Resultat dieser Generierung wird eine unvollständige HTML Seite (ohne HTML Block und ohne body-Tag) erstellt

The screenshot shows the code for `DemoContent.aspx` in the Visual Studio IDE. The code is as follows:

```
1 <%@ Page Title="Das ist der Seitentitel"
2     Language="C#" MasterPageFile="~/DemoSite.Master" AutoEventWireup="true"
3     CodeBehind="DemoContent.aspx.cs" Inherits="LearningASPNET_01.DemoContent" %>
4 <asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
5     <!-- Hier kommt der Inhalt für den Header Placeholder -->
6 </asp:Content>
7 <asp:Content ID="Content2" ContentPlaceHolderID="_cntMainPlaceholder" runat="serv
8     <!-- Hier kommt der Inhalt für den Inahlsplatzhalter -->
9 </asp:Content>
10
```

Annotations on the right side of the image:

- Definiert eine Inhaltsseite.
- Definiert den Browsertitel und überschreibt somit den Titel der Masterseite.
- Definiert eine die verwendete Masterseite.
- Definiert die Inhalte für den Headerbereich für diese Seite.
- Definiert die eigentlichen Inhalte der Seite.



Spezielle Themen

- Das Versenden von Emails funktioniert unter ASP.NET genau gleich wie in allen andern .NET Techniken
- Der Anwendungsfall Email versenden kommt aber in ASP.NET besonders häufig vor
- Sie verwenden den Namensraum `System.Net.Mail` und die darin enthaltenen Klassen:

Verzeichnis	Verwendung
<i>AlternateView</i>	Repräsentiert ein Format für die Ansicht des E-Mails.
<i>Attachment</i>	Repräsentiert eine dem E-Mail angehängte Datei.
<i>LinkedResource</i>	Repräsentiert eine eingebundene Ressource in einem Anhang (Attachment).
<i>MailAddress</i>	Repräsentiert eine E-Mail Adresse eines Absenders oder eines Empfängers.
<i>MailMessage</i>	Repräsentiert eine E-Mail Mitteilung, die mittels eines Objekts der Classe <i>SmtpClient</i> versendet werden kann.
<i>NetworkCredential</i>	Erzeugt ein Objekt, das für Passwort basierte Authentifizierung verwendet werden kann.
<i>SmtpClient</i>	Die Klasse erlaubt das Versenden einer Mitteilung mit dem SMTP-Protokoll.
<i>SmptException</i>	Typ der ausgelösten Ausnahme, wenn ein Objekt der Klasse <i>SmtpClient</i> eine Nachricht nicht versenden konnte.

```
protected void btnOrder_Click(object sender, EventArgs e) {
    try {
        // Message erstellen
        MailMessage objMail = new MailMessage();
        objMail.To.Add(tbEmail.Text);
        objMail.From = new MailAddress("Absender@Domain.ch");
        objMail.Subject = "Ihre Bestellung: " + lblProduct.Text;
        objMail.Body = "Besten Dank für Ihre Bestellung.\r\n" +
            "...";
        objMail.BodyEncoding = Encoding.GetEncoding("iso-8859-1");
        objMail.IsBodyHtml = false;

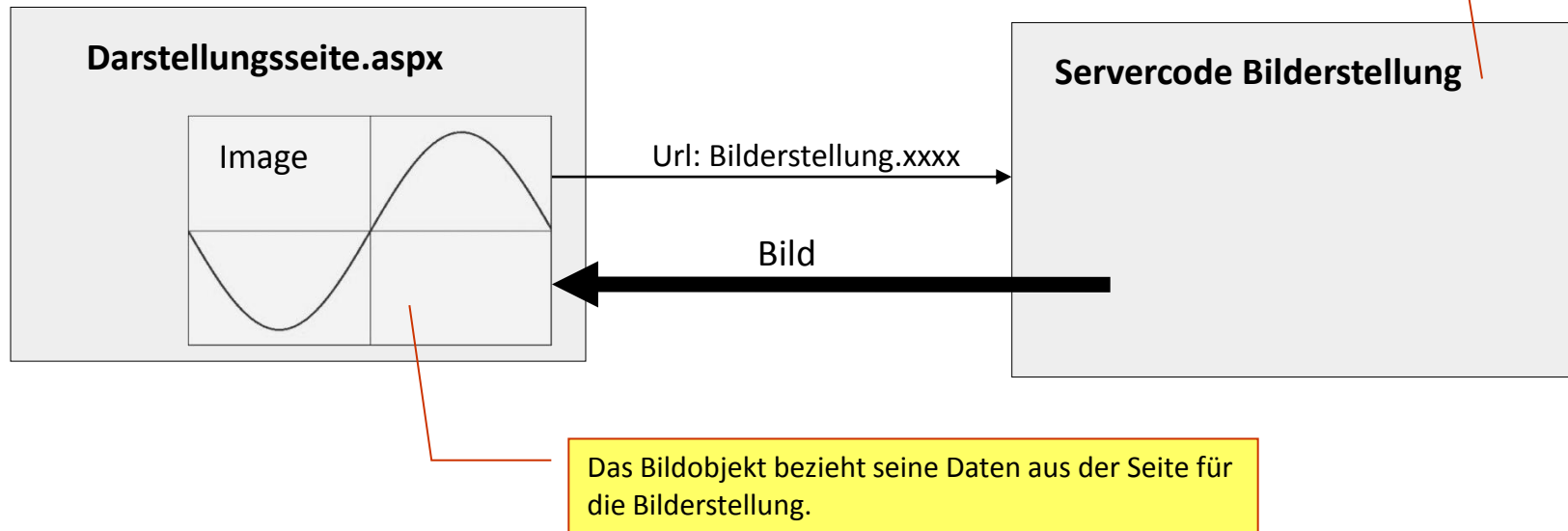
        // Attachments definieren
        string strFileName = this.Request.PhysicalApplicationPath + @"App_Data\MyDemoFile.Ext";
        objMail.Attachments.Add(new Attachment(strFileName));

        // Mail versenden
        SmtpClient objMailClient = new SmtpClient();
        objMailClient.Credentials = new System.Net.NetworkCredential("Sender@MyDomain.ch", "Passwort");
        objMailClient.Host = "Smtp.MyMailServer.ch";
        objMailClient.Send(objMail);

        // OK Seite anzeigen
        this.Response.Redirect("DownloadOk.aspx", false);
    }
    catch (Exception exc) {
        // Fehlerseite anzeigen
        this.Response.Redirect("DownloadFailed.aspx", false);
    }
}
```

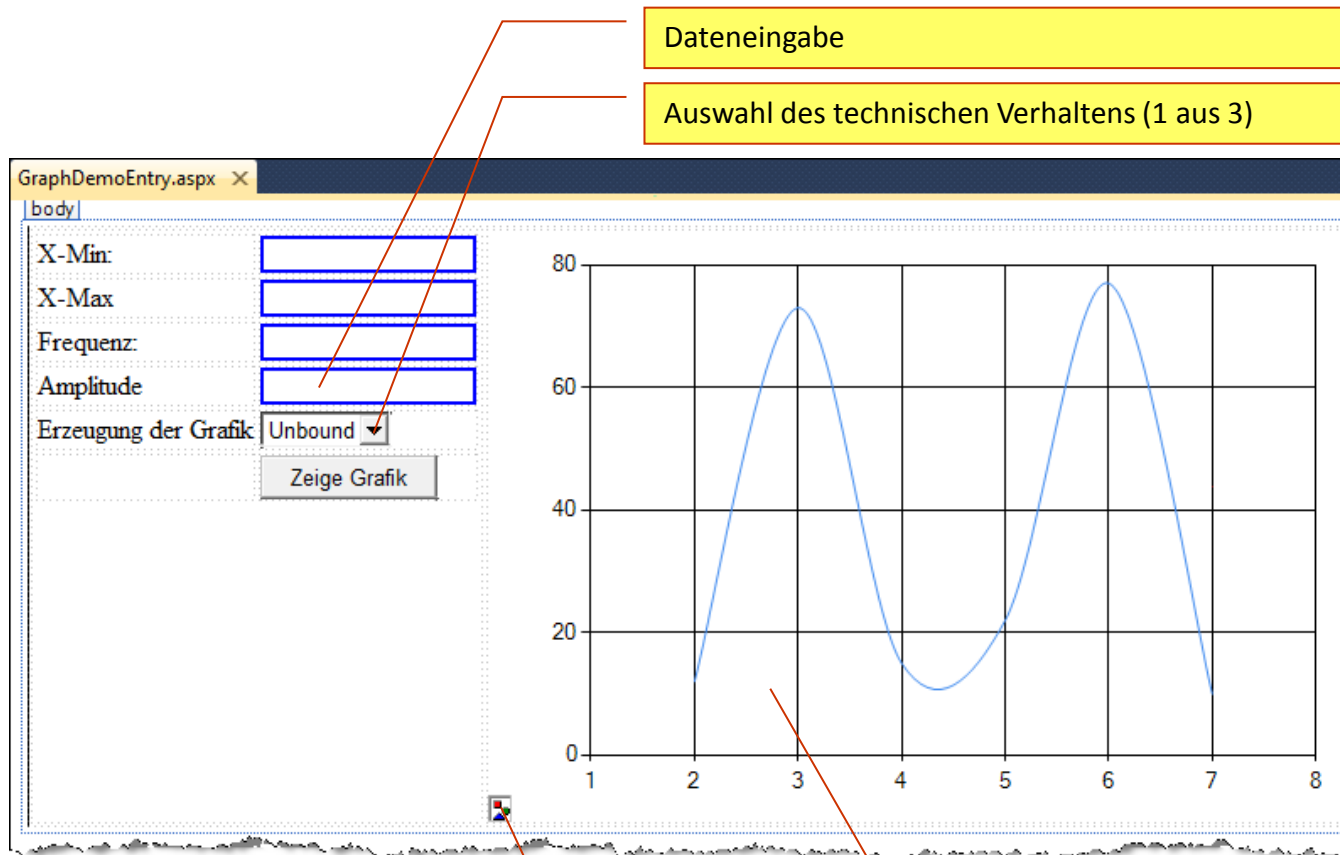
- Oft werden Daten aus Messreihen, gespeichert in Datenbanken oder andern Formen, als Bild wiedergegeben
- Dabei soll das Bild nicht als Datei erstellt, sondern lediglich als Bild aus dem Speicher des Servers an den Client übermittelt werden

Die Seite für die Bilderstellung benötigt keine optischen Elemente, sondern speichert das fertige Bild in den Datenstrom der Antwort.



- Für die Lösung der anstehenden Aufgabe müssen wir wissen, dass ein Bild in einer Seite eine Anforderung mit dem Bild zum Server sendet
- Diesen Umstand nützen wir aus und finden drei verschiedene Lösungsansätze
 1. Das Bild mit dem Chart-Steuerelement erstellen
 2. Dem Bild die URL einer eigenen aspx-Datei angeben und das Bild selber berechnen
 3. Dem Bild die URL auf einen eigenen HTTP-Handler angeben und das Bild selber berechnen
- Die Varianten 2 und 3 unterscheidet sich im Wesentlichen dadurch, dass der HTTP-Handler kein eigentliches UI hat, was in diesem Fall auch nicht notwendig ist.
- Der Grundaufbau besteht aus einer Seite, die entweder ein Bild darstellt oder ein Chartelement enthält (Bemerkung: In der Praxis implementieren Sie nur eine der 3 Varianten)
- Je nach gewählter Nutzungsart durch den Benutzer, wird im Code Behind die eine oder andere Art eingeschaltet

- Die Nutzseite enthält Steuerelemente für die Dateneingabe und die Auswahl des technischen Verhaltens



Vorbereitetes Chart-Steuerelement (unsichtbar konfiguriert).

Vorbereitetes Image-Steuerelement (unsichtbar konfiguriert).

```
protected void Page_Load(object sender, EventArgs e) {  
    if (!IsPostBack)  
    {  
        // Initialisierung der Daten  
        ...  
    } else {  
        // Speichere Werte in der Sitzung  
        Session["XMIN"] = _tbXMin.Text;  
        ...  
        // Typ auswählen  
        switch (_ddlTyp.SelectedValue) {  
            case "1": // Variante aspx  
                _imgGraph.ImageUrl = "~/GraphDemo/CreateGraph.aspx";  
                _imgGraph.Visible = true;  
                _chartDemo.Visible = false;  
                break;  
            case "2": // Variante Chart-Steuerelement  
                InitializeChartData();  
                _imgGraph.Visible = false;  
                _chartDemo.Visible = true;  
                break;  
            case "3": // Variante HTTP-Handler  
                _imgGraph.ImageUrl = string.Format("~/MyOwnGHandler.wlab?XMin={0}&XMax={1}&Freq={2}&Ampl={3}",  
                    _tbXMin.Text, _tbXMax.Text, _tbFrequency.Text, _tbAmplitude.Text);  
                _imgGraph.Visible = true;  
                _chartDemo.Visible = false;  
                break;  
            default:  
                break;  
        }  
    }  
}
```

In der Variante aspx, wird das Bild sichtbar geschaltet und die Eigenschaft ImageUrl auf die Datei CreateGraph.aspx definiert. Diese URL wird von Image benutzt um das Bild abzurufen. Die Daten werden in dieser Variante über das Sessionobjekt übergeben.

In der Variante Chart-Steuerelement werden in der Methode InitializeChartData() dem Steuerelement die Daten übermittelt. Anschliessend wird das Steuerelement sichtbar geschaltet. Der Rest wird vom Steuerelement selber erledigt.

In der Variante HTTP-Handler, wird das Bild sichtbar geschaltet und die Eigenschaft ImageUrl auf den HTTP-Handler definiert. Diese URL wird von Image benutzt um das Bild abzurufen. Die Daten werden in dieser Variante über den Querystring übergeben.

- Die Variante aspx besteht aus einer normalen .aspx-Seite
- Der Designerteil wird nicht benötigt und bleibt wie er generiert wurde
- Der Code Behind erstellt die Grafik und übermittelt diese an den Aufrufer

```
protected void Page_Load(object sender, EventArgs e)
{
    // Aufbau der Daten
    try
    {
        // Lesen der Daten
        float fXMin = Convert.ToSingle(Session["XMIN"]);
        float fXMax = Convert.ToSingle(Session["XMAX"]);
        float fFrequency = Convert.ToSingle(Session["FREQ"]);
        float fAmplitude = Convert.ToSingle(Session["AMPL"]);

        // Herstellen des Bildes
        Bitmap objBmp = CwlbGraphics.CreateImage(fXMin, fXMax, fFrequency, fAmplitude,
                                                    Color.WhiteSmoke, Color.Red);

        Response.Clear();
        Response.ContentType = "image/jpeg";

        objBmp.Save(Response.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
    }
    catch
    {
        string strFileError = MapPath("~/Images") + "\\GraphError.jpg";
        System.Drawing.Image imgError = System.Drawing.Image.FromFile(strFileError);
        imgError.Save(Response.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
    }
}
```

Wiederherstellen der Daten aus dem Sessionstatus.

Erstellen der Grafik über eine Hilfsklasse.

Vorbereiten des Datenstroms mit dem richtigen mime-type.

Speichern der Grafik in den Datenstrom der Antwort.

- Die Berechnung speichert die Daten in das Chart-Steuerelement
- Die Berechnung des Bildes wird durch das Chart-Steuerelement selber vorgenommen
- Ebenso die Übermittlung der Abbildung (wird auch via HTTP-Handler gemacht)

```
// Berechne die Punkte
private void InitializeChartData()
{
    // Konvertiere Werte
    double dXMin = Convert.ToDouble(Session["XMIN"]);
    double dXMax = Convert.ToDouble(Session["XMAX"]);
    double dFrequency = Convert.ToDouble(Session["FREQ"]);
    double dAmplitude = Convert.ToDouble(Session["AMPL"]);
    double dYCurrent;

    // Berechne Punkte
    for (double dX = dXMin; dX <= dXMax; dX += 0.1f)
    {
        // Berechne neuen Y-Wert
        dYCurrent = dAmplitude * (float)Math.Sin(dX * dFrequency);

        // Punkt in Daten ergänzen
        _chartDemo.Series[0].Points.Add(
            new System.Web.UI.DataVisualization.Charting.DataPoint(dX, dYCurrent));
    }
}
```

Hinzufügen der berechneten Punkte in die Datenserie des Chart-Steuerelements

- Diese Lösung bedarf einer neuen Datei des Typs "ASP.NET Handler", die über den Projektitem-Assistenten erstellt wird
- Dieser Handler definiert eine neue Namenserverweiterung für unsere Anwendung und erledigt die Berechnung des Bildes in der gleichen Art und Weise wie im Lösungsansatz 1
- Der HTTP-Handler muss in der Konfiguration von ASP.NET vermerkt werden, denn ASP.NET benutzt die Namenserverweiterung des Pfades um herauszufinden welcher Code aufgerufen werden soll.
- Beachten Sie, dass der HTTP-Handler als Parameter ein Contextobjekt enthält. Mit Hilfe des Kontextobjekts können die Objekte der Anforderung und der Antwort abgefragt werden.
- In der hier gezeigten einfachsten Implementierungsart eines HTTP-Handlers ist die Session im Kontext nicht verfügbar
- Beachten Sie zudem, dass in der Praxis ein HTTP-Handler zwecks Wiederverwendbarkeit in einer separaten Assembly mit Signierung untergebracht werden sollte.

```
public class MyGraphicsHTTPHandler : IHttpHandler
{
    public bool IsReusable { get { return true; }}

    public void ProcessRequest(HttpContext context)
    {
        HttpRequest objRequest = context.Request;
        HttpResponse objResponse = context.Response;

        // Aufbau der Daten
        try
        {
            // Lesen der Daten
            float fXMin = Convert.ToSingle(objRequest.QueryString["XMin"]);
            float fXMax = Convert.ToSingle(objRequest.QueryString["XMax"]);
            float fFrequency = Convert.ToSingle(objRequest.QueryString["Frequency"]);
            float fAmplitude = Convert.ToSingle(objRequest.QueryString["Ampl"]);

            // Herstellen des Bildes
            Bitmap objBmp = CwlbGraphics.CreateImage(fXMin, fXMax, fFrequency, fAmplitude,
                                                    Color.Wheat, Color.Black);

            objResponse.Clear();
            objResponse.ContentType = "image/jpeg";
            objBmp.Save(objResponse.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
        }
        catch
        {
            string strFileError = context.Server.MapPath("~/Images") + "\\GraphError.jpg";
            System.Drawing.Image imgError = System.Drawing.Image.FromFile(strFileError);
            imgError.Save(objResponse.OutputStream, System.Drawing.Imaging.ImageFormat.Jpeg);
        }
    }
}
```

Daten aus dem QueryString übernehmen

Bild über die selbe Methode wie Lösung 1 berechnen.

Daten wie in Lösung 1 an aufrufer zurückgeben.

- Der eigene HTTP Handler muss in der web.config bekannt gemacht werden

```
<?xml version="1.0"?>
<configuration>
  <system.webServer>
    <handlers>
      <remove name="ChartImageHandler" />
      <add name="MyOwnGHandler" preCondition="integratedMode" verb="GET,HEAD,POST"
        path="MyOwnGHandler.wlab"
        type="LearningASPNET_01.MyGraphicsHTTPHandler, LearningASPNET_01, ..." />
    </handlers>
  </system.webServer>
  <system.web>
    <httpHandlers>
      <add path="MyOwnGHandler.wlab" verb="GET,HEAD,POST"
        type="LearningASPNET_01.MyGraphicsHTTPHandler, LearningASPNET_01" validate="false" />
    </httpHandlers>
    <compilation debug="true" targetFramework="4.0">
    </compilation>
    <pages styleSheetTheme="Demo">
    </pages>
  </system.web>
</configuration>
```

Serverkonfiguration.

Sitekonfiguration