Grundlagen serverseitiger Programmierung

Programmierumgebungen für dynamische Webseiten

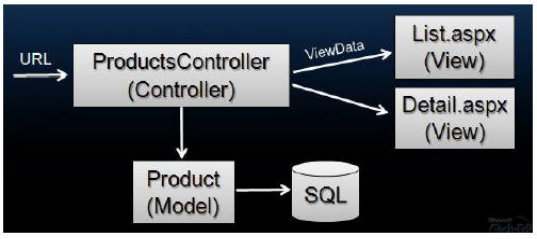
- CGI (Common Gateway Interface), Perl (Practical Extraction and Report Language)
  - über CGI-Schnittstelle aufgerufene Perl-Skripte
  - breite Unterstützung und gut geeignet für kleine Anwendungen
- PHP (PHP Hypertext Preprocessor)
  - in HTML eingebettet
  - PHP bietet mehr Geschwindigkeit, komfortablere Bibliotheken und wesentlich bessere Möglichkeiten zur DB-Anbindung
  - sehr spezialisierte Sprache, eingeschränkte Kommunikation mit anderen Applikationen, schwieriges Debugging und nicht-requestgetrieben; Hintergrundprozesse umständlich
- ASP (Active Server Pages)
  - fast nur mit dem MS Internet Information Server (IIS) verfügbar
- ASP.NET
  - Einbettung in das .NET-Framework
  - sehr leistungsfähig, aber wenig portabel (→ Mono)
- JSP (Java Server Pages) + Java Beans
  - JSP hat Ähnlichkeiten zu ASP und bietet vergleichbare Features
- Java, Flash, …
  - Ergebnisse nicht von HTML, CSS und JavaScript abhängig

Vorteile und Herausforderungen von Webapplikationen

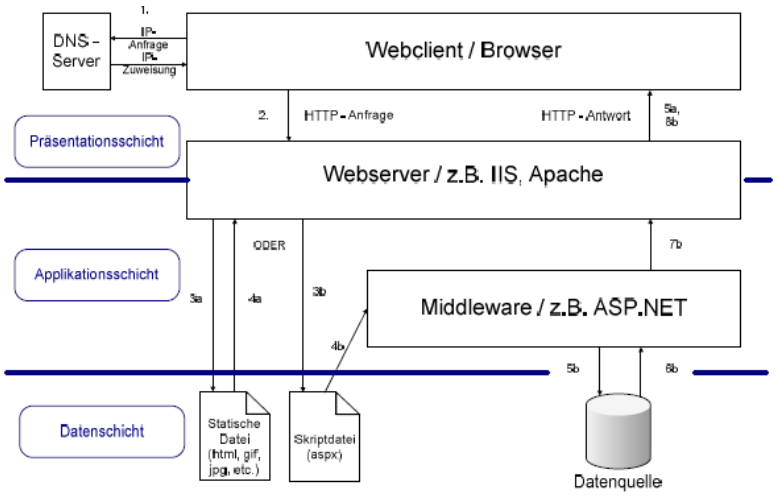
Vorteile	Herausforderungen
▪ clientseitig geringere Kosten ▪ beim Anwender muss keine SW installiert werden, ein einfacher Webbrowser ist ausreichend ▪ Nutzung über das Internet / von unterwegs ▪ Anwendungen sind multiuser- und netzwerkfähig ▪ leichte Wartbarkeit ▪ Benutzer setzen stets die aktuellste SW-Version ein ▪ konsistente Datenhaltung und zuverlässiges Backup ▪ Sicherheit, Skalierung, Load Balancing, Fail Over ▪ Aufbau in Mehrschichtarchitektur	▪ Parallelität ▪ Programmablauf (Zustandslosigkeit von HTTP, beliebige Navigationspfade) ▪ Zugriff auf den Datenbestand / DB ▪ Round-Trips / Caching ▪ Darstellung im Browser ▪ Laufzeit des Programms ▪ Sicherheit

Model View Controller Architecture

- Model
  - Komponenten einer Anwendung (basieren i.d.R. auf einem Datenmodell)
  - Eigenschaften: muss einen Zustand verwalten können, muss darstellbar sein, reagiert auf Nachrichten
- View
  - Präsentation des Modells für den Benutzer
  - verantwortlich für die Beschaffung, Darstellung und Änderungen im Modell
  - lässt sich sehr einfach austauschen
- Controller
  - verwaltet die Sichten, nimmt Benutzeraktionen entgegen



statische vs. dynamische Webseiten



- HTTP ist verbindungs- und zustandslos
- nach der Antwort wird die physikalische Verbindung zwischen Client und Server getrennt und alle Werte gehen verloren
- Problem: Wie kann trotzdem eine Kommunikation stattfinden?
- Lösung: Werte / Variablen müssen auf dem Client oder auf dem Server gespeichert werden

Ansätze zum Weiterreichen von Daten

1) Übertragung per „URL-Rewriting“
   - Daten werden an die URL angehängt (z.B. http://www.xy.de/seite.aspx?userid=tb)
   - in Formularen und Links verwendbar, für den Benutzer sichtbar, begrenzte Länge
   - jede Seite muss ein Skript sein, Übertragung per GET
   - muss in jeder URL stehen (auch bei Weiterleitung auf externe Seiten)
2) Übertragung per Hidden-Field
   - Daten werden im Body des Requests übertragen
   - nur in Formularen verwendbar, für den Benutzer nicht sichtbar, unbegrenzte Länge
   - jede Seite muss ein Skript sein und per Submit-Button verlassen werden
   - Übertragung per POST
3) Speicherung auf Clientseite mit Hilfe von Cookies
   - Werte werden vom Server auf dem Rechner des Benutzers abgelegt
   - Problem: Cookies können clientseitig gelöscht oder abgelehnt werden
   - mögliche Felder: Name, Wert, Lebensdauer, Domäne, Pfad, SSL-Info
   - sind nur von der Domäne aus lesbar, von der sie erzeugt wurden
   - maximal 4 KB Daten
   - 2 Arten: mit Laufzeit oder ohne Laufzeit (solange der Browser aktiv ist)
   - Vorteil: Benutzer hat die Transparenz über die gespeicherten Daten
   - Nachteile: browserabhängig, sensitive Daten werden übertragen
4) Speicherung auf Server- und Clientseite
   - ASP.NET bietet ein Session-Management, mit dessen Hilfe das Verwalten von Sessions durch die Vergabe von Session-IDs geregelt werden kann
   - Wiedererkennung und Zuordnung von Usern mittels einer Session-ID (120 Bit)
   - die Übertragung der Session-ID erfolgt per Cookie oder URL (Cookieless Session)
   - Vorteile: Infos werden dauerhaft, sicher & zentral gespeichert, sensitive Daten sind geschützt
   - Nachteil: Speicherbedarf auf dem Server

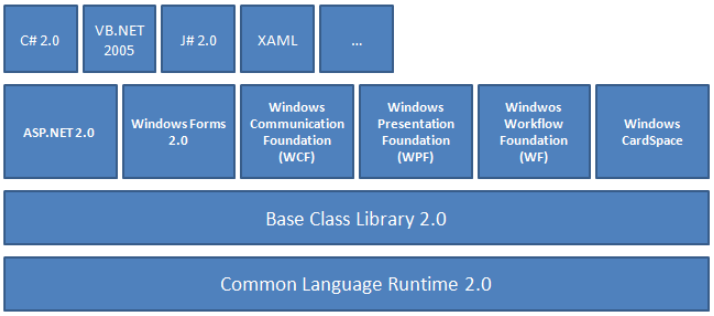
ASP.NET 2.0

Active Server Pages .NET

- Nachfolger der Active Server Pages
  - Vorreiter im Bereich Webserver-Scripting, Vorbild für JSP und PHP
- ASP.NET umfasst vier Gebiete
  - Webforms (.aspx)
  - Webservices (.asmx)
  - eigene HTTP-Handler (.ashx)
  - eigene HTTP-Module (z.B. URL-Rewriting, Authentifizierung, Wasserzeichen-Erstellung)

Eigenschaften von ASP

- ASP.NET nicht auf Scripting-Sprachen beschränkt, sondern unterstützt C#, VB.NET, J# etc.
- Code in ASP.NET wird nicht interpretiert, sondern liegt bereits als kompilierter Byte-Code vor (Intermediate Language)
- jede Seite ist ein eigenes Objekt (mit Eigenschaften, Vererbung etc.)
- Sessionverfolgung (auch ohne Cookies)
- Authentifizierung und Autorisierung
- Unterstützung von WebServices
- unterschiedliche Formen des Caching
- Master Pages, Themes, Skins

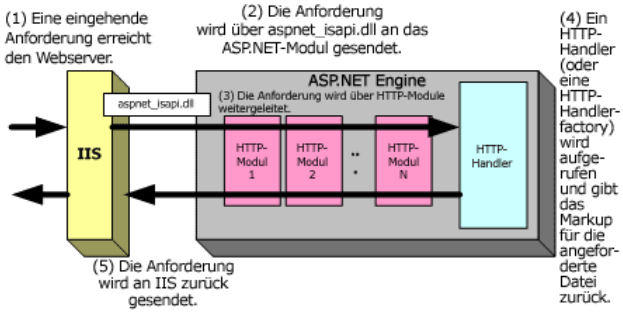


- Darstellung und Bearbeitung von Daten aus DBs
- Layout und Programmcode können getrennt werden
- jede Seite wird als Formular dargestellt, damit Textboxen, HiddenFields etc. genutzt werden können

Ereignisbehandlung

- gesamter Programmcode liegt in serverseitigen Ereignisbehandlungsroutinen für die Server Control-Ereignisse
- es gibt zwei Arten von Ereignissen
  - Ereignisse, die im Rahmen der verschiedenen Verarbeitungsschritte einer APSX-Datei auf dem *Server* ausgelöst werden (z.B. Page_Load(), TextBox1_PreRender())
  - Ereignisse, die Aktionen und Zustandsänderungen (z.B. neuen Werten) auf dem *Client* entsprechen (z.B. Button1_Click(), TextBox1_TextChanged()) → JavaScript nötig
- Pseudo-Ereignisse
  - nicht jede Änderung in einem Eingabefeld im Browser kann/sollte sofort an den Server übertragen werden
  - die meisten Ereignisse werden zeitverzögert „gefeuert“
  - Postback: das Zurücksenden von Daten durch eine Webseite an dieselbe Seite auf dem Server
  - alle Roundtrips basieren auf einem HTTP-POST
  - durch clientseitigen Skriptcode (z.B. Java) wird erreicht, dass ein Klick auf einen Link oder die Wertänderung in einem Eingabefeld auch einen Postback auslösen kann
  - ob ein Server Control bei einem Klick oder einer Wertänderung einen Postback auslöst, ist konfigurierbar
  - Server bemerkt erst nach dem Roundtrip das Ereignis; der Server erhält keine Ereignismeldungen, weil es in HTTP keine solchen Ereignismeldungen vom Client an den Server gibt
  - Server löst erst bei festgestellten Inhaltsänderungen Ereignisse aus

Seitenbehandlung in ASP.NET



- HTTP-Module
  - beeinflussen den gesamten Ein- und Ausgabedatenstrom
  - unabhängig von Verknüpfungen mit Dateierweiterungen
  - typische Anwendungen: Anforderungsüberprüfung, Authentifizierungsüberprüfung, Protokollierung von Anforderungsinformationen
  - Aufruf von allen Anforderungen und Antworten
  - Ereignisauslösung während der Anforderung einer Ressource
  - verfügbare Ereignisse: BeginRequest, AuthenticateRequest, AutorizeRequest, ResolveRequestCache, …
  - Implementierung der iHttpModule-Schnittstelle
  - Großteil der Implementierung in der global.asax
  - Konfiguration in der web.config
  - es erfolgt kein direkter Zugriff auf *Request* oder *Response*
- HTTP-Handler
  - der Prozess, der als Antwort auf eine Anforderung an eine ASP.NET-Webanwendung ausgeführt wird
  - mehrere Arten von Handlern
    - .NET-Handler: Seitenhandler, Webdiensthandler, Benutzersteuerelementhandler
    - benutzerdefinierte Handler
  - ein Handler ist für die Darstellung eines bestimmten Ressourcentyps (aspx, html …) zuständig
  - Erweiterung um eigenen Handler möglich
  - Forbiddenhandler für Direktaufrufe von .cs, .asp, .csproj, .config
- Unterschied HTTP-Module / HTTP-Handler
  - HTTP-Module werden für alle Anforderungen und Antworten aufgerufen, während HTTP-Handler nur als Antwort auf bestimmte Anforderungen ausgeführt werden

ASP.NET Lebenszyklus

- Page_Init()-Methode
  - Initialisierung des ASP.NET Arbeitsprozesses
  - Seite selber und alle Steuerelemente werden initalisiert
- LoadViewState()-Methode *(nur bei Postback)*
  - der Zustand (Attribute) der veränderten Steuerelemente wird aus dem Hiddenfield geladen
- LoadPostData()-Methode *(nur bei Postback)*
  - Daten, die der Benutzer in ein Formular eingetragen hat, werden den Steuerelementen zugewiesen
- Page_Load()-Methode
  - Implementierung der Geschäftslogik

- RaisePostDataChangeEvent()-Methode *(nur bei Postback)*
  - Änderungsprüfung des Zustandes der Steuerelemente seit dem letzten Postback
  - Abarbeitung der Ereignisse
- RaisePostBack-Event()-Methode *(nur bei Postback)*
  - Abarbeitung des Ereignisses, dass zu dem Postback geführt hat
- PreRender()-Methode
  - letzte Möglichkeit für die Beeinflussung der Steuerelemente
- SaveViewState()-Methode
  - Zustände der Seiten- und Steuerelemente werden abgespeichert, um für das nächste Postback als Vergleichswert zu dienen
- Render()-Methode
  - Objekt System.Web.UI.Page wird in HTML-Code umgewandelt
- Dispose()-Methode
  - nicht mehr benötigte Objekte werden zerstört
  - Leerung des Speichers
- Unload()-Methode
  - i.d.R. kein benutzerspezifischer Code mehr

ViewState

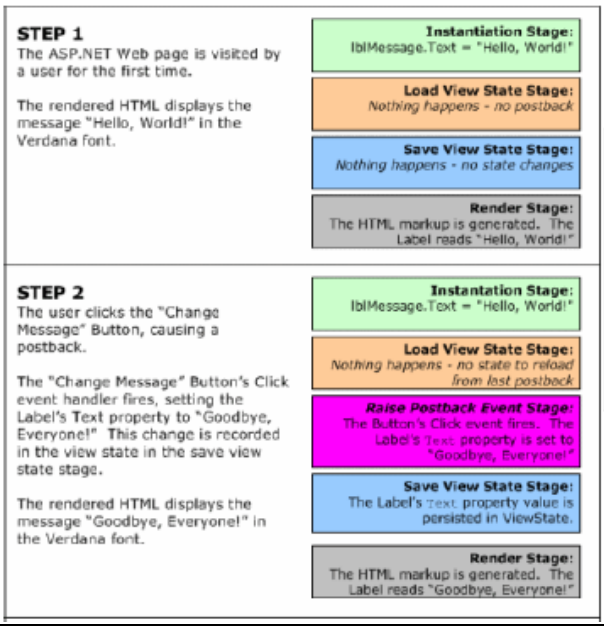
- speichert Werte per Schlüsselnamen wie ein Hashtable, z.B. ViewState[„Key1“]=„abc“
- ist eine geschützte Eigenschaft der Klasse System.Web.UI.Control, von welcher alle Seiten und Serversteuerelemente erben
- jedes Steuerelement hat seine eigene ViewState, wird als Hiddenfield realisiert
- verwaltet den dynamisch veränderten Teil der Seite, Formularinhalte sind davon ausgenommen
- Beispiel

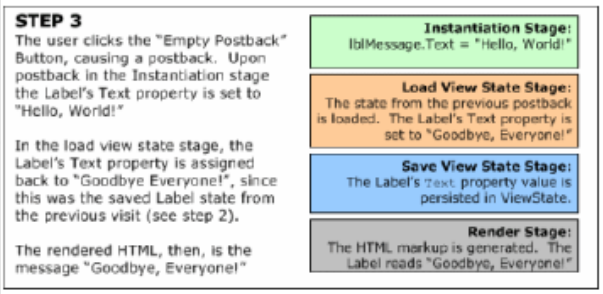
```
<asp:Label runat="server" ID="lblMessage"
  Font-Name="Verdana" Text="Hello, World!"></asp:Label>
<br />

<asp:Button runat="server"
  Text="Change Message" ID="btnSubmit"></asp:Button>
<br />

<asp:Button runat="server" Text="Empty Postback"></asp:Button>
```

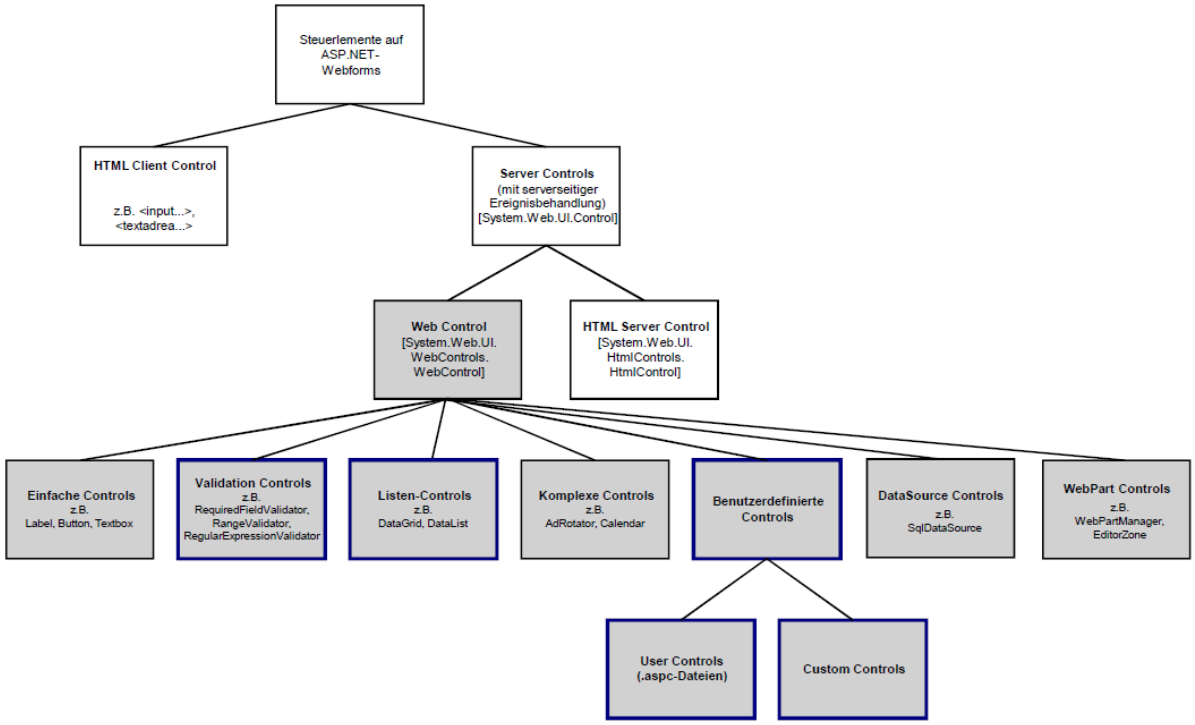
```
private void btnSubmit_Click(object sender, EventArgs e)
{
  lblMessage.Text = "Goodbye, Everyone!";
}
```





ASP.NET Controls

Überblick Steuerelemente



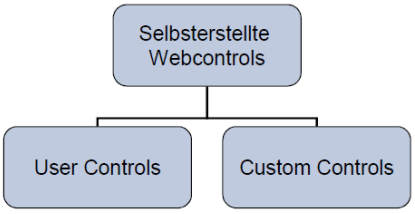
Typen von Steuerelementen

- HTML Client Controls
  - bekannte standardisierte HTML-Tags
  - Abfrage weiterhin über *Request*-Objekt möglich
  - Ausgabeformat im Browser
  - Beispiel: <input type="submit" name="SubmitButton1" value="einfacher HTML-Button">
- HTML Server Controls
  - konzeptionelle Mischung aus Standard-HTML-Elementen und WebControls
  - entspricht normalem HTML-Tag mit Zusatzattribut runat="server"
  - keine Ereignisbehandlung, keine Datenbindung
  - Erzeugung eines .NET-Objektes auf dem Server
  - eingeschränkte Funktionalität
  - Einsatz: Migrationszwecke
  - Beispiel: <input id="Button1" type="button" value="konvertierter HTML-Server-Steuer-element-Button" name="Button1" runat="server">
- Web Server Controls
  - vordefinierte Tags mit erweiterter Funktionalität
  - keine 1:1 Abbildung zu HTML-Elementen
  - komplexe Controls, wie DataGrid oder Calender
  - browserspezifische Umsetzung, Control kümmert sich selbst um die beste Darstellung
  - Möglichkeit zum direkten Postback
  - Ereignisweiterleitung/-behandlung
  - einheitliches Objektmodell
  - abgeleitet von System.Web.UI.WebControls
  - Beispiel: <asp:Calender id="MeinKalender" runat="server" />

Ausprägungen von Steuerelementen

- Validierungs-Server Controls
  - Server Controls, deren Aufgabe es ist, den Inhalt anderer Server Controls nach bestimmten Kriterien zu prüfen
  - Prüfen: Feld nicht leer, gleicher Inhalt in zwei Feldern, liegt Wert in einem bestimmten Wertebereich, komplexe Regeln mit Hilfe von regulären Ausdrücken oder eigenen Funktionen
  - schlägt die Gültigkeitsprüfung fehl, geben die Validierungs-Server Controls eine beliebige Meldung aus
  - ValidationSummary kann die Meldungen aller einzelnen Validation Controls zusammenfassen
  - sie arbeiten auf Wunsch nicht nur serverseitig, sondern durch den Einsatz von clientseitigem Skript-Code erzeugen Sie eine Fehlermeldung ohne Postback zum Server bzw. verhindern sogar den Postback zum Server bevor alle Eingaben korrekt sind
- Listen Controls
  - Steuerelemente, die eine Liste von Einträgen mit Einfach- und Mehrfachauswahl darstellen
    - DropDownList, ListBox, CheckBoxList, RadioButtonList
  - Befüllung der Listen
    - deklarativ durch Tags, programmatisch durch Items-Collection oder Datenbindung

benutzerdefinierte Steuerelemente



- User Controls (früher Pagelets)
  - sind selbst entwickelte Webcontrols in Form einer HTML-Quellcode-Datei, optional mit Code-Behind-Datei
  - Gruppierung von existierenden server controls
  - haben die Dateiextension .ascx und liegen innerhalb der Webanwendung
  - können beliebige Elemente enthalten (HTML + Serversteuerelemente)
  - zur Benutzung: Einbettung in andere Hosting Web Page
  - kann nicht alleine benutzt werden
  - Wiederverwendbarkeit von Layout und Quellcode
  - zentrale Aktualisierung
  - Beispiel: Fußzeile, die auf jeder Seite wiederholt werden soll
  - statisches Laden
    - per Drag&Drop wird das UserControl in die Entwurfsansicht der ASPX-Datei gezogen
    - in der Quell-Ansicht erscheint folgende Registrierung

      ```
      <%@ Register src="UserControls/WebUserControl1.ascx"
      tagname="WebUserControl1" tagprefix="uc1" %>
      ```

    - Verwendung

      ```
      <uc1:WebUserControl1 ID="webUserControl1" runat="server" />
      ```

  - dynamisches Laden
    - Einbindung des UserControls über einen PlaceHolder

      ```
      protected void Page_Load(object sender, EventArgs e) {
         Control myControl = Page.LoadControl("WebUserControl.ascx");
         PlaceHolder1.Controls.Add(myControl); }
      ```

- Custom Controls
  - selbstentwickelte Webcontrols rein programmcodebasiert (C#)
  - es gibt keine HTML-Quellcodedatei
  - kann in Form einer Assembly an jedem beliebigen Ort liegen
  - kann im Global Assembly Cache (GAC) liegen und so von jeder Webanwendung verwendet werden
  - der auszugebende Code wird über ein HtmlTextWriter-Objekt an ASP.NET geliefert
  - mögliche Anwendungen
    - Erweiterung eines bereits vorhandenen Steuerelements
    - mehrere Steuerelemente zu einem verbinden
    - Entwicklung eines eigenen Steuerelements
  - Registrierung

    ```
    <%@ Register TagPrefix="myCtl" Namespace="myControls" %>
    ```
  - Verwendung

    ```
    <myCtl:TextControl runat="server" MyText="Hallo Welt" />
    ```

ASP.NET AJAX

Grundlagen

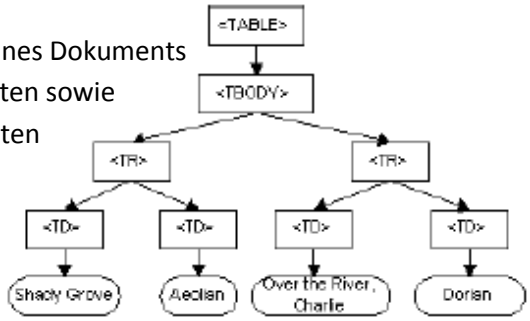
- Austausch von Seitenteilen anstelle des ständigen Neuladens von kompletten Webseiten
- Web-Anwendungen sollen sich wie Desktop-Anwendungen verhalten
- Daten werden auf Benutzeranforderung im Hintergrund geladen
- durch eine asynchrone Ausführung wird der Browser in der Zwischenzeit nicht blockiert
- für anwendungsartige Webseiten ist die Funktion vergleichbar mit Applets oder Flash
- Daten werden auf der Clientseite via JavaScript verpackt und in Form von XML über HTTP versendet
- möglich durch das XML HTTP-Request-Objekt, das von jedem aktuellen Browser unterstützt wird

Ablauf

- jede Benutzeraktion wird an die (sich auf Clientseite befindliche) AJAX-Engine delegiert
- AJAX-Engine steuert die Anfragen an den Server und die Veränderung der Webseite
- Anfragen an den Server mit z.B. SOAP über XMLHttpRequest()
- Veränderungen an der Webseite über JavaScript DOM

Document Object Model

- Programmierschnittstelle für den Zugriff auf HTML und XML
- Definition durch das World Wide Consortium
- dynamischer Zugriff auf Inhalt, Struktur und Layout eines Dokumentes
- Dokumente werden logisch wie ein Baum dargestellt
- insbesondere erlaubt DOM
  - die Navigation zwischen den einzelnen Knoten eines Dokuments
  - das Erzeugen, Verschieben und Löschen von Knoten sowie
  - das Auslesen, Ändern und Löschen von Textinhalten



Vorteile/Nachteile

Vorteile	Nachteile
▪ kein Browser-Plugin notwendig, nur JavaScript ▪ serverseitige Browsererkennung (Ziel: u.U. auf JavaScript verzichten zu können) ▪ Vermeidung des ständigen Neuaufbaus von Seiten ▪ Last auf dem Serversystem sinkt	▪ Unterstützung und Aktivierung von JavaScript notwendig ▪ nicht barrierefrei ▪ umfangreiche Tests notwendig (wenn AJAX von Hand programmiert wird) ▪ Back-Button funktioniert nicht immer ▪ Adresse der Seite ändert sich nicht mehr ▪ Latenzzeit

MS AJAX Produkte

- AJAX Library bietet AJAX-Funktionen und JavaScript-Erweiterungen
- ASP.NET AJAX-Erweiterungen umfassen serverseitige Unterstützung für die AJAX-Entwicklung (in Ergänzung zur Library)
- AJAX Control Toolkit (optionale Erweiterung, Sammlung für DHTML-Widgets für JavaScript-basierte Zusatzfunktionen)

AJAX-Framework

- Integration der Scripting-Funktionalität des Clients mit ASP.NET in einer umfassenden Plattform
  - clientseitiges Framework
    - umfangreiche Bibliothek zur Programmierung mit JavaScript (ohne Kenntnisse in JavaScript zu besitzen)
    - .NET-ähnliche Syntax für JavaScript, deklarative Definition von HTML-Tags
    - Datenbindung, eigene Eventhandler
  - serverseitiges Framework
    - teilweises Aktualisieren einer Seite mit einem UpdatePanel (ohne Postback)
    - ControlExtender zur Erweiterung von Steuerelementen mit clientseitigen Funktionen (z.B. Eingabevervollständigung)
    - TimerControl (Neuladung via Zeitsteuerung)
    - eigenes Template für neue Anwendungen
  - Installation
    - Nutzung des Templates
    - Integration in eine bestehende Anwendung erfordert u.U. umfangreiche Änderung an der web.config

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server" EnablePartialRendering="true"/>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
                <asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click" /><br />
                <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
            </ContentTemplate>
        </asp:UpdatePanel>
        <div>
            <asp:UpdateProgress ID="UpdateProgress1" runat="server">
                <ProgressTemplate>
                    Loading ...
                </ProgressTemplate>
            </asp:UpdateProgress>
        </div>
    </form>
</body>
</html>
```

ScriptManager ist Kernstück, welches JavaScript einbindet;
EnablePartialRendering muss immer gesetzt sein, damit AJAX funktioniert

UpdatePanel ist Bereich einer Seite, der dynamische Inhalte besitzt

UpdateProgress (max. 1 x pro Seite) zeigt den Inhalt an,
wenn grade ein AJAX-Request verarbeitet wird

Sessionmanagement, Konfiguration

Seitenzustand (ViewState)

- Ziel: Erhalt des Zustandes von Steuerelementen und Werten beim Selbstaufruf einer Seite
- Lebensdauer: bis zum Übergang auf eine andere Seite
- wird durch das versteckte HTML-Feld ID_VIEWSTATE realisiert
- ViewState Attribut steht sowohl der Page-Klasse als auch einzelnen Steuerelementen zur Verfügung
- es können nur Seitenzustände von Instanzen, die serialisierbar sind, gespeichert werden
- kann aus Gründen der Performance manuell ausgeschaltet werden

Sitzungszustand (session state)

- Ziel: Erhalt von Werten zwischen Seitenübergängen
- Lebensdauer: bis zum Ablauf einer bestimmten Zeit
- das ASP.NET-Framework weist jedem neuen Benutzer eine Sitzungsnummer (Session ID) zu
- diese Nummer wird per Cookie oder in der URL (Cookieless Session) übertragen
- persistente Cookies werden auf der Festplatte des Clients gespeichert, nicht persistente Cookies verbleiben im Speicher des Browsers und werden nach einer bestimmten Zeit gelöscht
- parallel dazu wird für jeden User auf dem Webserver ein Session Objekt angelegt
- im Session Objekt werden alle Session Variablen gesetzt und gelesen
- in einer internen DB wird festgelegt, welche Variablen und Werte einer bestimmten ID zugeordnet sind
- ASP.NET bietet die Option Sitzungstabellen auf einem dedizierten Server oder in einer SQL Server-DB zu speichern
- Cookieless Session
  - jede URL innerhalb einer Seite wird mit einer SessionID modifiziert
  - funktioniert nur bei relativen Verweisen
- WebFarmen
  - Sessionvariable wird im Speicher des Servers abgelegt, auf dem sie erzeugt wurde
  - Session ist auf anderen Servern nicht verfügbar
  - Session Handling
    - Session-Objekt (in-process)
    - State Server (out-process)
    - SQL-Server (persistent storage)
  - Unterscheidung nicht im Progammcode, nur in der Konfiguration
- Zuweisung

```
Session[„Zeichenkette“] = Objekt;
```

- Auslesen

```
string sessionValue = (String)Session[„Zeichenkette“];
```

- Konfiguration erfolgt in der web.config
  - <sessionState>
    - mode=“StateServer“: Daten werden im StateService auf einem beliebigen Rechner gespeichert
    - mode=“InProc“: Speicherung der Sitzungstabelle im Hauptspeicher
    - mode=“SQLServer“: Auswahl, wenn Daten dauerhaft gespeichert werden sollen

Anwendungszustand (applicaton state)

- Ziel: erhalt von Werten über alle Seitenaufrufe von allen Benutzern hinweg
- Lebensdauer: bis zum Herunterfahren des Webservers oder bis zum Austausch der global.asax
- jede Webanwendung ist eine Instanz von `System.web.HttpApplicationState`, der Zugriff erfolgt über die Application-Eigenschaft des `HttpContextes`
- hier sollten Informationen abgelegt werden, die über alle Sessions verfügbar sein sollen, z.B. Anzahl der Benutzer oder DB-Verbindungen
- Zugriff aus allen Ressourcen der Webanwendung
- für die Verwaltung der globalen Zustände bietet ASP.NET die Objektmengen Application und Cache
- Verwendung der Objektmenge Application
  - Verwendung analog zur Session
  - Sperren der Application-Variable beim Schreiben
  - Beispiel aus der global.asax

```
public void Application_OnStart() {
       Application["Counter"] = 0;
}
public void Application_OnSessionStart() {
       Application.Lock();
       Application["Counter"] = (int)Application["Counter"] + 1;
       Application.Unlock();
}
```

- Verwendung der Objektmenge Cache
  - Transient Application State
  - kontrolliert durch das Cache-Objekt
  - verwendet gemeinsam genutzten Speicher innerhalb der Webanwendung
  - Auflistung ist mit einem Verfallskriterium ausgestattet
  - Unterschiede gegenüber dem Application-Objekt
    - Abhängigkeiten (z.B. von einer Datei)
    - automatisches Sperren der Daten
    - Ressourcen-Management
    - Callbacks (wenn der Cache gelöscht wird)
  - statische Variablen
    - können jederzeit definiert werden
    - es wird nur eine Instanz erzeugt
    - Deklaration in der global.asax in Verbindung mit der @Application Classname="" Direktive
  - möglicher Einsatz: Produktdaten in einem Onlineshop werden aus dem Cache gelesen, dieser wird automatisch alle x Zeiteinheiten aktualisiert
  - Zuweisung

```
Cache[„Zeichenkette"] = Objekt;
```

  - Auslesen

```
object Objekt = Cache[„Zeichenkette"];
```

  - explizite Zuweisung

```
Cache.Insert(„Zeichenkette", Objekt, Dependency");
```

    - Methode Add() fügt nur hinzu, ersetzt aber nicht den Inhalt

- CacheDependency

```
using System.Web.Caching;
…
private void Page_Load(object sender, System.EventArgs e)
{
   if (Cache["meineDaten"] == null)
   {
      DataSet myData = new DataSet();
      myData.ReadXML(Server.MapPath("daten.xml"));
      CacheDependency myDependency =
               new CacheDependency(Server.MapPath("daten.xml"));
      Cache.Insert("meineDaten", myData, myDependency");
   }
   DataGrid1.DataSource = (DataSet)Cache["meineDaten"];
   DataGrid1.DataBind();
}
```

global.asax

- Hauptaufgabe: Definition von Ereignisbehandlungsroutinen für seitenübergreifende Ereignisse wie z.B. der Start einer Sitzung
- Nebenaufgabe: globale Instanziierung von Objekten, die auf allen Seiten verfügbar sind
- Ereigniskategorien
  - Ereignisse, die bei jedem Seitenaufruf ausgelöst werden
  - Ereignisse, die fallweise ausgelöst werden
- Zugriff auf die Session-Variable möglich
- bei Änderung der Datei wird die Anwendung neu gestartet
- kein Zugriff über den Webserver (URL) möglich
- Datei existiert optional einmal pro Webanwendung
- wenn die Datei global.asax existiert, kompiliert ASP.NET sie in eine Klasse, die von der HttpApplication-Klasse abgeleitet ist, und verwendet diese abgeleitete Klasse zur Darstellung der Anwendung

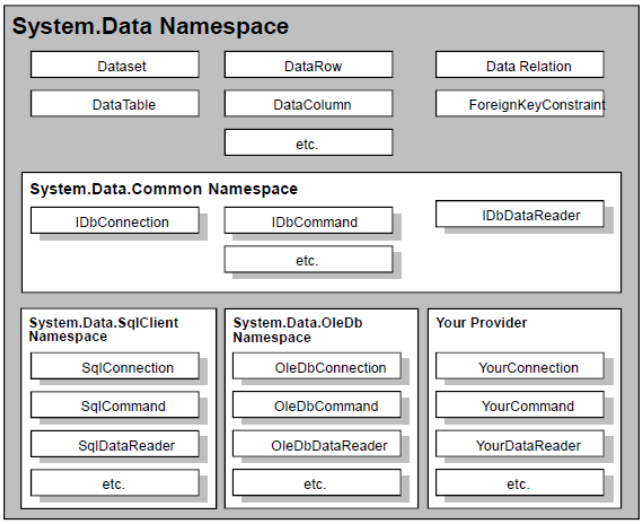
web.config

- Konfiguration von ASP.NET Anwendungen
- auf XML basierende Textdatei
- einfache Replikation / Wiederherstellung / Migration
- erbt von der Datei machine.config
- pro Webanwendung eine oder mehrere web.config-Dateien (pro Webverzeichnis eine)

ADO.NET

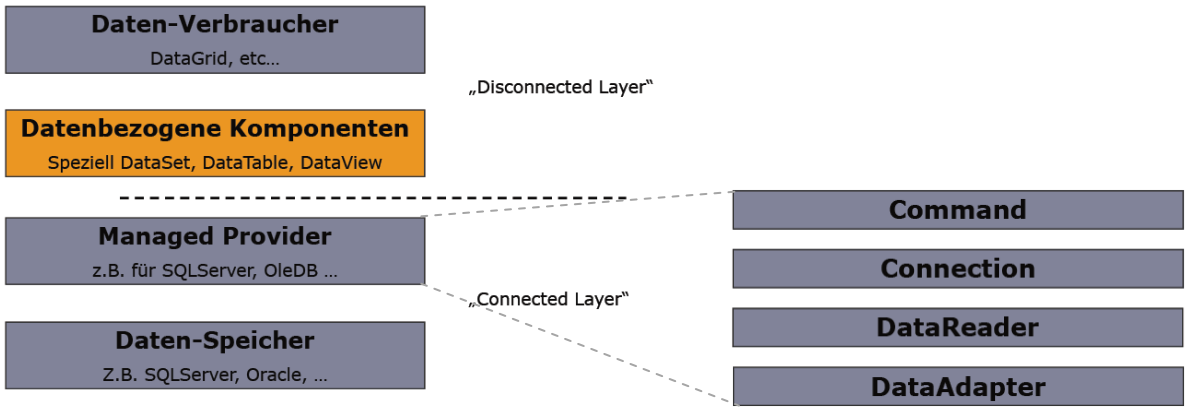
Grundlagen ADO.NET

- Teil der .NET-Plattform
- Sammlung von Klassen, die den Zugriff auf Datenbanken gewährleisten
- Datenmodell ähnlich zu ADO (ActiveX Data Objects)
- einheitliches Objektmodell
- exzellente Unterstützung für „disconnected n-tier“ Applikationen
- starke Unterstützung für XML
  - XML ist die Grundlage für ADO.NET
  - keine 2 APIs mehr für XML & DataAccess
- unterstützt Multiple Active Result Sets (MARS) (Mehrfachverwendung einer Verbindung)



System.Data.Common stellt Interfaces bereit; wenn nur gegen Interfaces implementiert wird, lässt sich die Datenbank einfach austauschen ohne den Code anpassen zu müssen

Schematischer Aufbau ADO.NET



- Managed Provider
  - sind die ADO.NET Version von Providern (ADO, OLE/DB) oder Treibern (ODBC)
  - Provider laufen in der gemeinsamen Laufzeitumgebung (CLR)
  - nicht unbedingt der einzige Weg Daten einzulesen im .NET Framework (XML)
    - i.d.R. aber der Standardweg
  - sie bestehen aus einem bestimmten Set von „managed classes", die bestimmte Interfaces implementieren
  - derzeit liefert Microsoft erst 5 Managed Provider: Universaltreiber für ODBC/OLEDB, Spezialtreiber für Oracle & SQL-Server, relationaler Zugriff, XML-Zugriff (SQLXML), weitere folgen
  - Aufbau
    - Connection & Command sind aus ADO bekannt
    - der DataReader entspricht in etwa einem „forward-only, read-only" RecordSet
      - dient dem besonders schnellen Zugriff
    - der DataAdapter ist die Schnittstelle zwischen DataSet und Managed Providern
  - Connection Object
    - Repräsentation einer Verbindung zu einer Datenquelle
    - mit einer Connection ist es möglich …
      - die Verbindung zur Datenquelle anzupassen
      - Transaktionen handzuhaben (Begin, Commit, Abort)
    - jeder Data Provider hat eine eigene Version
    - Connections werden vom .NET-Framework wegen der Performance gepoolt

```
using System.Data.SqlClient;
…
SqlConnection sqlConnection = new SqlConnection(ConfigurationManager.
   ConnectionStrings[”AdventureWorksDB“].ToString());

Webconfig:
<configSections>
   <connectionStrings>
         <add name=”AdventureWorksDB” connectionString=”Data Source=
           .\SQLEXPRESS;AttachDbFilename=|DataDictionary|\AdventureWorks
           _Data.mdf;Integrated Security=True;User Instance=True”
           providerName=”System.Data.SqlClient” />
   </connectionStrings>
```

- Command Object
  - stellt ein auszuführendes Kommando dar
    - nicht unbedingt ein SQL Kommando
  - mit dem ADO.NET Command Object ist es möglich
    - ein Statement, welches auf dem Server ausgeführt werden soll, zu definieren
    - Parameter Informationen für diese Kommando anzugeben
    - Rückgabewerte nach der Kommandoausführung zu erhalten
  - kann Parameter enthalten
    - Werte, die bei Ausführung eines Statements genutzt werden können

```
SqlCommand sqlCommand = new SqlCommand(“SELECT * FROM Production.Product
  WHERE SessionId = @SessionId;”, sqlConnection);
SqlParameter sqlParam = new SqlParameter(“@SessionId”, Session.
  SessionID);
sqlCommand.Parameters.Add(sqlParam);
```

- DataReader Object
  - der DataReader bietet einen forward-only, read-only Datenstrom
    - stellt die Ergebnisse einer ausgeführten Abfrage/Kommandos dar
    - eignet sich besonders für Steuerelemente wie Listen- und Kombinationsfelder
  - der DataReader bietet die Möglichkeit, einen Ergebnis-Datentrom von einer Datenquelle zu erhalten
  - Unterstützung von DataBindings in WebForms
  - Einsatzmöglichkeit: Erstellung einer Schleife, die einzelne Zeilen der Daten, die das DataReader-Objekt liefert, mit Hilfe von DataReader.Read() durchläuft
  - Voraussetzung: die Verbindung muss geöffnet sein

```
SqlConnection.Open();
SqlCommand cmd = new SqlCommand(„SELECT Name FROM Production.Product“,
   SqlConnection);
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
   Console.WriteLine(“Name = “ + dr[“Name”]);
}
SqlConnection.Close();
```

- DataAdapter
  - weiß, wie eine Tabelle aus der DB geladen wird und schreibt Änderungen zurück (`Fill(DataSet), Update(DataSet)`)
    - voreingestellte Kommandos sind überschreibbar (insert/update/delete), z.B. um Stored Procedures anzugeben oder Default-Kommandos mit CommandBuilder zu erzeugen

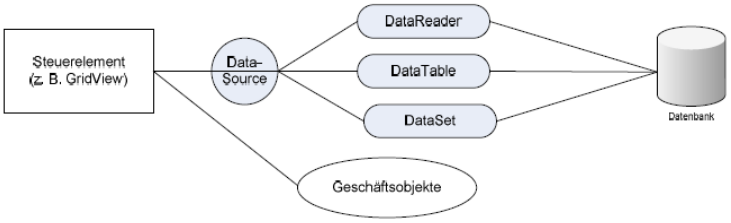
- erlaubt es, ein DataSet aus mehreren Datenquellen zu füllen
- schlägt die Brücke zwischen „connected Layer“ und „disconnected Layer“
- → DataAdapter hat eine starke Bindung zum DataSet

```
SqlDataAdapter sqlAdapter = new SqlDataAdapter(sqlCommand);
DataTable table = new DataTable();
sqlAdapter.Fill(table);
foreach (DataRow row in table.Rows)
      {…}.
sqlAdapter.Update(table);
```

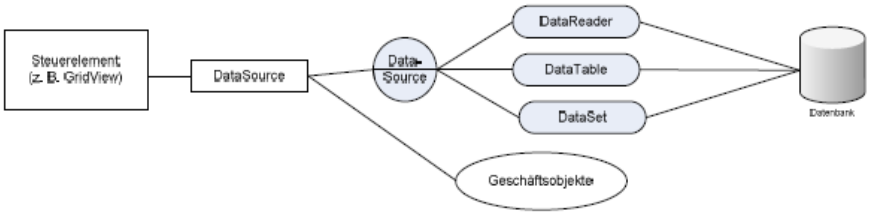
- Steuerung des DataAdapters: SelectCommand, InsertCommand, UpdateCommand, DeleteCommand
- Erzeugung aus Informationen des DataAdapters

- kein Öffnen und Schließen der Verbindung notwendig beim DataAdapter

- datenbezogene Komponenten
  - DataSet
    - ist disconnected von der Datenquelle
    - stellt eine In-Memory Database (IMDB) dar, mit der Möglichkeit SQL-ähnliche Befehle abzusetzen
    - hat keine Kenntnis über die Datenquelle oder deren Eigenschaften
    - kann eine komplexe, relatione Daten-Struktur sein
    - schlägt die Brücke zwischen relationalen und hierarchischen Daten → XML-Unterstützung
    - Strong Typing ist möglich
    - kann sehr einfach per WebService im Internet übertragen werden
  - XML-Unterstützung im DataSet und DataTable (nur eine Tabelle)
    - wird prinzipiell durch 4 Methoden gewährleistet
      - WriteXml(), WriteXmlSchema(), ReadXml(), GetXml()
    - Bearbeiten/Lesen/Konvertieren ist in beide Richtungen möglich (relational ⟷ XML)
    - bis auf kleine Einschränkungen können jede Form von XML-Daten in ein DataSet umgewandelt werden und umgekehrt
- Datenverbraucher
  - Bindung von Datenquellen an Steuerelemente, die Datenmengen ausgeben können
    - Repeater, DataList, DataGrid, TreeView …
  - Bindung von Datenmengen
    - DataSet, DataTable, Array, … (alle Objekte, die die IEnumerable Schnittstelle implementieren)
  - dadurch entfällt das zeilenweise Auslesen der Datenquelle und das zeilenweise Erzeugen der Ausgabe
  - Methode `DataBind()` veranlasst den Bindungsvorgang
  - die Datenbindung bleibt bei einem Postback erhalten
  - Datenbindung
    - direkte Datenbindung
      - direkte Bindung der Daten an das Steuerelement über die Eigenschaft DataSource



- indirekte Datenbindung
  - hier erfolgt ein Verweis auf ein Datenquellensteuerelement, das die Daten enthält und dem Datensteuerelement zur Verfügung stellt

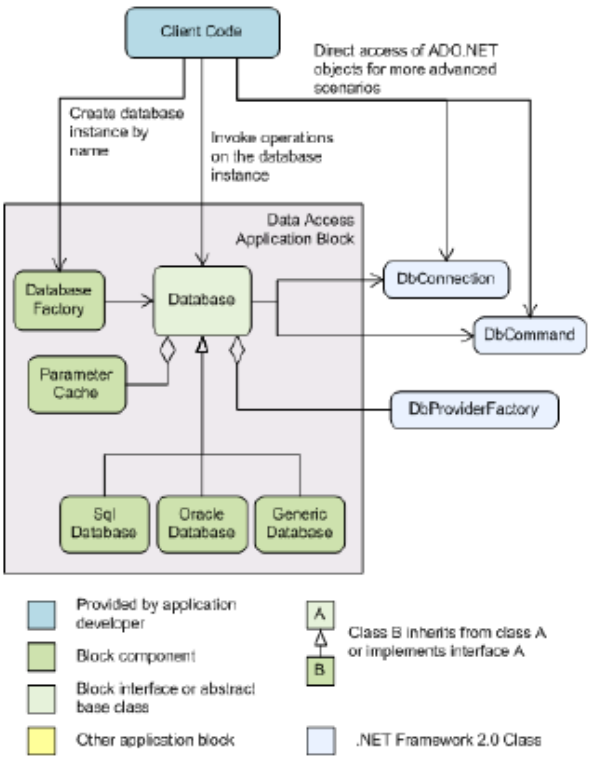



```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
   AllowPaging="true"></asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString=
   "<%$ ConnectionStrings:AdventureWorksConnectionString %>"
   SelectCommand="SELECT LastName FROM Person.Contact ORDER BY LastName">
</asp:SqlDataSource>
```

- DataSource erwartet einen Verweis auf die Datenmenge, die gebunden werden soll
- die Methode `DataBind()` führt die Bindung der Daten aus
- für die DataSource können neben den ADO.NET-Objekten wie DataTable, DataReader etc. auch alle .NET-Klassen, die die Schnittstelle IEnumerable unterstützen, angegeben werden
- das GridView-Steuerelement wird bei einem Postback anhand der im ViewState gespeicherten Informationen neu erstellt
- das GridView-Steuerelement unterstützt das Sortieren, Aktualisieren und Löschen anderer Datenquellen – sie müssen allerdings einen entsprechenden Ereignishandler mit der Implementierung für diese Vorgänge bereitstellen

Data Access Application Block

- Bestandteil der Enterprise Library
  - Bibliothek mit Lösungen für häufig vorkommende Entwicklungsszenarien
- Database Factory (Factory Pattern)
- vereinfachter & einheitlicher Zugriff auf Daten
- bisherige Zugriffsmethode: für jeden Tabellenzugriff werden einzelne Zugriffsmethoden (get, update, …) definiert, die alle gleiche grundlegende Aktionen wie z.B. das Auslesen des connection-Strings beinhalten
- Zugriff mittels Application Block: Bereitstellung einer Schnittstelle für Routineaktionen
- Vorteile
  - entsprechende Codeeinsparung
  - zentraler Zugriff auf die DB → leichtere Wartung und Testdurchläufe
  - Versionsunabhängigkeit von der Datenzugriffs-API
  - datenbankunabhängig
- Connection Management: DAAB verwaltet den Verbindungsstatus (connection pooling)



- Beispiel

```
string sSQL = "SELECT CustomerID, City  FROM Customers WHERE City Like 'M%'";
Database dbNorthwind = DatabaseFactory.CreateDatabase("Northwind");
DBCommandWrapper cmdCust = dbNorthwind.GetSqlStringCommandWrapper(sSql);
IDataReader rdrCust = dbNorthwind.ExecuteReader(cmdCust);
DataSet dsCust = dbNorthwind.ExecutedDataSet(cmdCust);

Webconfig:
<configuration>
<connectionStrings>
   <add name="Northwind" providerName="System.Data.SqlClient" connectionString=
      "server=(local); integrated security=SSPI;Database=Northwind" />
</configuration>
```

- `ExecuteNonQuery()`
  - Einsatz bei Update, Insert oder Delete Statements
  - Rückgabewert ist die Anzahl der betroffenen Datensätze bei Durchführung des Commands

```
Database db = DatabaseFactory.CreateDatabase(„AdventureWordkDb");
DbCommand dbCommand = db.GetSqlStringCommand("UPDATE ShoppingBasket SET Shop-
pingBasket.Quantity = 1 WHERE (SessionId = @SessionId);");
db.AddInParameter(dbCommand, "@SessionId", DbType.Guid, Session.SessionID);
db.ExecuteNonQuery(dbCommand);
```

- `ExecuteScalar()`
  - Einsatz bei einem einzelnen Rückgabewert wie die Anzahl von Datensätzen oder die aktuelle Uhrzeit auf dem Server

```
Database db = DatabaseFactory.CreateDatabase(„AdventureWordkDb");
DbCommand dbCommand = db.GetSqlStringCommand("SELECT COUNT(*) FROM
   Production.Products;");
object returnValue = db.ExecuteScalar(dbCommand);
Console.WriteLine(returnValue);
```

Fehlerbehandlung

Fehlerarten

- Fehler in der Programmierung
- Fehler aus der Fehlbedienung von Anwendungen
- Zugriff auf eine Ressource ist nicht möglich
- Fehler bei Versionswechsel der Entwicklungsumgebung
- Fehler beim Lesen einer Konfigurationsdatei
- Systemfehler, z.B. nicht genügen verfügbarer Arbeitsspeicher
- Fehler beim Analysieren einer Seite, eines Benutzersteuerelements oder einer ähnlichen Datei während der Kompilierung zu einer Assembly

Fehlermeldungen

- gelten nur für .aspx und .html-Seiten
- Problem: Fehlermeldung wird an den Client geschickt und enthält Quelltext
- Webanwendung kann in 2 Modi betrieben werden
  - Debugmodus (Anzeige von Fehlertext und Codezeile, nicht im Produktiveinsatz), Deaktivierung in der web.config: <compilation debug="true">
  - Auslieferungsmodus
- ASP bietet die Möglichkeiten
  - eine Fehlerseite ohne jegliche Fehlerinformationen anzuzeigen
  - eine eigens entwickelte Fehlerseite anzuzeigen

benutzerdefinierte Fehlermeldungen

- Konfiguration in der web.config
  `<customErrors mode="On/Off/RemoteOnly" />`
  - On: Aktivierung der benutzerdefinierten Fehlerbehandlung
  - Off: Deaktivierung und Anzeige von detaillierten Fehlermeldungen
  - RemoteOnly (Standard): zeigt bei lokalen Aufrufen detaillierte, bei entfernten Aufrufen neutrale Fehlerseite
  - allgemeiner Verweis auf eine andere Seite
    `<customErrors mode="On" defaultRedirect="Fehler.aspx" />`
  - Definition für bestimmte Fehlerseiten nach dem customErrorsTag
    `<error statusCode="400" redirect="Fehler/Errorxy.aspx">`
- Fehlerbehandlung auf Seitenebene
  - über das Ereignis Page_Error(), Bsp: Erfragung der letzten Exception

```
protected void Page_Error(object sender, EventArgs e)
{
        Exception ex = Server.GetLastError();
}
```

- Fehlerbehandlung auf globaler Ebene

```
protected void Application_Error(object sender, EventArgs e)
{
        Exception ex = Server.GetLastError();
}
```

Protokollierung von Fehlern

```
void Application_Error(object sender, EventArgs e)
{
        String Message = "Message:\n" + Server.GetLastError().Message;
        String LogName = "Application";
        if (!EventLog.SourceExists(LogName)) {
                EventLog.CreateEventSource(LogName, LogName);
        }
        EventLog Log = new EventLog();
        Log.Source = LogName;
        Log.WriteEntry(Message, EventLogEntryType.Error);
}
```

Anmeldung

Begriffsabgrenzung

- Bei der Authentifizierung handelt es sich um die Überprüfung der tatsächlichen Identität des Endbenutzers. Für diese Überprüfung werden die Anmeldeinformationen des Benutzers (häufig Benutzername und Kennwort) mit einer Liste von bekannten Daten abgeglichen. Wenn die Anmeldeinformationen korrekt sind, authentifizieren wir den Benutzer, andernfalls nicht.
- Die Autorisierung findet nach der Authentifizierung des Benutzers statt und entscheidet über die Berechtigung des Benutzers innerhalb der Anwendung. Dazu kann u.a. gehören, welche Seiten dem Benutzer angezeigt werden.

grundsätzlicher Aufbau der .NET-Authentifikation

1) Während des `AuthenticationRequest` prüft das `FormsAuthenticationModule` die Gültigkeit des Authentifizierungstickets. Dieses kann einerseits in einem Cookie vorhanden sein, oder als cookieless Format in der URL enthalten sein. Ist das Ticket gültig können seine Informationen als Instanz von `FormsAuthenticationTicket` abgerufen werden.

2) Während dem `AuthenticationRequest` versuchen andere Module wie das `UrlAuthorizationModule` Zugriff auf die angeforderte Seite zu bekommen. Wenn der User nicht vorher angelegt wurde oder ihm das Zugriffsrecht auf die Seite nicht zugewiesen wurde, schlägt die Autorisierung fehl.
3) Falls die Autorisierung fehl schlägt, wird die aktuelle Anfrage sofort abgebrochen und an die `EndRequest` Phase weitergeleitet. Das `FormsAuthenticationModule` läuft auch während des `EndRequests` und meldet eine Seite mit dem `Response.StatusCode` 401. Daraufhin leitet das Modul die Anfrage auf die erstellte Login Seite um.

Arten der Anmeldung

1) klassischer Ansatz über Session-Variablen
   - Prüfung der Formulardaten gegen eine DB o.Ä.
   - Setzen einer Session-Variable mit BenutzerID
   - prüfen auf jeder Seite, ob BenutzerID gesetzt ist
   - Vorteil: Nachvollziehbarkeit des Seitenablaufes
   - Nachteil: Prüfung muss auf jeder einzelnen Seite erfolgen (dezentral)
2) formularbasierte Authentifizierung
   - Beschränkung auf bestimmte Verzeichnisse/Benutzer/Gruppen möglich
   - meist genutzte Form der Authentifizierung
   - Konfiguration in der web.config

```
<authentication mode="Forms">
      <forms name="appNamePath" path="/" loginUrl="login.aspx"'
         protection="All""timeout="30">
            <credentials passwordFormat="Clear">
                  <user name="tb" password="test" />
                  <user name="uebung" password="test" />
            </credentials>
      </forms>
</authentication>
```

   - Rechtevergabe

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

   - Ausprägungen für die Rechtevergabe

```
<[allow|deny] users roles verbs />
```

     - users: Name der Nutzer, ein „?“ steht für anonyme Nutzer
     - roles: identifiziert eine Rolle
     - verbs: HTTP verbs wie GET, POST
   - Eventhandler

```
private void Button1_Click(object sender, System.EventArgs e)
{
      if (FormsAuthentication.Authenticate(tbxUsername.Text,
         tbxPassword.Text))
            FormsAuthentication.RedirectFromLoginPage(tbxUsername.Text,
               cbx.PersistLogin.Checked);
      else
            ErrorMessage.Text = "Please re-enter your credentials…";
}
```

   - Informationen über den angemeldeten Benutzer

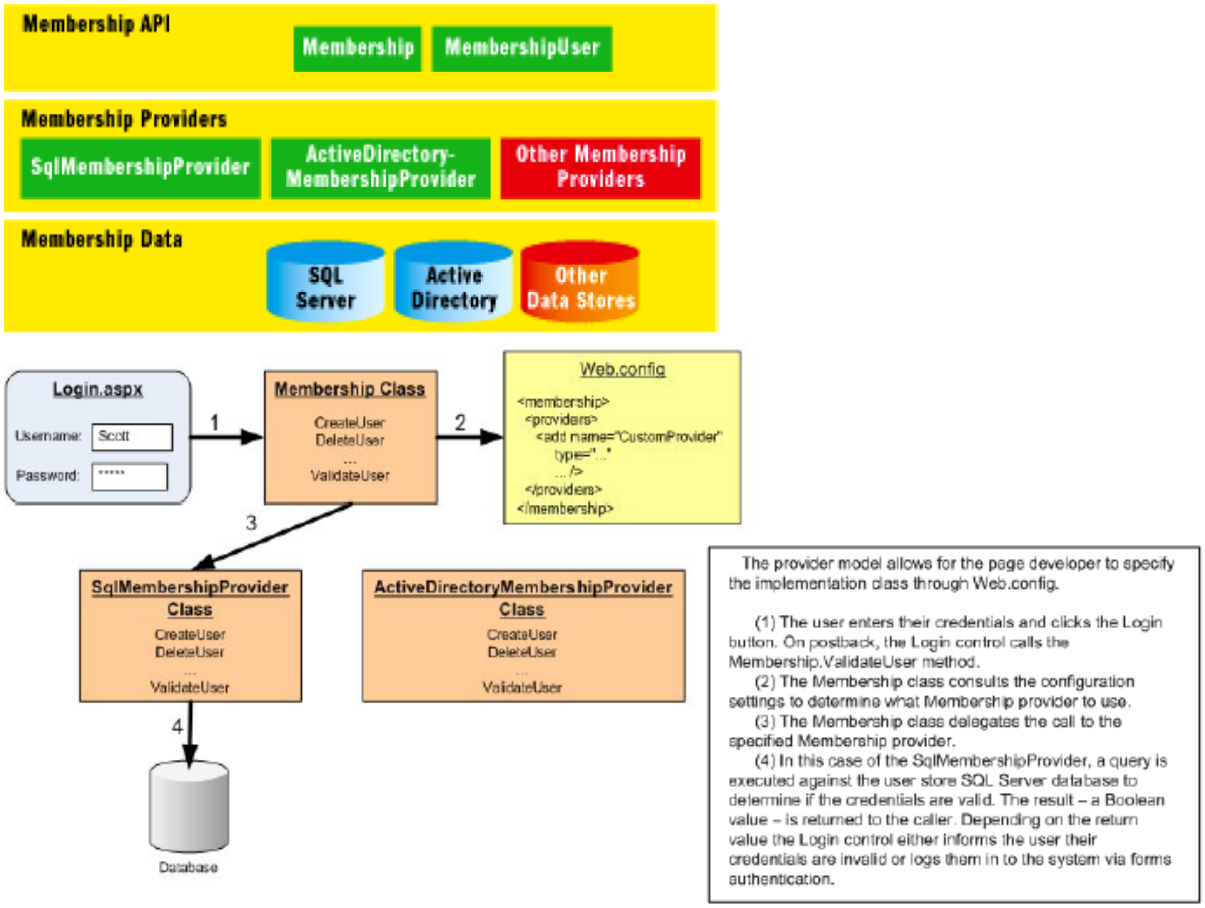
```
Page.User.Identity
```

   - Abmeldung

```
FormsAuthentication.SignOut();
```

- Vorteile
  - alle nicht angemeldeten Benutzer werden automatisch auf die Anmeldeseite geleitet
  - einmalige Implementierung
  - funktioniert auch ohne Einsatz von Cookies
- Nachteile
  - eignet sich nur bei wenigen Benutzern, wenn die Benutzer in der web.config manuell erfasst werden
  - Eingabedaten werden im Klartext übertragen (besser: SSL)

3) Membership Modell



- Prüfung von Benutzerinformationen für Web-Anwendungen
  - Erstellen von neuen Benutzern
  - Speicherung der Mitgliedsinformationen in SQL Server, Active Directory und anderen Datenquellen
  - Authentifizieren von Benutzern
  - Verwalten von Kennwörtern
- neue Controls zur Anmeldung
  - Standardbenutzercode muss nicht für jede Anwendung neu geschrieben werden
- drei verschiedene Anbieter: Membership, Profile, Personalization
- Mitgliedschaftsanbieter
  - Feature erstellt eine Verknüpfung zwischen den Frontend-Funktionen (Anmeldesteuerelemente und Benutzerverwaltungssite)
  - ein Mitgliedschaftsanbieter kapselt den gesamten Datenzugriffscode, der erforderlich ist, um Benutzer und Rollen zu speichern und abzurufen
  - die Komponente kann problemlos mit einem anderen Anbieter ersetzt werden

- benutzerdefinierter Mitgliedschaftsanbieter
  - eigener Code zur Mitgliedschaftsverwaltung
  - Mitgliedschaftsdaten werden in einem benutzerdefinierten Datenspeicher gespeichert
- Konfiguration in der web.config
- Vorteile
  - bereits vorhanden Benutzerverwaltungen können übernommen werden (z.B. Active Directory, SQL DB)
  - einfache Bedienoberfläche
- Nachteil: schwierige Integration von Fremddatenbanken

4) Webserveranmeldung

```
<authentication mode="[Windows/Forms/Passport/None]">
```

1. Windows Authentifizierung: Die Login-Seiten geben die User Information weiter an den ISS. Der ISS wickelt dann die Authentifizierung mit dem jeweils virtuellen Verzeichnis, in dem die Applikation läuft, ab. Dabei nutzt der ISS das Windows OS und die Active Directory Domain Strukturen. So kann der Zugriff auch von externen gespeicherten Windows Einstellungen abhängen. Auf diese Weise kann ein direkter Login auf eine gewünschte Seite durch Windows Einstellungen, die an den Webserver weitergeleitet werden, verhindert werden.
2. Passport Authentifizierung: Ein von Microsoft bereitgestellter zentraler Dienst, der die SSO Möglichkeit und Membership Dienste anbietet. Hierbei werden die Login Zugangsdaten an den Microsoft Passport Server, in dem die Userprofile zentral abgespeichert werden, weitergereicht.
3. Forms: Die Login-Anfrage wird in einem Formular mit Hilfe eines Login-Buttons an den Server geschickt. Anschließend wird auf dem lokalen Client ein Cookie gespeichert, welches dann bei allen weiteren Anfragen mitgesendet wird. Der Client bleibt während einer bestimmten Zeit authentifiziert.

Steuerelemente der Anmeldung

- Login
  - Anzeige der Benutzeroberfläche zur Benutzerauthentifizierung
  - enthält Textfelder für den Benutzernamen und das Kennwort
  - Kontrollkästchen, mit dem Benutzer angeben können, ob sie ihre Identität mithilfe der ASP.NET-Mitgliedschaft speichern möchten
  - kann als eigenständiges Steuerelement auf einer Haupt- oder Homepage oder auf einer gesonderten Anmeldeseite verwendet werden
  - es ist kein eigener Code für die Durchführung der Authentifizierung erforderlich
  - mit dem Authenticate-Ereignis kann benutzerdefinierter Authentifizierungscode hinzugefügt werden
- LoginView
  - zeigt nicht authentifizierten Benutzern einen Link zur Anmeldung und authentifizierten Benutzern einen Link zur Abmeldung
  - Einstellung der Darstellung über die LoginText- und LoginImageUrl-Eigenschaft
- LoginName
  - zeigt den Benutzernamen an, wenn sich der Benutzer mit der ASP.NET-Mitgliedscaft angemeldet hat
  - wenn die Windows-Authentifizierung verwendet wird, zeigt das Steuerelement den Windows-Kontonamen an

- PasswordRecovery
  - sendet dem Benutzer eine E-Mail-Nachricht mit einem Kennwort
- ChangePassword
  - Änderung des Benutzerpassworts
  - bietet auch die Möglichkeit, eine E-Mail-Benachrichtigung über das neue Kennwort zu senden
  - enthält zwei vorlagenbasierte Ansichten
    - ChangePasswordTemplate zeigt die Benutzeroberfläche an, die zum Erfassen der Daten verwendet wird
    - SuccessTemplate definiert die Benutzeroberfläche, die angezeigt wird, nachdem ein Benutzerkennwort erfolgreich geändert wurde
  - funktioniert sowohl mit authentifizierten als auch mit nicht authentifizierten Benutzern

WebServices, Remoting, WCF

WebServices

- Grundlagen WebServices
  - WebServices bilden ein verteiltes System
  - kein GUI-Austausch, sondern Daten-Austausch!
  - nicht nur Datenaustausch, sondern Anwendungsentkopplung!
  - kein dummer Client, sondern echtes Client/Server-Computing
  - Ad-hoc-Kommunikation: wenig Voraussetzungen
  - ein Weg-Dienst (im Gegensatz zur Systemintegration): Diensterbringung für den Dienstnutzer
  - Netz von SW-Diensten
  - verwendbar unabhängig von OS, Programmiersprache, Übertragungsprotokoll
  - kein eigenes Objektmodell
  - im Prinzip nur Textdateien, die zwischen Server und Client verschickt werden
  - bei klassischen Webform-Anwendungen Umwandlung der serverseitigen Daten in HTML, bei Webservices Datenaustausch über XML und SOAP
  - Nachteile bisheriger Web-Anwendungen: manueller Aufwand durch Medienbrüche in der Kommunikation, hohe Prozesskosten, geringe Effizienz
  - Vorteile durch XML WebServices: Automatisierung und Integration von Prozessen, geringere Kosten, größere Effizienz
- Dienstvermittlung und Rollen
  - Dienstanbieter
    - bietet eine Dienstleistung samt Dienstbeschreibung an
    - Implementierung, Betrieb, Wartung
  - Dienstnachfrager
    - sucht nach Dienstbeschreibungen und fordert sie an
    - nutzt einen Dienst
  - Dienstmakler (Trader)
    - speichert und kategorisiert Dienstbeschreibungen
    - bietet Suche nach Diensten an
- Bestandteile
  - Simple Object Access Protocol (SOAP)
    - W3C-Standard, Vorgänger XML-RPC

- Serialisierung in XML, Datentransport mit HTTP/SMTP
- „kein entfernter Methodenaufruf“ → MOM (message-oriented Middleware)
- Aufbau: Envelope, Header, Body

- Web Service Description Language (WSDL)
  - Beschreibung, welche Operationen ein WebService anbietet (IDL)
  - XML-Format
  - abstrakte Beschreibung von Operationen und Nachrichten
  - unabhängig vom Nachrichtenformat und Netzwerkprotokoll
  - Aufbau
    - types: verwendete Datentypen
    - message: Methode mit Parameter/Rückgabewerte
    - portType: enthält n Operationen (= Methoden), ein portType pro Protokoll
    - binding: Bindung zwischen Porttyp, Protokoll, Operation und Datenübergabeformat
    - service: n Bindings, für jeden Port eine URL
- WebService Discovery (DISCO)
  - programmatisches Suchen von WebServices
  - dateibasierter Mechanismus zur lokalen Suche
  - liefert den Verweis auf das WSDL-Dokument
  - muss nicht auf dem selben Server sein wie WSDL-Dokumente
  - Nachfolger: Web Service Inspection Language
- Universal Description, Discovery and Integration (UDDI)
  - einheitlicher Suchdienst für WebServices
  - globales Verzeichnis
  - Suche nach bestimmten Kriterien
  - Anwendungen können zur Laufzeit Funktionalitäten erweitern

- WebDienst mit .NET
  - Voraussetzungen: ISS 6.0, ASP.NET, .NET Framework
  - Implementierung als ASMX-Datei
    - beginnt mit Direktive „@ WebService“
    - optionale WebService/WebMethod-Attribute
    - WebMethoden als öffentliche Methoden der Webdienst-Klassen
  - HTTP, XML, SOAP bleiben verborgen
  - Beispiel

- Service.asmx

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

- Service.cs

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

        //Auskommentierung der folgenden Zeile bei Verwendung von Designkomponenten aufheben
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }

}
```

Remoting

- Abgrenzung zu WebDiensten
  - engere Client/Server-Bindung
  - bessere Eignung bei starker wechselseitiger Kommunikation
  - schlankere Binärprotokolle
  - kein WebServer notwendig
  - vergleichbar mit DCOM, CORBA, Java RMI
  - → Intranet-Lösung (mit Windows Forms)
- Grundlagen
  - entfernte Objektinstanziierung
  - Vorgehensweise
    - Deklaration remotingfähiger Klassen
    - Registrierung der remotingfähigen Klassen
    - Registrierung des Kanals auf Server und Client
    - Registrierung der Remoteklassen in der lokalen Anwendungsdomäne
  - weitere Hinweise
    - Host und Client müssen beide gestartet werden
    - Host darf nach Registrierung nicht beendet werden
  - Bibliothek: System.Runtime.Remoting
- remotefähige Klassen
  - Benutzung herkömmlicher Klassen nur von Clients in der selben Anwendungsdomäne
  - remotefähige Klassen von `MarshalByRefObject` ableiten
  - .NET Framework erzeugt automatisch Proxy
  - → Objekt wird nicht in die Anwendungsdomäne des Clients kopiert
  - → Client hat nur einen Verweis auf das Objekt (Proxy)

```
public class RemoteableClass : MarshalByRefObject { … }
```

- Registrierung remotefähiger Klassen
  - Aktivierung aus anderen Anwendungsdomänen ermöglich
    - remotefähige Klassen
    - URI zur Erzeugung der Objektinstanz (Name des Objekts)
    - Aktivierungsmodus Singleton oder SingleCall (für jeden Aufruf ein Objekt)

```
RemoteConfiguration.RegisterWellKnownServiceType(typeof(RemoteableClass),
   "RemoteObject",WellKnownObjectMode.SingleCall);
```

- Registrierung eines Kanals
  - Kommunikation zwischen Client und Server ermöglichen
    - TCP Channel: leistungfähiger, schlanker
    - HTTP Channel: geeignet für ISS
  - Server: `ChannelServices.RegisterChannel(new TcpServerChannel(8000))`
  - Client: `ChannelServices.RegisterChannel(new TcpClientChannel(8000))`
- Nutzung remotefähiger Klassen
  - Registrierung auf dem Client
    - remotefähige Klasse
    - URI zur Erzeugung der Objektinstanz (Name des Objekts) aus Sicht des Clients

```
RemoteConfiguration.RegisterWellKnownClientType(typeof(RemoteableClass),
   "tcp://localhost:8000/RemoteObject");
```

  - → Instanziierung der Klasse RemoteableClass möglich

- deklarative Registrierung
  - programmgesteuerte Registrierung
    - Änderung der Registrierungsdaten erfordert Kompilierung

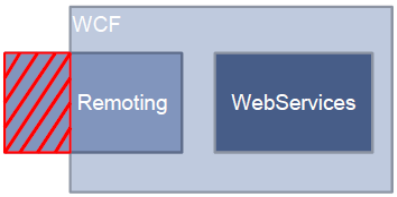
  → deklarative Registrierung mit Config-Dateien
  - Server: `RemotingConfiguration.Configure(„ServerApplication.exe.config");`
  - Client: `RemotingConfiguration.Configure(„ClientApplication.exe.config");`
- Serialisierung von Objekten
  - Vorteil: Objekt wird auf den Client übertragen
  - Objekt muss als serialisierbar gekennzeichnet werden
    - Attribut Serializable, Ableiten nicht notwendig

```
[Serializable]
public class Remoteable Class() { … }
```

- Remote-Methodenaufruf unterscheidet sich nicht vom lokalen Aufruf; nur durch die Registrierung des Channels wird ein Proxy erzeugt, der weiß, wenn eine Instanz gebildet werden soll, dann wird eine entfernte Instanz erstellt

Windows Communication Foundation (WCF)

- Bibliothek zur Entwicklung von servicebasierten verteilten Applikationen
- Bestandteil von .NET 3.0
- Vereinheitlichung der Konzepte von WebServices und Remoting sowie weiterer Ansätze (z.B. MSMQ, DCOM)
- Client/Server-Modell
- verschiedene Protokolle möglich: TCP, SOAP, etc.

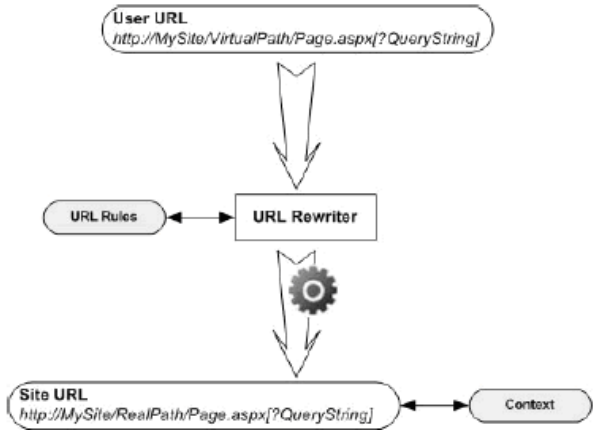


- Server
  - stellt ServiceContract in Form eines Interfaces zur Verfügung
    - ist eine Vereinbarung über die Methoden und deren Parameter
  - kann als Windows-Dienst oder Konsolen-Applikation laufen
  - hört auf einem einstellbaren Server Port (Endpoint)
    - Endpoint kann ausgeschaltet werden, dann läuft das Programm nur noch bei den Clients, die schon einen Proxy erstellt haben
  - verschiedene Protokolle über den gleichen Port möglich (HTTP, TCP, etc.)
- Client
  - fordert den ServiceContract über den Endpoint an
  - erzeugt daraus automatisch eine Proxy-Klasse
  - im Gegensatz zum Remoting muss auf dem Client nicht mehr der gesamte Code verfügbar sein, sondern nur noch eine Proxy-Klasse
  - One-Way/Two-Way-Communication möglich
    - asynchrone Kommunikation über MSMQ möglich
  - Aufruf

```
private void button1_Click(object sender, EventArgs e)
{
       localhost.HelloServiceClient proxy = new
          Client.localhost.HelloServiceClient();
       string resukt = proxy.HelloWorld(textBox1.Text);
       MessageBox.Show(result);
}
```

URL-Rewriting

- Aufgabe: Umleiten von Seiten, Veränderung der URL
- Ziele
  - einfache, kurze URLs statt komplexe, parameterüberladene Anfragen
  - Indizierung durch Suchmaschinen einfacher
- Funktionsweise



- ASP.NET 2.0 Lösung
  - Konfiguration in der web.config
  - nur für statische URLs

```
<urlMappings enabled="true">
      <add url="~/Info/Copyright.aspx" mappedUrl="~/Help/Copyright.aspx" />
      <add url="~/Support/Contacts.aspx" mappedUrl="~/Help/Contacts.aspx" />
</urlMappings>
```

  - Nachteile
    - nicht für das Umleiten von vielen Seiten geeignet
    - Änderungen der URL sind für den Benutzer sichtbar