Extra Notes of ASP.NET

Partial Classes in C#

A partial class is a special feature of C#. It provides a special ability to implement the functionality of a single class into multiple files and all these files are combined into a single class file when the application is compiled. A partial class is created by using a *partial* keyword. This keyword is also useful to split the functionality of methods, interfaces, or structure into multiple files.

AutoEventWireup attribute in ASP.NET

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

The ASP.NET page framework supports an automatic way to associate page events and methods. If the AutoEventWireup attribute of the Page directive is set to true, the page framework calls page events automatically, specifically the Page_Init and Page_Load methods. In that case, no explicit Handles clause or delegate is needed.

xmlns attribute

Xmlns stands for extensible markup language namespace.
The xmlns attribute specifies the xml namespace for a document.
Syntax :
<html xmlns="http://www.w3.org/1999/xhtml">
Attribute Values :

Value	Description
http://www.w3.org/1999/xhtml	The namespace to use (for XHTML documents)

Object Sender, EventArgs e

It is a standard in .NET that event handlers have this signature, usually declared by some delegate types for example EventHandler.

Sender references the object that raised the particular event, for example with Button's Click it would let you reference the Button control via sender argument.

Object Sender is a parameter called Sender that contains a reference to the control/object that raised the event.

EventArgs e is a parameter called e that contains the event data. EventArgs represents the data related to that event, and can be used to pass parameters, state information whatsoever to the event handler for the handling code to decide what to do.

Example:

```
protected void button1_Click (object sender, EventArgs e)
{
  Button button1 = sender as Button;
  button1.Text = "Submit";
}
```

When Button is clicked, the button1_Click event handler will be fired. The "object sender" portion will be a reference to the button which was clicked

SqlCommandBuilder Class

The SqlCommandBuilder class automatically generates commands that are used to resolve changes made to a DataSet with the associated SQL Server database.

The SqlDataAdapter does not automatically generate the SQL statements required to resolve (settle) changes made to a DataSet with the associated instance of SQL Server.

However, you can create a SqlCommandBuilder object to automatically generate SQL statements for updates if you set the SelectCommand property of the SqlDataAdapter. Then, any additional SQL statements that you do not set are generated by the SqlCommandBuilder.

In other words,
The CommandBuilder class can automatically generate commands used by the DataAdapter to update changes made to a DataSet back to the data source.

The CommandBuilder class helps developers generate update, delete., and insert commands for a data adapter.

The CommandBuilder class is useful because it lets you update the data source with changes made to the DataSet using very little code.

Connection Strings for different RDBMS

1) MS SQL Server –
Data Source=<server name>;Initial Catalog=<database name>;Integrated Security=true

2) MS Access –
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=<path of database file>

3) Oracle –
Data Source=<name of oracle instance>;Integrated Security=yes/SSPI

(SSPI = Security Support Provider Interface. It is a component of Windows API that performs security-related operations such as authentication.)

4) ODBC Data Source –
Dsn=<dsn name>;uid=<username>;pwd=<password>

(DSN = Data Source Name)

Global.asax file

The global.asax file resides in the root directory of an ASP.NET web application and is called ASP.NET application file.
It contains the code that is executed when certain events, such as start of an application or error in an application, are raised by the ASP.NET web application.
The global.asax allows us to write event handlers that react to global events in web applications. Global.asax files are never called directly by the user, rather they are called automatically in response to application events.
Events and states, such as session and application state that are specified in the global.asax file are applied to all the resources of the web application.
The code in global.asax file contains methods with predefined names. They do not contain any HTML or ASP.NET tags. They are optional, but a web application has no more than one global.asax file.
The global.asax file is as given below :

```
<%@ Application Language="C#" %>
<%@ Import Namespace="mix" %>
<%@ Import Namespace="System.Web.Optimization" %>

<script runat="server">

    void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup
        BundleConfig.RegisterBundles(BundleTable.Bundles);
        AuthConfig.RegisterOpenAuth();

    }
```

```
    void Application_End(object sender, EventArgs e)
    {
        //  Code that runs on application shutdown

    }

    void Application_Error(object sender, EventArgs e)
    {
        // Code that runs when an unhandled error occurs

    }

</script>
```

ASP.NET Page Life Cycle

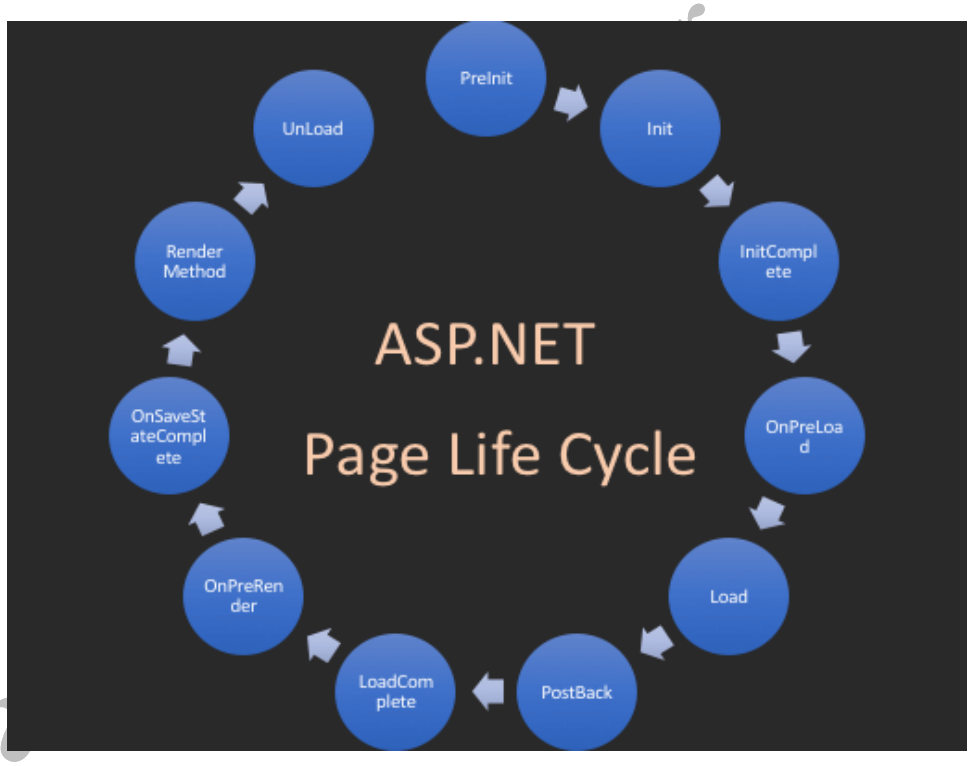
When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps.

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available. You can write your own code in it to override the default code.

The following are the various stages or events of ASP.Net page life cycle :

1) **PreInit** : PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
2) **Init** : Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.
3) **InitComplete** : InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
4) **OnPreLoad** : PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
5) **Load** : The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
6) **Control PostBack Events** : ASP.NET now calls any events on the page or its controls that caused the PostBack to occur. Checks the IsValid property of the Page and of individual validation controls before performing any processing.
7) **LoadComplete** : The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler.

8) **OnPreRender** : The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
9) **OnSaveStateComplete** : State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.
10) **Render Method** : This is a method of the page object and its controls and not an event. The Render method generates the client-side HTML, DHTML and script that are necessary to properly display a control at the browser.
11) **UnLoad** : The UnLoad stage is the last stage of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.



LINQ

LINQ stands for **Language Integrated Query**. LINQ is the uniform query syntax used to retrieve data from different sources and formats. LINQ is a structured query syntax built in C# and VB.NET to retrieve data from different types of data sources such as collections, generics, ADO.Net DataSet, XML Documents, web service and MS SQL Server and other databases.

LINQ comes with **LINQ Providers (or technologies)** for in-memory object collections, Microsoft SQL Server databases, ADO.NET datasets and XML documents. These different providers define the different flavors of LINQ. They are as follows –

1. LINQ to Objects
2. LINQ to XML
3. LINQ to SQL
4. LINQ to DataSet
5. LINQ to Entities

Advantages of LINQ

1. **Familiar language**: Developers don't have to learn a new query language for each type of data source or data format.
2. **Less coding**: It reduces the amount of code to be written as compared with a more traditional approach.
3. **Readable code**: LINQ makes the code more readable so other developers can easily understand and maintain it.
4. **Standardized way of querying multiple data sources**: The same LINQ syntax can be used to query multiple data sources.
5. **Compile time safety of queries**: It provides type checking of objects at compile time.
6. **IntelliSense Support**: LINQ provides IntelliSense for generic collections.
7. **Shaping data**: You can retrieve data in different shapes.
8. **Built-in Methods** : LINQ provides a lot of built-in methods that we can be used to perform the different operations such as filtering, ordering, grouping, etc. which makes our work easy.

Using LINQ in ASP.NET

Use **System.Linq** namespace to use LINQ. LINQ Queries uses extension methods for classes that implement IEnumerable or IQueryable interface. The **Enumerable** and **Queryable** are two static classes that contain extension methods to write LINQ Queries.

LINQ Query Syntax

var variablename = from <range variable> in <IEnumerable<T> or IQueryable<T> Collection>
<Standard Query Operators> <lambda expression>
<select or groupBy operator> <result formation>