



ASP.NET 2.0 Introduction



What is ASP.NET?

- Web application framework use to build web applications and web services.
- Part of Microsoft's .NET platform
- ASP.NET is built on the Common Language Runtime, allowing programmers to write ASP.NET code using any Microsoft .NET language like C#, VB.NET. J#
- Runs on IIS (Internet Information Services, Microsoft's Internet server)
- Successor to Microsoft's Active Server Pages (ASP) technology.



The Microsoft .NET Framework

- environment for building, deploying, and running Web applications and Web Services.
- Advantages
 - Easier and quicker programming
 - Reduced amount of code
 - Declarative programming model
 - Richer server control hierarchy with events
 - Larger class library
 - Better support for development tools



ASP.NET File

- File extension .aspx
- Can contain HTML, XML, and scripts
- Scripts in an ASP.NET file are executed on the server
- Supports
 - Visual Basic, C# , J# and C++
 - ADO.NET



Features

1. Controls

- ASP.NET controls
 - All Html controls and other sophisticated input controls
 - Event Aware Controls
 - All the asp objects on the web page is event aware
- ASP.NET Components
 - components are heavily based on XML. Example the old AD Rotator now uses XML to store advertisement information and configuration.



1. Security

- Authentication
- Authorization

2. Scalability

- Application can scale over more than one server. Enhanced sever to server inter-communication.

3. Improved Performance

- The first request for an ASP.NET page on the server will compile the ASP.NET code and keep a cached copy in memory.



Versions

Classic ASP or ASP

2002

Simple web based application – with request, response, session application-level logic , support for VB only

ASP.NET 1.0

Web UserControls , custom HTTP handlers, web service pages, web.config file, support for all .net languages, code behind files

2005

ASP.NET 2.0

Late 2006

ASP.NET 3.0

ASP.NET 3.5



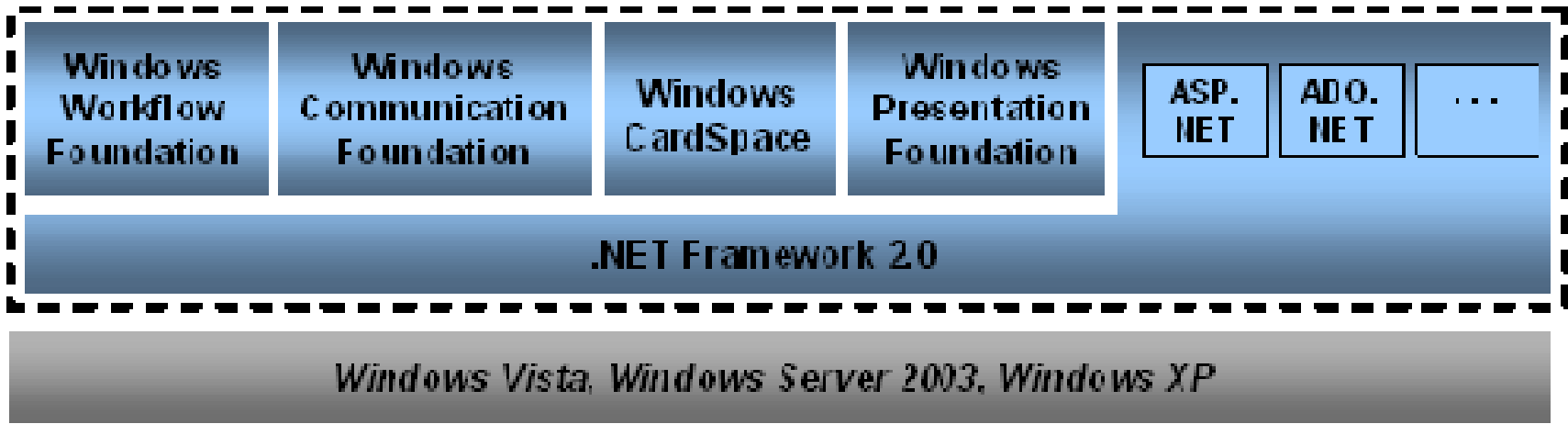
ASP.NET 2.0

- New controls
 - data controls (GridView, FormView, DetailsView)
 - Login controls
 - Navigation controls
 - Declarative data access controls(SqlDataSource, ObjectDataSource, XmlDataSource controls)
- Master pages
- Themes
- Skins
- Personalization services
- New localization technique
- Support for 64-bit processors
- Provider class model



ASP.NET 3.0

Applications





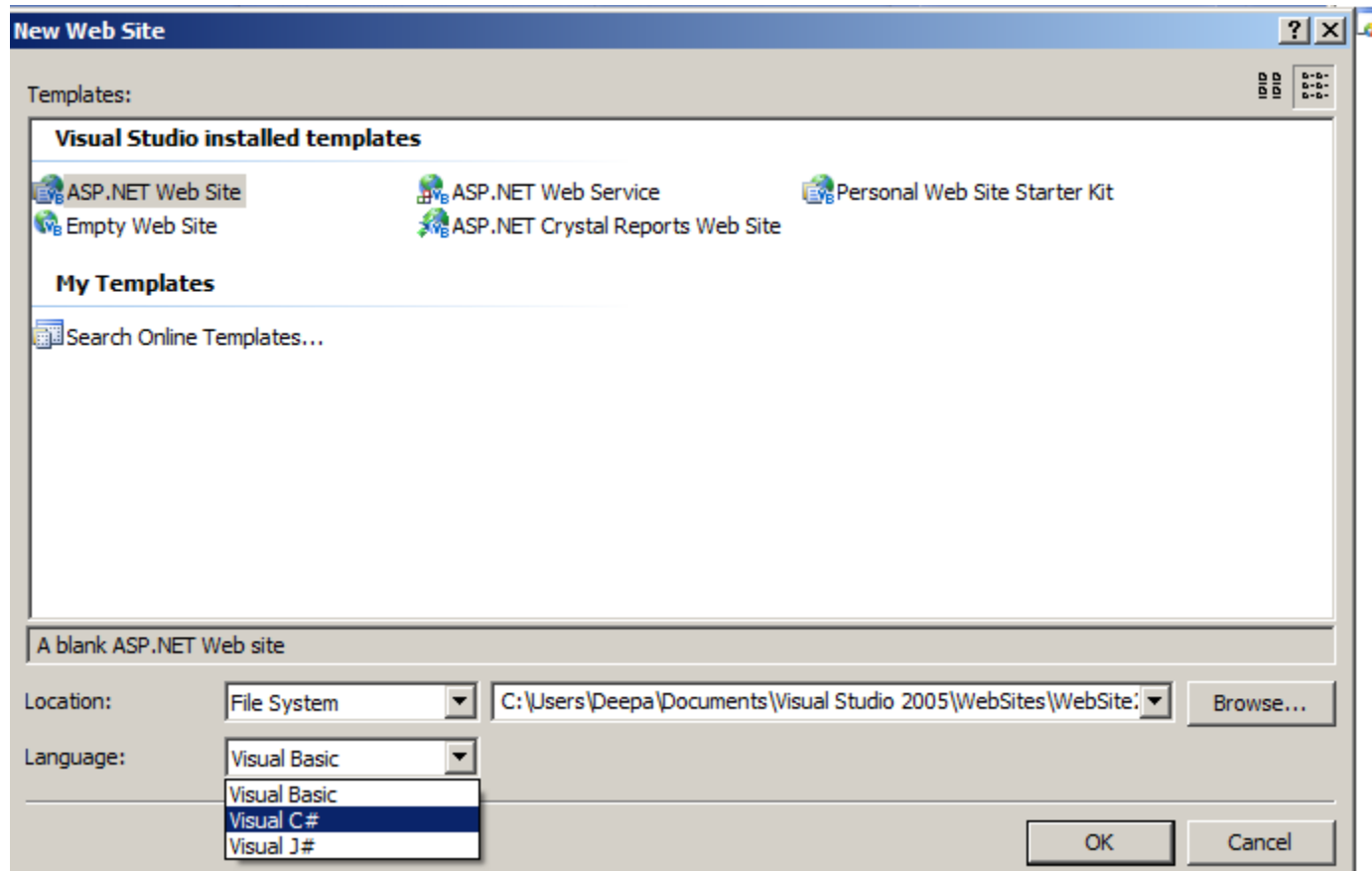
ASP.NET 3.5

- AJAX
- LINQ
- New data controls (ListView, DataPager)



A simple ASP.NET page

- Create a new web site.





What do you see?

- A new website with the name you specified (or the default name) gets created.
- A folder `App_Data` is created which will contain data files such as xml and data base files.
- **Default.aspx** is already created which is the first page in the web application.
- **Default.aspx.cs** is 'code-behind' class for **Default.aspx**.
- You have design as well as source view for all the asp pages.



Default.aspx

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

Page directive

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
```

```
</head>
```

```
<body>
```

ASP.NET Script

Remove the form tag and add this

```
<%Response.Write("hello"); %>
```

```
</body>
```

```
</html>
```

File → View in Browser : to test

Default.aspx.cs



```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class _Default : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e)
    {
    }}
}
```

- Code that is required when the page loads up can be put here
- Note that there is no link up between the asp.net page and the code here.
- This is automatically achieved → inferred by the **AutoEventWireUp** attribute set to true.



Getting the html form data

- Add the following in the `default.aspx` file

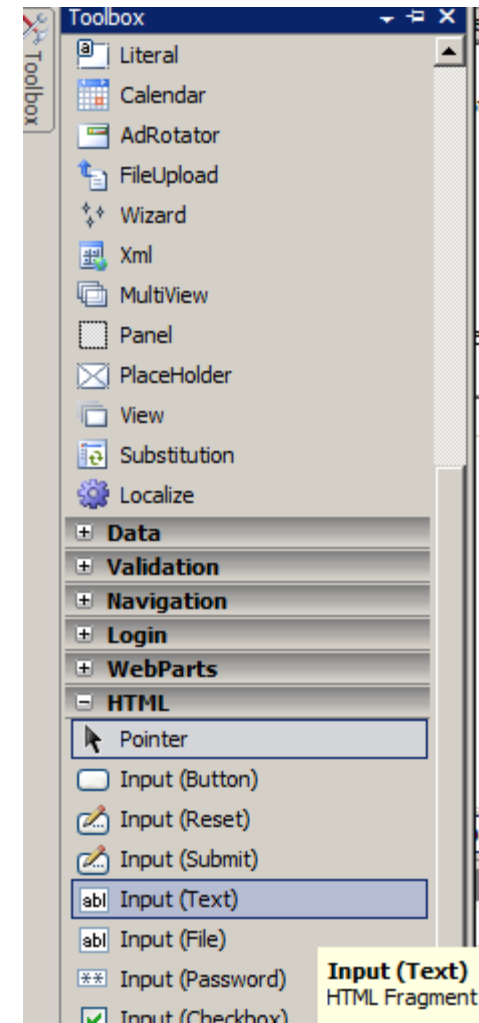
```
<form method="post"
  action="hello.aspx">
```

```
Your Name <input type="text"
  name="user" />
```

```
<input type="submit" />
```

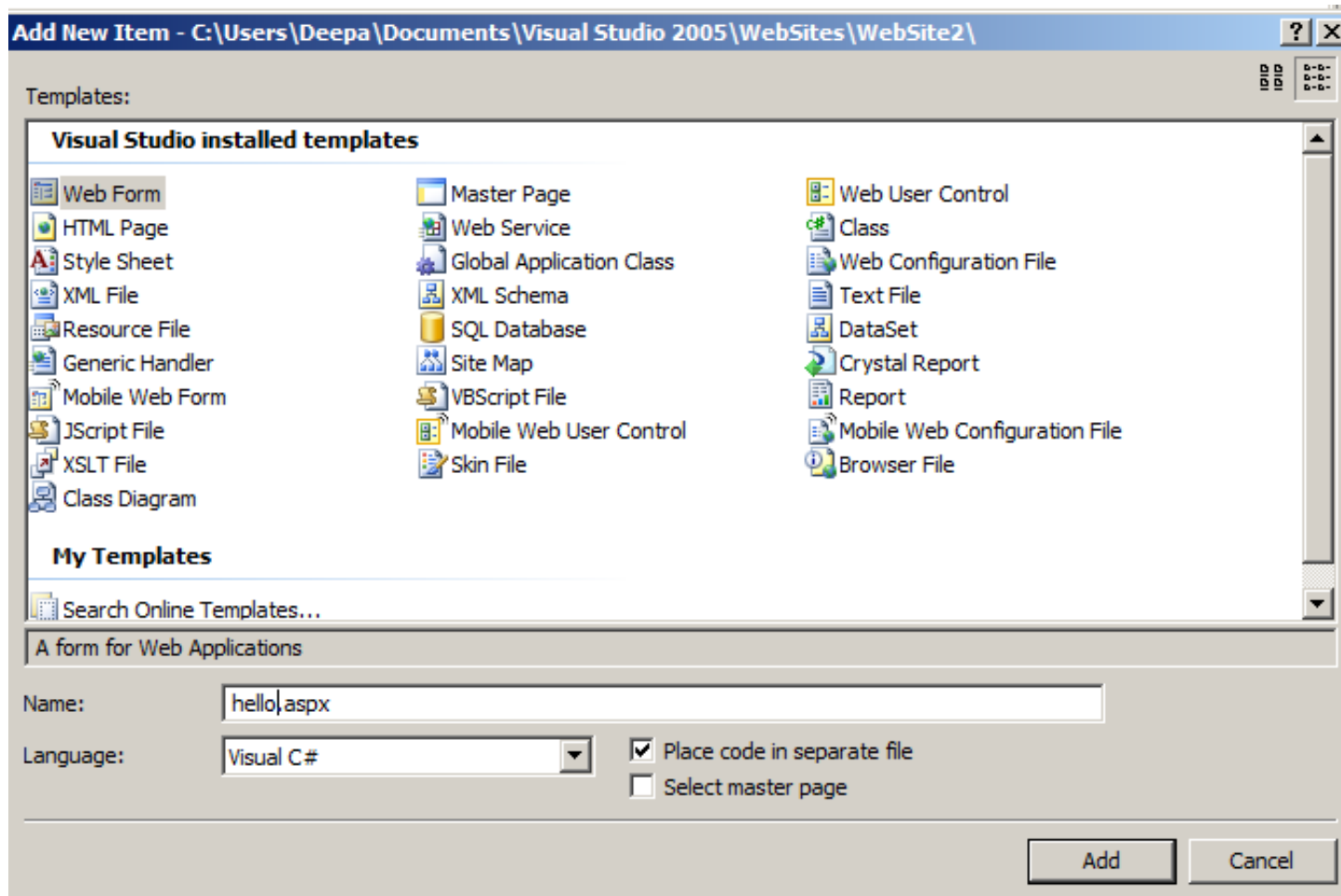
```
</form>
```

Or drag and drop from the toolbox in the design view.





- Create a new aspx file





Fetching the form data

hello.aspx

- ```
<%string name =
Request.Form["user"];
Response.Write("Hello "+ name);
%>
```
- The form parameters can be fetched using Request.Form Collection in ASP.NET ( as well as in ASP).



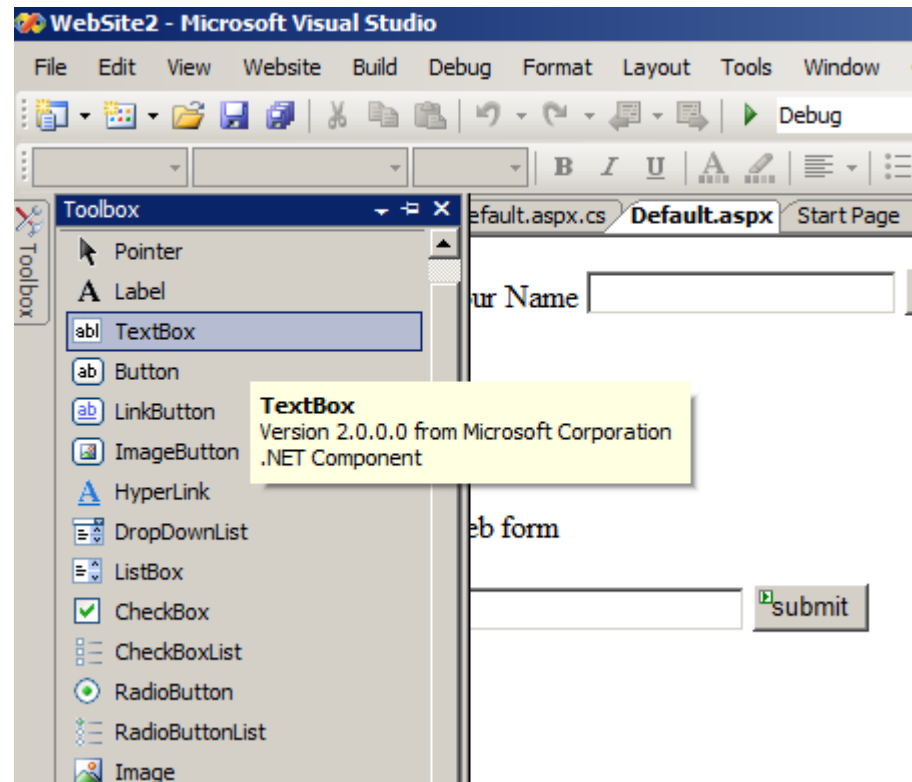
# Web Forms

- ASP.NET pages are officially called Web Forms
- Using web forms we can create a web application using the same control-based interface as a Windows application.
- Therefore instead of using html form tags, windows control can be used in the pages.
- The ASP.NET ISAPI extension reads the aspx file and generates the corresponding objects, and fires a series of events.



# Hello example using webform

- In the design view. Drag the textbox and the button into the webform like we did for the windows form.





- Add the form tag as shown below in the source view

*Add this*

```
<form runat="server">
```

```
<asp:TextBox ID="NameBox"
runat="server">
```

```
</asp:TextBox>
```

```
<asp:Button ID="Button1"
runat="server" OnClick="Button1_Click"
Text="submit" />
```

```
</form>
```

*Add this*

*Automatically generated on dragging controls*



- Double click on the button and add the event code

```
protected void
Button1_Click(object sender,
EventArgs e)
{
 string name = NameBox.Text;
 Response.Write("hello from
webform "+ name);
}
```

[Default.asp.cs](#)

Execute the file

Response is generated in the same page.



# Advantages of web form controls

- Allows object-oriented way of working with the control
- Easier to set the control interfaces or properties like colour of the font etc.
- Provides simple to understand event-driven model
- automatic postback can be triggered for any control → on selecting an item on the list box or on clicking a button.
- View State → state is automatically maintained.
- XHTML Compliance



# ASP.NET Web Page Code Model

- ASP.NET provides two models for pages
  - the single-file page model
  - code-behind page → default and recommended
- Normally an ASP.NET Web page has two parts:
  - visual element tags like server controls tags and static text.
  - programming logic for the page, which includes event handlers and other code.



# Single-file page model

- Both the visual element and the programming logic is written in the same page.

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(Object sender,
EventArgs e) {
Label1.Text =
DateTime.Now.ToString();
}
</script>
<html><head>
<title>Single-FileModel</title>
</head>
```

*code*



```
<body>
 <form runat="server">
 <asp:Label id="Label1"
runat="server" Text="Label">
 </asp:Label>

 <asp:Button id="Button1"
 runat="server"
 onclick="Button1_Click"
 Text="Button">
 </asp:Button>
 </form>
</body>
</html>
```

*Control tags*



# Code-behind page

- An ASP.NET Web page has constitute two files:
  - File containing visual element tags like server controls tags and static text.
  - File containing programming logic for the page, which includes event handlers and other code.

# Default2.aspx



```
%@ Page Language="C#" AutoEventWireup="true"
 CodeFile="Default2.aspx.cs"
 Inherits="Default2" %>
<asp:Content ID="Content1"
 ContentPlaceHolderID="Title"
 Runat="Server">
<html>
 <head runat="server" >
 <title>Code-Behind Page Model</title>
 </head>
 <body>
 <form id="form1" runat="server">
 <div> <asp:Label id="Label1" runat="server"
 Text="Label" > </asp:Label>

 <asp:Button id="Button1" runat="server"
 onclick="Button1_Click" Text="Button" >
 </asp:Button> </div> </form> </body>
 </html>
```

# Default2.aspx



```
%@ Page Language="C#" AutoEventWireup="true"
 CodeFile="Default2.aspx.cs"
 Inherits="Default2" %>
<asp:Content ID="Content1"
 ContentPlaceHolderID="Title"
 Runat="Server">
<html>
 <head runat="server" >
 <title>Code-Behind Page Model</title>
 </head>
 <body>
 <form id="form1" runat="server">
 <div> <asp:Label id="Label1" runat="server"
 Text="Label" > </asp:Label>

 <asp:Button id="Button1" runat="server"
 onclick="Button1_Click" Text="Button" >
 </asp:Button> </div> </form> </body>
 </html>
```



# Default2.aspx.cs

```
using System; using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class Default2 :
 System.Web.UI.Page
{
 protected void Page_Load(object sender,
 EventArgs e) { }

 protected void Button1_Click(object sender,
 EventArgs e) {
 Label1.Text = "Clicked at " +
 DateTime.Now.ToString();
 }
}
```



# Advantages of single file model

- For a small simple page all code in one file is simpler to handle.
- slightly easier to deploy or transfer because you need to send only one file.
- Because there is no dependency between files, a single-file page is easier to rename.

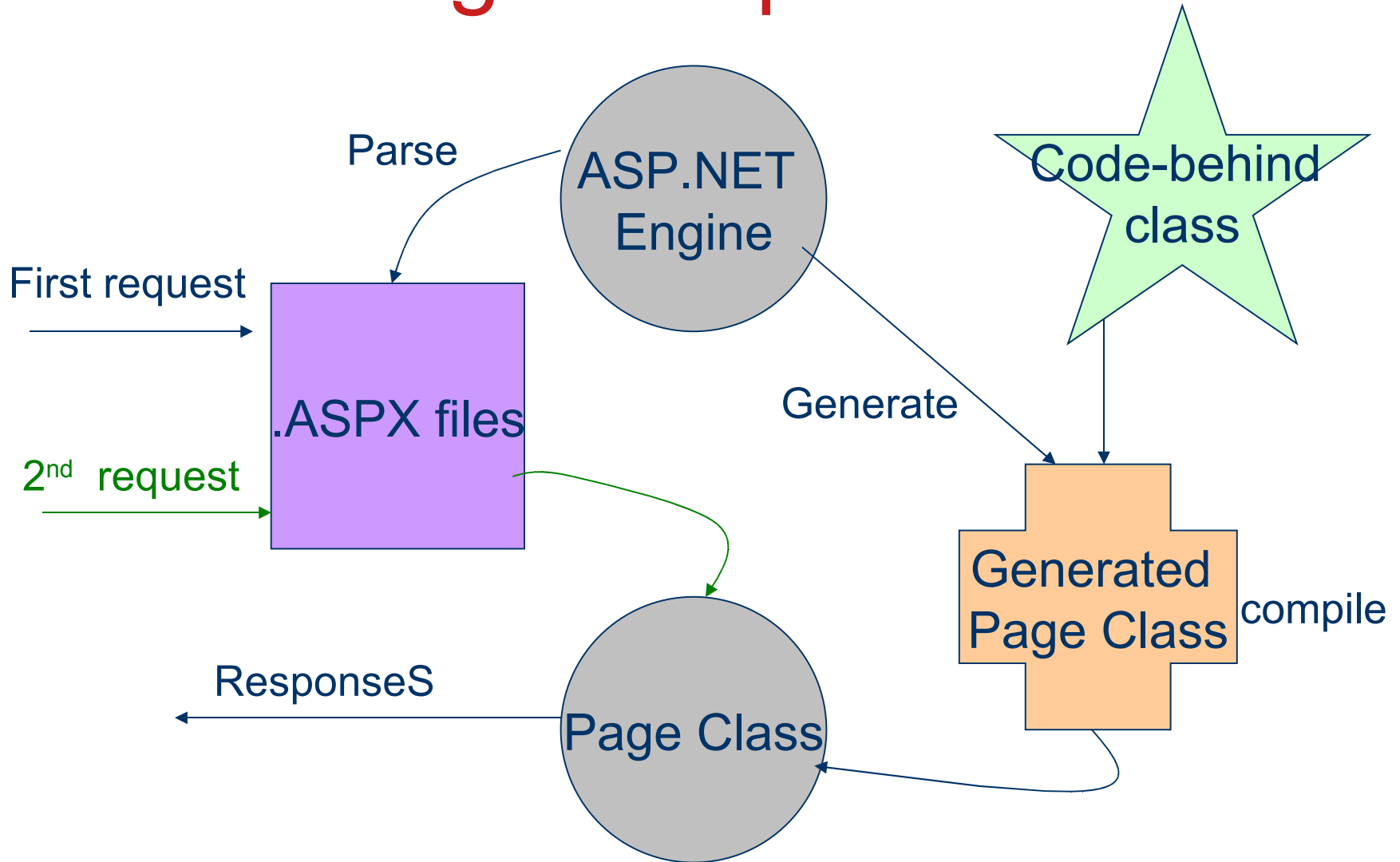


# Advantages of Code-Behind Pages

- A clean separation of design and code
- The design code is usually mark up code which is achieved through visual programming (drag drop). This code is generated automatically by visual studio. This approach keeps the code that programmer writes is kept separate from the code that is automatically generated.
- Page designers is not exposed to the code that programmers write
- Same code can be reused for multiple pages.



# Page compilation





# Page Life Cycle

- When an ASP.NET page is requested, the page goes through a life cycle.
- The life cycle include things like initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering etc.
- The callbacks to the various stages that the page goes through is provided in the form of events.
- One could make use of these callback and grab an opportunity to do something just when the page is at one of these stages.



# Some important page life cycle events- listed sequentially

- **PreInit**
  - Before any initialization happens to the page
  - Uses
    - Check the **IsPostBack** property to determine whether this is the first time the page is being processed.
    - Set a master page dynamically.
    - Set the Theme property dynamically.



- **Init**
  - Raised after all controls have been initialized.
  - Used to read or initialize control properties.
- **InitComplete**
  - Raised by the Page object.
  - Used for processing tasks that require all initialization be complete.
- **PreLoad**
  - After this event, view state is loaded and then any postback data included with the Request instance.
  - Used if you need to perform processing on your page or control before the Load event.



- **Load**
  - The Page calls the OnLoad event method on the Page, then recursively does the same for each child control, which does the same for each of its child controls until the page and all controls are loaded.
  - Used to set properties in controls and establish database connections.
- **Control events**
  - Raised when the user triggers an event on the controls
  - Used to handle specific control events, such as a Button control's Click event etc.



- **Unload**
  - Raised when the page unloads
  - This event occurs for each control and then for the page.
  - used to do final cleanup like closing file, database connection etc.

# ASP.NET Application Folders



- **App\_Code** folder
  - .cs (or .vb) files are kept.
  - .wsdl file
  - Dataset files
- All these are automatically available to your code.
- *Write a Calculator class that has two int attributes and has methods to add, subtract etc. Get the two numbers and the type of operation from ASPX page. Use the Calculator class to get the result of the page.*



- **App\_Data** folder
    - MS Access → .mdb files
    - MS SQL → .mdf files
    - XML files
  - **App\_Themes** folder
    - Skin files
    - CSS files
    - Images
  - **App\_GlobalResources** folder
    - .resx files
  - **Global.asax** file
    - This file has events that can be used to initialize the objects in the application-level and session-level.
- later
- 
- Two red arrows originate from the 'App\_Themes' folder section. One arrow points from the 'Images' sub-item to the word 'later'. The other arrow points from the 'App\_GlobalResources' folder section to the word 'later'.

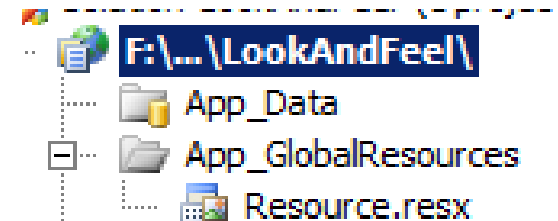
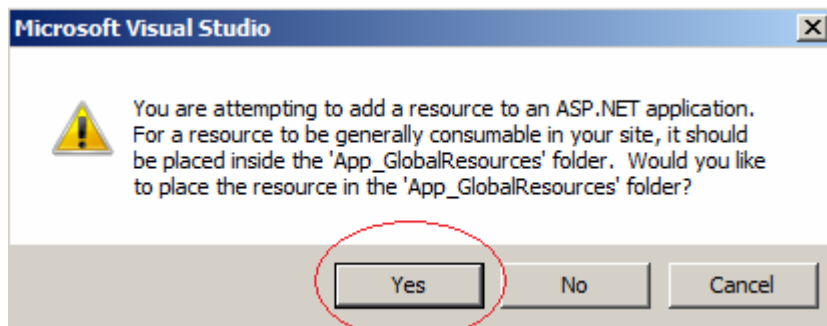
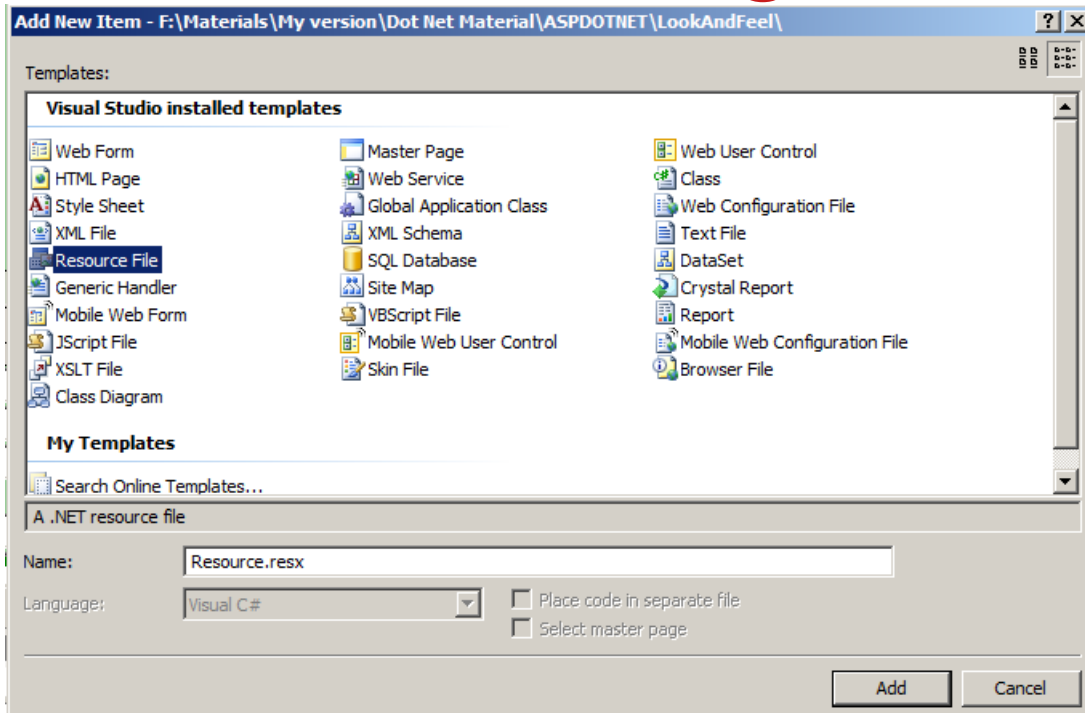


# Resource files

- Resource files are text files that contain name-value pairs which form data dictionary for the application.
- The frequently used text in the pages like titles, error messages and greetings can be placed in this file and a name can be associated with this text.
- This helps in managing the change easily.
- This is also usually used when you want to customize content based on the things such as culture.



# Creating resource file





# Entries in Resource file

- Type name-value pair as resource file and save the file

	Name ▲	Value
	Greetings	Hello
	ErrorMessage	Some error occurred due to which the request was not handled
✎	Title	My New Site
*		



# Referring the resource references

- Create a page similar to this and add the following code in the source.



```
<script runat="server">
protected void Page_Load(object sender,
EventArgs e) {
 Page.Title = Resources.Resource.Title;
}
protected void Button1_Click(object
sender, EventArgs e) {
Label1.Text =
Resources.Resource.Greetings; }
</script>
```



# @Page Directive

- Instructions written at the top of an ASP.NET page
- Has settings related to how a page should rendered and processed
- This is the page directive which automatically gets inserted at the top of every page
  - `<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>`



# Page Directive attributes

- **AspCompat**
  - if true allows the page to be executed on a single-threaded apartment. setting this attribute to true can degrade performance.
- **Language**
  - language being used in the code-behind. Can be Visual Basic, C#, or JScript .NET
- **AutoEventWireup**
  - If true, every page will have an automatic way to bind the events to methods in the same .aspx file or in code behind.



- **ValidateRequest**

- If true (default), request validation checks all input data against a hard-coded list of potentially dangerous values. If a match occurs, an

**HttpRequestValidationException**

object is thrown. This feature is enabled in the machine configuration file (Machine.config).

This can be disabled in application configuration file (Web.config) or on the page by setting this attribute to false.

- **Theme**

- Used to specify the theme for the page. This is a new feature available in Asp.Net 2.0.



- **MasterPageFile**
  - Specify the location of the MasterPage file to be used with the current Asp.Net page.
- **EnableViewState**
  - If true the view state is maintained across page requests. The default is true.
- **ErrorPage**
  - Specifies a target URL for redirection if an unhandled page exception occurs.
- **Inherits**
  - Specifies a code-behind class for the page to inherit. This can be any class derived from the Page class.



- **CodeFile**
  - Specifies the code-behind file with which the page is associated
- **Title**
  - sets the page title other than what is specified in the master page.
- **Culture**
  - If auto, enables the page to automatically detect the culture required for the page.
- **UICulture**
  - Specifies the UI culture setting to use for the page. Supports any valid UI culture value.
- **SmartNavigation**
  - If true returns the post-back to current position of the page. The default is false.



- **EnableSessionState**
  - Session state is enabled when set to true (default).
- **Debug**
  - If true compiles the page with debug symbols in place. Default is false.