The following is a list of sections in this document. You can use these as hyperlinks to navigate to any particular section.

|  |  |
| --- | --- |
| [Starting a Simple Web App in Visual Studio](#_Starting_a_Simple) | [Class Libraries](#_Class_Libraries) |
| [Page Directive](#_Page_Directive) | [New Table from Visual Studio – Fast](#_New_Table_from) |
| [Table Modifications](#_Table_Modifications) | [GridView Control](#_GridView_Control) |
| [Validation Controls](#_Validation_Controls) | [JQuery Error in Response to Using a Validation Control](#_JQuery_Error_in) |
| [Validation Summary](#_Validation_Summary) | [Performing Validation on the Server Side](#_Performing_Validation_on) |
| [Prevent Very Long Text Strings](#_Prevent_Very_Long) | [Easy to Use Setting Default Behavior](#_Easy_to_Use) |
| [Insert a User Control into a Page](#_Insert_a_User) | [Insert a User Control Dynamically](#_Insert_a_User_1) |
| [SQL Server Database Facility](#_SQL_Server_Database) | [Incompatible .mdf Files That Were Downloaded from Pluralsight Exercise Material](#_Incompatible_.mdf_Files) |
| [Error Message: connection to SQL Server](#_Error_Message:_connection) | [SQL Data Source - Data Binding Expressions](#_SQL_Data_Source) |
| [Introduction to SQLDataSource Control](#_Introduction_to_SQLDataSource) | [GridView SelectedValue](#_GridView_SelectedValue) |
| [Setting Up ObjectDataSource](#_Setting_Up_ObjectDataSource) | [Starting the EntityDataSource Demo](#_Starting_the_EntityDataSource) |
| [Entity Framework Overview](#_Entity_Framework_Overview) | [Entity Framework – Code First](#_Entity_Framework_–) |
| [Entity Framework – Code First – When Classes Change](#_Entity_Framework_–_1) | [Master Pages](#_Master_Pages) |
| [AJAX Basics](#_AJAX_Basics) | [AJAX Script-Manager Demo](#_AJAX_Script-Manager_Demo) |
| [AJAX Script-Manager Demo.2](#_AJAX_Script-Manager_Demo.2) | [Adding AJAX to Previously Built Page – Entity-framework Time Tracker with Master Page](#_Adding_AJAX_to) |
| [Authentication and Authorization](#_Authentication_and_Authorization) | [ASP.NET Dynamic Data](#_ASP.NET_Dynamic_Data) |
| [Configuring IIS Locally](#_Configuring_IIS_Locally) | [Deploying Web Application by Hand](#_Deploying_Web_Application) |
| [Automatic Deploy](#_Web_Deploy)ment |  |

# Starting a Simple Web App in Visual Studio

Solution: *IntroductionToASP\_NET*

The section [Introduction to ASP.NET].[Introduction to Visual Studio] was evidently written around 2010, and the instructor was using an older version of Visual Studio. Therefore, it is a challenge to follow by example from the course. The following is what I did to build a simple web application with Visual Studio 2019. (The instructor also mentioned simple Web Sites. I tried to build a simple Web Site. I managed to create the solution and the project, but I was unable to add a C# web page to the project.)

* Open Visual Studio 2019. Visual Studio displays 2 lists of options. The list in the right panel is titled **Get Started**.
* Click **Create a new project**. Visual Studio displays a dialog titled **Create a new project**.
* If you don’t see **Blank Solution**, type “blank” into the search box in the upper-right corner. Visual Studio shows **Blank Solution** in the right-hand panel.
* Click **Blank Solution** in the right-hand panel; then click the **Next** command button. Visual Studio displays a dialog titled **Configure your new project**.
* Type the solution name into the **Project name** text box. Then click the **Create** command button. Visual Studio displays a blank solution with the given name.
* Right-click the solution name in the **Solution Explorer** panel. Visual Studio displays a pop-up menu.
* Select **Add… > New Project…** in the pop-up menu. Visual Studio displays a dialog titled **Add a new project**.
* Select **ASP.NET Web Application (.NET Framework)**. Then click the **Next** command button. Visual Studio displays a dialog titled **Configure your new project**.
* Type the project’s name into the **Project name** text box. Examine the path in the **Location** text box, and modify it as desired. Finally click the **Create** command button. Visual Studio displays a blank solution with the given name. Visual Studio displays a dialog titled **Create a new ASP.NET Web Application** with **Empty** highlighted.
* Click the **Create** command button. Visual Studio displays the new project in the right-hand panel.
* You can right-click the project name and select **Add > Web Form** to add one or more web pages to the project.

# Class Libraries

Solution: enhancement of *IntroductionToASP\_NET*

If you add the class library to the solution, then when the solution is built, changes to the code in the class library will be compiled, and “dependencies” is not an issue.

You will need to

1. Add the class-library’s project name to the references of any other project that uses a class in the library.
2. Add “using” statement to a c# source file where the reference occurs.

# Page Directive

The **Page** directive is at the top of the .aspx code of a web form. Its structure is

<%@ Language=”C#” AutoEventWireup=”true” . . . %>

A feature that might be convenient for debugging purposes is **Trace=”true”**; adding this (temporarily) to the **Page** directive induces the browser to display a wealth of information such as Session State, Cookies, etc.

For more information use Google with the keywords “asp.net page directive trace” or refer to <https://docs.microsoft.com/en-us/previous-versions/aspnet/bb386420(v%3Dvs.100)>.

Also in the **Page** directive you can use **MaintainScrollPositionOnPostback=”true”**; when this is included, Java Script is inserted into the page that prevents the scrollbar from changing when the user does something that results in a postback on the page.

The syntax for a comment in a .aspx page is <%-- *comment here --*%>.

The syntax for a comment in an XML file – for example web.config or an ordinary HTML file - is

<!-- *comment here --*>.

# New Table from Visual Studio – Fast

Solution: *IntroASP\_NETusingWebControls*

Using the **Design** view (tab at the bottom of .aspx document), click **Table** from the menu bar at the top; Visual Studio, in response, presents a convenient dialog one can use to build the table.

# Table Modifications

Solution: *IntroASP\_NETusingWebControls*

To insert a row into the table from **DesignView** –

* Click mouse to set focus on a cell in the row.
* Use the pull-down menu (menu bar at the top) click **Table > Insert > Rows or Columns…** Visual Studio pops up a dialog titled **Insert Rows or Columns**.
* Follow instructions in the dialog (rows radio button, number of Rows or Number of Columns, Above selection or Below selection radio button); then click **OK**.

To delete a row from the table from **DesignView** –

* Click mouse to set focus on a cell in the row.
* Use the pull-down menu (menu bar at the top) click **Table > Delete > Delete Rows**.

To obtain the effect of **colspan=*number*** from **DesignView** –

* Use the mouse to select 2 or more adjacent cells in the row.
* Use the pull-down menu (menu bar at the top) click **Table > Modify > Merge Cells**.

# GridView Control

Solution: *IntroASP\_NETnorthwindDrillDown*

Drag the **GridView** icon from the **Toolbox** (in the **Data** section) to the page in **Design** view (tab at the bottom of .aspx document). A recommended 1st action is (1) hover the mouse on the **Gridview** control on the form; (2) click the right-arrow at the upper-left corner; the response is an adjacent dialog titled **GridView Tasks**; (3) click **Auto Format…** ; the response is a floating dialog titled **AutoFormat**; (4) select a scheme from the right panel; a scheme with alternating backgrounds is very pretty; (5) click **OK**.

# Validation Controls

Solution: *IntroASP\_NETusingWebControls*

<asp:RequiredFieldValidator>

<asp:RangeValidator >

<asp: CompareValidator>

<asp: RegularExpressionValidator>

<asp: CustomValidator>

<asp: ValidationSummary>

# JQuery Error in Response to Using a Validation Control

Solution: *IntroASP\_NETusingWebControls*

When I followed the instructors example in section [ASP.NET Web Form Features].[Validating User Input], the code compiled successfully. But when I tried to run the code, I received a run-time error containing

## *WebForms UnobtrusiveValidationMode requires a ScriptResourceMapping for 'jquery'. Please add a ScriptResourceMapping named jquery(case-sensitive).*

This error will probably not occur when I use out-of-the-box facilities to build projects and solutions. (Right now I am testing with a simple web app – see Starting a Simple Web App in Visual Studio – above.) Research via Google yielded a workaround. (1) Open Web.Config for editing. (2) Add

<appSettings>

<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />

</appSettings>

immediately after <configuration>. (3) Save and close Web.Config.

# Validation Summary

Solution: *IntroASP\_NETusingWebControls*

* Place each validation control (Required, Range, etc.) at some position close to the input control that it is validating.
* In the **Properties** window for the validation control, set the properties
  + **ControlToValidate** (or **ControlToCompare**)
  + **ErrorMessage**
  + **Text** (usually an asterisk)
  + (possibly other properties depending upon the type of validator)
  + (optional) you may want to select a different **ForeColor** property to make the asterisk stand out
* Place the **ValidationSummary** controlin a place where it would be easily noticed.

# Performing Validation on the Server Side

Solution: *IntroASP\_NETusingWebControls*

“Server Side” refers to executable code in the server (i.e. code behind). Validation on the Client side is done by Java Script that performs the validation as prescribed by the property settings in the validation controls. This is not normally turned off; [validator].**EnableClientScript** is usually set to **True**. But a hacker can rip out the java script, which makes the web site vulnerable to bad data that can be injected into such places as the database. To protect against this malicious practice, insert server-side validation code to prevent processing data from the input controls that are being validated.

Page.Validate();

performs the same validation as the client-side code. It sets the value of Page.IsValid to false if any of the validation tests fails. Moreover, Page.Validate() also displays the error message as if the client-side validation had been performed. (You can test this by setting **EnableClientScript** to false for one of the validators; and then enter erroneous input for the associated input control.)

# Prevent Very Long Text Strings

A name longer than 30 characters is probably not valid. One might consider it good practice to set the **MaxLength** property of a text box to 30 for a user that is entering a name.

# Easy to Use Setting Default Behavior

Solution: *IntroASP\_NETusingWebControls*

Many internet users will appreciate the ability to type the **Enter** key, instead of having to (1) hover over a particular command button, and (2) press the left-mouse-click button. This can be done easily by typing – into the <form> tag - ‘defaultbutton=’ followed by the ID of the command button - enclosed in quotation marks. (Intellisense does most of the typing for you.)

Many internet users will appreciate it if the focus (position of the cursor) is automatically at the 1st logical control when the page is 1st displayed. This can be done easily by typing – into the <form> tag - ‘defaultfocus=’ followed by the ID of the control to get the focus - enclosed in quotation marks. (Intellisense does most of the typing for you.)

Setting the focus can also be done programmatically (server-side) by using **Page.SetFocus(*identifier of control*)**.

# Insert a User Control into a Page

Solution: *IntroASP\_NETusingWebControls*

In **Design** view . . .

* Find the user control in the Solution Explorer.
* Use the mouse to drag the .ascx file into a cell (div, table cell, etc.). (The cell could also be a **PlaceHolder**, but this is more likely to be used for inserting the user control dynamically via instructions in the code-behind.)

Examine the markup in **Source** view. Notice that Visual Studio has inserted the directive

<%@ **Register src**=”*PathOfUserControlSourceCode*” **tagname**=”*SomeName*” **tagprefix**=”*String*” %>

and Visual Studio has inserted the directive

< *SomeName*: *String* **ID=”” runat=”**server” />

*SomeName* and *String* are names that Visual Studio has introduced arbitrarily. We can change them to whatever we want; but the changes must be in both of the directives cited above.

# Insert a User Control Dynamically

Solution: *IntroASP\_NETusingWebControls*

Refer to an example of doing this in solution **IntroASP\_NETusingWebControls**, page **WebForm1**.

# SQL Server Database Facility

To shadow the instructor in the section **Demo: Binding Data with the SqlDataSource**, I needed SQL Server on my computer.

As a remedy, I installed SQL Server 2014 Express Advanced (installation file SQLEXPRADV\_x64\_ENU.exe). I chose this version, because I am working under Windows 7, and my computer uses a 64-bit processor. (This was guesswork, and I hope I made the right choice.) The only critical decision that I had to make during installation referred to authentication; I chose Windows Authentication.

I continued by loading the Northwind database into my newly installed instance of SQL Server Express. I LATER FOUND OUT THAT THIS WAS AN UNNESSARY STEP FOR SHADOWING THE INSTRUCTOR. THE EXAMPLES SEEM TO BE WORKING WITH .MDF FILES INSTEAD OF WITH DATABASES ATTACHED TO SQL SERVER EXPRESS.

* Start up SQL Server Management Studio. The window titled Microsoft SQL Server Management Studio appeared.
* Right-click the **Databases** folder in the left panel. SQL Server Management Studio displays a pop-up menu.
* Click **Attach…** SQL Server Management Studio displays a pop-up Dialog titled **Attach Databases**.
* Click the **Add…** command button. SQL Server Management Studio displays file browser in a window titled **Locate Database Files – *name of my instance of SQL Server Express***.
* This browser does not behave as expected. For example it did not find **Northwind.mdf** in my desktop folder. In fact the only folder which seemed to work was **C:\Program Files\Microsoft SQL Server\MSSQL12.SQLEXPRESS\MSSQL\DATA**, and I copied **Northwind.mdf** to this folder, selected **Northwind.mdf**, and clicked the **OK** button. Even this caused a path-related error message. I resolved it by removing the Log file where it displayed in the **Locate Database Files** dialog. Finally SQL Server Management Studio imported the **Northwind.mdf** database successfully.

THE FOLLOWING NOTES DESCRIBE HOW TO USE SQL Server Management Studio.

Pluralsight supplies databases via .mdf files in the Exercise Material. Moreover, some of the later sections of this Pluralsight course entail creating and examining databases in .mdf files. Use the following steps to view or modify .mdf file in SQL Server Management Studio.

* Double-click the icon to start Microsoft SQL Server Management Studio. Management Studio displays a **Connect to Server** dialog.
* If “(Localdb)\MsSqlLocalDb” does not already show up in the 2nd combo box (titled **Server name:**), one is permitted to type into the combo box; type “(Localdb)\MsSqlLocalDb”. Then click the **Connect** button. Management Studio displays a tree-view in the left pane.
* Click the “+” to the left of **Databases**. Management Studio displays the LocalDb databases (.mdf files) on the computer.
* You can drill-down any of these to examine or modify the desired database.

You can also manipulate .mdf files from a command-line prompt.

* Open a command-line prompt from the Windows Start Icon.
* Type “sqlcmd –S (Localdb)\MsSqlLocalDb”. This invokes SQL Server, and it invites Transaction SQL statements by displaying the “1>” prompt.
* When you are finished with SQL Server, you can dismiss it via the “quit” command

IF YOU WANT TO GET RID OF A .mdf-file DATABASE, DO NOT DELETE THE .mdf FILE. INSTEAD, USE EITHER SQL Server Management Studio OR THE SQL-SERVER COMMAND-LINE FACILITY TO PERFORM THE FOLLOWING TRANSACT-SQL STATEMENT.

DROP DATABASE *.mdf file name*

If *.mdf file name* contains any punctuation, and if it is not already enclosed in square brackets (i.e. […]), enclose the database name in square brackets. If the .mdf file is already deleted, you can use the same command to delete the memory of the database from sql-server express, and ignore the can’t-find-physical-file error message.

# Incompatible .mdf Files That Were Downloaded from Pluralsight Exercise Material

Pluralsight supplies database files (.mdf) via the Exercise Material associated with the course. In more than one context – while working with the .mdf’s – I have seen the following error message.

This database file is not compatible with the current instance of SQL Server.

To resolve this issue, you must upgrade the database file by creating a new data connection, or you must modify the existing connection this database file.

For more information, see https://go.microsoft.com/fwlink/?LinkID=235986

The resolution that I found was in a U-tube lesson: (<https://www.youtube.com/watch?v=TIeyu4_cbJ8>):

* Attempt to open ***Whatever*.mdf**  - under **App\_Data** in **Solution Explorer**. Visual Studio fails, but brings up the **Server Explorer** and sets the focus to ***Whatever*.mdf**. Notice the red X on the adjacent icon
* Right-click ***Whatever*.mdf** in the **Server Explorer**. Visual Studio displays a pop-up menu.
* Click **Modify Connection…** Visual Studio displays a pop-up dialog titled **Modify Connection**.
* Click the **OK** command button. Visual Studio displays a warning message

The database file that you are attempting to connect to is not compatible with the current instance of SQL Server.

. . .

Do you want to upgrade the database file now?

Click the **Yes** command button. Visual studio modifies ***Whatever*.mdf** and displays an expansion of ***Whatever*.mdf** in **Server Explorer**. (The Red X disappears when you expand the **Tables** node.)

# Error Message: connection to SQL Server

I have seen the following error message in Visual Studio and when starting SQL Server Management Studio:

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 26 – Error Locating Server/Instance Specified) (Microsoft SQL Server)

It might be a consequence of the fact that SQL Server has been stopped as a service. Try logging in to SQL Service Management Studio. If you see the same error message, the following might resolve the issue.

* Click the Windows **Start** button (lower-right corner of screen).
* Click **All Programs**, and then navigate to **SQL Server *yyyy* Configuration Manager** (possibly via the **Microsoft SQL Server *yyyy* > Configuration Tools**. Windows displays a pop-up dialog titled **SQL Server Configuration Manager**.
* In the left panel make certain that **SQL Server Services** is selected.
* Check whether **SQL Server (SQLEXPRESS)** is stopped. If so, right-click **SQL Server (SQLEXPRESS)**, and then click **start** in the pop-up list. Windows restarts **SQL Server (SQLEXPRESS)**.

# SQL Data Source - Data Binding Expressions

‘<%#’ indicates a data-binding expression.

<%# Eval(…) %>

gets data from somewhere and inserts into control; argument can be an expression or a string literal – the name of a database field

<asp:GridView id=”gvCustomers” runat=”server”>

. . .

</asp:TemplateColumn>

<ItemTemplate>

<%# Eval(“ContactName”) %>

</ItemTemplate>

</asp:TemplateColumn>

. . .

</asp:GridView>

<%# Bind(…) %>

similar to Eval(), but it can also write data from the control to a database

<%# XPath(…) %>

used to connect data in an XML tree (in server memory) to a control

# Introduction to SQLDataSource Control

Solution: *IntroASP\_NETnorthwindDrillDown*

The section **Demo: Binding Data with the SqlDataSource** illustrates use of <asp:DropDownList . . . > containing

DataSourceID=”countryDS”

and

<asp:SqlDataSource ID=”countryDS” runat=”server”

ConnectionString=”<%$ ConnectionStrings:countryCS %>”

SelectCommand=”*database SELECT query*”

</asp:SqlDataSource >

where

the connection string “countryCS” is specified in Web.config. **SelectCommand** (in <asp:SqlDataSource> above) does not have to contain explicit query text; instead it might be

SelectCommand=”CountryStoredProc”

SelectCommandType=”StoredProcedure”

The Pluralsight instructor illustrates how the markup and Web.config are inserted automatically via a wizard. He demonstrates opening a connection with the Northwind database by selecting its database file **Northwind.mdf**. (**Northwind.mdf** is provided as part of the exercise material for this section - **3: Working with the Data Source Controls**). But while shadowing the instructor - after browsing for **Northwind.mdf** and clicking the button **Test Connection** – I received database-not-compatible error message from visual studio; refer (above) to the section of this document titled “Incompatible .mdf Files That Were Downloaded from Pluralsight Exercise Material”.

# GridView SelectedValue

Solution: *IntroASP\_NETnorthwindDrillDown*

The section **Demo: Binding Data with the SqlDataSource** illustrates use of <asp:GridView> and <asp:DetailsView>.

The **SELECT** statement for <asp:GridView> refers to the **SelectedValue** property of <asp:DropDownList>, and we understand that this is obtained from the clause DataValueField="Country" in <asp:DropDownList>

The **SELECT** statement for <asp:DetailsView> refers to the **SelectedValue** property of <asp:GridView>, but the instructor did not tell us how the **SelectedValue** property of <asp:GridView> is expressed in the markup.

From some Google research I found out that GridView’s **SelectedValue** property is expressed via the **DataKeyNames** clause; in our demo example this is DataKeyNames="CustomerID". The name in this clause refers to a field name in the data source that is populating the grid. What this means is

For the selected row in the grid, the **SelectedValue** equals the value in the data source’s field – identified by name - for this row. It is not even necessary for that field to be displayed in the grid.

Generalizing - DataKeyNames=*list of field names separated by commas* , which results in the **DataKeyNames** property being a list of strings. I discovered by experimenting that the **SelectedValue** uses the 1st of these field names.

How did the wizard decide to generate “CustomerID” for the **DataKeyNames** property? I don’t know. Possibly it chose based on the primary key. In my opinion the safe way to use the GridView’s **SelectedValue** property is to examine the markup.

# Setting Up ObjectDataSource

I don’t have a web project for this demo.

These are the steps described in [Working with DataSourceControls].[Demo: Binding Data with the Object Data Source].

* If Visual Studio does not yet contain the template **Linq to SQL Classes**, you will need to install it. Refer to instructions in <https://www.completecsharptutorial.com/mvc-articles/add-missing-linq-sql-class-vs-2017-2019.php>.
* The following should be components in your Visual Studio solution:
  + An **App\_Data** folder. Right-click the project name in the solution folder, and select **Add > [Add ASP.NET Folder] > App\_Data**.
  + Copy the database .mdf file and its log file (e.g. Northwind.mdf and Northwind\_log.mdf) into the **App\_Data** folder.
  + An empty folder for SQL Server data; its name is probably immaterial; I used “Data”.
* Right-click the data folder (e.g. “Data”). Visual Studio displays a pop-up menu.
* Click **Add > New Item…** Visual Studio displays a pop-up control titled **Add New Item – Solution Items**.
* Click **Installed** in the left pane. Type “linq” in the search box in the upper-right corner. Visual Studio displays **LINQ to SQL Classes** in the center pane.
* Click **LINQ to SQL Classes**, type an appropriate name (ending in “.dbml) into the **Name** textbox (at bottom of control), and then click the **Add** command button. Visual Studio dismisses the dialog and displays a new surface, whose tab-name is the same as the name supplied at the previous step.
* Open the .mdf file, navigate the database in the Server Explorer, and drag one or more tables to the .dbml surface. Finally close the .dbml surface, and click the **Save** command button in response to the question “Save changes to the following items?”.
* Note that Visual Studio has populated the data folder with a .dbml file, a .dbml.layout file, and a .designer.cs file.
* Open the .designer.cs file. Note the **namespace** statement at or near row 12. The name that follows **namespace** needs to be copied into a **using** statement that refers to classes in the .dbml. Note, also, the **class** declaration at or near row 26. This class’s name will also be needed for C# code.
* Open the .dbml file (using an XML text editor). Note the connection string (at or close to the 2nd line). The value will be needed in your web.config file.

# Starting the EntityDataSource Demo

Solution: *IntroASP\_NET\_entityDataSource*

The **EntityDataSource** is explained (in detail) in [Working with DataSourceControls].[Demo: Binding Data with the Entity Data Source]. This section uses AdventureWorksLT\_Data.mdf, and while attempting to look at its contents, I received database-not-compatible error message from visual studio; refer (above) to the section of this document titled “Incompatible .mdf Files That Were Downloaded from Pluralsight Exercise Material”, where I describe how to resolve the error.

The steps to set up this exercise are as follows

* Create a new project. Add an **App\_Data** folder (right-click the project name in the solution folder; select **Add > [Add ASP.NET Folder] > App\_Data**). Add AdventureWorksLT\_Data.mdf and its log file into **App\_Data**.
* Right-click the project name. Navigating from the pop-up menu **Add > New Item…** , Visual Studio displays a pop-up dialog titled **Add New Item – *project name***.
* Click **Data** in the left pane. Click **ADO.NET Entity Data Model** in the center pane. Type an appropriate name into the **Name** textbox at the bottom; use the extension .edmx (metadata file for Entity Data Model). Click the **Add** command button. Visual Studio displays a new pop-up dialog titled **Entity Data Model Wizard**.
* Click the icon **EF Designer from Database**. Click the **Next** command button.
* Make certain that the check box **Save connection settings in Web.Config as:** is checked. Click the **Next** command button. Visual Studio displays the next step of the wizard (Choose Your Version).
* I selected Entity Framework 5.0, because with version 6 we will encounter an incompatibility when we begin using the Entity Data Source. Click the **Next** command button.
* Click the arrow next to **Tables** to choose entities. In this example we want the **Customer** table, and I had to expand the **SalesLT** node to gain access to various tables; then I clicked the checkbox next to **Customer**. Accept the defaults at the bottom of the dialog. Click the **Finish** command button. When the wizard finishes, it has created the .edmx, and it displays it with a picture of the **Customer** table. Click OK when the wizard asks for permission to make modifications to the computer.
* At this point we have the option of removing one or more fields from the **Customer** table. The instructor has bypassed this option.
* Build the Solution.
* Add a web form titled default.aspx.

What happens now is similar to what was done with other methods; the controls (DropDownList, GridView, etc.) are connected to the database via the Entity-data object. An example of the results are in IntroASP\_NET\_entityDataSource->Default2.aspx.

# Entity Framework Overview

The Entity Framework facility provided via Visual Studio is one example of ORM, which stands for Object Relational Mapping. At the time this Pluralsight class was written, Entity Framework’s version number was 4.1. From my experience, which I described in the section “Starting the EntityDataSource Demo”, I was given the choice between Entity Framework 5.0 and Entity Framework 6.0; therefore, I suspect that when I download Entity Framework the version number will be higher than 4.1.

There are at least 3 flavors.

1. Schema First – the C# classes are derived from the schema of an existing database.
2. Model First – create an Entity Framework Designer diagram, from which the database schemas are derived.
3. Code First – create ordinary C# classes, from which the database schemas are derived.

The Plurasight course will focus on the Code First flavor.

# Entity Framework – Code First

Solution: *IntroASP\_NET\_EntityFr\_TimeTracker*

**AFTER FOLLOWING THE INSTRUCTIONS FROM PLUALSIGHT – SELECTING AN ORDINARY ASP.NET WEB APPLICATION – I TRIED A 2ND TIME WITH AN MVC WEB APPLICATION (AS SUGGESTED BY** <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application> ) **FROM BOTH ATTEMPTS I ACHIEVED LIMITED SUCCESS; THE LocalDB FILES WERE CREATED, BUT I WAS UNABLE TO VIEW THEM WITH VISUAL STUDIO’S SERVER EXPLORER.**

**THE SOLUTION IntroASP\_NET\_EntityFr\_TimeTracker.sln USES THE ORDINARY-ASP.NET MODEL. THE LocalDB THAT IT PRODUCES IS Time\_Tracker.mdf.**

**THE SOLUTION IntroASP\_NET\_EntityFrTimeTracker.sln USES THE ORDINARY-ASP.NET-WITH-MVC MODEL. THE LocalDB THAT IT PRODUCES IS Tim\_Tracker.mdf.**

We need to use NuGet to Install EntityFramework. My installation of Visual Studio already had Package Manager available. I installed EntityFramework with the following steps.

* Starting from **Tools** on the menu bar – click **Nuget Package Manager > Package Manager Console**. Visual Studio opened a window titled **Package Manager Console** – with PM> prompt for each command.
* At the PM> prompt type “install-package EntityFramework” followed by the **Enter** key. Visual Studio downloads and installs the package Entity Framework 6.4.0. The web page <https://docs.microsoft.com/en-us/ef/ef6/fundamentals/install>reassures that Entity Framework 6.4 is backward compatible with Entity Framework 4.1.

CodeFirst relies on something called a DbContext to transform C# classes into database table schema. After classes are defined (**Employee** and **TimeCard** in our example), these class names can be used in a generic class. We already have the example **List<TimeCard>** as one of the fields in **Employee**. Analogously **DbSet** is also a Generic Class that takes a <T> parameter; this is done for **Employee** and **TimeCard** in the definition of the **TimeTracker\_DB\_Context** class.

After (1) a connection string was added to Web.config, (2) the **TimeTrackerRepository** class was written,(3) the default.aspx form was built, and (4) the **GridView** control was wired to an <ASP:ObjectDataSource>, which (in turn) was wired to a public function (GetEmployees()) in **TimeTrackerRepository**, the Pluralsight instructor tried running the code to find errors (if any). He did not find any errors, but I found three in my shadowing code.

1. A run-time error occurred pertaining to my version of Web.config file. Evidently the <connectionStrings> declaration must NOT precede the <configSections> declaration. I resolved the problem by moving <connectionStrings> to a position immediately after </configSections>.
2. A run-time error occurred, which complained about access to the database file in the **<App\_Data>** folder. I guessed (correctly) that my web application was trying to write an empty database file into the **<App\_Data>** folder. Since my project does not contain the **<App\_Data>** folder (I don’t believe that the instructor mentioned this as a requirement), I resolved the problem by adding the **<App\_Data>** folder to my project.
3. A run-time error occurred, which appeared to be related to the connection string in Web.config. I found a resolution at [Connection strings and models - EF6 | Microsoft Docs](https://docs.microsoft.com/en-us/ef/ef6/fundamentals/configuring/connection-strings); I understood this article to say that Entity Framework Code First does not need a connection string in Web.config. Instead it will connect to LocalDB (a .mdf file in <App\_Data> using a name supplied in the constructor of the DB Context class. Therefore, the workaround was (1) to comment-out the <ConnectionStrings> portion of Web.config, and (2) to add a constructor to TimeTracker\_DB\_Context class. That constructor supplies the name (for example “TimeTracker”) in the argument that it passes to the base class (DbContext); refer to the code in TimeTracker\_DB\_Context.cs.

After these work-arounds the code ran without a run-time error, and I noted that my web application had written TimeTracker.mdf file and its log file into **<App\_Data>.**

When I tried to use Visual Studio’s Server Explorer to examine TimeTracker.mdf, I was greeted by another run-time error.

The database C:\users\sncole . . . \app\_data\TimeTracker.mdf cannot be opened because it is version 852. This server supports version 782 and earlier. A downgrade path is not supported.

A work-around for this error is to use SQL Server Management Studio to examine TimeTracker.mdf. Refer to the section of this document titled SQL Server Database Facility.

I was not able to perform the last steps of the Entity Framework demo – where the instructor used Visual Studio and the DB Context to produce a dialog that invites the user to enter new employees into the Employees table. Instead, what I did was

1. Examine the structure of the automatically generated TimeTracker tables using Management Studio, which I described 2 paragraphs above.
2. Insert 2 rows – John Doe and Jane Doe – into the Employees table by using Management Studio and 2 INSERT queries.
3. Apply “View in Browser” to Default.aspx to invoke the GridView. The two Employees rows were shown.

Compare (1) the schemas of the two tables (seen via SQL Server Management Studio) and (2) the Entity Framework source code of the two tables (class Employee in Employee.cs and class TimeCard in TimeCard.cs). Observe that Employee\_ID is a field in the schema of table TimeCard, but not expressed in class TimeCard; also observe that List<TimeCard> is a member of class TimeCard, but not in the schema of table Employee. Entity Framework has (correctly) built the foreign key Employee\_ID, which is needed for a one-to-many relationship between table Employee and table TimeCard. Evidently Entity Framework inferred this relationship from our use of List<>.

In summary the wiring in this demo is as follows.

* The GridView refers to the data in <asp:GridView . . . DataSourceID=”TimeTrackerODS” . . . >
* ”TimeTrackerODS” is the ID of an <asp:ObjectDataSource> element. This element refers to a class and a member function, both of which are declared in TimeTrackerRepository.cs. The class is “TimeTrackerRepository”, and the member function is GetEmployees(); both of these are specified in <asp:ObjectDataSource>.

Therefore, when the GridView is invoked for display, this demo creates an element of type **TimeTrackerRepository**, and it calls its member function **GetEmployees()**.

* The class “TimeTrackerRepository” contains a member \_Context of type TimeTracker\_DB\_Context, and which is created by invoking the constructor TimeTracker\_DB\_Context().

Therefore, the constructor **TimeTracker\_DB\_Context()** is invoked when **TimeTrackerRepository** is invoked.

* The class **TimeTracker\_DB\_Context** is declared in TimeTracker\_DB\_Context.cs, and it is derived from the class DbContext. Evidently classes derived from DbContext will synchronize database tables with any classes cited in **DbSet<>** members. **TimeTracker\_DB\_Context** has two **DbSet<>** members – the 1st is a collection of elements of type **Employee**, and the 2nd is a collection of elements of type **TimeCard**. If the database doesn’t exist, Entity Framework will create the database and its tables. The database name might be specified in a connection string; but the connection string is optional, and if it is missing EntityFramework uses a string passed to the base class, DbContext; in this example the string is “TimeTracker”.
* The classes **Employee** and **TimeCard** are defined in Employee.cs and TmeCard.cs, respectively.

# Entity Framework – Code First – When Classes Change

Solution: *IntroASP\_NET\_EntityFrTimeTracker*

At the beginning of the previous section – in capital letters – I said that there are two copies of the code-first sample web application – one is ordinary and the other with MVC added. For the enhancements in this section I am using the latter (IntroASP\_NET\_EntityFrTimeTracker.sln).

The instructor had to add Global.Asax.cs. This was not necessary for my code, because Global.asax.cs was already part of my solution, and it (conveniently) contained the member function **Application\_Start()**; I merely added the Database.SetInitializer() statement at the start of this function.

The statement

emps.ForEach(e => context.Employees.Add(e));

deserves explanation. The first part - emps.ForEach(e => is an example of C# **Lambda** expressions. I’m not very familiar with Lambda expressions, but I get the idea of what it is trying to express, i.e. an abbreviated way of saying

foreach (Employee e in emps)

The 2nd part - context.Employees.Add(e) becomes clear if you remember that **Employees** is a member of **TimeTrackerDBContext** of type **DbSet<Employee>**; (see TmeTrackerDBContext.cs). **DbSet<>** refers to a particular flavor of collection; **DbSet<>** provides an **Add()** method, that can be used to add a **List<>** to the collection. If you don’t remember what **Employees** is, . . .

* Use Visual Studio’s **View** drop-down menu to activate the **Object Browser**.
* Use **Object Browser**’s **Search** combo box (immediately below **Browse:**) to find “Employees”.
* etc.

To repeatedly test the code, alternately comment-out and then remove the comment from the statement (in Employee.cs)

public int CompanyID { get; set; }

You may encounter the run-time error

## *Cannot drop database "TimTracker" because it is currently in use.*

This can occur when you are in the process of examining a table in TimTracker.mdf (with Sql Server Management Studio). Close Management Studio to remedy this error.

# Master Pages

Solution: *IntroASP\_NET\_EntityFrTimeTrackerMP*

The Demo in this section builds on the Demo built for Entity Framework. Therefore, I copied files and code from the Web Application whose solution is IntroASP\_NET\_EntityFrTimeTracker.sln (ordinary Web Application with MVC). (Refer to MakeAnAlmostExactCopyOfSolution.docx.) In particular I needed to install the Entity Framework NuGet package, and I am using TiTracker.mdf for the LocalDb.

To add a Master Page, right-click the project, select **Add > New Item…** , click **Web Forms Master Page**, choose an appropriate name in the **name** text box (at the bottom of the dialog), and click the **Add** command button.

The Pluralsight instructor forgot to tell you, but he added

<div style=”clear:both;” />

immediately before “<br/ >” at the end of the header markup. This is appropriate because the CSS style “float:right;” is still in effect at this place in the markup, and without “clear:both;” it would affect positioning in the Content page that immediately follows.

To add a web form to the project that is embedded in the master page, select **Add > New Item…** , click **Web Form with Master Page**, choose an appropriate name in the **name** text box (at the bottom of the dialog), and click the **Add** command button; Visual Studio displays a 2nd control titled **Select a Master Page**. Select the master page, and click **OK**.

This section also included the use of a 2nd master page, one that omits all of the head-of-page and foot-of-page markup (as one way of providing the user a printable version of the web page). The user selects this option by clicking an anchor tag containing the text “Print me”. The technique used is as follows.

* When the user clicks “Print me” this invokes the anchor tag.
* The **href** property of the anchor tag is the same page (“TimesheetViewer.aspx”), which amounts to a post-back. But appended to “TimesheetViewer.aspx” is the query string “?print=true”.
* The query string is processed in the code-behind procedure **Page\_PreInit()**. (It would not work to process the query string in **Page\_Load()**, because the header and footer material in Default.Master would already have been loaded onto the page.
* **Page\_PreInit()** contains a test to find out whether the query string contains “Print”. If the answer is **true**, the master page is switched to Print.Master.

# AJAX Basics

AJAX is an abbreviation for Asynchronous Java Script and XML.

XML is an abbreviation for Extensible Markup Language.

JSON is an abbreviation for JavaScript Object Notation.

DOM is an abbreviation for Document Object Model.

REST is an abbreviation for Representational State Transfer.

Partial Page Updates is one of AJAX’s features.

To use AJAX, we need to drag a ScriptManager control to the web page.

<asp:ScriptManager id=”*ScriptManagerID*” runat=”server” />

Related controls are UpdatePanel, UpdateProgress, ScriptManagerProxy, and Timer.

In the code-behind one can refer to Boolean entities ***ScriptManagerID*.IsInAsyncPostBank** and **Page.IsPostBack**.

The Boolean property **EnablePartialRendering** is often useful. When set to **true**, AJAX is behaving normally – i.e. performing partial-page updates. When set to **false**, it behaves as if AJAX is not being used. For example,

<asp:ScriptManager id=”*ScriptManagerID*” runat=”server” EnablePartialRendering =”false” />

could be useful during debugging, because run-time error messages - during partial-page updates - are sometimes invisible. Another possibility is to avoid partial-page updates with a very-old browser.

Protected void Page\_Init (Object sender, EventArgs e)

{

if ( !SupportXMLHttp() )

{

*ScriptManagerID*.EnablePartialRendering = false;

}

}

# AJAX Script-Manager Demo

Solution: *Intro\_ASP\_NET\_AJAX\_EnabledSite*

Start with an ordinary web-application project. Add page Default.aspx.

1. Drag the **ScriptManager** control onto the page – inside the <div> which is immediately inside the <form>. Set its name to “sm”.
2. Add a Java script to the **Scripts** folder of the project. Name it Custom.js.
3. An easy way to view the properties page for the **ScriptManager** is (a) right-click the **ScriptManager** in design view; (b) click **Properties**.
4. Click **[…]** to the right of the **Scripts** property. Visual Studio presents a pop-up dialog titled **Script Reference Collection Editor**.
5. Click **Add** command button – lower-left part of dialog. Click **Path** – in the right-hand panel. Type “~/Scripts/Custom.js” to the right of **Path**. Click **OK**. Visual Studio adds markup inside <ASP: ScriptManager> (in Default.aspx).

The take-away from this demo is the employing of the java-script function pageLoad(), which induces java-script to be delayed until the page has been loaded.

# AJAX Script-Manager Demo.2

Intro\_ASP\_NET\_AJAX\_EnabledSite2

The example is a copy of the example of the previous section, except that <scripts> and <asp:ScriptReference> are no longer needed. We still need to retain <asp:ScriptManager>; we need exactly one of these when we are using AJAX facilities.

The example contains a label and a command button. When the user clicks the command button, the current time is written into the label. Without the enclosing <asp:UpdatePanel> and <ContentTemplate> a small flicker is visible in the icon of the browser’s tab label (top of screen). The flicker disappears when <asp:UpdatePanel> and <ContentTemplate> enclose the button and label.

One can get positive confirmation that the amount of data posted back from the server is minimized as a consequence of <asp:UpdatePanel> and <ContentTemplate>; this is by using a tool called Fiddler.

* Start Fiddler. Click the tab **Inspectors** at the top of the right panel.
* Run the Demo. Click the “Get Time” command button.
* Go back to Fiddler. You will see a substantial amount of activity listed in the left panel.
* Click the 1st row in the left panel labeled **Localhost: *some\_number***. Fiddler shows some information in the two right-hand panels.
* Click the **TextView** tab in the upper-right and upper-left panels. Fiddler shows a substantial amount of activity in the right-hand panel. This reflects information from the server to display the page.
* Click the 2nd row in the left panel labeled **Localhost: *some\_number***. Fiddler shows much less information in the lower-right-hand panel. This indicates that only a portion of the page has been updated.

It is instructive to repeat the steps above after changing Default.aspx by (1) commenting out the <asp:ScriptManager> declaration, and (2) commenting out the <asp:UpdatePanel> and <ContentTemplate> declarations; then compare what Fiddler shows now with what it showed when the Script Manager was activated. (Don’t forget to undo the comments.)

With regard to forcing an unhandled exception, the instructor suggested changing the property **EnablePartialRendering** of the **ScriptManager** control from **True** to **False**, when you get an unhandled exception, and you need more information about the nature of the error (e.g. where it occurred in the code). My experience with Visual Studio 2019 is somewhat different from what the instructor showed.

I do not see any indication of an error, when I

* right-click Default.aspx
* click **View in Browser** in the pop-up menu

Instead the application seems to stop executing the button-click event-handler function. But when I change **EnablePartialRendering** of the **ScriptManager** control from **True** to **False**, a System Error is displayed on the screen with details that are useful to debug the error.

If I run the application by clicking the **IIs Express(*BrowserName*)** icon in Visual Studio’s tool bar, the execution breaks with a very user-friendly diagnostic showing the code where the exception occurred. But this diagnostic tool would probably not be available when the execution is in **Release** mode on a remote server.

The final part of the demo assumes that it is impossible (or terribly inconvenient) to keep the command button inside the update panel. The remedy is to use a trigger feature of the update panel. The easy way to code the trigger is to bring up the **Properties** page of the update panel, and to click […] adjacent to the **Triggers** property. This brings up a dialog **UpdatePanelTrigger Collection Editor**. Use the **Add** command button to add a trigger, and specify **AsyncPostBackTrigger**. The remaining steps to specify the **Click** event of **SubmitButton** are self-explanatory. The partial-post-back behavior is then restored. This can be verified by noticing a lack of flicker in the icon of the browser’s tab label. It can also be verified (as before) by using Fiddler.

# Adding AJAX to Previously Built Page – Entity-framework Time Tracker with Master Page

Solution: *IntroASP\_NET\_EntityFrTimeTracker\_MP\_AJAX*

The demo in this section builds on the Demo built for Entity Framework and Master page. Therefore, I copied files and code from the Web Application whose solution is IntroASP\_NET\_EntityFrTimeTrackerMP.sln . (Refer to MakeAnAlmostExactCopyOfSolution.docx.) In particular I needed to install the Entity Framework NuGet package, I am using TiTracker.mdf for the LocalDb, and I included all of the features demonstrated earlier for Master Page.

In this demo the appropriate place for <asp:ScriptManager> is in the Master Page.

We start out by putting the markup of both grid-view controls into <asp:UpdatePanel> and <ContentTemplate>. As a consequence, it appears that AJAX has reduced the amount of data posted back to the page. But an examination using Fiddler indicates that both grids are being updated during each post-back; the 1st grid (containing the employees) is being updated unnecessarily, because nothing changes in this grid.

To prevent the application from uploading data to the employees grid . . .

* Change the scope of <asp:UpdatePanel> and <ContentTemplate> so that it encloses only the 2nd grid-view control.
* Go to the **Properties** page of <asp:UpdatePanel>.
* Click […] to the right of **Triggers** property. Visual Studio, in response, displays a pop-up dialog titled **UpdatePanelTrigger Collection Editor**.
* Add an **AsyncPostBack** trigger. For this trigger refer to GridView1, event **SelectedIndexChanged**.

When you run the application, you will notice no flicker during post-back. Moreover, you can prove that <asp:UpdatePanel> and its trigger are working by using Fiddler.

If the web application takes a noticeable amount of time to respond to a post-back request, the user may get impatient and click more buttons (which increases the load on the web application). As a remedy for the impatient user, it is good practice to use an <asp:UpdateProgress> control. Embedded in the <asp:UpdateProgress> declaration is a <ProgressTemplate> which introduces whatever you want to be displayed to denote “delay”. Many websites use an animated GIF for this purpose. To obtain an animated GIF . . .

* Go to the website <http://www.ajaxload.info/>.
* Select a GIF from the **Indicator type:** dropdown list.
* Click the **Generate it !** command button. The website displays it in the red panel. You may want to try out several such GIFs.
* When you have made your choice, right-click the moving image in the red panel, and click **save image as…** The web site opens a pop-up dialog allowing you to choose where to deposit the GIF.
* Navigate to a folder in the Web application; the navigator dialog allows you to add a new folder in case you need to do so. (It might be a good idea to create a new folder in the project with the name **Images**.) After selecting the folder to hold this GIF, click the **Save** command button.
* Use **Add > Existing Item…** to make the GIF (and possibly its containing folder) part of the solution.
* Drag the GIF from the **Images** folder to a place immediately after <ProgressTemplate>.

# Authentication and Authorization

I AM WRITING THIS SUMMARY AFTER HAVING REVIEWED THE “SECURITY” SECTION OF “INTRODUCTION TO ASP.NET 4 WEB FORMS”. THE TEXT THAT FOLLOWS WAS WRITTEN EARLIER - WHILE I WAS TRYING TO SHADOW THE INSTRUCTON. SHADOWING WAS FUTILE, BECAUSE VISUAL STUDIO HAS EVOLVED AFTER THE COURSE WAS RELEASED, AND IT IS ALMOST CERTAIN THAT MANY OF THE TECHNIQUES THAT PLURALSIGHT DESCRIBES IN “SECURITY” ARE OBSOLETE – AT LEAST WHEN USING VISUAL STUDIO 2019 (OR LATER). THE “ASP.NET Configuration” (WHICH PROVIDED A DATABASE OF AUTHENTICATED USERS AND THE WEB PAGES AT WHICH THEY WERE AUTHORIZED) APPEARS TO HAVE BEEN REPLACED WITH “ASP.NET CORE IDENTITY”; REFER TO <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio>

The first part of the section **Security** deals with authentication and authorization. The section starts by presenting two demos, one for Forms Authentication and the second for Windows Authentication.

In the Forms Authentication demo, the instructor starts by creating an ordinary web application with a project named “FormsAuthenticationDemo”. To shadow the instruction I similarly started a new web application: I named the solution Intro\_ASP\_NET\_FormsAuthentication”, and I named the project “FormsAuthentcation”. I tried running it immediately (out of the box), and I got a run-time error

CS0426: The type name 'SiteMaster' does not exist in the type 'FormsAuthentication'

Line 190: [TemplateContainer(typeof(FormsAuthentication.SiteMaster))]

After some digging with Google I found that someone earlier had stumbled on the same error, and his project name was also **FormsAuthentication**. Refer to [asp.net - Master page inherits tag question - Stack Overflow](https://stackoverflow.com/questions/2465460/master-page-inherits-tag-question). It seems that the name “FormsAuthentication” is a bad choice for the project’s name. I tried again – changing the name to “Forms\_Authentication”, and the problem went away.

The instructor showed how his web application presented a welcome screen with a hyperlink for logging in. My web application also presented a welcome screen, but there was no hyperlink for logging in.

In the Windows Authentication demo, the instructor started by creating an empty Web application with a project named “WindowsApplicationDemo”. I similarly created an empty project. After adding a web form Default.aspx, the instructor right-clicked the project name to invoke the dialog for the project’s properties. Then he clicked a checkbox named N**TLM Authentication**. I similarly added Default.aspx to my project, and invoked the dialog for the project’s properties. But this dialog did not include any indication of any kind of authentication.

The next two subtopics in this section of the course deals with the ASP.NET Configuration Website. (aspnet.regsql.exe provides a wizard that can be used to migrate the ASP.NET Configuration database from a LocalDB to a shared database server.) I could not shadow the instructor’s demo, because of differences between the instructor’s version of Visual Studio and the version that I am using (Visual Studio 2019).

* The instructor demonstrated obtaining access to “ASP.NET Configuration” via the **Project** pull-down menu. This option is not in the project that I created.
* Web.config in the instructor’s demo differed substantially from my project’s web.config. <connectionStrings>, <authentication>, <membership>, <profile>, and <role Manager> are missing from web.config.

Admittedly these may be available in newer templates – for example those containing the name “Core”.

I DID NOT PROCESS THE SUBSECTIONS “Demo: Login and Registration” NOR “Demo: Login Views”.

# ASP.NET Dynamic Data

Shortly after describing the features of ASP.NET Dynamic Data, the instructor goes into details with a demo. At the outset he creates a Visual Studio Web Site or a Visual Studio Web Site Web Application, which is specifically designated to include the Dynamic Data feature. The templates in Visual Studio 2019 do not include these templates. Therefore, I suspect that the Dynamic Data feature is now obsolete.

# Configuring IIS Locally

The section [Web Deployment].[Configuring IIS locally] demonstrates how to make sure (1) that IIS is installed on the development computer, and (2) that ASP.NET is properly integrated with IIS. Following the instructor’s directions I found that IIS was already installed. The 2nd part (making sure that ASP.NET is properly integrated with IIS) turned out to be a little more difficult. I found that starting a command prompt with administrative privileges for Windows 7 entails the following.

* Press the WINDOWS keyboard key. Windows 7 brings up the START menu with the cursor in the box with the prompt *Search for programs and files*
* Type “cmd”. Then press Ctrl+Shift+Enter. Windows should open a command-prompt window with the prompt **C:\Windows\system32>**.
* Type the instructions given by the Pluralsight instructor.

cd ..\microsoft.net\framework\v4.0.30319 [*enter key*]

.\aspnet\_regiis.exe –i [*enter key*]

# Deploying Web Application by Hand

After it has been verified that the web application runs under Visual Studio, it is good practice to deploy it to a local folder on the development computer - to verify (at least partly) that there will not be rude surprises when it is deployed to the outside world.

* The source for copying is the top folder of the web-application folder, where it is maintained for Visual Studio development. In my case this is C:\users\sncole\source\repos\*folder\_containing\_visual\_studio\_files\_of\_a\_web\_app*
* The destination is c:\inetpub\wwwroot – a folder that also contains (at a minimum) aspnet\_client, iisstart.htm, and welcome.png.
* Copy the source folder and all of its contents to the destination folder.
* Invoke the **Internet Information Services (IIS) Manager**. If you click the **Start Button**, you may find an icon for **IIS Manager** in the left-hand panel. If not, invoke **Control Panel > Administrative Tools**, and you will see **Internet Information Services (IIS) Manager** as a shortcut in the right-hand panel. In response **Internet Information Services (IIS) Manager** displays a dialog.
* In the left-hand panel of the dialog expand the root of the tree, expand the **Sites** node, and expand the **Default Web Site** node. In response **Internet Information Services (IIS) Manager** displays existing deployed web sites and the folder that was copied (in an earlier step of this process) to the destination folder. It also changes the name of the dialog to **Default Web Site Home**.
* Right click **Default Web Site**. In response **Internet Information Services (IIS) Manager** displays a pop-up menu.
* Click **Add Application…** in the pop-up menu. In response **Internet Information Services (IIS) Manager** displays a pop-up dialog titled **Add Application**.
* In the **Alias:** box in the dialog, type a name you would want the browser to display when it invokes the 1st web page of the web application. To minimize confusion one usually types the name of the visual studio project folder. If you use a different name, **Internet Information Services (IIS) Manager** will display both the alias name and folder name under the **Default Web Site** node. Otherwise, it will replace the folder’s icon with a web-site icon – with the same name.
* Click the […] icon to the right of the **Physical Path:** box in the dialog. **Internet Information Services (IIS)** Manager introduces a pop-up file browser. Navigate in the file browser via c:\inetpub\wwwroot to the folder at the head of the web-application project. If the visual-studio solution folder is the parent of the project folder, be careful to navigate down to the project folder. If the solution and project are in the same folder, this is not an issue. Click the **OK** command button in the file browser. **Internet Information Services (IIS) Manager** dismisses the file browser. Click the **OK** command button in the **Add Application. Internet Information Services (IIS) Manager** dismisses the dialog, and displays a new web-site icon under **Default Web Site** in the left panel of the **Default Web Site Home** dialog.
* Click the **Browse \*:81 (http)** link in the right-hand pane of the **Default Web Site Home** dialog. **Internet Information Services (IIS) Manager** responds by compiling the new web site. It is now available to respond the computer’s local web browser. Unless there is an error message, you can use the URL (e.g. “LocalHost:81/*web\_application\_name*” in a browser to test the deployed web application.

When deploying one particular web application – a variation of one of the demos described earlier in this document – I encountered the error message in response to the last step in the deployment.

## HTTP Error 403.14 - Forbidden

### The Web server is configured to not list the contents of this directory.

followed by

* A default document is not configured for the requested URL, and directory browsing is not enabled on the server.

I interpreted the error messages above to mean that the default web page for this web application had not been specified, especially since I intended the page **default0.aspx** to be the starting page. In visual studio I had right-clicked **default0.aspx** in the solution explorer, and clicked **Set As Start Page** in the pop-up menu, but evidently this setting does not apply to the deployed web application. Instead, I found (via Google) the following instructions to correct the error.

* Return to IIS Manager.
* Click the erroneous web application listed in the tree in the left-hand pane under **Default Web Site**.
* In the center pane – at the bottom of the page – make certain that the **Features View** tab is selected.
* Double-click the **Default Document** icon in the **IIS** group. IIS responds by displaying a list of web pages titled **Default Document**. (Default0.aspx was not in this list.)
* In the **Actions** pane in the right-hand panel, make certain that **Disable** is displayed. If **Enable** is displayed, click it.
* Click **Add…** in the **Actions** pane. IIS responds by displaying a pop-up dialog titled **Add Default Document**.
* Type the name (e.g. Default0.aspx) in the text box. Then click **OK**. IIS responds by displaying the name of the added page at the top of the **Default Document** list.
* Click the **Default Web Site** node in the left pane to dismiss **Default Document** in the center pane.
* Expand the **Default Web Site** node, click the name of the erroneous web application, and click **Browse** (again) in the right-hand pane. This should have corrected the error.

I suspected that – under the covers – IIS was making a change to web.config in response to the **Default Document** manipulations above. Comparing web.config in the Visual Studio development folder with web.config under c:\inetpub\wwwroot I found that the following had been added at the end of web.config:

<system.webServer>

<defaultDocument enabled="true">

<files>

<add value="Default0.aspx" />

</files>

</defaultDocument>

</system.webServer>

To avoid the process of correcting the missing-default problem after every deployment, I recommend that web.config be copied back to the Visual Studio development folder.

This **Deploying Web Application by Hand** section concludes with a description of how to respond to a data-base error by making a change to a property of the web application’s application pool. I did, indeed, encounter a database run-time error when I tried deploying and running the first version of an entity framework sample program; refer to the section [Entity Framework – Code First](#_Entity_Framework_–). The error was

# Server Error in '/EntityFr\_TimeTracker' Application.

## *A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: SQL Network Interfaces, error: 50 - Local Database Runtime error occurred. Cannot create an automatic instance. See the Windows Application event log for error details. )*

This is a different error from the one experienced by the Pluralsight instructor, and changing an application-pool setting did not resolve it. This error is possibly related to version number incompatibility between the Local Database (852) and the Entity Framework software used in the web application (782). Refer to a description of this incompatibility in section [Entity Framework – Code First](#_Entity_Framework_–).

# Automatic Deployment

The final file 3 Pluralsight sections of [Web Deployment] -- [Introducing WebDeploy], [Demo: Using WebDeploy], and [SQL Deploy and Web config transforms]-- illustrate by using features of Visual Studio that are no longer available as of 2019. I found the following two articles that seem to contain some of the same features, but I have not tried to use them.

<https://docs.microsoft.com/en-US/visualstudio/deployment/quickstart-deploy-to-a-web-site?view=vs-2019>

or

<https://docs.microsoft.com/en-us/visualstudio/deployment/tutorial-import-publish-settings-iis?view=vs-2019>