ASP.NET Basics

Janne Kemppi

January 2015

opiframe

13.11.2014

Web Pages

Web Pages

- **A Web Page is a web document which is suitable for World Wide Web hosting and can be accessed with a web browser.**
  - Typically written with HTML language.
  - Web Browser displays a web page on display or mobile phone.

Static and Dynamic Web Pages

- **Web browser can request a web page from web server in network.**
  - Web browser uses HTTP protocol to make these requests.
- Web server delivers responses in two ways:
  - Static web page is delivered exactly as stored.
  - Dynamic web page is generated by web application. Typically it's output depends on user input. It is thus responsive.

Web Applications

- Web Application’s functionality depends on two issues:
  - **Client-side scripting** (JavaScript and extensions) that executes in web wrowser. These are often embedded directly to HTML and XHTML language document files.
    - Client-side scripting is typically used for graphical ”eye candy” such as animated gifs where refreshment of web page would not look good.
  - **Server-side software** (PHP, Python, Ruby, Node.js, ASP.NET) executes when user requests a document within web server. They produce output that web wrowser understands.
    - Server-side software is typically used for ”number crunching” in background and fetching information from data bases.

XHTML Basic Functionality

* **Web browsers use XHTML and HTML languages defined by W3C standards (http://www.w3.org/).**
  * HTML is widely used with XHTML slowly replacing it.
* Note! Any web page that uses HTML or XHTML can be validated using following address:
  * http://validator.w3.org/check

XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-type"
content="text/html;charset=UTF-8" />
<title>Title text</title>
</head>
<body> <p>Body text </p> </body>
</html>
```

HTML Elements

- HTML has large number of elements which can be displayed using tags.
  - The **h1** for header.
  - The **p** for paragraph.
  - The **br** for line break.
- Each HTML element has also numerous attributes:
  - The **style** for using CSS style sheets.
  - The **id** for element’s identification.
  - The **src** for suorce for images.
  - The **title** for extra information shown as tool tip.
  - The **value** for text content for input element.

CSS

* **Cascading Style Sheets (CSS) is used to alter how HTML/XHTML elements are shown.**
  * Practically all modern web browsers support CSS.
  * They are used for uniform graphical styling (or ”eye candy”).
* CSS can be set in three places:
  * **Inline Style Sheet** is embedded to HTML code. It overrides Internal Style Sheets and External Style Sheets.
  * **Internal Style Sheet** is within HTML document but it is separate from HTML code. It overrides External Style Sheet.
  * **External Style Sheet** is a separate document with CSS code.

CSS

- Example of Inline Style Sheet:

```
<body> <p style="Color:Blue">Body text </p> </body>
```

- Example of Internal Style:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd"> <html xmlns="http://www.w3.org/1999/xhtml">
<head>
        <meta http-equiv="Content-type" content="text/html;charset=UTF-8" />
        <style type="text/css">
                p {Color:Blue;}
        </style>
        <title>Title text</title>
</head>
<body>
        <p>Body text </p>
</body>
</html>
```

- Example of External Style (replaces style element of Internal Style).

```
<link rel="stylesheet" type="text/css" href=”MyStyle.css"/>
```

Exercise 01

- Lets make our first static Web page.
  - Exercise 01a: Make a static web page
  - Exercise 01b: Add simple Cascading Style Sheet (CSS) to web page
- Use validator to check your XHTML code.

opiframe

ASP.NET

ASP.NET

* **ASP.NET is a server-side software framework designed to produce dynamic web pages.**
  * It was released with .NET Framework 1.0 in 2002.
  * Has extension for reading SOAP messages
* Future ASP.NET versions (”ASP.NET vNext”) will be:
  * More modular (for working with .NET Entity Framework)
  * New open-source compiler platform (will be cross-platform?)

ASP.NET Architecture

* **ASP.NET architecture consists of Core Services, Web Forms and MVC.**
  * **ASP.NET Core Services** handle HTTP request and how response is handled. They consist of several classes that implement sessions, state management, caching etc.
  * **ASP.NET Web Forms** (*.aspx) are used to create individual dynamic web pages. They consist of static XHTML mark-up and corresponding code-behind file (typically in C#) where dynamic code is implemented.
  * **ASP.NET MVC Framework** is an extension of ASP.NET that allows programming ASP.NET pages using model-view-programming (MVC) pattern.

ASP.NET Tools

- Following development tools are typically used for making ASP.NET software:
  - Microsoft Visual Studio
  - Microsoft Visual Web Developer Express
  - Microsoft Expression Web
  - Microsoft SharePoint Designer
  - ASP.NET Intellisense Generator
  - CodeGear Delphi
  - Macromedia Homesite
  - MonoDevelop
  - SharpDevelop
  - Eiffel for ASP.NET
  - Adobe Dreamweaver

ASP.NET Hosting

- **Web Applications must be hosted in a web server.**
  - Web server responds to a HTTP request by rendering web application to web browser.
- Large number of web hosts are available:
  - Apache Tomcat is very common web host used extensively with Java applications (JSF).
  - IIS (Microsoft Internet Information Server) is conventional web service that supports local and perimeter network. It can host ASP.NET, PHP and Node.js websites. Its main advantage is total control.
  - Windows Azure is a cloud platform that hosts web services. It is Microsoft-managed. Its main advantage is flexible scale.

Making ASP.NET Web Application

- Basic Web Page can be contructed with Visual Studio as following:
  - File → New Project...
  - Choose ”ASP.NET Web Application” → ”Empty Project”
  - Select ”Add” and ”Add New Item...”
    - Select a ”Web Form” and name it ”WebFormFirst”
- What you just did is a basic one page web forms application with a single ASP.NET page and code behind file!
  - Select design and write obligatory ”Hello World!”
    - Switch between design and Source to see change
  - Debug and see it in a web browser.

Making ASP.NET Web Application

- Select the web form to use Designer (down) and then select a button from Toolbox in left.
  - Change it's text to Press Me!
- Then add a Label (with ID Label1) from Toolbox
  - Add an Event that triggers with Click.
  - Add following to event handler in code-behind:
    - Label1.Text = DateTime.Now.ToString();
- Debug the web application

Making ASP.NET Web Application

* Add following line to use supported function in ASP.NET page:

<p> <% =DateTime.Now %></p>

* Debug the web application to test out ASP.NET’s methods…

ASP.NET Razor View Engine

* **ASP.NET uses ASP.NET syntax and Razor view engine for making simple text markup.**
  * Idea is to allow user write ASP.NET objects using HTML style.
* Normally ASP.NET elements can be written to HTML file as following: ”<%” and ”%>”.
* Razor follows same principle with keyword ”@” and ”@{ }”.
  * Razor is used extensively with ASP.NET MVC pages.

ASP.NET Intrinsic Objects

- **ASP.NET Core Services employ large number of Intrinsic Objects.**
- Most ASP.NET objects are invisible (not shown in web pages). Each HTML and ASP.NET element is effectively an object. Following namespaces are important:
  - **System** defines classes and interfaces used for client server communication.
  - **System.Web** defines classes for managing cookies, file transfers, output control, page contents, page requests, page transmissions, and access to serverside utilities.

HttpContext

- **A HttpContext object is always created when web browser sends a request to ASP.NET application (server side).**
  - HttpContext has Current object that refers to requested page.
- HttpContext.Current refers to following objects that handle aspects of request:
  - Response
  - Request
  - Server
  - Application
  - Session

Response

- **The Response object contains methods and properties related to browser output.**
  - It is typically used to change response output:

Response.Write(”Hello World Response!”);

Response.Write(”<p>This is formatted text.</p>”);

- Or set movement to a new web page:

Response.Redirect(”http://www.opiframe.com”);

Request

- **The Request object contains methods and properties related to web browser.**
    - It has information concerning cookies, browser type and passing values to directly:

```
Response.Write(Request.Browser.Browser);
Response.Write(Request.PhysicalApplicationPath);
```

Server

* **The Server object contains methods and properties related to web server.**
  * It is most commonly used to change server values (for example timeout values).

```
Response.Write(Server.MachineName);
Response.Write(Server.ScriptTimeout);
```

* Server object has methods execute and transfer for running different pages.

Application

* The Application object contains all classes related to currently running application.

```
Response.Write(Application.Contents);
```

opiframe

Session

- **The Session object contains methods and properties related to individual user(s) or instance(s) of a web site.**

```
Response.Write(Session.SessionID);
```

- Session object is also used to store information for the duration of session.

```
 if (Session["abcd"] != null)
        {
           Response.Write("<p>Session key abcd holds value " + 
Session["abcd"].ToString() + "</p>");
        }
        else
        {
           Session["abcd"] = 1234;
        }
```

opiframe

State

- **Web communication is stateless.**
  - Whenever web browser makes a request to server side the previous state in browser is lost and replaced with a new content.
- This limitation is bypassed with several techniques with modern web browsers:
  - **Cookies** store data to client side computer’s memory or hard drive. They are seldom used with ASP.NET coding.
  - **Query String techniques** add data to end of Page URL for Server to use when it processes and creates new pages. User can see (and manipulate) by hand these values so they are a security risk.
  - **Hidden elements** (”Hidden fields”) use invisible input fields within display to store data. They clutter the client side markup code.

State

* ASP.NET solution to statelessness problems is to use one of the following approaches:
  * **Session state** (server side technique) is based on Session object storing the session ID number and using it to access previous data.
    * Data is stored as long as the same session exists. This is main method when data to be stored is user based (for example a new store or banking service).
    * Session's length can be controlled with Session object's Abandon, Clear and RemoveAll methods. Session.Timeout is also useful.

```
Response.Write(Session.SessionID);
Session["MyValue"] = “abcd";
// Writes nothing if there is a null value
Response.Write(Session["MyValue"] == null);
```

State

- **Application state** (server side technique) is based on Application object storing the data for everyone using the application.
  - Data is stored as long as the application is running.

```
Application.Lock();
Application["MyValue"] = “abcd";
Application.UnLock();
```

State

- **View state** (client side technique) is based on storing the Web form control’s data into it. It is effectively a hidden element technique.
  - Set EnableViewState property as True.
- **Control state** (client side technique) works like View state but stores data to a control element.

Exercise 02

- Lets make our first ASP.NET web form and study HttpContext object and its effects on request – reply cycle.
  - Exercise 02: Create a new ASP.NET Web Form and use HttpContext object to show information.

Interactive Web Pages

opiframe

Interactive Web Pages

- **Web applications generate events that are triggered by web browser's request and server's response.**
- These events can happen at various levels:
  - Page
  - Application
  - Session
  - Control
- Whether events are triggered and handled depends on setting of **AutoEventWireup** attribute at various levels.
  - C# (true) and VB (false) due event system implementation.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile= "WebFormFirst.aspx.cs" Inherits= "MyApp.WebFormFirst" %>

Page Events

- **Page event is triggered when there is a change in web page.**
- Event is caught in Web form's code-behind according to following list during page's life cycle:
  - **PreInit** is used to test if this is a postback page.
  - **Init** is used to have control properties read or initialized.
  - **InitComplete** is used to change view state.
  - **PreLoad** is seldom used.
  - **Load** is used for setting control properties and database connections.
  - **LoadComplete** is used for those controls that need to be placed on form before their property changes take place.
  - **PreRender** is used to possible final changes. After this page is rendered!
  - **PreRenderComplete** is used for binding data to controls.
  - **SaveStateComplete** is used to change page while any value changes are not used for next postback.
  - **Unload** is fired first for each control found on the page and then to page itself. It is typically used for closing files and database connections. New page can be called to pop up.

Page Events

```
protected void Page_Load(object sender, EventArgs e)
{
        Response.Write(“Page_Load Event”);
}
```

opiframe

Application Events

- **Application event is triggered when there is a change in web application.**
- Web application related events are related either to web application’s own life cycle or web browser’s request cycle. Following events depend on web application’s own life cycle:
  - **Application_Start** is used to initialize variables used through the running of web application.
  - **Application_End** is used to free application-level resources or save logging information.
  - **Application_Error** is triggered when an error occurs on web application. This is used mostly in debugging at development stage to catch errors missing from other error handlers.
  - **Application_LogRequest** is used for custom logging.

Application Events

- Following events are triggered at web application according to page's handling request cycle:
  - **BeginRequest**
  - **AuthenticateRequest**
  - **DefaultAuthentication**
  - **AuthorizeRequest**
  - **ResolveRequestCache**
  - **AcquireRequestState**
  - **PreRequestHandlerExecute**
  - **PostRequestHandlerExecute**
  - **ReleaseRequestState**
  - **UpdateRequestCache**
  - **EndRequest**
- Note: All Application events are coded in Global.asax file.

Session Events

* **Individual browser sessions have life cycle roughly similar to page life cycle.**
* Each session is related to individual user and has two events related to starting and ending a session.
  * **OnStart** is triggered when user opens any page on web application for first time (i.e. does not have current session open).
    * Typically used for shopping cart opening for user in web stores.
  * **OnEnd** is triggered when session is terminated. This typically happens due timeout.
    * Typically any saved data is lost and cleanup code is run.
* Note: All Session events are coded in Global.asax file.

Global.asax

- Application and session events are ran in web server. The associated code is stored in global.asax.
  - Events are processed from global.aspx file and nothing from this file is passed to web browser (client side).

Global.asax

```
<script language="C#" runat="server">
public void Application_OnStart()
{
		Application[“MyVisitors"] = 0;
}
public void Session_OnStart()
{
Application.Lock();
Application["MyVisitors"] = (int) Application["MyVisitors"] + 1;
Application.UnLock();
}
</script>
```

Control Events

- **Control events are triggered by control’s life cycle and user’s activities.**
  - Most control events are same as Page events.
  - Most control’s have also events special to that control type.
- Note: Click and Command events are both similar except information it will hold and both are fired when user clicks control.

Control Events

* Button click event set in web form:

```
<asp:Button Text="Press Me!" runat="server" id="Button1"
OnClick="Button1_Click" />
```

* Button event handler in code back:

```
protected void Button1_Click(object sender, EventArgs e)
{
        if (TextBox1.Text != "") {
                Label1.Text = "Hello World, " + TextBox1.Text;
        }
        else {
                Label1.Text = "Information is missing!";
        }
}
```

Postback Property

* **ASP.NET allows separating some code that is only run when page is loaded for the first time with Postback property of ViewState object.**
* Normally dynamic web page is completely reloaded at every request as server processes it and sends back its response for rendering. This causes web page initialize whenever browser sends a request to server (with Load event).

```
private void Page_Load(object sender, System.EventArgs e)
{
        if(Page.IsPostBack)
        {
                // This code is executed when page reloads!
        }
        else
        {
                // This code is executed only once as page is rendered for first time!
        }
}
```

Exercise 03

- Lets add features to our ASP.NET web form.
  - Exercise 03: Lets make a new ASP.NET web form page and add a number of event handlers to it.

Navigating Between Pages

- ASP.NET web applications support multiple web pages with data transfer between them.
- Following two methods are typically employed:
  - **Response.Redirect** is most common method to navigate between web pages and different web sites.
    - It is typically used with Session object to store the information.
  - **Server.Transfer** is used to transfer user to another web page within web site without changing the URL. Web browser appears to be showing the same page.
    - It is typically used with a property from previous page being used to transfer information.
    - Only works with same server and between active service pages.

Controls

* **ASP.NET web applications support wide variety of controls.**
* Most commonly used controls are:
  * **HTML controls** create true HTML elements.
  * **Standard controls** are commonly used general purpose controls.
  * **Validation controls** for validating user input.
  * **Data Controls** for accessing database.
  * **AJAX Extensions** for client-side scripting support.
* Note: User Controls are not controls but they can be made to behave like controls.

Controls

* Microsoft suggest following guidelines in selection of controls:
  * User controls should be used when the web site has multiple pages with mostly similar controls. Reusing simplifies code.
  * Server controls should be used with very interactive web page. This enhances security of both data and source code. This is standard solution with Microsoft products.
  * HTML controls should be used when web page requires only simple (and unsecure) client-side scripting (typically JavaScript) to work. This is standard solution with PHP.
  * Validation controls should be always used when users generate a lot of input, which must be highly accurate for service to function correctly. This is standard solution with web based services to wider public.

HTML Controls

- HTML Controls are widely used HTML language based controls. They are ”old way” of doing things:
  - **Input (Button)** is a Button control.
  - **Input (Reset)** is a button control that clears a form from form viewer data.
  - **Input (Submit)** triggers a processing event. It is typically launched to process form data.
  - **Input (Text)** is a textbox control.
  - **Input (File)** is a input textbox that gets from local computer pathname for file to be uploaded to server.
  - **Input (Password)** is input textbox masking its input. It is used for passwords.
  - **Input (Check box)** is a Checkbox control.
  - **Input (Radio)** creates a Radio button control.
  - **Input (Hidden)** is a Label that is hidden from user.
  - **Textarea** is a multiline text box control with scroll bars.
  - **Table** is used to create a table.
  - **Image** displays an image element from a source image.
  - **Select (Dropdown)** is a dropdown box control.
  - **Horizontal Rule** draws a horizontal line across web page.
  - **Div** is used to create a flow layout panel control.

Standard Controls

* Standard Controls are ASP.NET controls for common general purpose interaction with user.
    * **Label** is used to display raw text.
    * **Button** is standard push button control.
    * **TextBox** is used for typed input.
    * **LinkButton** is a button that is displayed as hyperlink.
    * **ImageButton** is an button control displayed as image.
    * **HyperLink** is used to navigate between web sites without code-behind.
    * **DropDownList** is a list of items.
    * **ListBox** is a list of items always displayed
    * **CheckBox** is a checkable control.
    * **CheckBoxList** is a list of checkable items.
    * **RadioButton** is a radiobutton in set in a group with GroupName property.
    * **RadioButtonList** is a list of radio button items.
    * **Image** shows an image.
    * **Calendar** is used to display an almanac.
    * **Table** is used to create an table.

Validator Controls

* **Validation Controls are used for validating user input.**
  * **CompareValidator**
  * **CustomValidator**
  * **DynamicValidator**
  * **RangeValidator**
  * **RegularExpressionValidator**
  * **RequiredField Validator**
* Note: **ValidationSummary** is used to report validation errors.

User Controls

- **User controls can be used to create custom control functionality.**
  - They are System.Web.UI.UserControl objects
  - Stores as *.ascx extensions.
- Note: User Controls may look similar to web pages but they cannot be run directly in web browser.

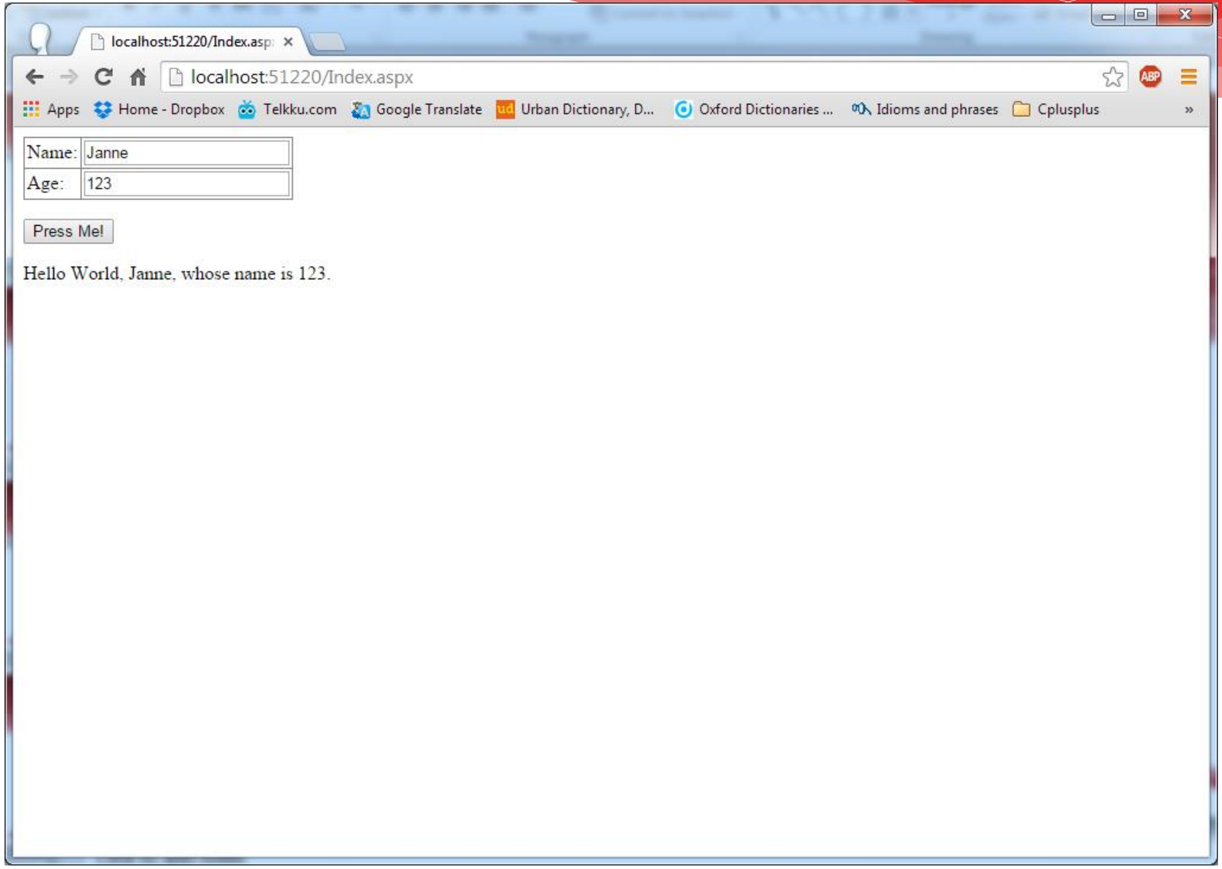
Configuration Files

- **Most ASP.NET configuration files allow changing their content without shutting down the web site.**
- The .NET configuration system uses single master file (machine.config) with all necessary settings for entire .NET system. Local configuration files override this file as necessary. This happens very often.
  - Windows-based applications can override this with app.config file.
  - Web applications can override this with web.config file.
- Typically overriding is done by changing the web.config file in shared or local directory.
- You can see your machine.config file in:
  - **C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFI3**

Exercise 04

- Lets make our next ASP.NET web form.
  - Exercise 04: Lets make a simple ASP.NET web form page and use it to implement a table.

Exercise 04



Name:	Janne
Age:	123

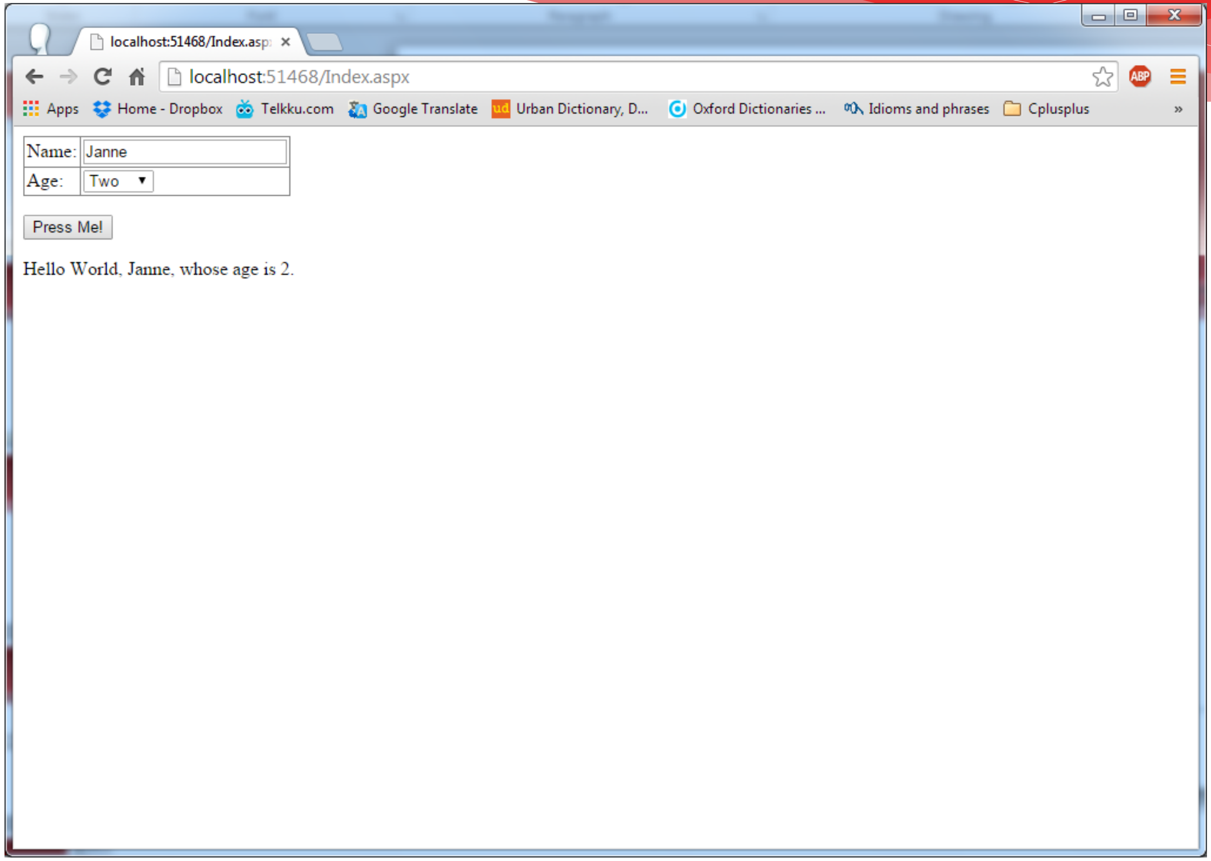
Hello World, Janne, whose name is 123.

opiframe

Exercise 05

- Lets make next ASP.NET Exercise.
  - Exercise 05: Change previous exercise result so that age is selected from a dropdown list filled with strings.

Exercise 05



Name:	Janne
Age:	Two

Press Me!

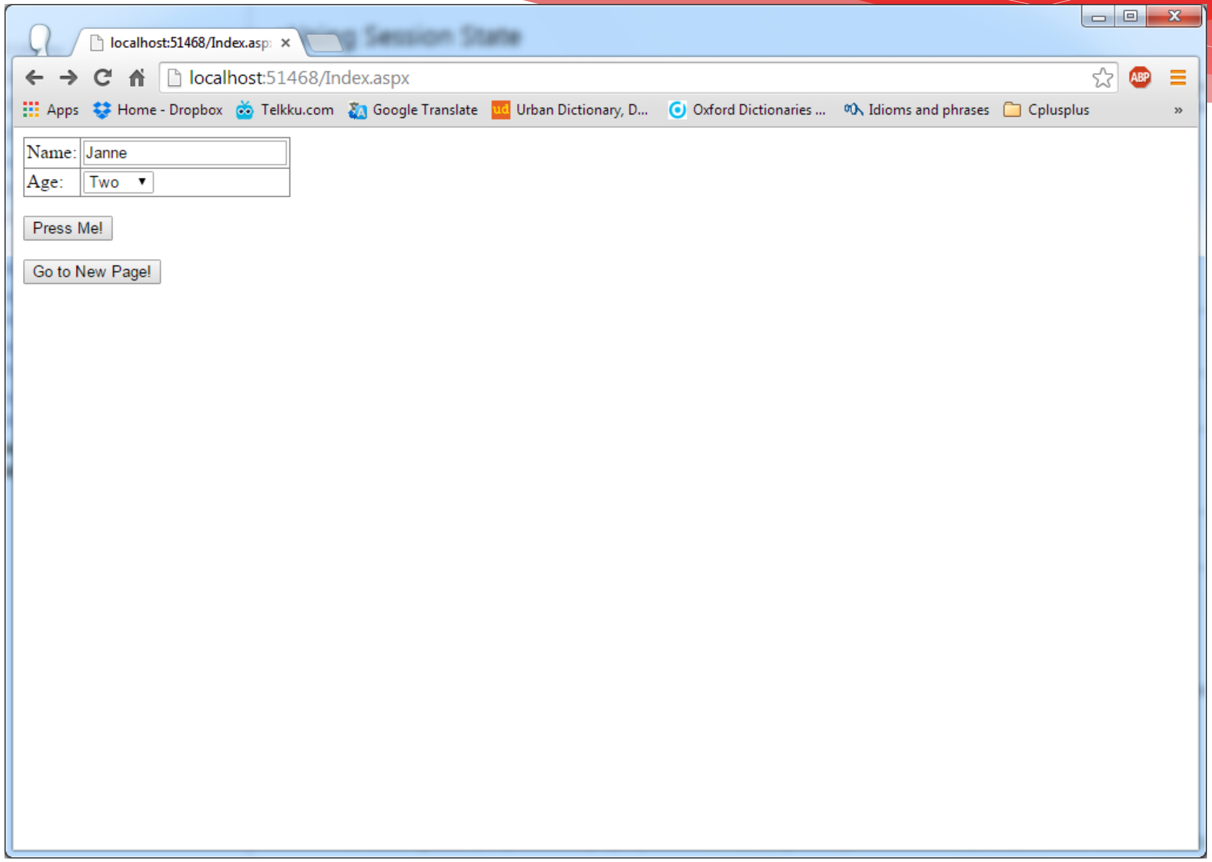
Hello World, Janne, whose age is 2.

opiframe

Exercise 06

- Lets make next ASP.NET Exercise.
  - Exercise 06: Change previous exercise result so that there is a new button that allows changing pages and data is shown in the new second page.

Exercise 06

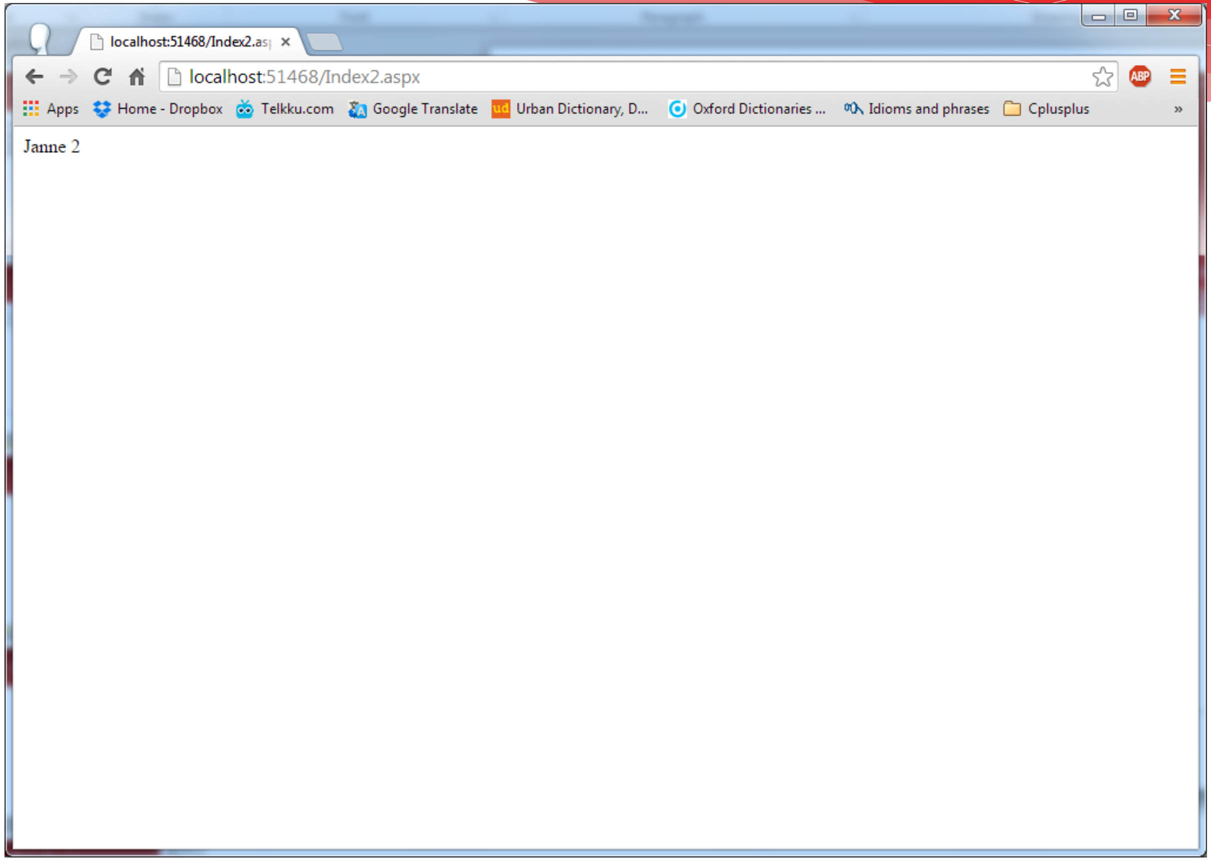


Name:	Janne
Age:	Two

Press Me!

Go to New Page!

Exercise 06



```
Janne 2
```

Data Services

Data Services

* ASP.NET uses XML files to store configurations and data structures.
* Works best with simple data structures.
  * A single sheet with large number of rows.
  * List with large number number of aggregate data objects.
* XML was developed from SGML (used in 1980's).
  * Uses SGML rules and HTML format.
  * Files are written as *.xml.
* Note: XML file can be validated with:
  * **http://validator.w3.org**

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<!—— This is a comment ——>
<animallist >
<title>
</title>
<item>
        <cat>Siamese</cat>
        <weight amount=123>ValueA</weight>
</item>
<item>
        <cat>Street</cat>
        <weight amount=456>ValueB</weight>
</item>
</animallist>
```

DTD Validation

- **XML and XHTML document can also include language rules that follow DTD (Document Type Definition).**
    - DTD rules should be added to XML file so elements are always validated. Use: http://validator.W3.org . It is called “schema”.

```
<!DOCTYPE animallist [
<!ELEMENT title (#PCDATA)>
<!ELEMENT cat (#PCDATA)>
<!ELEMENT weight (#PCDATA)>
<!ELEMENT animallist (title, item*)>
<!ELEMENT item (cat, weight)>
<!ATTLIST weight amount CDATA "123">
]>
```

XML Parsering

* XML files and feeds are used with XmlDataSource element.
    * This data source is typically shown with a Repeater element

```
<asp:Repeater ID="Repeater" runat="server" DataSourceID="RSSDataSource">
  <ItemTemplate>
     <asp:HyperLink runat="server"
        Text='<%# XPath("title") %>'
        NavigateUrl='<%# XPath("guid") %>' />
     <asp:Label runat="server"
        Text='<%# DateTime.Parse(XPath("pubDate").ToString()) %>' />
     </br>
   </ItemTemplate>
  </asp:Repeater>

<asp:XmlDataSource ID="RSSDataSource" DataFile="http://yle.fi/uutiset/rss/paauutiset.rss"
XPath="rss/channel/item" runat="server"></asp:XmlDataSource>
```

Exercise 11

- Lets make our next ASP.NET web form.
  - Exercise 11: Lets make a simple ASP.NET web form page and parser some XML data from RSS feed of YLE.
  - http://yle.fi/uutiset/rss/paauutiset.rss

Exercise 12

- Lets make our next ASP.NET web form.
  - Exercise 12: Lets make a simple ASP.NET web form page and parser some XML data from a XML file (XmlFileCat.xml shown in next slide) with repeater.

Exercise 12

* Employ following as your XmlFileCat.xml file:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is a comment -->
<animallist >
  <title>
  </title>
  <item>
    <cat>
      <race>Siamese</race>
      <name>Sari</name>
    </cat>
    <weight amount="123">ValueA</weight>
  </item>
  <item>
    <cat>
      <race>Street</race>
      <name>Sanna</name>
    </cat>
    <weight amount="456">ValueB</weight>
  </item>
</animallist>
```

DataBase

- **Web applications uses databases to store most of their data.**
  - Databases use typically SQL language. It was created 1980's from SEQUEL language invented in 1970's (by IBM).
  - Almost all databases are relational databases where tables can relate to each other through keys.
- Data is stored in tables.
  - Each **row** is one record in table.
  - Each **column** is one data item type.

DataBase

- DataBases are managed with with Database Management Systems.
  - Microsoft SQL Server is typically managed with SQL Server Management Studio.
- Actual database is accessed from ASP.NET application with two objects:
  - **DataReader** is used to access DB for reading it.
  - **DataSet** is used to both reading and changing DB.
- Note: Database access uses System.Data.SqlClient namespace.

DataReader

* DataReader object is used to read-only from DB.

```
 SqlConnection sc = new SqlConnection();
 sc.ConnectionString="Data Source= MYCOMPUTERNAME\\SQLEXPRESS;Initial Catalog=
MyDBName;Integrated Security=True";

 sc.Open();
 SqlCommand cmd = new SqlCommand ("Select * from AnimalList", sc);

 SqlDataReader myreader = cmd.ExecuteReader();
 while (myreader)
 {
                myreader.Read();
                Response.Write(Convert.ToString(myreader[0]) + "<br>");
                myreader.Read();
                Response.Write(Convert.ToString(myreader[1]) + "<br>");
}
sc.Close();
```

Connection String

- Following terms should be used:
  - The **DataSource** uses Server, Addr, Address, Network Address as synonyms.
  - The **Initial Catalog** uses Database as synonym.
  - The **Integrated Security** has three settings:
    - **True** for using Windows integrated security (and bypassing User ID and Password settings).
    - **False** for using User ID and Password settings for DB server.
    - **SSPI** uses Windows security if no User ID/Password is set and DB server's password when it is not present.
  - The **MultipleActiveResultSets** boolean value determines if application is allowed to do multiple simultaneous queries.
    - This is often True with Entity Framework and lazy loading enabled.
- Note: Use http://www.connectionstrings.com/ for checking out connectionstrings to different DB servers.

Connection String

* The connection string is typically stored in web.config file:

```
<?xml version="1.0"?>
<configuration>
<connectionStrings>
<add name=”ConnectionStringName" providerName="System.Data.SqlClient" connectionString="Data Source=.\SQLEXPRESS;Initial Catalog=DatabaseName;Integrated Security=True;MultipleActiveResultSets=True" />
</connectionStrings>
</configuration>
```

DataSet

* DataSet works in similar fashion to DataReader:

```
SqlDataAdapter da = new SqlDataAdapter ("Select * from
AnimalList", sc);
DataSet set = new DataSet();
da.Fill(set, "MyTable");
Response.Write(set.Tables[0].Rows[0][0].ToString());
Response.Write(set.Tables[0].Rows[0][1].ToString());
```

* The DataSet can be used to store data to database.

```
da.Tables["MyTable"].Rows[0][0] = ”Jenny";
da.Tables["MyTable"].Rows[0][1] = "123";
da.Update(set, "MyTable");
```

Entity Framework

- Entity Framework is a object-relational mapping framework to ADO.NET (and subsequently all .NET applications).
  - Used to do ”DB First” programming.
  - Can be used with both DB and XML files.
- Entity Framework does not come automatically with Web Forms programs in Visual Studio. It would be found in References of your program (EntityFramework and EntityFramework.SqlServer).
- Entity Framework must be downloaded separately.
  - References → Manage NuGet Packages...
  - Select ”EntityFramework”
- Web.config file should show this framework in use alongside References section of your program.

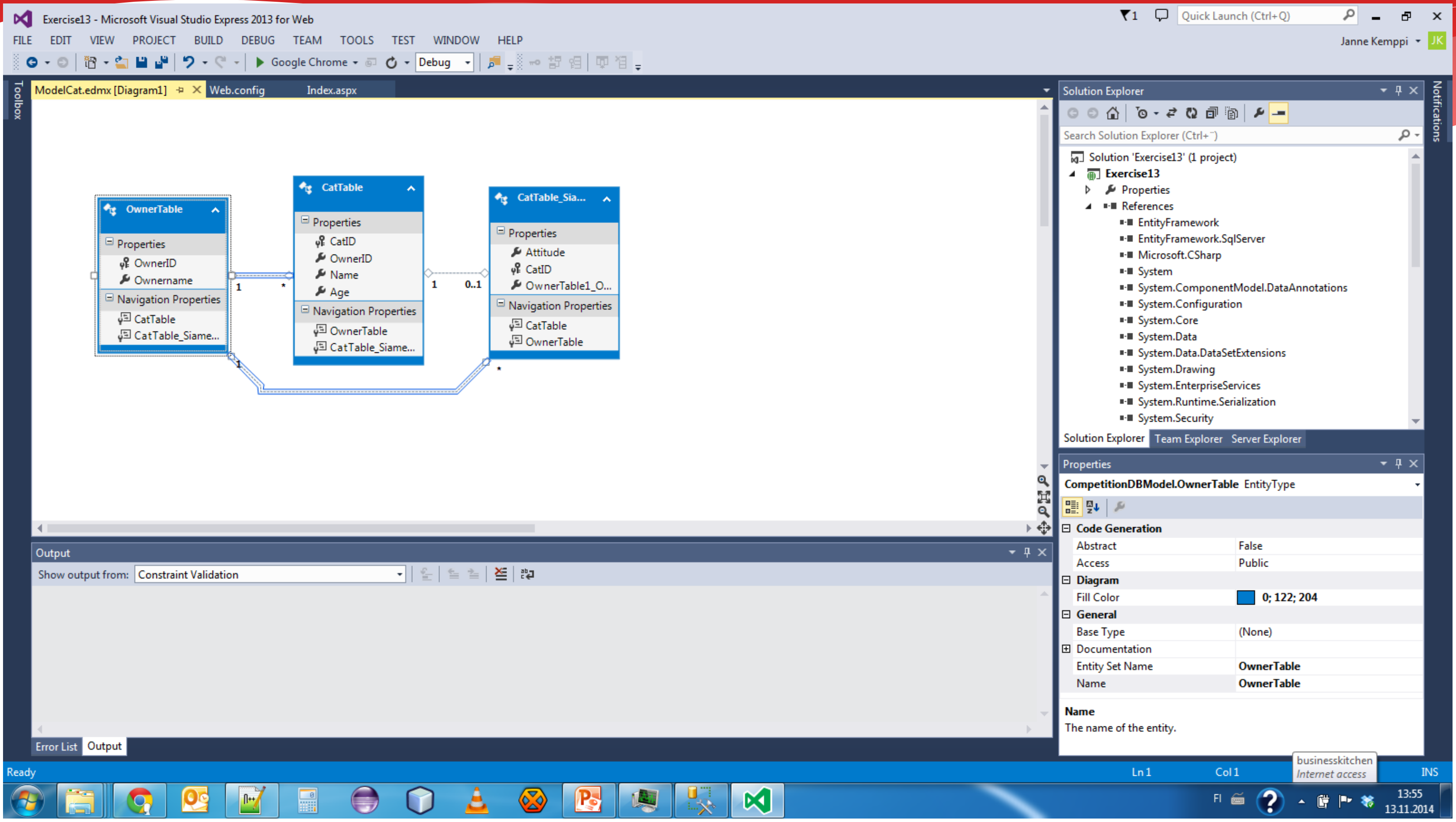
Entity Framework

- First step is to add a connection to database server:
  - TOOLS → Connect to Database…
  - Database server’s address is used as ”Server Name”
  - Selecting ”Refresh” should make everything work.
- Second step is selecting the database to use:
  - Pick from ”Select or enter database name”
- Test Connection makes sure there is a valid connection to database server and database within it.
  - If not, check out how you do authentication…
- Finally press OK if everything works. The Data Explorer should show your DB as a Data Connection…

Entity Framework

- Third step is to bring model of the DB with Entity Framework.
  - Models directory → Add… → ”ADO.NET Entity Data Model”
  - Give model a name it is used in program.
  - The model is created with wizard. The ”EF Designer from Database” creates model from existing DB. It is easiest way.
  - Most important part of wizard is the selection of what goes into model. Typically we use Tables and Views and some of them should be selected.

Entity Framework



Solution Explorer

Search Solution Explorer (Ctrl+¨)

- Solution 'Exercise13' (1 project)
  - Exercise13
    - Properties
    - References
      - EntityFramework
      - EntityFramework.SqlServer
      - Microsoft.CSharp
      - System
      - System.ComponentModel.DataAnnotations
      - System.Configuration
      - System.Core
      - System.Data
      - System.Data.DataSetExtensions
      - System.Drawing
      - System.EnterpriseServices
      - System.Runtime.Serialization
      - System.Security

Solution Explorer | Team Explorer | Server Explorer

Properties

CompetitionDBModel.OwnerTable EntityType

Code Generation	
Abstract	False
Access	Public
Diagram	
Fill Color	🟦 0; 122; 204
General	
Base Type	(None)
Documentation	
Entity Set Name	OwnerTable
Name	OwnerTable

Name
The name of the entity.

Entity Framework

* Each table is now a class in program:

```
namespace Exercise13.Models
{
  using System;
  using System.Collections.Generic;

  public partial class CatTable
  {
    public int CatID { get; set; }
    public int OwnerID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }

    public virtual OwnerTable OwnerTable { get; set; }
    public virtual CatTable_SiameseTable CatTable_SiameseTable { get; set; }
  }
}
```

Entity Framework

* The contect file creates the class that is used as a context to fetch data from and to.

```
namespace Exercise13.Models
{
  using System;
  using System.Data.Entity;
  using System.Data.Entity.Infrastructure;

  public partial class CompetitionDBEntities : DbContext
  {
    public CompetitionDBEntities()  : base("name=CompetitionDBEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
      throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<CatTable> CatTable { get; set; }
    public virtual DbSet<CatTable_SiameseTable> CatTable_SiameseTable { get; set; }
    public virtual DbSet<OwnerTable> OwnerTable { get; set; }
  }
}
```

Entity Framework

- Ultimately the idea is to display the data in a table.
  - Web Forms → "Design" view
  - Toolbox → Data and choose some display method (typically GridView).
  - "Choose Data Source" and select one from available.
    - Typically Entity (since we use Entity Framework)
    - Object context is tied to selected model (DataSet) with Named Connection
    - EntitySetName should be table to be presented.
- Note: If something goes wrong look at *.edmx file and see if ProviderManifestToken is appropriate (usually 2008)

Entity Framework

- The combination of entity data source and gridview is following:

```
 <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False"
DataSourceID="EntityDataSource1">
        <Columns>
            <asp:BoundField DataField="Name" HeaderText="NAME"
ReadOnly="True" SortExpression="Name" />
            <asp:BoundField DataField="Age" HeaderText="AGE"
ReadOnly="True" SortExpression="Age" />
        </Columns>
    </asp:GridView>
```

Entity Framework

```
<asp:EntityDataSource ID="EntityDataSource1" runat="server"
ConnectionString="name=CompetitionDBEntities"
DefaultContainerName="CompetitionDBEntities"
EnableFlattening="False" EntitySetName="CatTable" Select="it.[Name],
it.[Age]">
</asp:EntityDataSource>

<asp:EntityDataSource ID="EntityDataSource2" runat="server"
ConnectionString="name=CompetitionDBEntities"
DefaultContainerName="CompetitionDBEntities"
EnableFlattening="False" EntitySetName="OwnerTable"
Select="it.[Ownername]">
</asp:EntityDataSource>
```

Exercise 13

- Lets make our next ASP.NET web form.
  - Exercise 13: Lets make a simple ASP.NET web form page and parser some data from a DB.

Client-side Scripts

Client-side Scripts

* **Client-side scripting is used to carry out operations that do not require web browser sending a request to server.**
  * This makes page much more responsive (no waiting for round trip).
  * Downside is threat of malware running in web browser's virtual machine.
  * Typically done with JavaScript.

Client-Side Scripts

* Typical JavaScript code

```
<script>
        function MyFunc()
        {
                var myVar = 123;
                alert(’text’ + myVar);
        }
</script>
...
<input type="button" onClick=”MyFunc( )" value=”Test" />
```

Client-side Scripts

* Typically javascripts are stored in a separate file (*.js) called from page with **src** attribute.

```
<script type="text/javascript" src=”Test.js">
</>
</script>
```

Client-Side Scripts with DOM

- **XML Document Object Model (DOM) specification defines the structure of the Web page and all its components.**
  - Any XML structure holding HTML data is XHTML.
- Client-side script can manipulate XHTML documents changing display to match DOM without recreating web page (or going to web server).
- XML uses a tree-branching with root, children, siblings and leaf as terms.
- Note: DOM structure can be browsed with IE with Developer Toolbar.

Client-Side Scripts with AJAX

* **Ajax concept means that client-side script (JavaScript) makes a call to server-side for updating page without waiting for page request and refresh.**
  * Jesse James Garret: Ajax: A New Approach to Web Applications (2005.)
* ASP.NET's implementation of Ajax is named AJAX. It is JavaScript extension with following controls:
  * Pointer
  * ScriptManager
  * ScriptManagerProxy
  * Timer
  * UpdatePanel
  * UpdateProgress

AJAX

* **ScriptManager** control works essentially in same way as partial page rendering. It is inherited from master page. It is enabled with **EnablePartialRendering** attribute.
* **ScriptManagerProxy** control is essentially same as ScriptManager but it works similarly for this page only. This overrides master page inheritance.
* **Timer** control is used to delay execution of script with **Interval** property. It is often used to display real-time clock but it is not entirely reliable with 1sec or less interval.
* **UpdateProgress** control responds to server triggering.
* **UpdatePanel** control is a collection of controls which are updated if one of the child controls is changed (if **ChildrenAsTriggers** is set to ”True”). **UpdateMode** attribute sets if panel is always updated when any panel is updated (”Always”) or only if specifically asked (”Conditional”).

AJAX

* ScriptManager can also be used to manage a collection of scripts as well as registered services.

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
<Scripts>
        <asp:ScriptReference Path=”Test.js" />
</Scripts>
<Services>
        <asp:ServiceReference Path=”Test.asmx" />
</Services>
</asp:ScriptManager>
```

AJAX Extender Controls

- AJAX allows any controller become extender control.
- Extender controls are two-edged sword:
  - You get new controls free! ☺
  - The controls do not have quality control or long-term maintenance or updates. Can become unavailable suddenly.
- Make your own extender controls:
  - Get what you want and maintenance expertise is in-house.
  - Can be very expensive. Requires expertise.
- Companies are often better served by buying extender controls from another company:
  - Costs money. ☹
  - Safe for long-term functionality and reliability → Quality.

Client-Side Libraries

- ***Client-side libraries*** **hold client-side programming code run in the web browser.**
  - Necessary files are downloaded and referenced from HTML browser.
- JavaScript libraries have different purposes:
  - New and more complex controls (widgets).
  - Less coding with factories. The jQuery is often used in this.
- Several client-side libraries:
  - The jQuery in http://jquery.com
  - The Prototype JS in http://prototypejs.org
  - The Dojo in http://dojotoolkit.org

Exercise 14

- Lets make our next ASP.NET web form.
  - Exercise 14: Lets make a simple ASP.NET web form page and use JavaScripting to change text to upper or lower set.

Web Application Debugging

Web Application Debugging

- **Web application debugging is difficult because some errors are not caught by application based error loggers.**
  - ASP.NET uses custom debugging settings to determine which application gets what in error message.
- Custom Debugging is set in web.config file with **<customErrors>** tags.
  - The **defaultRedirect** attribute overrides the detailed error page with custom error page in URL address.
  - The **mode** sets whether the detailed and custom error pages are shown in local machine or remote machine.
    - The **RemoteOnly** means that detailed errors are shown in local machine and custom errors in remote machine. This is normal setting in production environment.
    - The **Off** means that that all see detailed errors. This is normal setting in development environment.

Page Tracing

- **Page tracing is a technique where logical errors are caught by simply moving step by step as page is being handled.**
  - Tracing works only in local machine.
- Page tracing is set as following:
  - Web Pages set tracing in a page basis in Visual Studio by setting **Trace** attribute of Page element to ”true”.
  - Web Application set tracing in a page basis in Visual Studio by setting **system.Web** attribute in **web.config** file with maximum numbers of trace requests server will allow.

<trace enabled="true" requestLimit=”10" />

  - Then add **Trace.axd** file to your program.

Error Status Codes

- Contents of custom error pages depend on error being generated.
  - HTTP Status Codes can be found at: http://www.w3.org/.

```
<customErrors mode="On"  defaultRedirect="PageA.htm">
<error statusCode="404" redirect  =“PageB.aspx"/>
<error statusCode="501" redirect  =“PageC.aspx"/>
</customErrors>
```

Web Application Configuration & Deployment

Web Application Configuration & Deployment

* **Web application security is based on authentication and authorization.**
  * **Authentication** means ensuring that user is who she claims to be. This is done by identifying anonymous users.
    * Typical solution is a login page.
  * **Authorization** means ensuring that user can only access resources she needs. This is done by assigning rights to identified user.
    * This is done by redirecting user who does not have rights back to a web page where she does have rights to be (typically web page she came back to).

Authentication

* **ASP.NET has several authentication methods which are set in web.config file (using <authentication> tag).**
  * **Windows Live ID authentication.** Value is ”Passport”.
  * **Microsoft IIS (Internet Identification Services).** Value is ”Windows”. This is typically used with intranets.
  * **Forms authentication.** Value is ”Forms”. This means Web application itself has code to deal with authentication issues.
  * **No authentication.** Value is ”None”.

Forms Authentication

* **Forms authentication is used when web application handles its authentication needs.**
    * Developer decides where and how user Ids and passwords are stored. Typically small applications use web.config file and large applications DB or XML files.

```
<authentication mode="Forms">
<forms>
<credentials passwordFormat="Clear">
          <user name=“Janne" password=“janne" />
          <user name="Sari" password=”sari” />
</credentials>
</forms>
</authentication>
```

Forms Authorization

* **Forms authorization is used when web application handles its authorization needs.**
* Forms authorization is set with <authorization> tag. It has several possible values:
  * The ”**allow**” and ”**deny**” are set at element level.
  * The ”**users**”, ”**roles**” and ”**verbs**” are set at attribute level.
  * The ”**?**” (anonymous users) and ”*****” (authenticated users) are possible attribute values.

```
<authorization>
        <allow users=”Janne"/>
        <deny users="?"/>
</authorization>
```

opiframe

Forms Authorization

* The next step is adding code for redirecting user whenever necesssary to login page (for example Login.aspx).

```
FormsAuthentication.SignOut();
Response.Redirect("Login.aspx");
```

* The login page should have two textboxes and a button.

```
if (FormsAuthentication.Authenticate (Box1.Text, Box2.Text)
        FormsAuthentication.RedirectFromLoginPage (Box1.Text,
false);
else
        Response.Write(“Failed to login");
```

Impersonation

- Impersonation can be used to control user authorization when ASP.NET and IIS are used to control access.
  - First, IIS checks that incoming user comes from a valid IP address.
  - Second, ASP.NET then checks if **inpersonation** has been set or not:
    - If set, ASP.NET assumes authentication is successful and then continues with authorization.
    - If not set (default), ASP.NET processes the authentication and authorization.

Impersonation

* Impersonation is set in web.config file.
* For example, all users are accepted:

<identityimpersonate="true" />

* Alternatively a user group might be accepted:

<identityimpersonate="true" userName="Accounting" pass word="Password" />

IIS

* **IIS is a extensible web server intended for hosting web sites and subsequently web applications.**
  * It acceps and supervises requests done via HTTP protocol.
  * HTTP protocol using requests can be used in gaming, data storage, enterprise applications, email server, FTP server etc.
* Note: You can check if your old Windows computer has IIS by looking for inetpub directory.
* Note: You can check if your computer is running IIS by starting a web browser and typing: http://localhost as URL.

Web Application Publishing

- **Web site applications are published as either development or production environment and need to maintained.**
  - Idea is to make and test updates and changes in development environment first and then upload them to production environment.
- Maintenance procedure depends on environment complexity:
  - Simple HTML page written with notepad can be updated and maintained by simply copying replacement files.
  - ASP.NET applications built an tested can be published directly with IDE such as VisualStudio.
- Major question is also who hosts service.
  - Company ran server needs to consider operating system, network and database support. This requires expertise.
  - Small company usually writes the web application and uses a hosting-service provided by other company.

Windows Installer

- **Windows Installer (Microsoft Installer, MSI) can be used to create a setup file.**
- Web sites are web applications are published in different ways:
  - Web sites are typically published by copying files to web server.
  - Web application configurations can be set with Visual Studio Deployment Wizard for various installations.
- Note: IIS has application pools for isolating multiple web applications running on same computer.

Questions?

opiframe