

Björn Persson

# Introduktion till ASP.NET 2.0

Webbaserad applikationsutveckling  
december 2005

## Om denna sammanfattning

Syftet med denna introduktion är att (förhoppningsvis) ge en första inblick i hur man skapar webbformulär (*web forms*) med ASP.NET 2.0. Introduktionen är på inget sätt uttömmande i ämnet och bygger på sammanfattningen *Introduktion till ASP.NET* (för version 1.x av ASP.NET). Jag börjar med att behandla hur man skapar ASPX-sidor på egen hand (projekt med inbäddad kod) för att sen visa hur man tar hjälp av Visual Studio 2005 (VS2K5, eller bara VS – projekt med *code behind*).

I kapitel med avancerade saker behandlas bl.a. kontrollerna ListBox och DataGrid samt hur man kan använda egna klasser (eller komponenter) från ASPX-sidor. Sist av allt avslutas med ett kapitel med "Frågor och svar".

För att kunna göra (och köra) exempel i introduktionen krävs *Internet Information Server* (IIS) och *.NET Framework 2.0* (med ASP.NET 2.0<sup>1</sup>) på datorn som webbserver ska exekvera på samt (lämpligen) *Visual Studio 2005* för exemplet med *code behind*. (Server och klient kan vara samma dator. ☺ Se sammanfattningen *Installera .NET 2.0* för hur IIS, .NET Framework, m.m. installeras samt lösningar på en del problem som kan uppstå.) Rekommenderat operativsystem för dator med webbserver är Windows 2000 (Professional eller Server), Windows XP Professional (inte Home Edition!<sup>2</sup>) eller Windows 2003 Server (även om det senare kan strula med säkerhetsinställningar...).

Kod har skrivits med typsnitt av fast bredd (Courier New) för att göras mer lättläst samt längre exempel har inneslutits i en ram (se exempel nedan). Kod som måste skrivas på en rad, men som inte får plats på en rad i detta dokument, har radbrutits och en ↵-symbol visar på denna "ofrivilliga" radbrytning.

I Visual Basic (VB) kan programsatser (*statements*) skrivas på flera rader genom att använda understrykningstecknet ("\_").<sup>3</sup> Detta underlättar läsning av kod då man slipper skrolla i sidled för att läsa längre programsatser. Viktigt är att placera ett mellanslag innan understrykningstecknet som i detta exempel:

```
enVariabel = 1 _  
             + 2    'Kommentarer i kod skriv med fet stil
```

Jag är givetvis tacksam för alla konstruktiva synpunkter på sammanfattningens utformning och innehåll.

Spara papper, skriv inte ut sammanfattningen! Ladda ner PDF-filen istället.

Eskilstuna, december 2005

Björn Persson, e-post: [bjorn.persson@mdh.se](mailto:bjorn.persson@mdh.se)

Ekonomihögskolan, Mälardalens högskola

Personlig hemsida: <http://www.eki.mdh.se/personal/bpn01/>

<sup>1</sup> IIS bör installeras före .NET Framework, annars installeras inte ASP.NET! Se sammanfattningen *Installera .NET Framework* för vad som behöver göras om .NET Framework installeras före IIS.

<sup>2</sup> Det finns ingen IIS till Windows XP Home Edition.

<sup>3</sup> I C#, och många andra språk, används ett tecken för att avsluta sats, t.ex. semikolon.

# Innehållsförteckning

<b>1</b>	<b>INTRODUKTION TILL ASP.NET .....</b>	<b>4</b>
1.1	Grundläggand om ASP.NET.....	4
1.1.1	Skapa projekt för inbäddad kod .....	5
1.1.2	Skapa ett projekt med <i>code-behind</i> .....	9
1.1.3	Kontroller i ASP.NET [ KONTROLLERA ] .....	12
1.1.4	Exempel med kontroller.....	19
<b>2</b>	<b>AVANCERAD ASP.NET [ KONTROLLERA ] .....</b>	<b>23</b>
2.1	Kontroller och databaser .....	23
2.1.1	Kontrollen ListBox .....	23
2.1.2	Kontrollen DataGrid .....	25
2.2	Använda egna klasser från ASP.NET [ ATT GÖRA ] .....	31
2.2.1	Klassen SQLProxy.....	31
2.3	Användarkontroller ( <i>user controls</i> ) [ ATT GÖRA ] .....	31
2.3.1	Skapa en enkel användarkontroll [ ATT GÖRA ] .....	31
2.3.2	Skapa en mer avancerad användarkontroll [ ATT GÖRA ] .....	31
2.3.3	Infoga en användarkontroll med egenskaper [ ATT GÖRA ] .....	31
<b>3</b>	<b>FRÅGOR OCH SVAR .....</b>	<b>32</b>
3.1	Datum som fil senast ändrades (skrevs till) .....	32
3.2	Använda Visual Studio.NET (2003) .....	32
3.2.1	Öppna ett existerande projekt.....	32
3.2.2	Webbprojekt med flera utvecklare [ ATT GÖRA ] .....	32
<b>4</b>	<b>LITTERATUR .....</b>	<b>33</b>

# 1 Introduktion till ASP.NET

Precis som vi kan använda *Active Server Pages* (ASP) i COM/COM+ så kan vi använda ASP.NET i .NET. Skillnaden är bl.a. att ASP.NET numera är kompilerad kod i motsats till ASP:s tolkade skriptkod. ASP.NET (liksom *.NET Framework*) finns i tre version: 1.0, 1.1 och 2.0. 1.0 måste installeras manuellt liksom 2.0, men 1.1 installeras (oftast) när vi installerar service pack 2 (SP2) för Windows XP.

Annat som skiljer ASP.NET från ASP är att filändelsen är .ASPX (istället för .ASP).

Syftet med den korta introduktion är att (förhoppningsvis) ge en första inblick i hur man skapar webbformulär (*web forms*) med ASP.NET 2.0. Introduktionen är på inget sätt uttömmande i ämnet. Jag börjar med att behandla hur man skapar ASPX-sidor på egen hand (projekt med inbäddad kod) för att sen visa hur man gör det mesta med Visual Studio (VS – projekt med *code behind*).

För att kunna göra (och köra) exempel i introduktionen krävs *Internet Information Server* (IIS) och *.NET Framework 2.0* på datorn som webbserver ska exekvera på samt (lämpligen) *Visual Web Developer 2005 Express Edition* (WebDev, som kan laddas ner gratis från Microsoft) eller *Visual Studio 2005* (VS2K5, eller bara VS, som inte är gratis) för exempel med *code behind*. (Server och klient kan vara samma dator. ☺) Rekommenderat operativsystem för dator med webbserver är Windows 2000 (Professional eller Server), Windows XP Professional (inte Home Edition!<sup>4</sup>) eller Windows 2003 Server (även om det senare kan strula med säkerhetsinställningar...).

Beskrivningar i denna sammanfattning bygger på att vi använder Visual Web Developer 2005 Express Edition, men bör fungera på ett liknande sätt i (fulla) Visual Studio 2005. En fördel med WebDev är att *Intellisense*<sup>5</sup> (*auto completion*) fungerar även med inbäddad kod numera (vilket den inte gjorde i tidigare versioner av Visual Studio.NET).

---

## 1.1 Grundläggand om ASP.NET

ASP.NET 2.0 fungerar på många sätt som tidigare versioner av ASP.NET, d.v.s. det grundläggande är det samma. ASPX-sidor kan skapas på två sätt: med inbäddad kod eller *code-behind*. I det första fallet placeras koden direkt i ASPX-sidans HTML-kod, d.v.s. inbäddad inuti HTML-koden. Detta sätt påminner om hur ”vanliga” ASP-sidor (ASP v.3 eller tidigare) skapas. Med *code-behind* så placeras programkoden (VB eller C#) i en separat fil som inkluderas längst upp i ASPX-filen. I båda fallen kompileras koden till en DLL-fil, d.v.s. slutresultatet blir ”det samma”. Observera att ASPX-filerna **alltid** måste distribueras till produktionsserver, oavsett om vi använder inbäddad kod eller *code-behind*.

När vi skapar webbprojekt så har vi ingen möjlighet att ändra sökvägen till var lösningsfilen (.SLN-filen) ska placeras – vi kan endast ange URL:en till webbapplikationen<sup>6</sup>. Vi kan dock ändra standardinställningar i VS innan vi skapar projektet.<sup>7</sup> Alla filer (förutom lösningsfilen) placeras på webbservern.

---

<sup>4</sup> Det finns ingen IIS till Windows XP Home Edition.

<sup>5</sup> *Intellisense* är Microsofts namn på *auto completion*, d.v.s. funktionen som bl.a. visar på vilka metoder som finns i objekt, m.m..

<sup>6</sup> En webbapplikation är alla filer i en mapp och dess undermappar på en webbserver. ASP.NET-filer som ligger i dessa mappar delar på minne, d.v.s. applikations- och sessionsobjekt är gemensamma.

<sup>7</sup> Detta har redan gjorts på alla datorer i Ekonomihögskolans datorsalar!

Som standard så är `DEFAULT.XXX` (ersätt `XXX` med `HTM`, `ASP` eller `ASPX`) namnet på "standardfiler" (eller indexfiler) i IIS.<sup>8</sup> En "standardfil" är den fil som laddas om man inte anger ett filnamn i en URL. T.ex. så är nedanstående URL:er samma i IIS:

```
http://www.eki.mdh.se/studentguide/  
http://www.eki.mdh.se/studentguide/default.htm
```

### 1.1.1 Skapa projekt för inbäddad kod

Nedan beskrivs hur vi använder WebDev för att skapa filer, m.m. för exempel. Om vi använder en vanlig texteditor så beskrivs detta kort inom hakparenteser, [ och ], där det gäller.

För att använda inbäddad kod så skapar man lättast ett tomt projekt samt "vanliga" HTML-filer (som vi byter filändelse på till `.aspx`).

1. Skapa ett nytt projekt genom att välja **New Web Site...** från File-menyn och välj sen **ASP.NET Web Site**. Resten av inställningarna i dialogrutan New Web Site beror bl.a. på var webbserver finns och hur vi väljer att kommunicera med (d.v.s. kopiera till/från) webbserver (via filsystem, HTTP eller FTP). I denna sammanfattning används HTTP att kommunicera med webbserver och därför väljs **HTTP** i listrutan vid Location samt en URL till webbserver (t.ex. `http://<adress-till-server>` / InbaddadKod – ersätt `<adress-till-server>` med `localhost` eller adress till webbserver<sup>9</sup> och InbaddadKod med sökväg på webbserver).

I projektet finns två filer, `Default.aspx` och `web.config`. Eftersom vi ska använda inbäddad kod i exemplet så måste vi ta bort `ASPX`-filen och lägga till en ny (för inbäddad kod).

2. Högerklicka på filen `Default.aspx` och välj Delete samt klicka sen på OK för att bekräfta radering av filen.
3. Lägg till en ny webbsida (webbformulär) genom att högerklicka på projektet (rad i fet stil som börjar med http) samt välj **Add New Item...**
4. Välj **Web Form** i listrutan Templates: och byt namn på fil till `Default.aspx` samt kontrollera att kryssrutan Place code in separate file **inte** är markerad.

[... eller skapa en ny fil med namnet `Default.aspx` under webbserverns mappstruktur. ]

5. Öppna filen (om den inte öppnades automatiskt) och byt till källkodsläge (genom att klicka på fliken Source längst ner i editorn).
6. Lägg till nedanstående kod längst upp i filen (utöver HTML-kod som behövs för webbsidor, d.v.s. HTML- och BODY-taggar).

```
<%@ Page Language="vb" %>
```

Koden ovan talar om vilket språk som skriptkoden har i webbsidan (inom `<% %>`-taggarna).

Det är nu dags att lägga till dynamisk kod. ☺

<sup>8</sup> Andra webbservrar (t.ex. Apache) använder filnamnet `INDEX.XXX` (ersätt `XXX` med `HTM` eller `HTML ...` eller `PHP`).

<sup>9</sup> På Ekonomihögskolan så används t.ex. Kompis-servrarna och Macbeth för webbapplikationer.

### 1.1.1.1 Lägga till dynamisk kod

Dynamisk kod är vad jag kallar kod som exekveras för att generera HTML-kod dynamiskt, d.v.s. skriptkod. Skriptkod kan infogas på ett antal olika sätt. Det klassiska (och det som kan användas för att ”infoga” det dynamiska i HTML-koden) är som med ASP v.3 (eller tidigare), d.v.s. mellan `<% %>`-taggarna. Denna typ av taggar bör dock undvikas till förmån för andra taggar, bl.a. etiketter (*labels*). Jag återkommer till denna andra typ av taggar... Först ett exempel med taggarna vi bör undvika. ☺

6. Lägg till nedanstående kod någonstans mellan HTML-taggar `<body ...>`<sup>10</sup> och `</body>`. Eftersom vi skriver ut HTML-koden dynamiskt så måste vi även skriva ut eventuella HTML-taggar också. Därför skriver vi ut stycketaggarna (`<p>`), lite ledtext samt aktuellt datum och tid.

```
<%  
    Response.Write("<p>Hej, tiden är " & DateTime.Now() & "</p>")  
%>
```

### 1.1.1.2 HTML-formulär

I ”vanlig” ASP krävdes en hel del logik för att hantera HTML-formulär. Men i ASP.NET har det blivit lättare – mycket av logiken sköts av ASP.NET. För att kunna hantera denna logik så har Microsoft (bl.a.<sup>11</sup>) introducerat något som kallar webbserverkontroller (eller bara webbkontroller, *web server controls*). I HTML-koden lägger vi till prefixet `asp:` först i taggen samt använder attributet och värdet `runat="server"` för att tala om att servern ska hantera logiken.

I nedanstående exempel visas först HTML-koden för en ”vanlig” textruta och sedan motsvarande webbkontroll.

```
<input name="HtmlText" type="text" value="Text i textruta">  
<asp:TextBox id="TextBox1" runat="server">Text i textruta</asp:TextBox>
```

HTML-koden (som skickas till webbläsare) för webbkontroll ovan blir följande:

```
<input name="TextBox1" type="text" value="Text i textruta" id="TextBox1" />
```

För att webbformulär (*web forms*) ska fungera så måste vi placera webbkontrollerna inom formulärtaggen. I formulärtaggen måste (eller bör) vi även använda attributen `id` och `runat` enligt exempel nedan.

```
<form id="FormNamn" runat="server">  
    <!-- Kontroller i formuläret -->  
</form>
```

En viktig detalj är att formulär i ASP.NET oftast skickas till den webbsida där formuläret finns, vilket är skälet till att vi kan utelämnas ACTION-attributet i FORM-taggen. ASP.NET

<sup>10</sup> Det kan finnas ett eller flera attribut i inledande body-taggen...

<sup>11</sup> Med bl.a. menar jag att det finns andra typer av kontroller – kontroller som jag inte kommer behandla här. ☺

lägger för övrigt till attributen ACTION och METHOD. D.v.s. (händelse-)hantering av formulär sker i samma webbsida som formuläret finns i. (I ASP kunde man t.ex. ha en ”vanlig” HTML-sida med formulär och en ASP-sida för att hantera data från formuläret.)

### 1.1.1.3 Hantera händelser från HTML-formulär

Tanken med de nya webbkontrollerna och ASP.NET är att utvecklingen av ett användargränssnitt för webben ska påminna om det för ”vanliga” Windows-applikationer. D.v.s. formuläret ska bygga på händelser (så som *Click*, d.v.s. att användare klicka på en knapp). Det är främst en typ av händelse, som genereras av webbkontroller, och händelsen *Page\_Load* som är intressant (men inte de enda...).

Händelsen *Page\_Load* kan t.ex. användas för att fylla webbkontroller med data från databaser. Vi behöver dock inte hantera saker som att se till att markerat alternativ i listrutor är det som användare valt när formulär skickades.

Metoderna som ska hantera händelserna, bl.a. *Page\_Load*, placeras inom en script-tagga – oftast innan öppnande html-tagga. Eventuellt resultat från exekvering, som ska visas i webbsida, placeras i variabler. På detta sätt separeras logik från presentation.

```
<SCRIPT LANGUAGE="VB" RUNAT="server">
  'Här placeras händelsehanterare och andra metoder
</SCRIPT>
<HTML>
  <!-- HTML-koden för webbsidan -->
```

Alla fördefinierade metoder och metoder för hantering av händelser (vad jag kan se nu) har två parametrar: källa för händelse (av typen *Object*) och argument från händelse (av typen *EventArgs*).<sup>12</sup> I exempel nedan visas metoden *Page\_Load( )* för att hantera händelsen *Page\_Load*.

```
<SCRIPT LANGUAGE="VB" RUNAT="server">
  Sub Page_Load(Source As Object, E As EventArgs)
    ' T.ex. initiering av kontroller med data från databas
  End Sub
</SCRIPT>
```

Ett typiskt exempel på en händelse är när användaren klickar på en knapp. I HTML-formulär är det främst Submit-knappen för formulär som gäller. I nedanstående formulär finns en textruta, en knapp och en av nyheterna i ASP.NET – en etikett<sup>13</sup>. Tanken är att vi ska skapa en händelsehanterare för knapptryckningen som läser texten från textrutan och skriver ut den i etiketten.

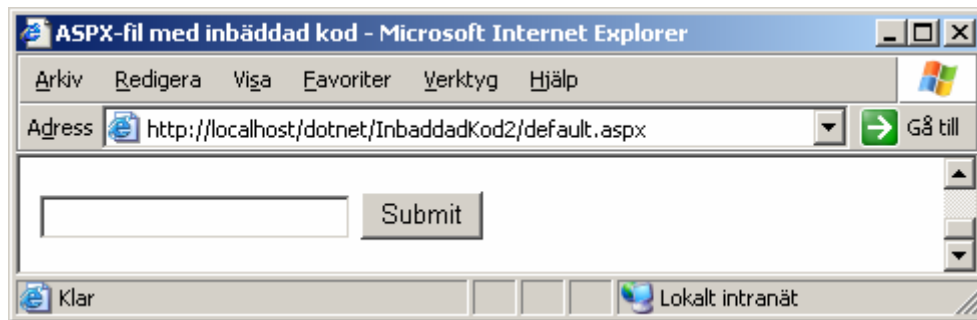
Koden för formuläret (i den resulterande webbsida) visas nedan.

```
<form ID="frmTest" RUNAT="SERVER">
  <asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
  <asp:Button ID="Button1" onClick="Control_Click" TEXT="Submit"
    RUNAT="SERVER" />
```

<sup>12</sup> Vi behöver inte använda oss av objekten i parametrarna... men dom måste finnas där. ☺

<sup>13</sup> En etikett är egentligen bara en webbkontroll som ”konverteras” till vanlig text när ASP.NET ”exekverar” kontrollen. Men när vi designar webbsidan med VS.NET så ”agerar” den som en etikett i ett ”vanligt” Windows-formulär.

```
<asp:Label ID="Label1" RUNAT="SERVER" />
</form>
```



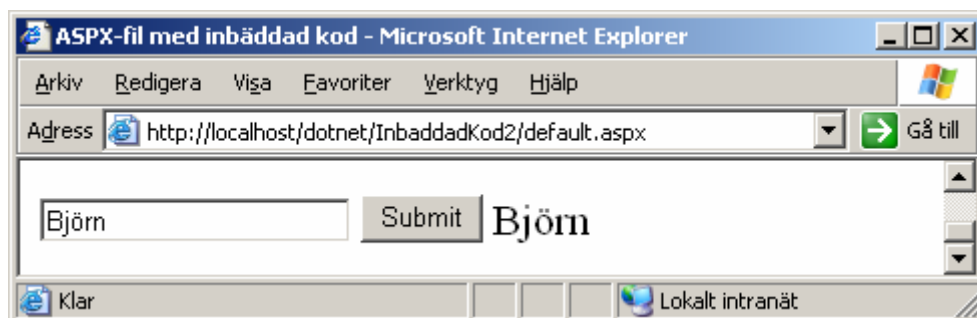
Som vi ser av koden ovan så har alla webbkontroller attributet och värdet `RUNAT="server"`. Detta talar om för ASP.NET att den ska hantera dess logik. Vi använder även attributet `ID` för att namnge webbkontrollerna.<sup>14</sup> På detta sätt kan vi använda värdet på attributet `ID` för att referera till kontrollerna i skriptkod (se mer i koden för händelsehanteraren nedan).

Textkontroller är annars "inget speciellt" – om vi vill att text som standard ska placeras i textrutan (då webbsida visas första gången) så kan vi placera texten mellan den öppnande och avslutande taggen. Knappen skiljer sig en aning från en "vanlig" HTML-knapp. Vi har i attributet `onClick` talat om viken metod i webbsidan som ska exekveras när formuläret skickas tillbaka till webbsidan-/servern. Definitionen för metoden visas i koden nedan. Etiketten är, som sagt, en ny typ av kontroll. Egentligen är det bara en plats i dokumentet vi kan referera till som ett namn (se kod nedan).

Koden för att hantera klickhändelsen (och resulterande webbsida) visas nedan.

```
<SCRIPT LANGUAGE="VB" RUNAT="server">
  Sub Control_Click(sender As Object, e As System.EventArgs)
    Label1.Text = TextBox1.Text
  End Sub
</SCRIPT>
```

Som vi ser i koden så använder vi namnen på kontrollerna (eller värdet på attributet `ID`) för att referera till kontrollerna. I koden ovan så tilldelar vi innehållet i textrutan (`TextBox1.Text`) till etiketten (`Label1.Text`).



<sup>14</sup> Inte alla HTML-taggar har attribut som `NAME`, etc.. Men genom att använda attributet `ID` (som alla HTML-taggar har) så kan vi ge alla kontroller unika namn. (Värden i attributet `ID` ska enligt HTML-specifikationen vara unika för alla kontroller.)



### 1.1.2 Skapa ett projekt med *code-behind*

I detta exempel ska vi göra samma sak som i exemplet ovan (med inbäddad kod). Skillnaden ligger i att vi ska använda oss av VS.NET för att göra det mesta av jobbet åt oss. Vi kommer (förhoppningsvis ☺) se hur lätt det är att skapa ett webbformulär och att det påminner mycket om sättet vi gör det i ett "vanligt" Windows-formulär.

1. Skapa ett projekt av typen ASP.NET Web Site (t.ex. `http://<adress-till-server>/CodeBehindVB`). Här skapas ett antal filer, bl.a. en som heter `Default.aspx` (en fil som vi inte behöver radera eftersom den bygger på *code behind*).
2. Högerklicka på filen i Solution Explorer och välj **Set As Start Page** i menyn som visas. (Detta steg behövs för att vi ska kunna "exekvera" applikationen som ett "vanligt" Windows-program, t.ex. genom att klicka på Kör-knappen.)

Om vi öppna filen (om den inte öppnades automatiskt) och byter till HTML-läge (genom att klicka på fliken Source längst ner i editor) så ser vi att det redan finns ett Page-direktiv.

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
```

Även här talar koden ovan om vilket språk som skriptkoden har i webbsidan (inom `<% %>`-taggarna). De tre nya attributen är

- `AutoEventWireup` – (inte viktigt nu ☺ – sök på *AutoEventWireup attribute* i MSDN för förklaring – klicka på `@Page`, eller *ASP.NET Web Server Control Event Model*, om Express Edition).<sup>15</sup>
- `CodeFile` – fil som innehåller kod (med bl.a. händelsehanterare).
- `Inherits` – klass som aktuell ASPX-sida ärver från (finns oftast i filen ovan).

Vill vi lägga till fler webbsidor så högerklickar vi på projektet (inte lösningen!) och väljer **Add New Item....** I dialogrutan som visas väljer vi **Web Form**, anger namn vi vill att webbsida ska ha och klickar OK. Kontrollera även att kryssrutan *Place code in separate file* är förbockad.

#### 1.1.2.1 Positionering av element i WebForms [ SKRIV OM ]

När vi använder *code behind* och WebForms så kan vi designa webbsidorna som om det vore ett "vanligt" Windows-formulär. D.v.s. vi kan placera etiketter och andra kontroller exakt där vi vill ha dem. (Observera att resulterande webbsida kan se mycket annorlunda ut – allt beroende på tillverkare av och version på webbläsare! <sup>16</sup>) Detta är "jättefint"... men det fungerar främst med Internet Explorer, liksom en del eventuella andra saker.

För att detta ska fungera så används attributet `MS_POSITIONING` med värdet `GridLayout` i `body`-taggen. Detta instruerar VS.NET att använda *Cascading Style Sheets* (CSS)<sup>17</sup> för att positionera HTML-elementen när vi designar formulär. Vill vi att en webbsida ska fungera som i "vanlig" HTML så använder vi istället värdet `FlowLayout` för attributet `MS_POSITIONING`. I nedanstående bild har en etikett placerats slumpmässigt i webbsidan.

<sup>15</sup> Eftersom VS vill ha den där så lämnar vi lämpligen attributet som det är.

<sup>16</sup> Bara för att jag tänkte behandla detta ämne så fungerade det strålande i både Netscape v.4.79 och Opera v.6.05...☺ Men jag vill trycka på att resultatet inte går att förutsäga i alla webbläsare (och då talar jag inte om textbaserade Lynx ☺).

<sup>17</sup> CSS kan användas för att ändra utseende på HTML-element men även för att positionera elementen.



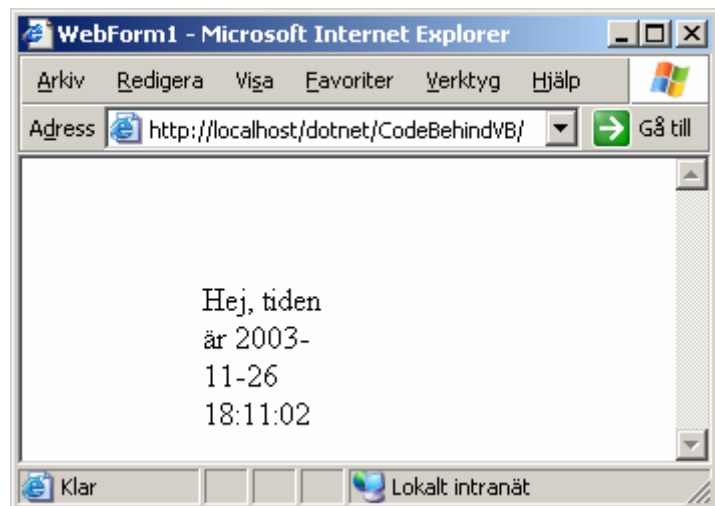
Den största skillnaden mellan GridLayout och FlowLayout är att vi lättast använder etiketter för utskrift med den första och som "vanligt" med den senare. Med "vanligt" menar jag som i vilken WYSIWYG<sup>18</sup>-editor för HTML som helst, t.ex. FrontPage.

Det är nu dags att lägga till dynamisk kod. ☺

### 1.1.2.2 Lägga till dynamisk kod

Även med *code behind* kan vi använda `Response.Write` för att skriva ut till webbsidan. Men vi använder lämpligen etiketter till detta – i stort sett ett krav om vi använder GridLayout.

3. Växla till designläget (klicka på fliken Design längst ner i editor). Dra och släpp en etikett (*Label*) från verktygsmenyn Toolbox (till vänster i VS).
4. Dubbelklicka på bakgrunden för att öppna kodfönstret och för att skapa metoden `Page_Load()`. Fyll i koden nedan i metoden.



```
Label1.Text = "Hej, tiden är " & DateTime.Now
```

Bygg projektet och "kör" applikationen (genom att t.ex. välja **Start** från Debug-menyn). Om du får ett meddelande om att det inte går att "debugga" på avstånd (*remote*) så svarar du OK/Ja/Yes (för att stänga av "debugging" för projektet) och kör igen (om det behövs).<sup>19</sup> Resultatet bör bli något i stil med skärmdump till höger.

<sup>18</sup> What-You-See-Is-What-You-Get.

<sup>19</sup> Jag har haft lite problem att få detta att fungera... Jobbar bl.a. på att få det att fungera på servern Kompis...

För att fortsätta utveckla webbapplikationen måste du stänga webbläsaren! (Det går att öppna en webbläsare och ”surfa” till webbsidan för att kunna växla mellan webbläsare och VS.NET. Glöm bara inte att bygga/kompilera applikationen innan du växlar till webbläsaren.)

### 1.1.2.3 HTML-formulär

Även med *code behind* så använder vi webbkontroller.

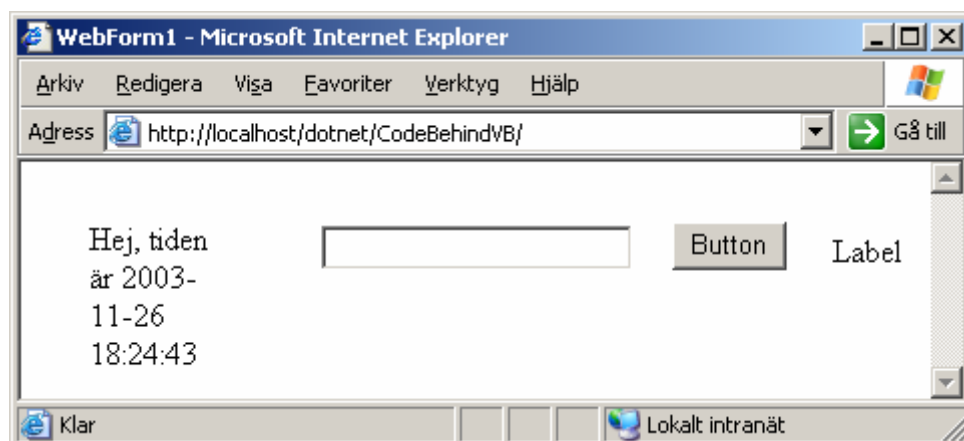
### 1.1.2.4 Hantera händelser från HTML-formulär

Återigen så hanteras formulär på samma sätt som med inbäddad kod, och åter igen använder vi VS för att göra mycket av jobbet... ☺

5. Dra och släpp en textruta, en knapp samt ytterligare en etikett på webbsidan (gärna i den ordningen och i rad som förra exemplet).
6. Dubbelklicka på knappen för att skapa händelsehanteraren för knappen (antagligen med namnet `Button1_Click()`) och fyll i nedanstående kod<sup>20</sup> i metoden.

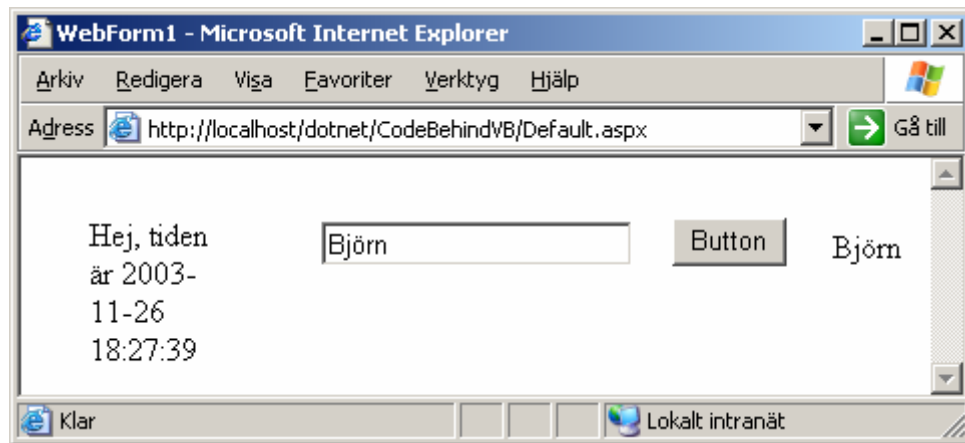
```
Label2.Text = TextBox1.Text
```

Bygg projekt och ”kör” applikationen (eller växla till en webbläsare och ”surfa” till webbsidan). Resultatet bör se ut något i stil med nedanstående skärmdump.



Prova att fylla i ditt namn i textrutan och klicka på knappen. Resultatet bör bli något i stil med nedanstående skärmdump.

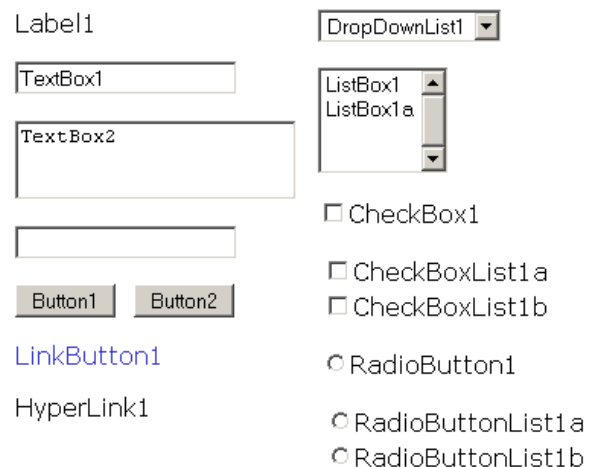
<sup>20</sup> Om webbkontrollerna har fått ett annat namn (om du ”lekt” lite) så får du anpassa namnen på kontrollerna. Har du följt beskrivningen ”till pricka” så bör det fungera. ☺



### 1.1.3 Kontroller i ASP.NET [ KONTROLLERA ]

Det finns ett antal olika sätt att skapa HTML-kontroller. I denna introduktion har jag valt att använda endast ett sätt – webbserverkontroller (eller ASP-kontroller, se bild till höger för exempel). Övriga typer är HTML-serverkontroller (*HTML server controls*), valideringskontroller (*validation controls*) och användarkontroller (*user controls* – kontroller som vi själva kan skapa), vilka ni får läsa om någon annanstans ☺.

Nedan beskrivs och visas ASP.NET-koden för webbkontroller och sedan koden i *code behind* (kallat kodfilen) för kontrollerna som visas i bilden ovan. En skillnad mot ASP.NET 1.x är att vi inte längre kan se variabler som deklarerats för motsvarande kontroller i webbformuläret (ASPX-filen). Sist av allt visas den resulterande HTML-koden (som webbkontrollerna resulterar i) som skickas till webbläsaren, främst för ”oss” som kommer från äldre teknologier som ASP och PHP för att (förhoppningsvis) lättare förstå relationen mellan webbkontroller och HTML-kontroller.



#### 1.1.3.1 Kodan för kontroller

I ASP-koden (eller snarare ASP.NET-koden) så har taggar för webbserverkontrollerna ett antal saker gemensamt:

- Kontrollerna börjar med `<asp:Kontrolltyp` (ersätt `Kontrolltyp` med typ av kontroll som önskas, t.ex. `TextBox`, `Button` eller `ListBox`).
- Attributet `id` används för att namnge kontrollen, d.v.s. unikt kunna identifiera dem. Detta namn används i kod som variabel namn för att manipulera kontrollerna.
- Attributet `runat`, med värdet `"server"`, som talar om för ASP.NET att koden ska tolkas ("exekveras") av server innan webbsida skickas till klient.
- Attributet `text` som innehåller texten som kontrollen ska visa (om någon). Detta attribut kan ibland utelämnas samt dess värde placeras mellan öppnande och avslutande tagg.

- Kontrollerna måste ha en avslutande tagg, `</asp:KontrollTyp>`, eller den öppnande taggen avslutas i XHTML-stil med ett snedstreck (`<asp:Kontrolltyp ... />`).

### 1.1.3.2 Egenskaper för kontroller

Egenskaper (*properties*) för kontroller avser främst kontrollers utseende, däribland text, men även dess beteende, d.v.s. hur de ska reagera vid t.ex. tangentbordstryckningar (kortkommando<sup>21</sup>, m.m.) och musrörelser. Dessa egenskaper kan ändras vid design i egenskapsfönstret (*Properties*) eller via kod, d.v.s. med objekten som motsvarar kontrollerna.

Egenskaperna för kontrollerna motsvarar till viss del attributen i ASP.NET-taggar. M.a.o. så har kontrollerna ett antal gemensamma egenskaper, bl.a.

- `Text` – text som ska visas i kontrollen.
- `ID` – namn på kontrollen.<sup>22</sup>
- `Visible` – om kontroll ska visas i den resulterande webbsidan.

### 1.1.3.3 Partiella klasser

I ASP.NET 2.0 används numera, vad som kallas, partiella klasser. Klasser för webbformulär delas upp i två delar: en del som innehåller kod som utvecklare skriver och en del som genereras av ASP.NET. Detta, anser jag, är en förbättring av hur ASP.NET fungerar då vi inte behöver ”stöka till” koden som vi måste redigera. Det innebär också att vi ”tappar kontrollen”, något som några av oss inte är så förtjusta i. ☺

Denna separation av genererad och egenskriven kod innebär bl.a. att vi inte längre ser deklARATIONER av variabler för motsvarande kontroller i ASP.NET-koden.

### 1.1.3.4 Etiketter

Etiketter (*labels*) är egentligen bara text, men med webbserverkontroller så är det objekt som vi kan ändra egenskapen `Text` för att ändra vad som visas för besökaren. D.v.s. det är den kontroll som är lämpligast för att visa (icke-redigerbar) data för besökare.

ASP-koden för etiketter i sin enklaste form visas nedan (som den visas i bild med exempel på kontroller ovan).

```
<asp:Label ID="Label1" runat="server">Label1</asp:Label>
```

Resulterande HTML-kod för etiketter blir följande:

```
<span id="Label1">Label1</span>
```

<sup>21</sup> Kortkommando är tangentbordstryckningar som oftast involverar tangenterna Shift, Ctrl och/eller Alt i kombination med en annan tangent.

<sup>22</sup> Denna egenskap kan kanske kännas lite överflödig eftersom dess värde bör motsvara namnet på objektets variabel. Men egenskapen kan vara användbar om vi skulle loopa över vektor med kontroller t.ex..

### 1.1.3.5 Texttrutor

Texttrutor är den huvudsakliga kontrollen för att hämta data från besökare av webbplatser. Det finns tre typer av texttrutor, motsvarande de i HTML-formulär, vars typ avgörs av attributet `TextMode` och dess värde:

- Avsaknaden av attribut (och värde) – enradig textruta.
- `"MultiLine"` – flerradig textruta.
- `"Password"` – lösenordsruta, d.v.s. textruta där tecken visas med asterisk.

ASP-koden för texttrutor i sin enklaste form, enradig textruta, och flerradig textruta visas nedan.

```
<asp:TextBox id="TextBox1" runat="server">TextBox1</asp:TextBox>
<asp:textbox id="TextBox2" runat="server"
  TextMode="MultiLine">TextBox2</asp:textbox>
```

Resultande HTML-kod för de tre typerna av texttrutor blir följande:

```
<input name="TextBox1" type="text" value="TextBox1" id="TextBox1" />
<textarea name="TextBox2" id="TextBox2">TextBox2</textarea>
<input name="TextBox3" type="password" id="TextBox3" />
```

### 1.1.3.6 Knappar ('vanliga')

Knappkontrollen motsvarar submit-knappar i HTML-formulär, en kontroll som bl.a. har följande egenskaper: `CommandName` och `OnClick`. Den senare kan användas för att koppla en metod att hantera knappens klickhändelse (se mer nedan när kod beskrivs) och den första för att ge kommandot ett namn (se nedan för hur det kan användas).

```
<asp:button id="Button1" runat="server" Text="Button1"></asp:button>
<asp:Button id="Button2" runat="server" Text="Button2" CommandName="cmdButton2"
  OnClick="Button2_Click"></asp:Button>
```

Deklaration av variabel vid *code behind* blir enligt nedan.

```
Protected WithEvents Button1 As System.Web.UI.WebControls.Button
```

Ett sätt att hantera klickhändelsen för knappen är att lägga till nedanstående metod. I VB.NET används det reserverade ordet `Handles` efter metodens signatur för att tala om vilken, eller vilka, eventuell händelser som metoden ska hantera (svara på).<sup>23</sup> Om metoden ska hantera flera händelser så separeras de med kommatecken (se mer nedan). Som argument till metoden skickas objekt som genererade händelse i argumentet `sender` samt eventuella övriga värden i Event-objektet i argumentet `e`.<sup>24</sup>

<sup>23</sup> I C# så sker "detta sätt" på ett något annorlunda sätt – med egenskapen `Click` (som inte finns i VB.NET!).

<sup>24</sup> I detta fall så ska inget skickas i Event-objektet, men det måste vara med.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    'Kod som ska exekveras när besökare klickar på knapp  
End If
```

Ett annat sätt att koppla metoder till händelser är att använda attributet `OnClick` i webbkontrollens ASP.NET-kod, ett sätt som fungerar i både VB.NET och C#, som den andra knappen i koden ovan. En skillnad (eller två ☺) är att metodens synlighet<sup>25</sup> inte får vara (som ovan) privat (och att vi inte använder det reserverade ordet `Handles`).

```
Protected Sub Button2_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs)  
    'Kod som ska exekveras när besökare klickar på knapp  
End If
```

Om en metod hanterar flera händelser, vilket är praktiskt om händelserna ska exekvera en större mängd gemensam kod, så kan vi använda attributet `CommandName` i webbkontrollen. Vi använder då webbkontroll som genererade händelse, i argumentet `sender`, för att hämta kommandonamnet, vilket vi kan avgöra vilken avvikande kod (relativt andra kontroller) som ska exekveras när kontroll genererar händelse.

```
Private Sub Button1_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles Button1.Click  
    Dim btnKnapp As Button, strKommandotext As String  
  
    btnKnapp = CType(sender, Button)           'Hämta ref till kontroll & konvertera  
    strKommandotext = btnKnapp.CommandName     'Hämta kommandonamn för kontroll  
  
    If strKommandotext = ".." Then  
        'Exekvera olika saker beroende på webbkontroll som genererade händelse  
    End If  
End Sub
```

Resultande HTML-kod för knappar visas nedan.

```
<input type="submit" name="Button1" value="Button1" id="Button1" />  
<input type="submit" name="Button2" value="Button2" id="Button2" />
```

### 1.1.3.7 Länknappar

Länknappar (*link buttons*) används istället för ”vanliga” knappar, d.v.s. inte för länkar till andra webbsidor. Funktionen är m.a.o. den samma som för knapparna ovan och de flesta egenskaperna är de samma, bl.a. `CommandName` och `OnClick`. Vi ersätter, i stort sett, bara `Button` med `LinkButton` i kod motsvarande knapparna ovan. Nedan visas ASP.NET-koden och variabel som deklarerar i *code behind*.

```
<asp:linkbutton id="LinkButton1" runat="server">LinkButton1</asp:linkbutton>
```

```
Protected WithEvents LinkButton1 As System.Web.UI.WebControls.LinkButton
```

<sup>25</sup> Synlighet är hur tillgänglig något är, d.v.s. om det är publikt, skyddat (*protected*) eller privat.

### 1.1.3.8 Länkar

En länk (*hyper link*) är en länk. ☺ Länkar genererar inga ASP-händelser, men skickar dock besökare till URL som länk refererar till. Intressanta egenskaper är (utöver de för de flesta kontroller ☺) `ImageUrl`, `NavigateUrl` och `Target`. `NavigateUrl` används för att sätta URL som länk refererar till och `Target` hur webbsida ska öppnas (t.ex. nytt fönster eller vilken ram). `ImageUrl` skulle kunna användas för att ändra bild för länk (om ej text används för länk). Nedan visas ASP.NET-kod och variabel i *code behind*.

```
<asp:hyperlink id="HyperLink1" runat="server">HyperLink1</asp:hyperlink>
```

```
Protected WithEvents HyperLink1 As System.Web.UI.WebControls.HyperLink
```

I kod så är det främst intressant att t.ex. ändra egenskaperna ovan beroende på val och händelser (orelaterat till länkkontrollen) i ett webbformulär.

### 1.1.3.9 DropDownList

Nedrullningsbara listrutor (*drop down lists*) är användbara för att låta besökare välja ett alternativ av många. Precis som HTML-kontrollen så har varje alternativ i en listruta ett värde som skickas (*value*) och ett värde som visas (*item*). Listan kan fyllas dynamiskt av bl.a. vektorer (*arrays*) eller data från databaser, men även statiska (fasta) värden m.h.a. `ListItem`-taggen. Intressanta egenskaper är bl.a.

- `AutoPostBack` – om formulär ska skickas när valt alternativ ändras i listruta.
- `DataSource`<sup>26</sup> – datakälla för alternativ i listruta.
- `Items` – alternativ i listruta.

```
<asp:DropDownList id="DropDownList1" runat="server">  
  <asp:ListItem Value="DropDownList1">DropDownList1</asp:ListItem>  
</asp:DropDownList>
```

```
Protected WithEvents DropDownList1 As System.Web.UI.WebControls.DropDownList
```

Två sätt att fylla en listruta med kod är att lägga till alternativ via dess egenskap `Items` eller tilldela en vektor till dess egenskap `DataSource`. Om vi använder egenskapen `DataSource` så måste vi även anropa listrutans metod `DataBind()` så att listrutan fylls. Exempel visas nedan, men observera att värde som skickas (*value*) och visas (*text*) är den samma. (För exempel hur vi kan visa ett värde, t.ex. beskrivning, och skicka ett annat, t.ex. primärnyckel – se kapitlet *Avancerad ASP.NET* nedan.)

```
Dim arrVektor() As String = {"Ett", "Två", "Tre"} 'Dekl. o skapa vektor (array)  
Dim arrVektor As ArrayList 'Dekl. var. för vektorklass  
  
ddlItems.Items.Add("Ett") 'Fyll listrutans vektor Items direkt  
ddlItems.Items.Add("Två")  
ddlItems.Items.Add("Tre")  
  
ddlVektor.DataSource = arrVektor 'Fyll listruta med vektor (array)
```

<sup>26</sup> I kombination med `DataSource` så är även egenskaperna `DataMember`, `DataTextField`, `DataTextFormatString` och `DataValueField` intressanta. Dessa behandlas i kapitlet *Avancerad ASP.NET* nedan.



```
ddlVektor.DataBind()

arlVektor = New ArrayList          'Fyll listruta med vektorklass (ArrayList)
arlVektor.Add("Ett")
arlVektor.Add("Två")
arlVektor.Add("Tre")
ddlArrayList.DataSource = arlVektor
ddlArrayList.DataBind()
```

I kodfilen får vi tillgång till ett antal fler intressanta egenskaper, bl.a.

- `SelectedIndex` – index (som tal) på valt alternativ i listruta.
- `SelectedItem` – bakomliggande objekt för valt alternativ i listruta. Denna egenskap används främst när komplexa objekt (som med t.ex. `Hashtable`) läggs till i listruta.
- och `SelectedValue` – text som visas för valt värde.

Listrutor har en händelse *Selected Index Changed* som uppstår när besökare väljer ett nytt alternativ i listruta. Denna händelse uppstår även om knapphändelse sker (d.v.s. klickas), d.v.s. vi skulle kunna hantera båda händelser (om det skulle vara nyttigt ☺).

```
Private Sub ddlHashtable_SelectedIndexChanged(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles ddlHashtable.SelectedIndexChanged
    'Kod att exekvera om valt alternativ i listruta ändras
End Sub
```

### 1.1.3.10 ListBox

Listrutor (*list boxes*) är användbara om besökare ska kunna välja ett eller flera alternativ från en lista. Funktionen för en listruta är till största del den samma som för nedrullningsbara listrutor, d.v.s. upprepas inte här. Den stora skillnaden är att besökare kan välja flera alternativ (se nedan).

```
<asp:ListBox id="ListBox1" runat="server">
    <asp:ListItem Value="ListBox1">ListBox1</asp:ListItem>
    <asp:ListItem Value="ListBox1a">ListBox1a</asp:ListItem>
</asp:ListBox>
```

```
Protected WithEvents ListBox1 As System.Web.UI.WebControls.ListBox
```

Om flera alternativ kan väljas i listruta så måste vi loopa över listrutans egenskap `Items` och testa om aktuellt alternativ är valt. Om vi har en listruta med namnet `lstArrayList` och en etikett med namnet `lblKlickadeAlt` så kan vi loop över listrutans alternativ, kontrollera om alternativet är valt och skriva ut värdet för valt alternativ i etiketten med nedanstående kod.

```
For Each li As ListItem In lstArrayList.Items
    If li.Selected Then
        lblKlickadeAlt.Text += " " + li.Value
    End If
Next
```

### 1.1.3.11 Kryssrutor

Kryssrutor används då besökare ska ha ett begränsat antal multipla alternativ att välja från (med listrutor så är antalet i stort sett obegränsat). Det finns två webbkontroller för kryssrutor

(*check boxes*): `CheckBox` för enstaka kryssrutor och `CheckBoxList` för en lista med kryssrutor. Fördelen med den senare är att vi kan skapa alternativen med kod när webbformulär laddas (precis som med listrutor, d.v.s. med egenskapen `Items` ☺) liksom vi kan loopa över vektorn `Items` för att avgöra vilka "alternativ" i kryssrutelistorna som bockats för.

En intressant (eller kanske inte ☺) egenskap med kryssrutornas taggar är att vi **inte** får placera text mellan öppnande och avslutande tagg utan måste använda attributet `Text`. Intressanta egenskaper är `Selected` som kan användas för att avgöra om kryssruta blivit förbockad.

```
<asp:checkbox id="CheckBox1" runat="server" Text="CheckBox1"></asp:checkbox>

<asp:checkboxlist id="CheckBoxList1" runat="server">
  <asp:ListItem Value="cbl1a">CheckBoxList1a</asp:ListItem>
  <asp:ListItem Value="cbl1b">CheckBoxList1b</asp:ListItem>
</asp:checkboxlist>
```

```
Protected WithEvents CheckBox1 As System.Web.UI.WebControls.CheckBox
Protected WithEvents CheckBoxList1 As System.Web.UI.WebControls.CheckBoxList
```

Om vi använder enkla kryssrutor (`CheckBox`) så kan vi kontrollera om respektive kryssruta bockats för med dess egenskap `Selected` enligt nedan. En egenskap som vi kan använda för att bocka för (eller avbocka) en kryssruta med kod genom att tilldela värdet `True` (eller `False`).

```
If CheckBox1.Checked Then
  'Kod att exekvera om kryssruta bockats för
End If
```

För att lägga till alternativ i en kryssrutelista (`CheckBoxList1`) med kod kan vi skriva kod enligt nedan.

```
CheckBoxList1.Items.Add("Fyra")
```

Om vi har en kryssrutelista med namnet `CheckBoxList1` så skulle vi kunna loop över dess vektor `Items` och skriv ut värdet för förbockade alternativ i en etikett (`lblForbockatAlt`).

```
For Each cb As ListItem In CheckBoxList1.Items
  If cb.Selected Then
    lblForbockatAlt.Text += cb.Value + " "
  End If
Next
```

Sammanfattningsvis så är kryssrutelistor praktiska för att skapa kryssrutor dynamiskt, men även om vi inte skapar dem dynamiskt så är de praktiska för att avgöra vilka kryssrutor som bockats för (genom att loopa över `Items` som ovan istället för en massa `If`-satser).

### 1.1.3.12 Radioknappar

Radioknappar påminner mycket om kryssrutor, dock med några skillnader. Även radioknappar används för att besökare ska kunna välja bland ett begränsat antal alternativ, men de placeras i grupper där endast ett alternativ ska kunna väljas. Det finns två webbkontroller även för radioknappar (*radio buttons*): `RadioButton` för att själv bygga upp grupper och `RadioButtonList` för att grupper ska behandlas som en kontroll. För att gruppera de enskilda radioknapparna använder vi egenskapen `GroupName`, d.v.s. vi ger deras egenskap samma värde (namn), så att endast en av radioknapparna med samma gruppnamn kan väljas.

```
<asp:RadioButton id="RadioButton1" runat="server" Text="RadioButton1"
  GroupName="Gruppl"></asp:RadioButton><BR>
<asp:RadioButton id="RadioButton2" runat="server" Text="RadioButton2"
  GroupName="Gruppl"></asp:RadioButton>

<asp:radiobuttonlist id="RadioButtonList1" runat="server">
  <asp:ListItem Value="rb1la">RadioButtonList1a</asp:ListItem>
  <asp:ListItem Value="rb1lb">RadioButtonList1b</asp:ListItem>
</asp:radiobuttonlist>
```

```
Protected WithEvents RadioButton1 As System.Web.UI.WebControls.RadioButton
Protected WithEvents RadioButtonList1 As System.Web.UI.WebControls.RadioButtonList
```

För att avgöra om en enskild radioknapp är markerad så används egenskapen `Checked`.

```
If RadioButton1.Checked Then
  'Kod att exekvera om radioknapp markerats
End If
```

Även radioknappslistor kan skapas med kod, d.v.s. dynamiskt.

```
RadioButtonList1.Items.Add("Fyra")
```

Om vi vill veta vilket alternativ som valts i en radioknappslista kan vi använda någon av egenskaperna `SelectedIndex`, `SelectedItem` och `SelectedValue`.

```
lblValtAlternativ2.Text = RadioButtonList1.SelectedValue
```

Sammanfattningsvis så är radioknappslistor mer praktiska, liksom kryssrutelistor.

## 1.1.4 Exempel med kontroller

Exempel nedan visar hur en webbsida med webbkontroller kan se ut och hur vi kan hämta värden från webbkontrollerna. Nedan visas en bild på formulär ifyllt av en besökare och resultatet efter att besökare skickat formuläret.

The first screenshot shows a web browser window titled 'kontroller\_exempel - Mozilla Firefox'. The browser's address bar shows 'Arktiv Redigera Visa Gå till Bokmärken Verktyg Hjälp'. The form contains the following fields and controls:

- Namn:** A text box containing 'Nils Nilsson'.
- Adress:** A multi-line text box containing 'Nygatan 1', '123 45', and 'Nystad'.
- Nationalitet:** A dropdown menu with 'Svensk' selected.
- Semestermål:** A list box with 'Kos', 'Torremolinos', and 'Venedig' (selected).
- Destinationer:** A group of checkboxes: 'Ja, jag önskar reklam för nedanstående länder' (checked), 'Grekland' (checked), 'Frankrike' (unchecked), and 'Spanien' (checked).
- Kön:** Radio buttons for 'Man' (selected) and 'Kvinna'.

On the right side of the form, there are labels for each control type: 'Enradig textruta', 'Flerradig textruta', 'Nedrullningsbar listruta', 'Listruta', 'Kryssruta', 'Kryssrutelista', and 'Radioknappslista'. A 'Skicka' button is at the bottom right. The status bar shows 'Klar'.

The second screenshot shows the same browser window after submission. The page content is:

Tack Nils Nilsson för din anmälan  
 Vi kommer skicka broschyrer till Nygatan 1 123 45 Nystad  
 Broschyrer kommer skickas med språket Svensk  
 Du är intresserad av resemålen: Torremolinos Venedig  
 Vi kommer skicka reklam till dig.  
 Vi kommer skicka broschyrer för: Grekland Spanien  
 Du är en Man

The status bar still shows 'Klar'.

ASP.NET-koden för formulär visas nedan.

```
<form id="Form1" method="post" runat="server">
  <H1>Exempel med webbkontroller</H1>

  <!-- Panel med formulär -->
  <P><asp:Panel id="Panel1" runat="server">
    <!-- Tabell för design (positionering) av webbkontroller -->
    <TABLE id="Table1" cellSpacing="1" cellPadding="1" width="800" border="0">
      <TR>
        <TD><STRONG>Namn:</STRONG></TD>
        <TD><asp:TextBox id="txtNamn" runat="server"></asp:TextBox></TD>
        <TD>Enradig textruta</TD>
      </TR>
      <TR>
        <TD><STRONG>Adress:</STRONG></TD>
        <TD><asp:TextBox id="txtAdress" runat="server" Rows="3"
          TextMode="MultiLine"></asp:TextBox></TD>
        <TD>Flerradig textruta</TD>
      </TR>
      <TR>
        <TD><STRONG>Nationalitet:</STRONG></TD>
        <TD><asp:DropDownList id="ddlNationalitet" runat="server">
          <asp:ListItem Value="Da">Dansk</asp:ListItem>
          <asp:ListItem Value="No">Norsk</asp:ListItem>
          <asp:ListItem Value="Sv">Svensk</asp:ListItem>
        </asp:DropDownList></TD>
        <TD>Nedrullningsbar listruta</TD>
      </TR>
      <TR>
        <TD><STRONG>Semestermål:</STRONG></TD>
        <TD><asp:ListBox id="lstSemestermal" runat="server"
          SelectionMode="Multiple">
            <asp:ListItem Value="Ko">Kos</asp:ListItem>
            <asp:ListItem Value="To">Torremolinos</asp:ListItem>
          </asp:ListBox></TD>
        <TD></TD>
      </TR>
    </TABLE>
  </P>
  <asp:Button id="Skicka" value="Skicka" runat="server">Skicka</asp:Button>
</form>
```

```

        <asp:ListItem Value="Ve">Venedig</asp:ListItem>
    </asp:ListBox></TD>
    <TD>Listruta</TD>
</TR>
<TR>
    <TD><STRONG></STRONG></TD>
    <TD><asp:CheckBox id="chbReklam" runat="server" Text="Ja, jag önskar
        reklam för nedanstående länder"></asp:CheckBox></TD>
    <TD>Kryssruta</TD>
</TR>
<TR>
    <TD><STRONG>Destinationer:</STRONG></TD>
    <TD><asp:CheckBoxList id="cblDestination" runat="server">
        <asp:ListItem Value="Gr">Grekland</asp:ListItem>
        <asp:ListItem Value="Fr">Frankrike</asp:ListItem>
        <asp:ListItem Value="Sp">Spanien</asp:ListItem>
    </asp:CheckBoxList></TD>
    <TD>Kryssrutelista</TD>
</TR>
<TR>
    <TD><STRONG>Kön:</STRONG></TD>
    <TD><asp:RadioButtonList id="rblKon" runat="server">
        <asp:ListItem Value="Man">Man</asp:ListItem>
        <asp:ListItem Value="Kvinna">Kvinna</asp:ListItem>
    </asp:RadioButtonList></TD>
    <TD>Radioknappslista</TD>
</TR>
<TR>
    <TD></TD>
    <TD align="right">
        <asp:Button id="btnSkicka" runat="server"
            Text="Skicka"></asp:Button></TD>
    <TD></TD>
</TR>
</TABLE>
</asp:Panel></P>

<!-- Etikett för resultat från formulär -->
<P><asp:Label id="lblNamn" runat="server"></asp:Label></P>
<P><asp:Label id="lblAdress" runat="server"></asp:Label></P>
<P><asp:Label id="lblNationalitet" runat="server"></asp:Label></P>
<P><asp:Label id="lblSemestermal" runat="server"></asp:Label></P>
<P><asp:Label id="lblReklam" runat="server"></asp:Label></P>
<P><asp:Label id="lblDestination" runat="server"></asp:Label></P>
<P><asp:Label id="lblKon" runat="server"></asp:Label></P>
</form>

```

Och koden för knappens klickhändelse visas nedan.

```

'Hämta namn, adress och nationalitet
lblNamn.Text = "Tack " & txtNamn.Text & " för din anmälan" 'Textruta
lblAdress.Text = "Vi kommer skicka broschyrer till " & txtAdress.Text 'Textruta
lblNationalitet.Text = "Broschyrer kommer skickas med språket " -
    & ddlNationalitet.SelectedItem.Text 'Nedrullningsbar listruta

'Hämta semestermål
lblSemestermal.Text = "Du är intresserad av rese målen: "

For Each lb As ListItem In lstSemestermal.Items 'Listruta
    If lb.Selected Then
        lblSemestermal.Text += lb.Text & " "
    End If
Next

'Hämta om reklam
If chbReklam.Checked Then 'Kryssruta
    lblReklam.Text = "Vi kommer skicka reklam till dig."
Else
    lblReklam.Text = "Vi kommer <b>inte</b> skicka reklam till dig."
End If

'Hämta destination
lblDestination.Text = "Vi kommer skicka broschyrer för: "

```

```
For Each cb As ListItem In cblDestination.Items 'Kryssrutelista
    If cb.Selected Then
        lblDestination.Text += cb.Text & " "
    End If
Next

'Hämta kön
lblKon.Text = "Du är en " & rblKon.SelectedItem.Text 'Radioknappslista

Panell.Visible = False 'Dölj panel med formulär
```

## 2 Avancerad ASP.NET [ KONTROLLERA ]

I denna del av sammanfattning beskrivs lite mer avancerade, eller komplexa, saker i ASP.NET. Här finns bl.a. ett ”enkelt” exempel för att visa hur en DataGrid kan användas (för fler exempel, se sammanfattningen *Webbsidor och databaser*).

---

### 2.1 Kontroller och databaser

#### 2.1.1 Kontrollen ListBox

Listrutor<sup>27</sup> (vanliga eller nedrullningsbara) består av ett värde som ska skickas (t.ex. en primärnyckel) och en text som ska visas för besökare (båda vilka kan vara samma). Vad vi vill hämta från en databas är alltså ett frågeresultat (SQL-sats) som består av två kolumner.<sup>28</sup> Vi måste, om värde och text ska vara olika, även tala om vilken kolumn som ska vara värde respektive text.

I detta exempel kommer vi använda en tabell kurser med kolumnerna kurskod, kursnamn, nivå och poäng. När vi hämtar kurserna kommer vi sammanfoga de tre sista kolumnerna för texten som ska visas för besökare och kurskoden kommer användas som värde som ska skickas.

Exempel nedan innehåller två filer:

- ASPX-filen `EnkelListruta.aspx`, kallat formuläret nedan, som innehåller HTML- och ASP.NET-koden.
- *code behind*-filen `EnkelListruta.aspx.vb`, kallat kodfilen nedan, som innehåller koden för logik.

##### 2.1.1.1 Tabell i exempel

Detta exempel använder en tabell med namnet `kurser` som innehåller fyra kolumner: `kurskod` (`CHAR(6)`), `kursnamn` (`CHAR(50)`), `niva` (`CHAR(1)`) och `poang` (`BYTE`). Eftersom exemplet använder Access (eller snarare Jet Engine ☺) som databas så används en MDB-fil (`C:\Student\Studenter.mdb`) och OleDb-drivrutiner.

##### 2.1.1.2 Det ”tomma” formuläret och kodfilen

I Page-direktivet anger vi

- språk (VB)
- hur namn på metoder ska fungera (egendefinierade, d.v.s. inte standardiserade)<sup>29</sup>
- kodfilens namn
- och klass som formulär ska ärva från (`<projektnamn>.<klassnamn>`).

Resten av dokumentet är i stort sett vanlig HTML. I resten av detta exempel kommer inte nedanstående kod vara med utan resterande kod kommer placeras mellan de två HTML-kommentaren i koden nedan.

---

<sup>27</sup> Faktiskt gäller mycket beskrivet i detta avsnitt även för kryssrute- och radioknappslistor.

<sup>28</sup> Om vi sammanfogar flera kolumner i en tabell så bör vi namnge den med ett alias, d.v.s. att vi använder det reserverade ordet `AS` i `SELECT`-klausulen av SQL-sats.

<sup>29</sup> Sök på ”@ Page directive” i hjälpen och leta upp länken ”Web Server Control Event Model” för mer information.

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="EnkelListruta.aspx.vb"
Inherits="Webbexempel.EnkelListruta"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <title>EnkelListruta</title>
  </HEAD>
  <body>
    <form id="Form1" method="post" runat="server">
      <!-- Här BÖRJAR innehållet i webbsida -->
      <!-- Här SLUTAR innehållet i webbsida -->
    </form>
  </body>
</HTML>
```

I kodfilen har två kommentarer placerats som visar var deklarationer av egna variabler (och eventuella konstanter) kommer placeras samt var egna metoder kommer definieras i koden.

```
Public Class EnkelListruta
  Inherits System.Web.UI.Page

  'Variabler deklareras HÄR

  #Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
  <System.Diagnostics.DebuggerStepThrough() Private Sub InitializeComponent()

  End Sub

  'NOTE: The following placeholder declaration is req. by the Web Form Designer.
  'Do not delete or move it.
  Private designerPlaceholderDeclaration As System.Object

  Private Sub Page_Init(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
  End Sub

  #End Region

  Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
  End If

  'Egna metoder kommer läggas till HÄR

End Class
```

### 2.1.1.3 Grundläggande taggar för kontrollen ListBox

Enklaste<sup>30</sup> formen av en ListBox innehåller en tagg:

- <asp:ListBox> – tagg för listrutekontrollen.

I ListBox-taggen används attribut för att namnge listruta (id).

```
<!-- Här BÖRJAR innehållet i webbsida -->
<asp:ListBox id="lstKurser" runat="server"></asp:ListBox>
<!-- Här SLUTAR innehållet i webbsida -->
```

<sup>30</sup> Nåja... det finns enklare former om vi överlåter till ASP.NET att göra mer. ☺



I kodfilen lägger vi till en variabel för listruta och vi behöver även en konstant för ConnectionString.

```
'Variabler deklareras HÄR
Private cstrConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Student\Studenter.mdb;" _
    & "Persist Security Info=False"
Protected WithEvents lstKurser As System.Web.UI.WebControls.ListBox
```

Vi lägger även till kod i Form\_Load() för att fylla listruta med data. I detta fall anropas en egen metod HamtaOchFyllListBox() så att vi kan fylla listruta igen vid andra händelser (än Form Load).

```
Private Sub Page_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    If Not IsPostBack Then
        HamtaOchFyllListBox()
    End If
End Sub

Private Sub HamtaOchFyllListBox()
    Dim adoConn As OleDb.OleDbConnection, adoCmd As OleDb.OleDbCommand
    Dim adoDA As OleDb.OleDbDataAdapter, adoDS As DataSet
    Dim strSql As String

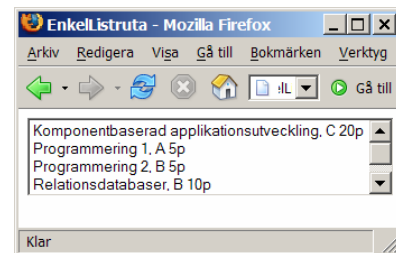
    strSql = "SELECT kurskod, (kursnamn & ', ' & niva & ' ' & poang & 'p') " _
        & "AS beskrivning FROM kurser ORDER BY kursnamn"

    adoConn = New OleDb.OleDbConnection(cstrConn) 'Skapa Connection-objekt
    adoCmd = adoConn.CreateCommand() 'Skapa Command-objekt
    adoCmd.CommandText = strSql 'Ange kommando att utföra
    adoCmd.CommandType = CommandType.Text 'Ange typ av kommando att utföra

    adoDA = New OleDb.OleDbDataAdapter(adoCmd) 'Skapa DataAdapter-objekt
    adoDS = New DataSet 'Skapa DataSet-objekt
    adoDA.Fill(adoDS, "Kurser") 'Fyll DataSet-objekt från DataAdapter

    lstKurser.DataSource = adoDS 'Ange datakälla för ListBox
    lstKurser.DataValueField = "kurskod" 'Ange kolumn för värde
    lstKurser.DataTextField = "beskrivning" 'Ange kolumn för text
    lstKurser.DataBind() 'Bind data till ListBox
End Sub
```

Resultatet visas i bild till höger. Om vi tittar på källkoden (HTML-koden) så ser vi att attributet value i taggen <option> visar kurskoden för respektive kurs.



## 2.1.2 Kontrollen DataGrid

Kontrollen DataGrid är en dynamisk kontroll vars storlek anpassas efter hur mycket data som en databastabell (eller frågeresultat, vektor, XML-fil/-sträng, m.m.) innehåller. Resultatet, efter att webbserver exekverat ASP.NET-koden, är en HTML-tabell.

Det finns ett antal olika sätt att använda en DataGrid, d.v.s. det jag beskriver nedan är bara **ett** sätt. Här beskrivs hur vi skriver koden för hand samt hur vi generera kolumner i tabell manuellt.

Exempel nedan innehåller två filer:

- ASPX-filen `EnkelTabell.aspx`, kallat formuläret nedan, som innehåller HTML- och ASP.NET-koden.
- *code behind*-filen `EnkelTabell.aspx.vb`, kallat kodfilen nedan, som innehåller koden för logik.

### 2.1.2.1 Tabell i exempel

Detta exempel använder en tabell med namnet `studenter` som innehåller två kolumner: `personnr` (`CHAR(10)`) och `namn` (`CHAR(50)`). Eftersom exemplet använder Access (eller snarare Jet Engine ☺) som databas så används en MDB-fil (`C:\Student\Studenter.mdb`) och OleDb-drivrutiner.

### 2.1.2.2 Det ”tomma” formuläret och kodfilen

I Page-direktivet anger vi

- språk (VB)
- hur namn på metoder ska fungera (egendefinierade, d.v.s. inte standardiserade)<sup>31</sup>
- kodfilens namn
- och klass som formulär ska ärva från (`<projektnamn>.<klassnamn>`).

Resten av dokumentet är i stort sett vanlig HTML. I resten av detta exempel kommer inte nedanstående kod vara med utan resterande kod kommer placeras mellan de två HTML-kommentaren i koden nedan.

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="EnkelTabell.aspx.vb"
Inherits="BjornsradiationNETVB.EnkelTabell"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head>
    <title>EnkelTabell</title>
  </head>
  <body MS_POSITIONING="FlowLayout">

    <form id="Form1" method="post" runat="server">
      <!-- Här BÖRJAR innehållet i webbsida -->
      <!-- Här SLUTAR innehållet i webbsida -->
    </form>

  </body>
</html>
```

I kodfilen har två kommentarer placerats som visar var deklARATIONER av egna variabler (och eventuella konstanter) kommer placeras samt var egna metoder kommer definieras i koden.

```
Public Class EnkelTabell
  Inherits System.Web.UI.Page

  'Egna variabler deklarereras HÄR

  #Region " Web Form Designer Generated Code "

  'This call is required by the Web Form Designer.
```

<sup>31</sup> Sök på ”@ Page directive” i hjälpen och leta upp länken ”Web Server Control Event Model” för mer information.

```

<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

End Sub

'NOTE: The following placeholder declaration is required by the Web Form
' Designer. Do not delete or move it.
Private designerPlaceholderDeclaration As System.Object

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
End Sub

'Egna metoder kommer läggas till HÄR

End Class

```

### 2.1.2.3 Grundläggande taggar för kontrollen DataGrid

Enklaste<sup>32</sup> formen av en DataGrid innehåller tre taggar:

- <asp:DataGrid> – tagg för DataGrid-kontrollen.
- <Columns> – används för att innesluta definitioner av kolumner i tabell.
- <asp:BoundColumn> – en kolumn bunden till en kolumn i databastabell.

I DataGrid-taggen används attribut för att namnge DataGrid (id) och för att tala om att inte ASP.NET ska generera kolumnerna i DataGrid (AutoGenerateColumns="False").

```

<!-- Här Börjar innehållet i webbsida -->
<asp:DataGrid id="dgdStuderter" runat="server" AutoGenerateColumns="False">
  <Columns>
    <asp:BoundColumn DataField="personnr"
      HeaderText="Personnummer"></asp:BoundColumn>
    <asp:BoundColumn DataField="namn" HeaderText="Namn"></asp:BoundColumn>
  </Columns>
</asp:DataGrid>
<!-- Här Slutar innehållet i webbsida -->

```

I kodfilen lägger vi till en variabel för DataGrid och vi behöver även en konstant för ConnectionString.

```

'Variabler deklarerar HÄR
Private cstrConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Student\Studerter.mdb;" _
    & "Persist Security Info=False"
Protected WithEvents dgdStuderter As System.Web.UI.WebControls.DataGrid

```

<sup>32</sup> Nåja... det finns enklare former om vi överlåter till ASP.NET att göra mer. ☺

Vi lägger även till kod i `Form_Load()` för att fylla `DataGrid` med data. I detta fall anropas en egen metod `HamtaOchFyllDataGrid()` så att vi kan fylla `DataGrid` igen vid andra händelser (än *Form Load*).

```
Private Sub Page_Load(...) Handles MyBase.Load
    'Put user code to initialize the page here
    If Not IsPostBack Then
        HamtaOchFyllDataGrid()
    End If
End Sub

'Egna metoder kommer läggas till HÄR
Private Sub HamtaOchFyllDataGrid()
    Dim adoConn As OleDb.OleDbConnection, adoCmd As OleDb.OleDbCommand
    Dim adoDA As OleDb.OleDbDataAdapter, adoDS As DataSet
    Dim strSql As String

    strSql = "SELECT personnr, namn FROM studenter"

    adoConn = New OleDb.OleDbConnection(cstrConn) 'Skapa Connection-objekt
    adoCmd = adoConn.CreateCommand() 'Skapa Command-objekt
    adoCmd.CommandText = strSql 'Ange kommando att utföra
    adoCmd.CommandType = CommandType.Text 'Ange typ av kommando att utföra

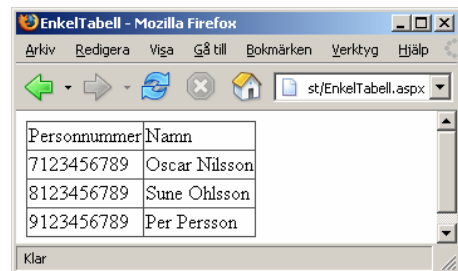
    adoDA = New OleDb.OleDbDataAdapter(adoCmd) 'Skapa DataAdapter-objekt
    adoDS = New DataSet 'Skapa DataSet-objekt
    adoDA.Fill(adoDS, "Studenter") 'Fyll DataSet-obj. fr DataAdapter

    dgdStudenter.DataSource = adoDS 'Ange datakälla för DataGrid
    dgdStudenter.DataBind() 'Bind data till DataGrid
End Sub
```

Resulterande webbsida bör se ut något i stil med bild till höger.

#### 2.1.2.4 Snygga till tabell

Tabellen i bild till höger är inte så tilltalande. Dags att snygga till den en aning.<sup>33</sup> För detta använder vi taggarna `HeaderStyle`, `ItemStyle` och `AlternatingItemStyle` för tabells rubrikrad, udda rader respektive jämna rader. I taggarna kan attributet `CssClass` användas för att använda stilmallar.



The screenshot shows a web browser window titled 'EnkelTabell - Mozilla Firefox'. The address bar shows 'st/EnkelTabell.aspx'. The table has two columns: 'Personnummer' and 'Namn'. The data rows are:

Personnummer	Namn
7123456789	Oscar Nilsson
8123456789	Sune Ohlsson
9123456789	Per Persson

Below the table is a button labeled 'Klar'.

```
<asp:datagrid id="dgdStudenter" runat="server" AutoGenerateColumns="False">
    <AlternatingItemStyle CssClass="listajamn"></AlternatingItemStyle>
    <ItemStyle CssClass="listaudda"></ItemStyle>
    <HeaderStyle CssClass="listarubrik"></HeaderStyle>
    <Columns>
        ...
    </Columns>
</asp:datagrid>
```

Nedan visas egenskaper för stilmallar som används ovan.

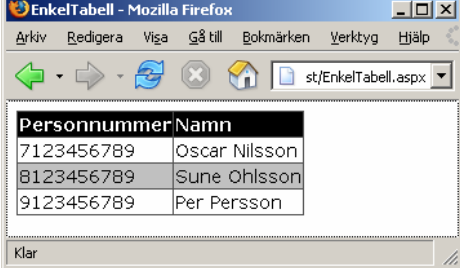
```
.listajamn
{
    background-color: #c0c0c0; color: black; font-size: 14px; font-family: Verdana
}
.listarubrik
{
```

<sup>33</sup> Med snygga till menar jag göra den mer lättläst – design är inte min starka sida.

```

background-color: black; color: white;
font-weight: bold; font-size: 14px;
font-family: Verdana
}
.listaudda
{
background-color: white; color: black;
font-size: 14px; font-family: Verdana;
}

```



EnkelTabell - Mozilla Firefox

st/EnkelTabell.aspx


Personnummer	Namn
7123456789	Oscar Nilsson
8123456789	Sune Ohlsson
9123456789	Per Persson

Klar

Resultatet av applicerade stilmallar visas i bild till höger.

### 2.1.2.5 Redigera poster

Om vi vill kunna redigera poster finns ett antal olika alternativ – ett är att kunna redigera dem i DataGrid. För detta kan vi lägga till en knappkolumn (*button column*), som antingen kan vara knappar eller länkar. Vi kan även välja vilka kolumner i tabell som går att redigera och inte – de senare markeras som endast läsbara (*read only*), lämpligen primärnycklar. I bild ovan används länkar för knappar och kolumnen personnummer har satts till endast läsbar. När användare klickar på Ändra-länken (*edit*) så visas studentens namn i en textruta samt Ändra-länken ändras till Uppdatera- och Avbryt-länkar (*update* resp. *cancel*).



EnkelTabell - Mozilla Firefox

rn/BjornradioNETVB/test/EnkelTabell.aspx

Personnummer	Namn	
7123456789	Oscar Nilsson	<a href="#">Ändra</a>
8123456789	Sune Ohlsson	<a href="#">Uppdatera</a> <a href="#">Avbryt</a>
9123456789	Per Persson	<a href="#">Ändra</a>

Klar

Eftersom vi ska kunna redigera poster, utom personnumret, så ändrar vi kolumnen med personnummer till endast läsbar med attributet `ReadOnly="True"`. Vi lägger även till en knappkolumn med taggen `<asp:EditCommandColumn>` med följande attribut:

- `ButtonType` – `LinkButton` för länkar och `PushButton` för knappar.
- `UpdateText` – text som ska visas i länk (eller knapp) för Uppdatera-länk.
- `CancelText` – text som ska visas i länk (eller knapp) för Avbryt-länk.
- `EditText` – text som ska visas i länk (eller knapp) för Ändra-länk.

För att HTML-tabellen inte ska ändra storlek när besökare klickar på Ändra-länken så kan vi ange bredd på tabell och kolumner. För detta används attributet `width` i DataGrid-taggen och attributet `ItemStyle-Width` i kolumntaggarna (`<asp:BoundColumn>` och `<asp:EditCommandColumn>`). Ändringar i ASPX-filen visas med grå bakgrund nedan.

```

<asp:DataGrid id="dgdStudenter" runat="server" AutoGenerateColumns="False"
Width="450">
...
<Columns>
<asp:BoundColumn DataField="personnr" ReadOnly="True"
HeaderText="Personnummer" ItemStyle-Width="150"></asp:BoundColumn>
<asp:BoundColumn DataField="namn" HeaderText="Namn"
ItemStyle-Width="160"></asp:BoundColumn>
<asp:EditCommandColumn ButtonType="LinkButton" UpdateText="Uppdatera"
CancelText="Avbryt" EditText="#196;ndra"
ItemStyle-Width="140"></asp:EditCommandColumn>
</Columns>
</asp:DataGrid>

```

Redigering av poster innebär att vi måste hantera tre händelser: Ändra, Uppdatera och Avbryt. Den första och den sista är mycket enkla medan den mellersta kräver lite mer kod.

För att svara på händelsen Ändra så kan vi lägga till den första av de två metoderna nedan. I den påbörjas redigering genom att sätta egenskapen `EditItemIndex` till index för post som ska redigeras. Detta värde kan vi hämta från parametern `e` till metod. Sist binder vi data till vår `DataGrid` igen.

Och för att svara på händelsen Avbryt så kan vi lägga till den andra metoden nedan. Är avbryts redigering genom att sätta egenskapen `EditItemIndex` till ett ogiltigt värde, t.ex. `-1` (eftersom vektorer är nollbaserade ☺). Återigen så binder vi data till vår `DataGrid`.

```
Private Sub dgdStudenter_EditCommand(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
    Handles dgdStudenter.EditCommand
    dgdStudenter.EditItemIndex = e.Item.ItemIndex 'Ange index att redigera
    HamtaOchFyllDataGrid() 'Hämta data för DataGrid
End Sub

Private Sub dgdStudenter_CancelCommand(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
    Handles dgdStudenter.CancelCommand
    dgdStudenter.EditItemIndex = -1 'Ange index att redigera
    HamtaOchFyllDataGrid() 'Hämta data för DataGrid
End Sub
```

**Observera** att om vi använder Access (eller snarare Jet Engine ☺) som databas så måste användarkontot ASPNET ha Ändra-rättigheter till MDB-fil i filsystemet.

Här börjar vi med att hämta personnummer (och tillika primärnyckel i tabell ☺) genom att hämta innehållet i `DataGrid`s första cell i aktuell rad. Nästa steg är att hämta en referens till textruta med det ändrade värdet på students namn. När vi har referensen till textrutan så kan vi hämta dess värde. Därefter kan vi skapa en SQL-sats och utföra den för att uppdatera tabell i databasen. Eftersom vi är klara med redigering så sätter vi `DataGrid`:s egenskap `EditItemIndex` till `-1` (d.v.s. ogiltigt värde) och binder data till `DataGrid`.

```
Private Sub dgdStudenter_UpdateCommand(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataGridCommandEventArgs) _
    Handles dgdStudenter.UpdateCommand
    Dim adoConn As OleDb.OleDbConnection, adoCmd As OleDb.OleDbCommand
    Dim adoDS As DataSet, txtNamn As TextBox
    Dim strPersonnr, strNamn, strSql As String

    strPersonnr = e.Item.Cells(0).Text 'Hämta personnummer från DataGrid

    txtNamn = CType(e.Item.Cells(1).Controls(0), TextBox) 'Hämta referens t textruta
    strNamn = txtNamn.Text 'Hämta innehåll i textruta

    strSql = "UPDATE tblStudenter SET namn='" & strNamn & "' WHERE personnr='" & _
        strPersonnr & "'"

    adoConn = HamtaConnection() 'Hämta Connection-objekt
    adoConn.Open() 'Öppna förbindelse till DB
    adoCmd = adoConn.CreateCommand() 'Skapa Command-objekt
    adoCmd.CommandType = CommandType.Text 'Ange typ av kommando
    adoCmd.CommandText = strSql 'Ange kommandotext
    adoCmd.ExecuteNonQuery() 'Utför kommando

    dgdStudenter.EditItemIndex = -1 'Ange index att redigera

    HamtaOchFyllDataGrid() 'Hämta data för DataGrid
End Sub
```

---

## 2.2 Använda egna klasser från ASP.NET [ ATT GÖRA ]

Ibland är det praktiskt att skapa klasser som vi kan återanvända i andra projekt. En sådan klass är SQLProxy som används för databasåtkomst.<sup>34</sup>

### 2.2.1 Klassen SQLProxy

Klassen SQLProxy innehåller nedanstående metoder.

- `GetReader(String)` – returnerar en `DataReader` för SQL-satsen i argumentet.
- `GetDataSet(String)` – returnerar ett `DataSet` för SQL-satsen i argumentet.

Det finns lite olika sätt att skapa klassen. Här kommer jag använda ett projekt – ett projekt som vi kan lägga till i lösningar där vi vill använda den.

---

## 2.3 Användarkontroller (*user controls*) [ ATT GÖRA ]

### 2.3.1 Skapa en enkel användarkontroll [ ATT GÖRA ]

### 2.3.2 Skapa en mer avancerad användarkontroll [ ATT GÖRA ]

### 2.3.3 Infoga en användarkontroll med egenskaper [ ATT GÖRA ]

När vi infogar en användarkontroll i ett webbformulär så skapas inte en motsvarande instansvariabel i code behind-filen. Vi måste alltså själva deklarera en variabel i code behind-filen för att kunna manipulera kontrollen med kod.

---

<sup>34</sup> Klassen SQLProxy är en ”standardklass”, d.v.s. inget jag kommit på själv. ☺

### 3 Frågor och svar

I detta kapitel beskrivs lösningar på problem jag ”råkat ut för” och mina (eller av mig stulna ☺) lösningar.

---

#### 3.1 Datum som fil senast ändrades (skrevs till)

I FrontPage finns en funktion för att infoga/uppdatera datum som fil senast ändrades. Jag har ännu inte hittat en motsvarande funktion i VS.NET. Men nedanstående lösning fungerar. ☺ Lösningen bygger på att en etikett med namnet lblFilnamn används för att presentera datumet.

Här hämtas först URL till aktuell fil med funktionen `CurrentExecutionFilePath()`, som används i anropet av metoden `MapPath()` för att erhålla fysisk sökväg till fil i serverns filsystem. Sen testas (lite onödigt?) att fil existerar (och för att visa hur vi kan kontrollera en fils existens ☺) innan anrop av `GetLastWriteTime()` som returnerar datum som fil senast uppdaterades. Denna kod kan t.ex. placeras i metoden `Page_Load()`.

```
string strPath = Server.MapPath(Request.CurrentExecutionFilePath); //Fysisk sökväg
if(File.Exists(strPath))
    lblFildatum.Text = File.GetLastWriteTime(strPath).ToString(); //Hämta datum
```

---

#### 3.2 Använda Visual Studio.NET (2003)

Beskrivningar i denna sammanfattning har gjorts utifrån kod skriven (och genererad ☺) i Visual Studio.NET 2003 (VS.NET 2K3, eller bara VS.NET). Här har jag samlat lite tips om hur vi använder VS.NET och hur vi kan låta VS.NET hjälpa oss göra utveckling lättare. Använd möjligheten att kunna placera flera projekt i samma lösning, vilket gör det lättare att dela på filer, eller kopiera filer, mellan projekt samt felsöka webbapplikationer som utvecklas i flera projekt (t.ex. komponentbaserade applikationer).

##### 3.2.1 Öppna ett existerande projekt

När vi skapar ett nytt webbprojekt så skapas en webbapplikation på webbserver. En webbapplikation är en mapp, alla filer i mappen samt alla undermappar (som inte tillhör en annan webbapplikation). Vi kan därför inte skapa ett nytt projekt för att redigera ett existerande webbprojekt (och därmed webbapplikation).

Istället, om vi inte redan har ett existerande lösning, så skapas en tom lösning som vi lägger till ett existerande projekt från webb i, vilket beskrivs nedan.

1. Högerklicka på lösningen i lösningsfönstret (*Solution Explorer*) och välj Add... från menyn som visas samt Existing Project From Web... från undermenyn som visas.
2. Fyll i URL till webbserver (**inte** projekt, d.v.s. utelämna sökväg) i dialogrutan Add Existing Project From Web som visas och klicka på OK.
3. Bläddra till mapp med projekt, markera XXPROJ-filen (ersätt XX med språk, t.ex. CS för C# och VB för VB.NET) och klicka på knappen Open.

##### 3.2.2 Webbprojekt med flera utvecklare [ ATT GÖRA ]

När vi är flera som jobbar på samma webbprojekt så uppstår problemet med samtidig användning av filer då alla kodfiler finns på en webbserver, och med användning avses redigering. Vi måste m.a.o. skydda (eller försöka skydda) utvecklare från att öppna och



redigera samma fil samtidigt. En lösning är att sitta i samma rum... ☺ Men en bättre är att använda källkodskontroll (*source control*), vilket beskrivs nedan (när jag väl kommit på hur ☺).

## 4 Litteratur

Nedan finns annan litteratur som använts som referensmaterial.

- Anderson, R. et al, *Professional ASP.NET*, Wrox Press, 2001.
- Barwell, F. et al, *Professional VB.NET*, Wrox Press, 2001.
- Goode, C. et al, *Beginning ASP.NET 1.0 with C#*, Wrox Press, 2002.
- Homer, A. & D. Sussman, *ASP.NET Distributed Data Applications*, Wrox Press, 2002.
- Robinson, Ed, et al, *Upgrading MS Visual Basic 6.0 to MS Visual Basic.NET*, Microsoft Press, 2002.
- Ullman, C. et al, *Beginning ASP.NET 1.1 with Visual C# .NET 2003*, Wiley Publishing, 2004.

samt webbsidor hos Microsoft (bl.a. MSDN) liksom webbplatserna asp.net och GotDotNet.com.