ASP.NET i ADO.NET

Wstęp

Podstawy komunikacji sieciowej

Serwer/klient
TCP/IP (protokół transportowy (pakiety), protokół sieciowy)
HTTP (protokół typu żądanie-odpowiedź z żądaniami GET i POST)

Historia ASP

Technologię Microsoft's Active Server Pages (ASP) wprowadzono w1996. Wcześniej dla platform Microsoftu używano Common Gateway Interface (CGI) oraz Internet Server Application Programming Interface (ISAPI), które przyczyniły się do rozwoju ASP.

CGI było pierwszą szeroko zaakceptowaną techniką na tworzenie serwerów Web serwujących dane dynamicznie. Rozszerzała ona funkcjonalność serwerów tak, że mogły one dynamicznie generować odpowiedzi HTTP używając programów napisanych w C lub którymś z języków skryptowych, jak np. Perl. Można było w ten sposób personalizować zawartość stron zgodnie z danymi o użytkownikach przechowywanymi w bazie danych.

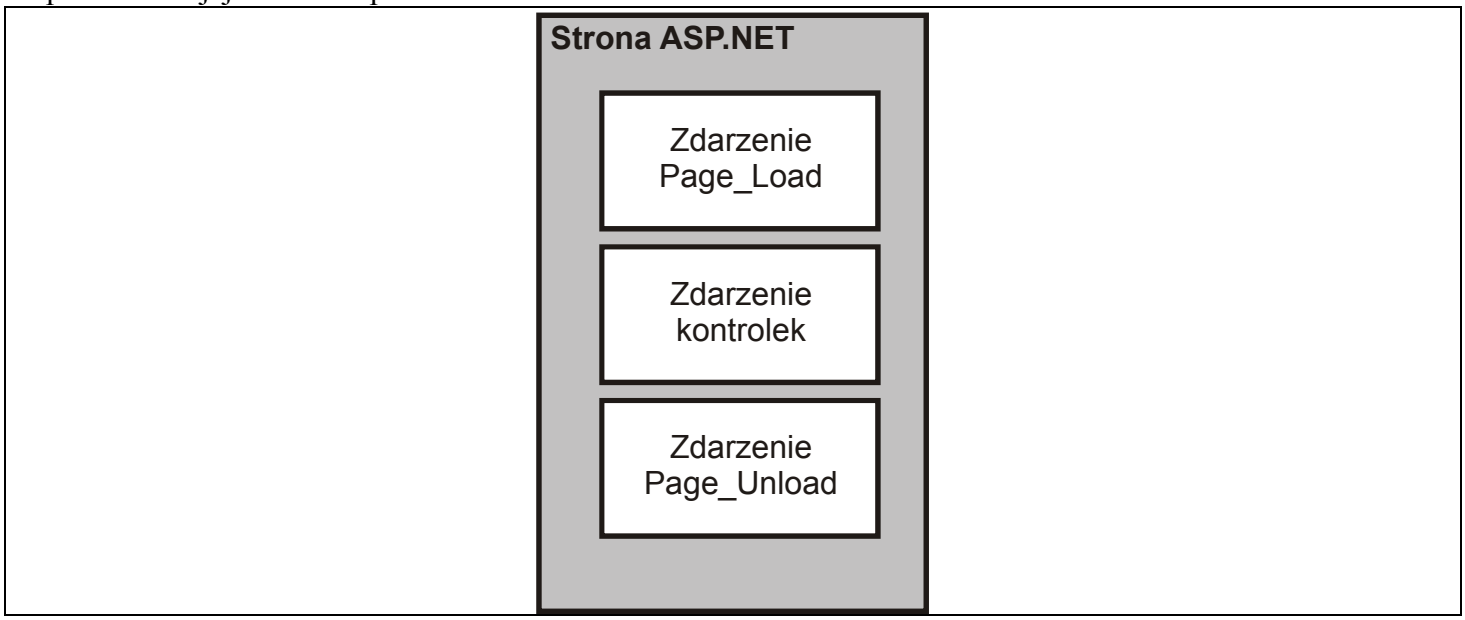
Wadą tego rozwiązania było tworzenie nowego procesu na każde żądanie HTTP oraz jego zabijanie po obsłużeniu żądania.

Platforma Microsoft's Active Server rozwiązywała ten problem, jednak przed wprowadzeniem ASP programiści byli zmuszeni do pisania rozszerzeń ISAPI lub filtrów. Główna różnica pomiędzy ISAPI oraz CGI polega na tym, że CGI bazuje na wykonywalnych plikach (EXEs) platformy Windows, rozszerzenia ISAPI zaś zaimplementowane są jako Dynamic Link Libraries (DLLs).

itd.....

Strony ASP.NET

W ASP.NET strony są obiektami, a więc mają swoje własności, metody i zdarzenia. Jedna z własności jest bardzo ważna: jest to isPostBack. Własność jest typu Boolean, i wskazuje, czy strona jest ładowana jako odpowiedź na jej odesłanie przez klienta.




```
<SCRIPT [LANGUAGE="codeLangauge"] RUNAT="SERVER" [SRC="externalfilename"]>
' Event Handling Code
```

```
</SCRIPT>
```

Dyrektywy stron ASP.NET

Directive	Description
@Page	The **@Page** directive defines page-specific attributes used by the ASP.NET page parser and compiler. An example of a **@Page** attribute is **Language**, which specifies the default language for the page.
@Control	The **@Control** directive defines control-specific attributes used by the ASP.NET page parser and compiler. An example of a **@Control** attribute is **Description**, which provides a text description of the control.
@Import	The **@Import** directive explicitly imports a namespace into a page. The only attribute supported by the **@Import** directive is **Namespace**, which indicates the name of namespace to import. (More on namespaces later.)
@Register	The **@Register** directive associates aliases with namespaces and class names for concise notation in custom server control syntax. For more information on the **@Register** directive, see Session 10.
@Assembly	An assembly is a unit of reusable code compiled into a .dll file. The **@Assembly** directive links an assembly against the current page, making all of the assembly's classes and interfaces available for use on the page. The only attribute supported by the **@Assembly** directive is **Assemblyname**, which indicates the name of the assembly to link. Assemblies that reside in an application \bin directory are automati-cally linked to pages within the application, therefore, they do not need to be linked using the **@Assembly** directive.
@OutputCache	The **@OutputCache** directive controls the output caching policy for the page. An example of a **@OutputCache** attribute is **Duration**, which specifies the time (in seconds) that the output cache for the page will be maintained.

Przykład:

```
<%@ Page Language="VB" Description="ASP.NET Page" %>
<%@ Import Namespace="System.Net" %>
<SCRIPT LANGUAGE="VB" RUNAT="server">
Sub Page_Load(Source As Object, E As EventArgs)
' Page_Load Code
        lblTest.Text = Page.isPostBack
End Sub
Sub Control_Click(Sender As Object, E As EventArgs)
        'Control_Click Code
        Response.Write("The Submit button was clicked!")
End Sub
Sub Page_Unload(Source As Object, E As EventArgs)
' Page_Unload
End Sub
</SCRIPT>
<html>
<head>
<title>ASP.NET Page</title>
</head>
        <body>
        <form ID="frmTest" RUNAT="SERVER">
                <asp:Label ID="lblTest" RUNAT="SERVER"/>
                </br>
                <asp:Button ID="btnSubmit" onClick="Control_Click" TEXT="Submit"
                RUNAT="SERVER"/>
        </form>
        </body>
</html>
```

False Submit	The Submit button was clicked! True Submit

Przestrzenie nazw:

Namespace	Description
System	Contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.
System.Collections	Contains interfaces and classes that define various collec-tions of objects, such as lists, queues, arrays, hash tables, and dictionaries.
System.IO	Provides access to the File and Directory objects, which enable you to add, move, change, create, or delete folders (directories) and files on the Web server.
	Includes the **HTTPRequest** class that provides extensive information about the current HTTP request, the **HTTPResponse** class that manages HTTP output to the client, and the **HTTPServerUtility** object that provides access to server-side utilities and processes. **System.Web** also includes classes for cookie manipulation, file transfer, exception information, and output cache control.
System.Web.UI	Contains the ASP.NET control classes.
System.Web.UI.HtmlControls	Contains the HTML server controls.
System.Web.UI.WebControls	Contains the Web controls.
System.Data	Provides access to general data access services.
System.Data.OleDb	Provides access to OLEDB -specific data access services.
System.Data.SQLClient	Provides access to SQL Server data access services.
SystemXML	Provides access to the services for manipulating XML.

Na łatwy początek

Na początek stwórzmy stronę internetową wyświetlającą hasło "Hello W3Schools":

Hello W3Schools!

Kod HTML

Kod HTML wyświetlający hasło "Hello W3Schools":

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3Schools!</h2>
</center>
</body>
</html>
```

Kod ten można umieścic w pliku "**firstpage.htm**" i przeglądnąć w przeglądarce internetowej.

Kod ASP.NET

Najprostszym sposobem konwersji strony HTML do strony ASP.NET jest zwykła zamiana rozszerzenia pliku z **.html** na **.aspx** (poniższy, znany już nam kod zapisać można w pliku "**firstpage.aspx**")

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3Schools!</h2>
</center>
</body>
</html>
```

Dynamiczna strona ASP.NET

Poniższy kod wyświetla nasz przykład jako stronę ASP.NET (pojawiła się instrukcja **Response.Write()**):

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3Schools!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

Kod ten umieścić można w pliku: **dynpage.aspx**

Kontrolki serwera

W klasycznym ASP wykonywalny kod kontrolek musi być umieszczony w kodzie strony w miejscu, w którym kontroli mają się pojawić. Co więcej, kod wykonywalny nie może być oddzielony od kodu HTML. W ASP.NET kod wykonywalny może być odseparowany od kodu html.

Kontrolki serwera są tagami, które rozumiane są przez serwer. Wyróżnia się 3 rodzaje kontrolek serwera:

HTML Server Controls – tradycyjne tagi HTML

Web Server Controls – nowe tagi ASP.NET

Validation Server Controls – kontrolki do walidacji wejścia

Kontrolki serwera HTML w ASP.NET

Kontrolki serwera HTML są w istocie tagami HTML rozpoznawanymi i wykonywanymi przez serwer. Domyślnie wszystkie elementy HTML w plikach ASP.NET są traktowane jako tekst. Aby elementy te stały się fragmentami kodu wykonywanego przez serwer, muszą one posiadać atrybut **runat="server"**. Aby rozróżnić kontrolki serwera dodaje się im atrybut **id**.

Uwaga: Wszystkie kontrolki serwera HTML muszą być umieszczone wewnątrz tagu <form> z atrybutem **runat="server"**. Atrybut ten określa, że dana forma powinna być przetworzona przez serwer. Ponadto określa, że kontrolki tam zawarte mogą być osiągalne przez skrypty serwera.

Uwaga: W ogólności przeglądarki internetowe dopuszczają istnienie tagów HTML, które nie są domknięte, albo też są źle zagnieżdżone. W ASP.NET takie rzeczy są niedopuszczalne.

Kontrolka serwera HTML	Opis
HtmlAnchor	Steruje elementem <a>
HtmlButton	Steruje elementem <button>
HtmlForm	Steruje elementem <form>
HtmlGeneric	Steruje elementami HTML nie określonymi przez specyficzne kontrolki serwera HTML jak <body>, , , itd.
HtmlImage	Steruje elementem <image>

HtmlInputButton	Steruje elementami <input type="button">, <input type="submit">, oraz <input type="reset">
HtmlInputCheckBox	Steruje elementem <input type="checkbox">
HtmlInputFile	Steruje elementem <input type="file">
HtmlInputHidden	Steruje elementem <input type="hidden">
HtmlInputImage	Steruje elementem <input type="image">
HtmlInputRadioButton	Steruje elementem <input type="radio">
HtmlInputText	Steruje elementami <input type="text"> oraz <input type="password">
HtmlSelect	Steruje elementem <select>
HtmlTable	Steruje elementem <table>
HtmlTableCell	Steruje elementami <td> oraz <th>
HtmlTableRow	Steruje elementem <tr>
HtmlTextArea	Steruje elementem <textarea>

W poniższym przykładzie zadeklarowana jest kontrolka serwera HtmlAnchor (przykład można zapisać w pliku z rozszerzeniem .aspx). Atrybut HRef kontrolki HtmlAnchor jest używany w procedurze obsługi zdarzeń Page_Load. Page_Load jest jednym z wielu zdarzeń, które rozumiane są przez ASP.NET:

```
<script runat="server">
Sub Page_Load
link1.HRef="http://www.w3schools.com"
End Sub
</script>
<html>
<body>
<form runat="server">
<a id="link1" runat="server">Visit W3Schools!</a>
</form>
</body>
</html>
```

Wykonywalny kod przeniesiony został poza dokument HTML

Kontrolki serwera Web w ASP.NET

Kontrolki serwera Web są specjalnymi tagami ASP.NET rozumianymi przez serwer. Podobnie do kotrolek serwera HTML, kontrolki serwera Web działają na stronie serwera i wymagają podania atrybutu runat="server". Kontrolki serwera Web niekoniecznie muszą mapować istniejące elementy HTML. Mogą natomiast reprezentować bardziej złożone elementy.

Składnia tworzenia kontrolek serwera Web jest następująca:

```
<asp:control_name id="some_id" runat="server" />
```

Kontrolki:

Kontolki serwera Web	**Opis**
AdRotator	Wyświetla sekwencję obrazów
Button	Wyświetla przycisk
Calendar	Wyświetla kalendarz
CheckBox	Wyświetla check box
CheckBoxList	Tworzy grupę elementów check box z możliwością wielokrotnego wyboru
DataGrid	Wyświetla pola ze źródła danych na siatce
DataList	Wyświetla elementy ze źródła dancyh używając szablonów
DropDownList	Tworzy rozwijalną listę

HyperLink	Tworzy hyperlink
Image	Wyświetla obraz
ImageButton	Wyświetla obraz z możliwością kliknięcia
Label	Wyświetla statyczną zawartość, która jest programowalna (pozwala zastosować styl do wyświetlanej zawartości)
LinkButton	Tworzy przycisk hyperlink
ListBox	Tworzy rozwijalną listę pojedynczego lub wielokrotnego wyboru
Literal	Wyświetla statyczną zawartość, która jest programowalna (nie pozwala zastosować stylu do zawartości)
Panel	Dostarcza kontener dla innych kontrolek
PlaceHolder	Rezerwuje przestrzeń dla kontrollek dodanych w kodzie
RadioButton	Tworzy przycisk opcji (radio button)
RadioButtonList	Tworzy grupę przycisków opcji
Repeater	Wyświetla powtórzoną listę elementów dowiązaną do kontrolki
Table	Tworzy tablicę
TableCell	Tworzy komórkę tablicy
TableRow	Tworzy wiersz tablicy
TextBox	Tworzy text box
Xml	Wyświetla plik XML lub rezultat transformacji XSL

W poniższym przykładzie zadeklarowana jest kontrolka serwera Web klasy Button – tj. przycisk z etykietą tekstową (cały kod umieszczony jest w pliku .aspx). Następnie utworzona jest procedura obsługi zdarzenia Click, która zmienia etykietę przycisku:

```
<script runat="server">
Sub submit(Source As Object, e As EventArgs)
button1.Text="You clicked me!"
End Sub
</script>
<html>
<body>
<form runat="server">
<asp:Button id="button1" Text="Click me!" runat="server" OnClick="submit"/>
</form>
</body>
</html>
```

Kontrolki walidacji serwera w ASP.NET

Kontrolki walidacji serwera wykorzystywane są do sprawdzania poprawności danych wprowadzanych przez użytkownika. Jeśli wynik walidacji jest negatywny, wyświetlana jest wiadomość o błędzie.

Każda kontrolka walidacji wykonuje specyficzny typ sprawdzania (tj. sprawdzanie ze względu na wystąpienie konkretnej wartość lub wartości z zadanego przedziału)

Walidacja domyślnie jest wykonywana podczas kliknięć na przyciskach Button, ImageButton, lub LinkButton (które są kontrolkami). Można zabronić wykonywania walidacji podczas klikania na przycisk przez ustawianie wartości własności CausesValidation na false.

Kontrolki:

Validation Server Control	**Description**
CompareValidator	Porównuje wartość jednej wartości kontrolki wejściowej do wartości innej kontrolki wejściowej lub do określonej wartości

CustomValidator	Pozwala napisać własną metodę walidacji wprowadzanych wartości
RangeValidator	Sprawdza, czy wprowadzona wartość jest z zadanego przedziału
RegularExpressionValidator	Zapewnia, że wartość (jej postać) z kontrolki wejściowej zgadza się zadanym wzorcem
RequiredFieldValidator	Tworzy kontrolkę wejściową jako pole wymagane
ValidationSummary	Wyświetla raport o wszystkich błędach walidacji, które wystąpiły na stronie Web

Składnia tworzenie kontrolki serwera Validation jest następująca:

```
<asp:control_name id="some_id" runat="server" />
```

W poniższym przykładzie zadeklarowana jest kontrolka TextBox, przycisk Button , oraz RangeValidator (wszystko umieszczone jest w pliku .aspx). Jeśli walidacja okaże się niepoprawna, pojawi się text "The value must be from 1 to 100!" na kontrolce RangeValidator:

```
<html>
<body>
<form runat="server">
Enter a number from 1 to 100:
<asp:TextBox id="tbox1" runat="server" />
<br /><br />
<asp:Button Text="Submit" runat="server" />
<br />
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
EnableClientScript="false"
Text="The value must be from 1 to 100!"
runat="server" />
</form>
</body>
</html>
```

Obsługa zdarzeń w ASP.NET

Procedura obsługi zdarzenie (event handler subroutine) wykonuje określony kod, gdy zajdzie konkretne zdarzenie. Dla przykładu jak niżej:

```
<%
lbl1.Text="The date and time is " & now()
%>
<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

nie wiadomo, kiedy faktycznie wykona się zadeklarowany kod.

Zdarzenie Page_Load

Page_Load jest jednym z wielu zdarzeń rozumianych przez ASP.NET. Zdarzenie Page_Load jest wyzwalane, kiedy dana strona jest wczytywana. Wtedy ASP.NET automatycznie wywołuje procedurę obsługującą to zdarzenie:

```
<script runat="server">
```

```
Sub Page_Load
lbl1.Text="The date and time is " & now()
End Sub
</script>
<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

Uwaga: Zdarzenie Page_Load nie posiada żadnych referencji do obiektów ani żadnych parametrów event contains no object references or event arguments!

Własność Page.IsPostBack

Procedura Page_Load wykonywana jest za każdym razem, kiedy strona jest ładowana. Jeśli kod procedury Page_Load ma być wykonany tylko podczas pierwszego załadowania strony, należy posłużyć się własnością Page.IsPostBack.

Jeśli Page.IsPostBack ma wartość false, to strona jest ładowana po raz pierwszy, jeśli ma wartość true, strona jest odsyłana z powrotem do serwera (po kliknięciu na przycisk):

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  lbl1.Text="The date and time is " & now()
end if
End Sub
Sub submit(s As Object, e As EventArgs)
lbl2.Text="Hello World!"
End Sub
</script>
<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
<h3><asp:label id="lbl2" runat="server" /></h3>
<asp:button text="Submit" onclick="submit" runat="server" />
</form>
</body>
</html>
```

W przykładzie za pierwszym załadowaniem strony wyświetlany jest tekst "The date and time is...." z bieżącą datą i czasem; oraz wyświetlany jest przycisk. Kiedy użytkownik naciśnie przycisk, procedura obsługi zdarzenia napisze "Hello World!" na drugiej etykiecie, ale tym razem data i czas już się nie zmienia.

Web Forms w ASP.NET

Wszystkie kontrolki serwera muszą wystąpić wewnątrz tagu <form>, oraz tag <form> musi posiadać atrybut runat="server". Atrybut runat="server" wskazuje, że forma powinna być przetwarzana na serwerze. Pokazuje ona również, że kontrolki mogą być osiągalne z poziomu skryptów serwera:

```
<form runat="server">
...HTML + server controls
</form>
```

Uwaga: Forma zawsze dostarcza się do strony sama. Jeśli zdefiniowany jest atrybut action, jest on ignorowany. Jeśli pominięty zostanie atrybut metody, będzie on ustawiony na wartość domyślną method="post". Jeśli pominięte zostaną nazwy oraz id atrybutów, ASP.NET automatycznie usupełni te braki.

Uwaga: Strona .aspx może zawierać tylko jedną kontrolkę <form runat="server">

Jeśli podczas projektowania zostanie źródło widoku na stronie .aspx zawierającej formę bez wyspecyfikowanej nazwy, metody, akcji lub id, wtedy ASP.NET samo doda brakujące atrybuty do formy. Będzie to wygląda mniej więcej tak:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
...some code
</form>
```

Dostarczanie formy

Formy często dostarczane są po kliknięcu na klawisz. Kontrolka serwera Button w ASP.NET ma następujący format:

```
<asp:Button id="id" text="label" OnClick="sub" runat="server" />
```

Atrybut id definiuje unikalną nazwę dla przycisku, oraz tekst etykiety, która na nim jest wyświetlana. Parametr onClick specyfikuje nazwę procedury obsługi zdarzenia związanej z kliknięcien na tym przycisku.

W poniższym przykładzie zadeklatowana jest kontrolka Button (wszystko w jednym pliku .aspx). Kliknięcie na przycisku uruchamia procedurę obsługi zdarzenia, która zmienia etykietę przycisku:

```
<script  runat="server">
Sub submit(Source As Object, e As EventArgs)
  button1.Text="You clicked me!"
End Sub
</script>

<html>
<body>

<form runat="server">
<asp:Button id="button1" Text="Click me!" runat="server" OnClick="submit"/>
</form>

</body>
</html>
```

Utrzymywanie ViewState

W klasycznym ASP wysłanie formy (jej dostarczenie) powodowało wyczyszczenie wszystkich wartości w niej występujących. Strona klienta nie zachowała ViewState, czyli niemożliwe było, aby raz wprowadzone dane mogły być powtórnie edytowane. W przypadku zasygnalizowania błedu w danych przez serwer pozostawało wprowadzanie wszystkich danych na nowo.

W ASP.NET strony wysłane pojawiają się w przeglądarce razem ze wszystkimi starymi wartościami. Dzieje się tak dlatego, że ASP.NET utrzymuje ViewState. ViewState określa status strony, kiedy wysyłana jest ona do serwera. Status jest definiowany za pomocą pól ukrytych umieszczanych na każdej ze stron z kontrolką <form runat="server">. Odpowiedni kod źródłowy mógłby wyglądać tak:

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTI0ODU5MDE1Ozs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />
.....some code
</form>
```

Utrzymywanie ViewState jest domyślnym ustawieniem dla ASP.NET Web Forms. Jeśli nie jest to pożądane, należy wstawić dyrektywę <%@ Page EnableViewState="false" %> na początku strony .aspx lub dodać atrybut EnableViewState="false" do kontrolki.

Poniższy przykład (plik demo_classicasp.aspx) pokazuje stary sposób (wartości na formie znikną po naciśnięciu klawisza):

```
<html>
<body>
```

```
<form action="demo_classicasp.aspx" method="post">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
Response.Write("Hello " & fname & "!")
End If
%>
</body>
</html>
```

A to jest nowy sposób (wartości nie znikają):

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Hello " & txt1.Text & "!"
End Sub
</script>
<html>
<body>
<form runat="server">
Your name: <asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Kontrolka TextBox

Kontrolka TextBox jest używana do tworzenia pól tekstowych służących do wprowadzania danych przez użytkownika. Kontrolka ta ma posiada atrybuty i własności, a w poniższym przykładzie pokazany jest sposób użycia niektórych z nich:

```
<html>
<body>

<form runat="server">

A basic TextBox:
<asp:TextBox id="tb1" runat="server" />
<br /><br />

A password TextBox:
<asp:TextBox id="tb2" TextMode="password" runat="server" />
<br /><br />

A TextBox with text:
<asp:TextBox id="tb4" Text="Hello World!" runat="server" />
<br /><br />

A multiline TextBox:
<asp:TextBox id="tb3" TextMode="multiline" runat="server" />
<br /><br />
```

```
A TextBox with height:
<asp:TextBox id="tb6" rows="5" TextMode="multiline"
runat="server" />
<br /><br />

A TextBox with width:
<asp:TextBox id="tb5" columns="30" runat="server" />

</form>

</body>
</html>
```

Dodawanie skryptów

Zawartość oraz ustawienia kontrolki TextBox mogą być zmieniane przez skrypty serwera, gdy strona jest dostarczona. Forma może być dostarczona przez naciśnięcie na przycisk lub kiedy użytkownik zmieni wartość w kontrolce TextBox.

W następującym przykładzie zadeklarowane są: kontrolka TextBox, kontrolka Button, kontrolka Label (wszystko w jednym pliku .aspx). Kiedy przycisk zostanie naciśnięty, wykona się procedura submit. Procedura ta zmieni tekst kontrolki Label:

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Your name is " & txt1.Text
End Sub
</script>
<html>
<body>
<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

W kolejnym przykładzie w jednym pliku .aspx znajdują się deklaracje dwóch kontrolek: TextBox oraz Label. Kiedy zmieniona zostanie wartość w TextBox i albo nastąpi kliknięcie poza TextBox, albo naciśnięty zostanie klawisz Tab key, wtedy wykona się podprocedura change. Procedura ta zapisze tekst z TexBox na kontrolce Label:

```
<script runat="server">
Sub change(sender As Object, e As EventArgs)
lbl1.Text="You changed text to " & txt1.Text
End Sub
</script>
<html>
<body>
<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server"
text="Hello World!"
ontextchanged="change" autopostback="true"/>
<p><asp:Label id="lbl1" runat="server" /></p>
```

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

Kontrolka Button

Kontrolka Button używana jest do wyświetlania przycisków. Przycisk może być przyciskiem wysyłającym (submit button) lub przyciskiem komendy (command button). Domyślnie każdy przycisk jest przyciskiem wysyłającym (submit button).

Przycisk wysyłający nie ma żadnej nazwy komendy. Przy kliknięciu odsyła jedynie stronę do serwera. Możliwe jest napisanie obsługi zdarzenia (event handler) sterującego akcjami wykonywanymi po kliknięciu na przycisk.

Przycisk komendy ma nazwę komendy oraz pozwala utworzyć wieloprzyciskowe kontrolki na stronie dokumentu. Można napisać obsługę zdarzeń do sterowania wykonywanymi akcjami po naciśnięciu klawisza komendy.

Kontrolki Button posiadają szereg atrybuty i własności. Przykład użycia kontrolki Button:

```
<html>
<body>

<form runat="server">
<asp:Button id="b1" Text="Submit" runat="server" />
</form>
</body>
</html>
```

Dodanie skryptu

Formy najczęściej dostarczane są poprzez kliknięcia na przyciskach. W poniższym przykładzie zadeklarowana jest kontrolka TextBox, Button oraz Label (wszystko w jednym pliku.aspx). Kiedy klawisz wysyłający zostanie naciśnięty, wykonana zostanie podprocedura pisząca tekst na kontrolce Label:

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Your name is " & txt1.Text
End Sub
</script>
<html>
<body>
<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Kontrolki z dowiązaniem danych

Poniższe kontrolki wspomagają dowiązania danych:

asp:RadioButtonList

asp:CheckBoxList

asp:DropDownList

asp:Listbox

Wybieralne elementy powyższych kontrolek zazwyczaj definiowane są przez jedną lub więcej kontrolek asp:ListItem:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>
</body>
</html>
```

Przy dowiązanie danych do wypełnienia listy elementów wybieralnych można użyć oddzielnych źródeł danych, jak bazy danych, pliki XML, lub skrypt.

Używając importowanych źródeł, dane są odseparowane od HTML. Dlatego żadne zmiany na elementach nie przenoszą się na dane w źródle danych.

Kolekcje ASP.NET

Obiekt ArrayList

Obiekt typu ArrayList jest kolekcją elementów zawierających pojedyncze, proste wartości. Elementy dodawane są do ArrayList za pomocą metody Add(). W poniższym przykładze tworzona jest czteroelementowa lista:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
end if
end sub
</script>
```

Domyślnie w ArrayList istnieje 16 pozycji. Jednak finalny rozmiar ArrayList można określić za pomocą metody TrimToSize():

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
end if
end sub
</script>
```

ArrayList można sortować alfabetycznie lub numerycznie metodą Sort():

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
```

```
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
end if
end sub
</script>
```

Aby sortować w odwrotnej kolejności, należy użyć metody Reverse() po metodzie Sort():

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
  mycountries.Reverse()
end if
end sub
</script>
```

Dowiązanie danych do kontrolki ArrayList

Obiekty ArrayList mogą automatycznie generować teksty i wartości dla następujących kontrolek:
asp:RadioButtonList
asp:CheckBoxList
asp:DropDownList
asp:Listbox
Aby dowiązać dane do kontrolki RadioButtonList, najpierw należy utwożyć kontrolkę RadioButtonList (bez żadnych elementów asp:ListItem) w pliki .aspx:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

Następnie należy dodać skrypt, który buduje listę oraz dowiązuje wartości do list kontrolki RadioButtonList:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
  rb.DataSource=mycountries
  rb.DataBind()
end if
```

```
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

Własność DataSource kontrolki RadioButtonList jest ustawiona na ArrayList definiując źródło danych kontrolki RadioButtonList. Metoda DataBind() kontrolki RadioButtonList dowiązuje źródło danych z kontrolką RadioButtonList.

Uwaga: wartości danych występują zarówno jako Text oraz Value właściwości dla kontrolki. Aby dodać jakąś Value która różna jest od Text, należy użyć albo obiektu Hashtable albo obiektu SortedList.

Obiekt Hashtable

Obiekt Hashtable zawiera pary {klucz,wartość}. Indeksy są używane jako klucze, zaś wyszukiwanie wartości odbywa się poprzez przeszukiwanie kluczy. Elementy do Hashtable dodawane są do Hashtable metodą Add().

Przykład, w którym tworzona jest Hashtable zawierająca nazwy państw:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
end if
end sub
</script>
```

Dowiązanie danych

Obiekt Hashtable może automatycznie generować tekst oraz wartości dla następujących kontrolek:

asp:RadioButtonList

asp:CheckBoxList

asp:DropDownList

asp:Listbox

Aby dowiązać dane do kontrolki RadioButtonList, na początek należy utworzyć kontrolkę RadioButtonList (bez żadnych elementów asp:ListItem) w pliku .aspx:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" />
</form>
</body>
</html>
```

Następnie można dodać skrypt budujący listę:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
```

```
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" />
</form>
</body>
</html>
```

Następnie dodawana jest podprocedura, która ma być uruchomiona, kiedy użytkownik kliknie na elemencie w kontrolce RadioButtonList . Kiedy przycisk wyboru (radio button) jest naciśnięty, na etykiecie pojawi się tekst:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Uwaga: Nie można wybrać porządku sortowania na obiektach dodanych do Hashtable. Do sortowania alfabetycznego lub numerycznego należy użyć obiektu SortedList.

Obiekt SortedList

SortedList ma cechy zarówno ArrayList, jak i HashTable. Elementami SortedList są pary klucz-wartość. Obiekt SortedList automatycznie sortuje elementy w porządku alfabetycznym lub numerucznym. Dodawanie

elementów odbywa się za pomocą metody Add(). Wymiar końcowy SortedList można ustalić za pomocą metody TrimToSize().
W przykładzie poniżej tworzona jest lista SortedList, do której dodawane są nazwy czterech państw.:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
end if
end sub
</script>
```

Dowiązanie danych

Obiekt SortedList może automatycznie generować tekst i wartości dla następujących kontrolek:
asp:RadioButtonList
asp:CheckBoxList
asp:DropDownList
asp:Listbox
Aby dowiązać dane do RadioButtonList, na początek należy utworzyć kontrolkę RadioButtonList bez żadnych elementów asp:ListItem na stronie .aspx:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" />
</form>
</body>
</html>
```

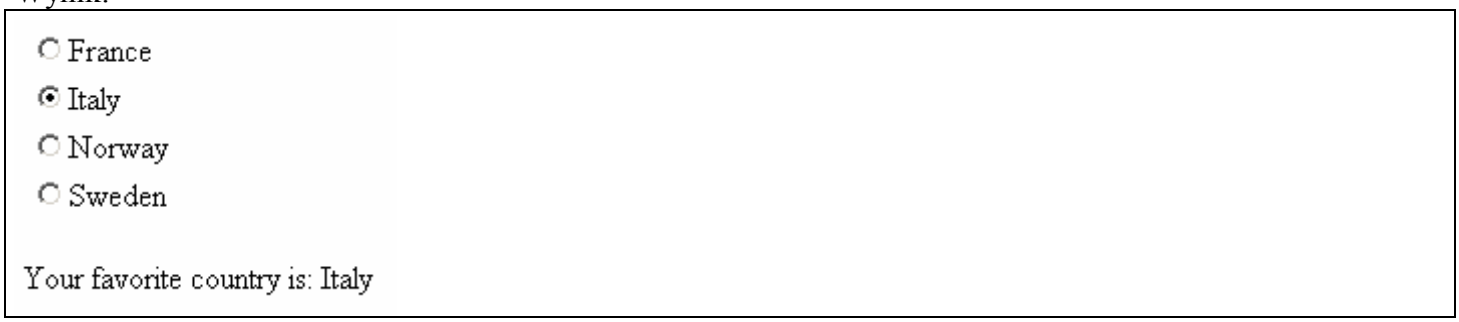
Następnie dodać należy skrypt budujący listę:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" />
</form>
</body>
</html>
```

Następnie dodawana jest podprocedura, która może być wywołana, kiedy użytkownik kliknie na elemencie kontrolki RadioButtonList. Gdy dojdzie do kliknięcia, pojawi się tekst na etykiecie:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Wynik:



Plik XML

Przykład pliku XML o nazwie "countries.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<countries>
<country>
<text>Norway</text>
<value>N</value>
</country>
<country>
<text>Sweden</text>
<value>S</value>
</country>
<country>
<text>France</text>
```

```
<value>F</value>
</country>
<country>
<text>Italy</text>
<value>I</value>
</country>
</countries>
```

Dowiązanie danych do kontrolki List

Pliki XML można dowiązywać do kontrolek List. Na początek należy zaimportować przestrzeń nazw "System.Data" (potrzebne, aby można było pracować z obiektami DataSet). W tym celu na początku strony .aspx umieszcza się dyrektywę:

```
<%@ Import Namespace="System.Data" %>
```

Następnie utworzony jest obiekt DataSet do wczytania pliku XML. Plik ten zostanie wczytany do DataSet kiedy strona załadowana zostanie po raz pierwszy:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
end if
end sub
```

Aby dowiązać DataSet do kontrolki RadioButtonList należy na stronie .aspx najpierw taką kontrolkę utworzyć (bez elementów asp:ListItem elements):

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" />
</form>
</body>
</html>
```

Następnie należy dodać skrypt budujący XML DataSet:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
  rb.DataSource=mycountries
  rb.DataValueField="value"
  rb.DataTextField="text"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
</form>
</body>
</html>
```

Następnie należy dodać podprocedurę, która wykonywana będzie, gdy użytkownik kliknie ne element kontrolki RadioButtonList. Po kliknięciu na przycisk (radio button) na etykiecie pojawi się tekst:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
  rb.DataSource=mycountries
  rb.DataValueField="value"
  rb.DataTextField="text"
  rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Wynik będzie taki sam jak w poprzednim przykładzie.

Dowiązywanie DataSet do kontrolki Repeater

Kontrolka Repeater używana jest do wyświetlania powtórzonej listy elementow, które są dowiązane do kontrolki. Kontrolka Repeater może być związana z tabelą bazy danych, plikiem XML lub inna listą elementów.W poniższym przykładzie pokazano dowiązanie pliku XML do kontrolki Repeater.
W przykładzie skorzystano z następującego pliku XML ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
```

```
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>
```

Na początek importowana jest przestrzeń nazw "System.Data" (potrzebna do pracy z obiektami DataSet). Import ten zapewnia umieszczenie na początku strony dyrektywy:

```
<%@ Import Namespace="System.Data" %>
```

Następnie tworzony jest DataSet do wczytywania pliku XML. Plik ten jest wczytywany podczas pierwszego załadowania strony:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

Następnie tworzona jest kontrolka Repeater.
na początek , i tylko raz, zawartości elementu <HeaderTemplate> wyprowadzane są na wyjściu,
następnie zawartości elementu <ItemTemplate> są powtórzone dla każdego „rekordu” w zbiorze DataSet,
i na koniec zawartości <FooterTemplate> są wyprowadzane na wyjściu.

```
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
...
</HeaderTemplate>
<ItemTemplate>
...
</ItemTemplate>
<FooterTemplate>
...
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Następnie dodajemy skrypt, który tworzy DataSet i dowiązuje mycdcatalog (obiekt typu DataSet) do kontrolki Repeater. Kontrolka Repeater wypełniana jest ponadto tagami HTML. Elementy danych dowiązane są do komórek w sekcji <ItemTemplate> za pomocą metody <%#Container.DataItem("fieldname")%>:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Oto wynik:

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988
Greatest Hits	Dolly Parton	USA	RCA	9.90	1982
Still got the blues	Gary Moore	UK	Virgin records	10.20	1990
Eros	Eros Ramazzotti	EU	BMG	9.90	1997

Użycie <AlternatingItemTemplate>

Wygląd alternatywnych wierszy na wyjściu opisać w elementcie <AlternatingItemTemplate> występującym po elemencie <ItemTemplate>. W poniższym przykładzie co drugi z wierszy będzie wyświetlony w kolorze szarym:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<AlternatingItemTemplate>
<tr bgcolor="#e8e8e8">
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</AlternatingItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Oto wynik:

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988
Greatest Hits	Dolly Parton	USA	RCA	9.90	1982
Still got the blues	Gary Moore	UK	Virgin records	10.20	1990
Eros	Eros Ramazzotti	EU	BMG	9.90	1997

Użycie <SeparatorTemplate>

Element <SeparatorTemplate> jest używany do opisania separatora pomiędzy rekordami. W poniższym przykładzie wstawiona zostanie pozioma linia pomiędzy każdą parą wierszy w tabeli:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="0" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<SeparatorTemplate>
<tr>
<td colspan="6"><hr /></td>
</tr>
</SeparatorTemplate>
<FooterTemplate>
</table>
```

```
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Oto wynik:

Title	Artist	Country	Company	Price	Year
Empire Burlesque	Bob Dylan	USA	Columbia	10.90	1985
Hide your heart	Bonnie Tyler	UK	CBS Records	9.90	1988
Greatest Hits	Dolly Parton	USA	RCA	9.90	1982
Still got the blues	Gary Moore	UK	Virgin records	10.20	1990
Eros	Eros Ramazzotti	EU	BMG	9.90	1997

Dowiązanie DataSet do kontrolki DataList

Kontrolka DataList, podobnie do kontrolki Repeater, jest używana do wyświetlania listy powtarzających się elementów (związanych z kontrolką). Tylko, że DataList domyślnie dodaje tabelę dokoła elementów. Kontrolka DataList może być dowiązana do tabeli bazy danych, pliku XML, lub innej listy elementów. Poniżej pokazane jest jak dowiązać plik XML do kontrolki DataList.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
```

```
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>
```

Na początek zaimportowana jest przestrzeń nazw "System.Data" (aby móc pracować z obiektami typu DataSet):

```
<%@ Import Namespace="System.Data" %>
```

Następnie utworzony jest DataSet dla pliku XML, który wypełniany jest danymi z pliku XML kiedy strona ładowana jest po raz pierwszy:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

Następnie tworzony jest DataList na stronie .aspx. Na początek wyświetlane są zawartości elementu <HeaderTemplate> i tylko raz na wyjściu, następnie powtarzane są zawartości elementu <ItemTemplate> dla każdego "rekordu" w DataSet, na koniec zawatrości elementu <FooterTemplate> są wyświetlane raz na wyjściu:

```
<html>
<body>
<form runat="server">
<asp:DataList id="cdcatalog" runat="server">
<HeaderTemplate>
...
</HeaderTemplate>
<ItemTemplate>
...
</ItemTemplate>
<FooterTemplate>
...
</FooterTemplate>
</asp:DataList>
</form>
</body>
</html>
```

Następnie dodajemy skrypt który tworzy DataSet, wiążąc go z mycdcatalog i kontrolką DataList. Wypełniamy również kontrolkę DataList przez <HeaderTemplate> który zawiera nagłówek tabeli, <ItemTemplate> który zawiera elementy danych do wyświetlenia, oraz <FooterTemplate> który zawiera tekst. Atrybut gridlines kontrolki DataList jest ustawiony na "both" aby wyświetlić granice tabeli:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
```

```
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:DataList id="cdcatalog"
gridlines="both" runat="server">
<HeaderTemplate>
My CD Catalog
</HeaderTemplate>
<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>
<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>
</asp:DataList>
</form>
</body>
</html>
```

Użycie stylów

Można również dodać styl do kontrolki DataList, aby ją upiększyć:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="true"
itemstyle-backcolor="#778899"
```

```
itemstyle-forecolor="#ffffff"
footerstyle-font-size="9pt"
footerstyle-font-italic="true">
<HeaderTemplate>
My CD Catalog
</HeaderTemplate>
<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>
<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>
</asp:DataList>
</form>
</body>
</html>
```

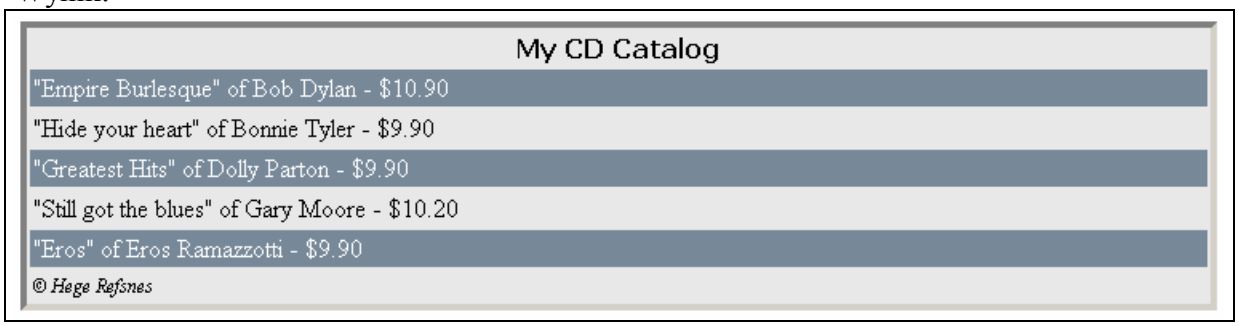
Użycie <AlternatingItemTemplate>

Można dodać element <AlternatingItemTemplate> za elementem <ItemTemplate> aby opisać wygląd naprzemiennych wierszy na wyjściu. Określenie stylu pojawia się w sekcji <AlternatingItemTemplate> w kontrolce DataList:

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="True"
itemstyle-backcolor="#778899"
itemstyle-forecolor="#ffffff"
alternatingitemstyle-backcolor="#e8e8e8"
alternatingitemstyle-forecolor="#000000"
footerstyle-font-size="9pt"
footerstyle-font-italic="True">
<HeaderTemplate>
```

```
My CD Catalog
</HeaderTemplate>
<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>
<AlternatingItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</AlternatingItemTemplate>
<FooterTemplate>
&copy; Hege Refsnes
</FooterTemplate>
</asp:DataList>
</form>
</body>
</html>
```

Wynik:



ADO.NET

ADO.NET jest częścią .NET Framework

ADO.NET składa się ze zbioru klas do pracy z danymi

ADO.NET całkowicie bazuje na XML

ADO.NET nie posiada, w przeciwieństwie do ADO, obiektu Recordset

Korzystanie z ADO.NET na stronach ASP niewiele różni się od sposobu, w jaki korzysta się z ADO.NET z poziomu aplikacji VB.

Utworzenie połączenia z bazą danych

W pierwszym kroku należy zaimportować przestrzeń nazw "System.Data.OleDb" (potrzebna, gdy chce się pisać aplikacje dla baz danych używających sterowników OLE DB (jak MS Access). Połączenie z bazą danych tworzone będzie w podprocedurze Page_Load. W tym celu utworzona zostanie zmienna dbconn jako nowy obiekt klasy OleDbConnection z ciągiem znaków określających połączenie (specyfikującym dostarczyciela OLE DB oraz lokalizację bazy danych:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=" &
server.mappath("northwind.mdb"))
dbconn.Open()
end sub
</script>
```

Utworzenie obiektu Command

Zmienna dbcomm służyć będzie do wykonywania komend, tj. wysyłania zapytań SQL do bazy danych. W rozważanym przypadku będzie ona instancją klasy OleDbCommand:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM Klienci"
dbcomm=New OleDbCommand(sql,dbconn)
end sub
</script>
```

Utworzenie DataReader

Klasa OleDbDataReader używana jest do odczytywania strumienia rekordów odczytanych ze źródła. Obiekt tej klasy jest tworzony w wyniku wywołania metody ExecuteReader obiektu klasy OleDbCommand:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM Klienci"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
end sub
</script>
```

Dowiązania do kontrolki Repeater

DataReader może być dowiązany do kontrolki Repeater:

```
<%@ Import Namespace="System.Data.OleDb" %>
<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=" &
server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM Klienci"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
customers.DataSource=dbread
customers.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>
<html>
<body>
```

```
<form runat="server">
<asp:Repeater id="customers" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Nazwa Firmy</th>
<th>Przedstawiciel</th>
<th>Adres</th>
<th>Miasto</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("NazwaFirmy")%></td>
<td><%#Container.DataItem("Przedstawiciel")%></td>
<td><%#Container.DataItem("Adres")%></td>
<td><%#Container.DataItem("Miasto")%></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Zamknięcie połączenia z bazą danych

Należy zawsze zamykać zarówno DataReader jak i połączenie, jeżeli dostęp do bazy danych nie jest już dłużej wymagany:

```
dbread.Close()
dbconn.Close()
```

Inny przykład:

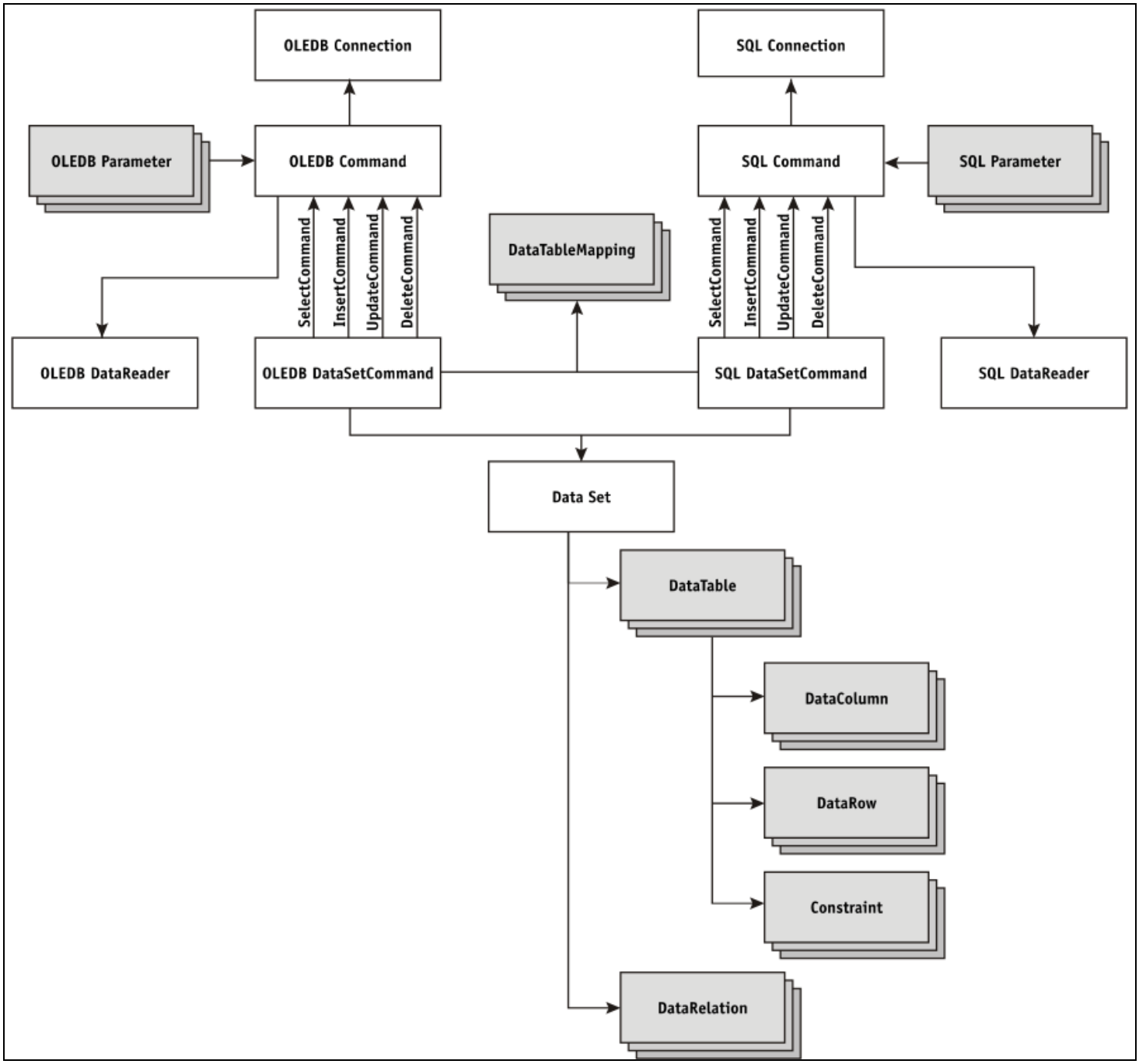
```
<%@ Import Namespace="System.Data.OleDb" %>
<%@ Import Namespace="system.data" %>
<%@ Page Language="VB" Debug="False" Trace="False"%>
<HTML>
<SCRIPT LANGUAGE="vb" RUNAT="server">
DIM oConn as OleDbConnection
DIM oCmd as OleDbDataAdapter
DIM oDS as new dataset
public sub page_load(sender as object,e as eventargs)
if page.ispostback=false then
BindData()
end if
end sub
Function BindData()
oConn=new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data source=" &

server.mappath("northwind.mdb"))
oCmd=new OleDbDataAdapter("select * from Klienci where Kraj='Polska'",oConn)
oCmd.Fill(oDS,"Klienci")
```

```
datagrid.datasource=oDS.tables("Klienci").defaultview
datagrid.databind()
End Function
</SCRIPT>
<BODY>
<FORM ID="form1" RUNAT="server">
<ASP:DATAGRID
ID="datagrid"
RUNAT="server"
AUTOGENERATECOLUMNS="True"
DATAKEYFIELD="IDklienta">
</ASP:DATAGRID>
</FORM>
</BODY>
</HTML>
```

Wynik:

IDklienta	NazwaFirmy	Przedstawiciel	StanowiskoPrzedstawiciela	Adres	Miasto	Region	KodPocztowy	Kraj	Telefon	Faks
WOLZA	Wolski Zajazd	Zbyszek Piestrzeniewicz	Właściciel	ul. Filtrowa 68	Warszawa		01-012	Polska	(26) 642-7012	(26) 642-7012


OLEDB Connection
SQL Connection
OLEDB Parameter
OLEDB Command
SQL Command
SQL Parameter
SelectCommand
InsertCommand
UpdateCommand
DeleteCommand
DataTableMapping
SelectCommand
InsertCommand
UpdateCommand
DeleteCommand
OLEDB DataReader
OLEDB DataSetCommand
SQL DataSetCommand
SQL DataReader
Data Set
DataTable
DataColumn
DataRow
Constraint
DataRelation