

Departamento de Lenguajes y Sistemas Informáticos

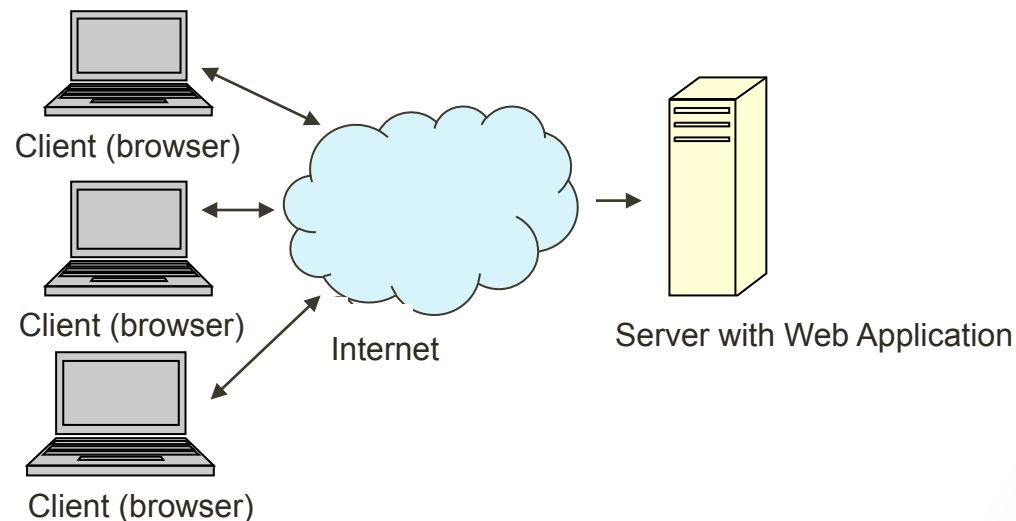
# Unit 9. Web Applications. Interface Layer (I)

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Escuela Politécnica Superior  
Universidad de Alicante

# Web applications vs desktop applications

- Easy user interface creation.
- Updates distribution easier and less costly.
- Distributed processing.
  - Much more easy to provide processing in the server side.
  - The Web provides standard protocols (HTTP, HTML, XML) to ease n-layer applications.



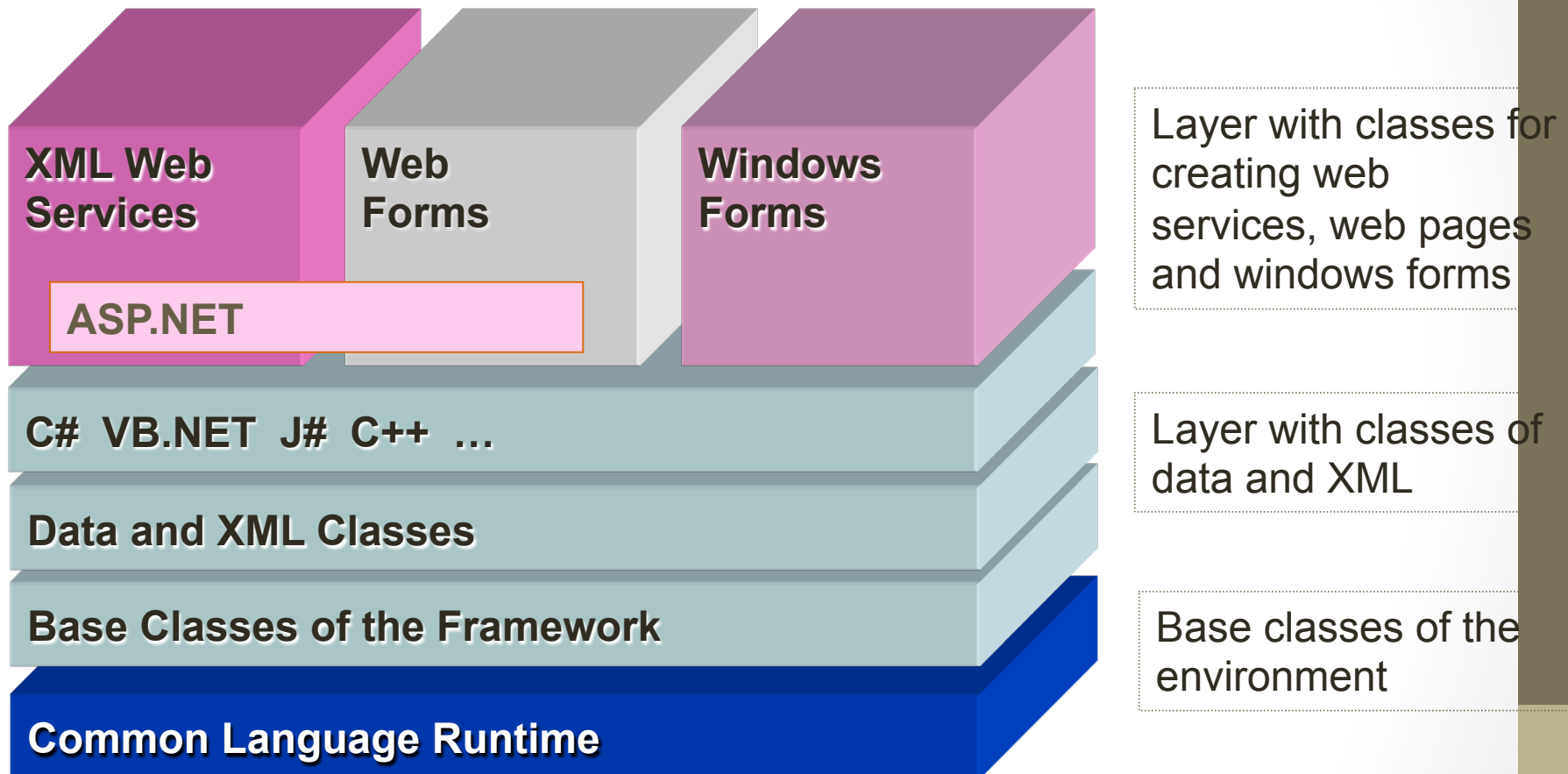
# Server dependent technology

- The main advantage is on the security the programmer has over his code:
  - He finds the files of the server that are executed when they are requested through the Web.
  - The users do not have access to the code, just to the resulting page on their browser

# What is... ASP.NET?

- Platform to build [Web applications and Web Services](#) that work under IIS.
- Part of the .NET framework
- Server side technology
- ASP.NET languages
  - Object oriented
  - Event driven
  - Compiled in the sever
- Support of multiple languages:
  - C#
  - VB.NET
  - Jscript.NET
  - J#

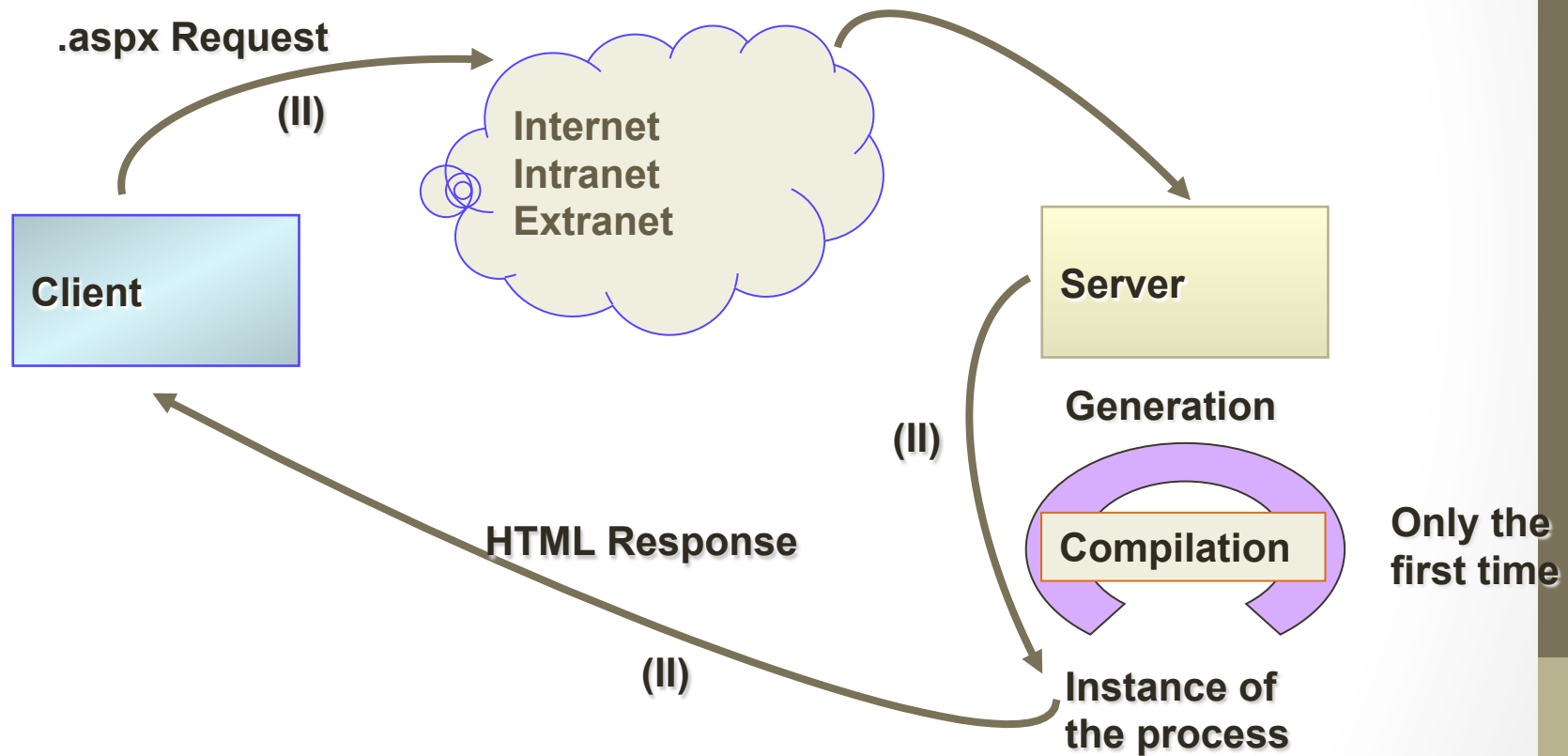
# Microsoft .NET Framework



# “Execution”

How does it work?

- Call to an ASP .NET web page



# Web Applications with ASP.NET

- Combination of files, pages, handlers, modules and executable code that can be invoked from a virtual directory.
- They can divided into several web pages.
- They share a set of resources and common configuration options..
- Each application has its own:
  - Cache set
  - Session data

# Where is the application stored??

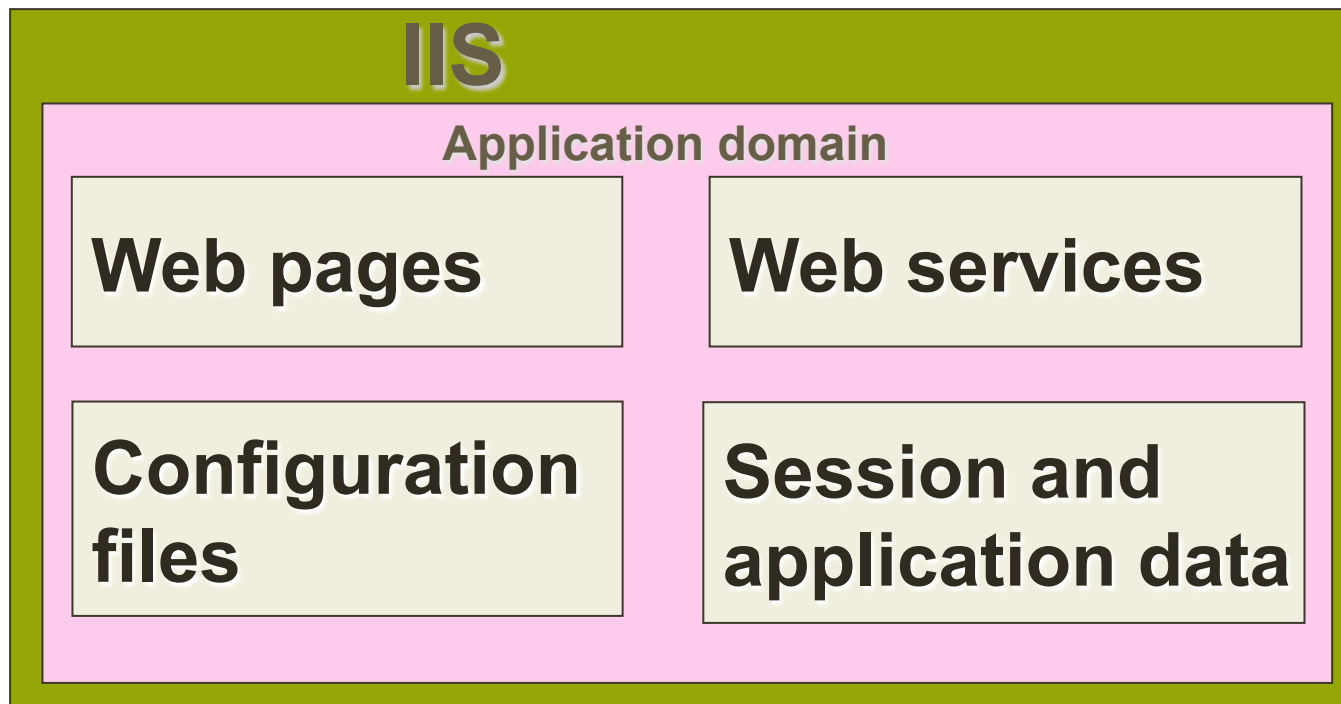
## → Virtual Directory

- A web application only exists in a localization that has been published by IIS as a Virtual Directory.
- A virtual directory is a shared resource identified by an alias that represents the physical location in the server.
- `//localhost` is the virtual root of the computer (`\InetPub\wwwroot`)



# Virtual directory

- Virtual directory: structure of basic agrupation that delimites an application.
- Creation and administration using IIS (Internet Information Server)



# Web Forms(I)

- Techniques for Rapid Application Development (RAD).
- We can:
  - Drag and drop controls in a form
  - Write the supporting code
- The application is developed for a web server.
- The users interact with the application through a browser.

# Web Forms(II)

- They provide an approach oriented to:
  - Objects
  - Events
  - Status management
- Programming in the server side for managing events in the client side:
  - They can be virtually executed in any browser compatible with HTML.

## Web forms(III)

- All the server controls have to appear inside a `<form>` tag, and this tag has to contain the `runat="server"` attribute.
- This attribute indicates that the form has to be processed on the Server.
- It also indicates that the controls contained can be accessed by server scripts:  
`<form runat="server">`  
    ...HTML + controles de servidor  
`</form>`

**A .aspx web page has to contain a unique control**  
`<form runat="server">`

# In Visual Studio: Website or Web Project?

- Website: set of independent Web pages.
  - For simple Web pages (e.g., personal Web page...)
- **Web Project:** set of Web pages linked with a project file.
  - For advanced applications
  - We can reference DLLs, etc

# Do we need Internet Information Server?

- Visual Studio has its own development server, therefore, for testing our application in our computer we do not need to have IIS installed.
- However, in order to deploy our application in a server, we would need it.

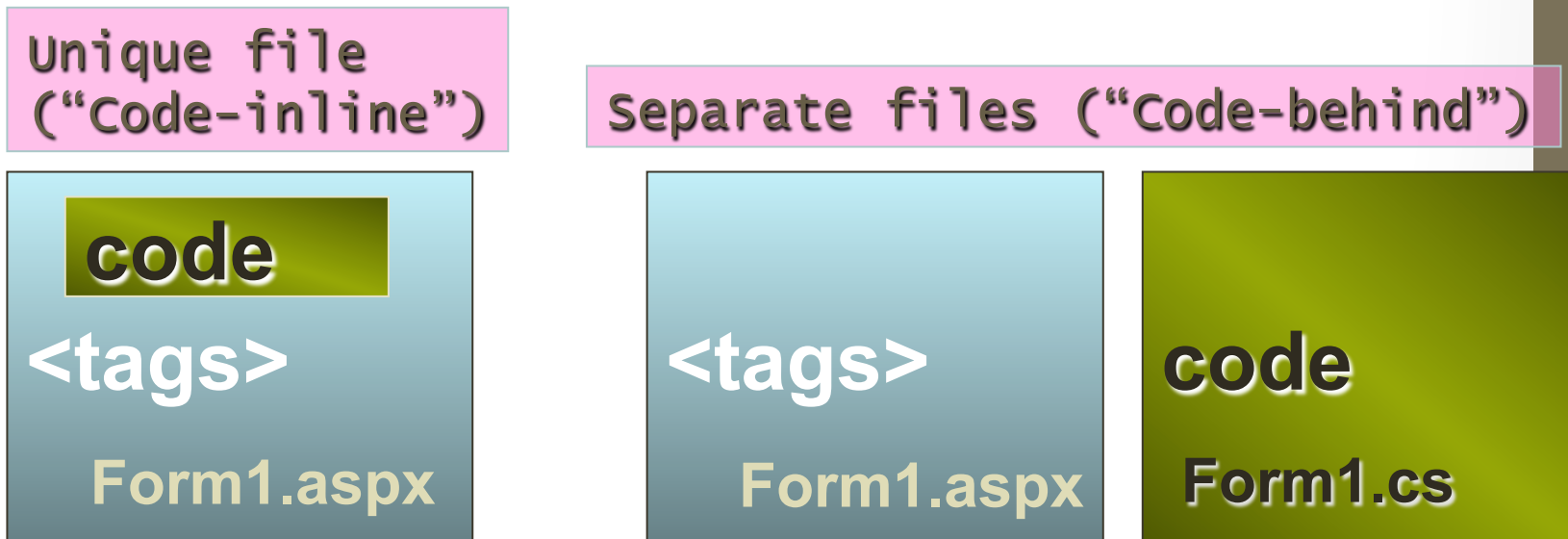
# In Visual Studio...

## Where is the application stored?

- **File system**
- C:\Documents and Settings\aaaa\Mis documentos\Visual Studio 2005\WebSites\WebSite2
  - [http://localhost:3371 /WebSite2 /Default.aspx](http://localhost:3371/WebSite2/Default.aspx)
- **Local IIS:** Folder of the default website (<http://localhost>)
  - For instance C:\Inetpub\wwwroot\WebSite1

# Code-inline vs Code-behind

- ▶ Declarative “Tags”
  - HTML, server controls, static text
- ▶ Different to ASP, good separation between the code and tags





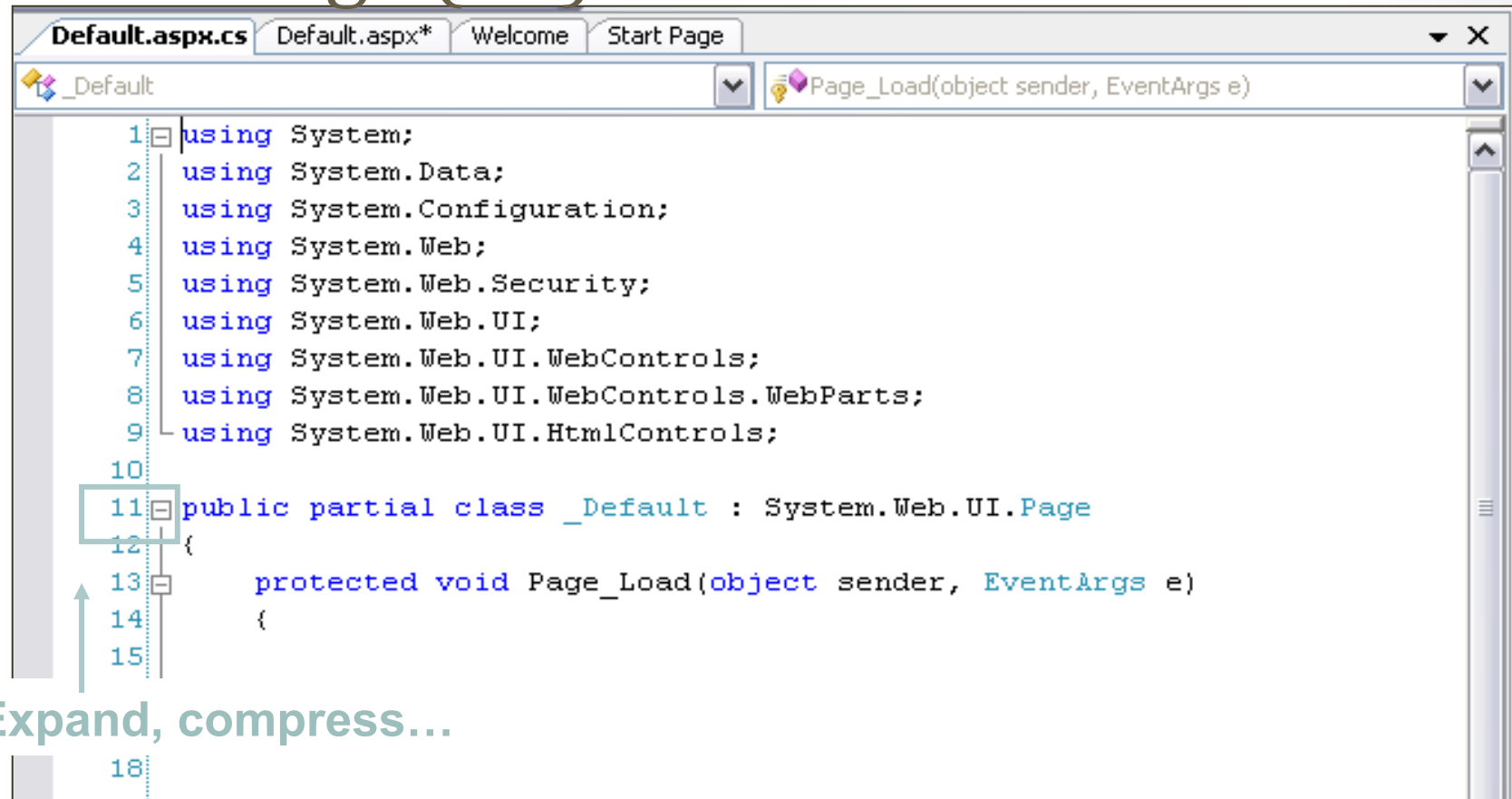
# Code-behind

- The code for managing events is located in a physical separated file from the page that contains the server controls and the tags.
  - Extension *.aspx*
  - Extension *.cs*, *.vb* ... (*code-behind*)
- USEFUL to have it separated :

It is common in group projects having

  - Designers working in the UI of the application
  - And developers working on the behaviour or code.

# Ver código (C#)



```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Web;
5 using System.Web.Security;
6 using System.Web.UI;
7 using System.Web.UI.WebControls;
8 using System.Web.UI.WebControls.WebParts;
9 using System.Web.UI.HtmlControls;
10
11 public partial class _Default : System.Web.UI.Page
12 {
13     protected void Page_Load(object sender, EventArgs e)
14     {
15
16
17
18
```

Expand, compress...

**Definition of a partial class**

**Procedure that handles the LOAD event of the page**

Cuadro de herra... Web.config

**Estándar**

- Puntero
- Label
- TextBox
- Button
- LinkButton
- ImageButton
- HyperLink
- DropDownList
- ListBox
- CheckBox
- CheckBoxList
- RadioButton
- RadioButtonList
- Image
- ImageMap
- Table
- BulletedList
- HiddenField
- Literal
- Calendar

Web.config Default.aspx\* Default.aspx\* Página de inicio Ex

**Propiedades**

**Button1** System.Web.UI.WebControls.Button

(Expressions)	
(ID)	<b>Button1</b>
AccessKey	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CausesValidation	True
CommandArgument	
CommandName	
CssClass	
Enabled	True
EnableTheming	True
EnableViewState	True
Font	
ForeColor	
Height	
OnClientClick	
PostBackUrl	
SkinID	
TabIndex	0
Text	<b>Button</b>
ToolTip	
UseSubmitBehavior	True
ValidationGroup	
Visible	True

**Text**

**\_Default**

```
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.
{
    protected void Page_Load(object sender,
    {
        this.
    }
}
```

AppRelativeTemplateSourceDirectory  
AppRelativeVirtualPath  
AsyncTimeout  
BuildProfileTree  
**Button1**  
Cache  
ChildControlsCreated  
ClearChildControlState  
ClearChildState  
ClearChildViewState

Lista de errores 0 errores 0 advertencias 0 mensajes

Descripción

Lista de errores Resultados

Listo

# Index

1. Master Pages
2. Server Controls
3. Events
4. LookandFeel using CSS
5. Navigating trthough WebForms

***1***

# Master Pages

# Master Page

- Master pages allow to create a coherent design for the web pages of the application.
- They can define the look, design and standard behaviour for all the pages (or group of pages) of our application in a single master page.
- Then, we create individual content pages that include the content we want to be shown.

# When requesting a page...

- When the users request content pages, these are combined with the master page for generating an output which combines the master page design with the content page.

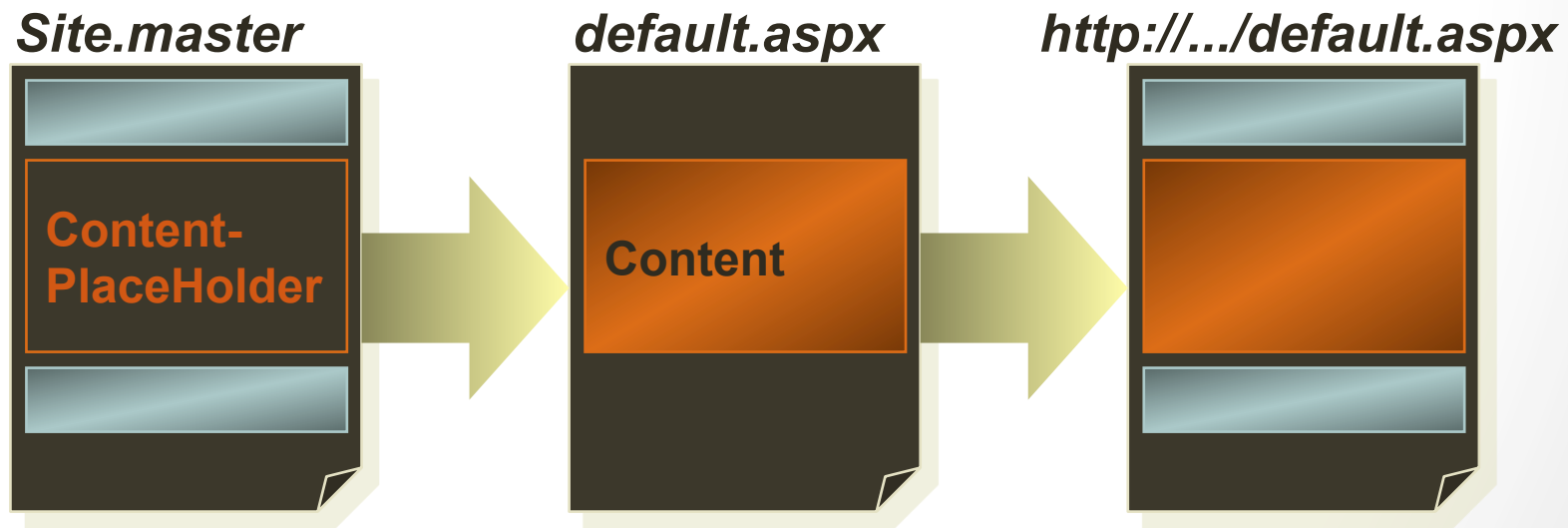
# Advantages

- We can make design modifications in a unique location; changes will be reflected in all the pages using the master page.
- Reuse of the user interface.
- Better final user experience: more coherent pages



# How it works...

- Master pages define the common content and the content containers (content placeholders)
- Content pages refer to the master pages and fill the containers with the content.



# At execution time

- IIS controls the master pages in the following sequence:
  - Users request a page writing the URL address of the content page.
  - When the page is captured, the **@ Page directive** is read. If it refers to a master page, the master page is also read. If the pages are requested for the first time, the 2 pages are compiled.
  - The master page with the updated content is combined in the page control tree of the content page.
  - The content of the individual content controls are combined with the correspondent **contentplaceholder** content of the master page.
  - The resulting combined page is shown in the browser.

# Page.Master

- New property (Master) of `System.Web.UI.Page`
- This property contains a reference to the content's master page, therefore it provides a content page with programatic access to the master page
  - It determines if the page has associated a master page
  - It allows accessing to the controls defined in the master page and we can write support code in the content page
  - It allows accesing to the public methods and properties defined in the master page
- Integration at level code of the master page and content

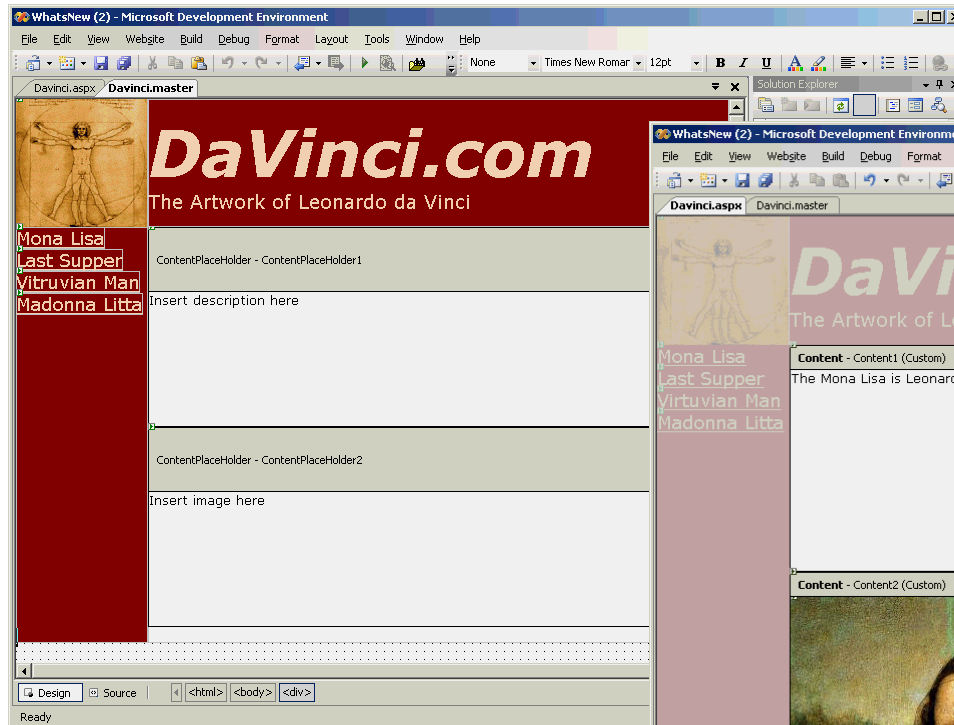
# From the browser...

- From the user perspective, the combination of the master pages and the content pages as a result of a single page. The URL address of this page is the one of the content page.

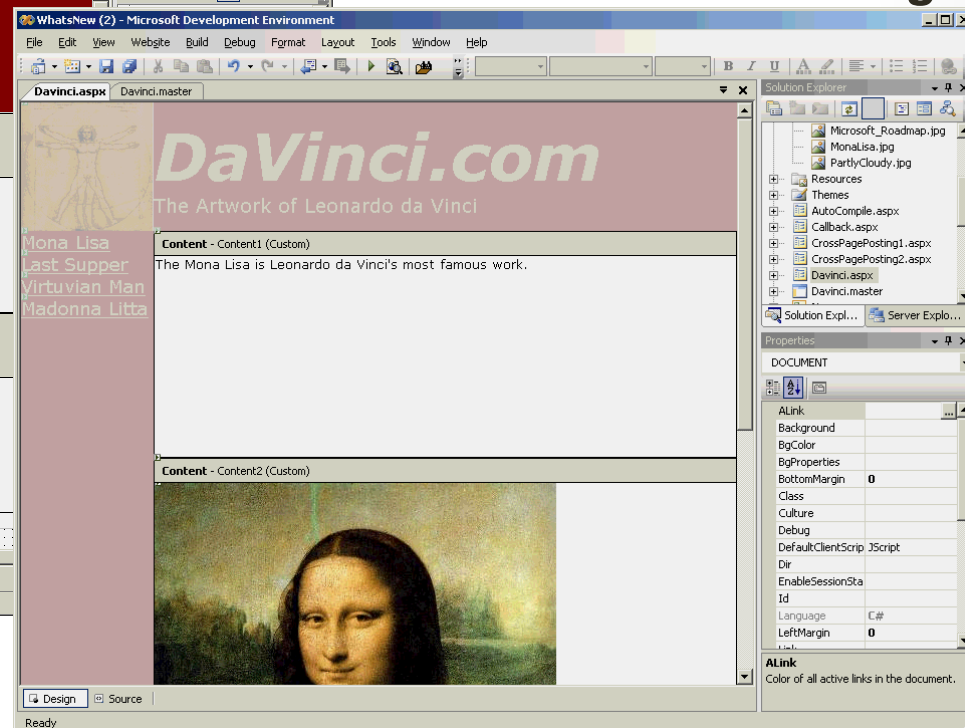
# In Visual Studio...

- Visual inheritance

## Master Page



## Content Page



2

Server controls

# The server controls...

- Have properties that can be established
  - declaratively (in the tag)
  - or programming (in the code).
- They have events that the developers can manage
  - For executing specific actions during the execution of the page
  - or as an answer of an action of the client side that sends the page to the server.

# Server controls

- Support for advanced features: binding data, templates..
- We use the prefix `asp:` and the attribute `runat="server"`.  
`<asp:TextBox id="text" runat="server" />`



# ASP.NET types of controls

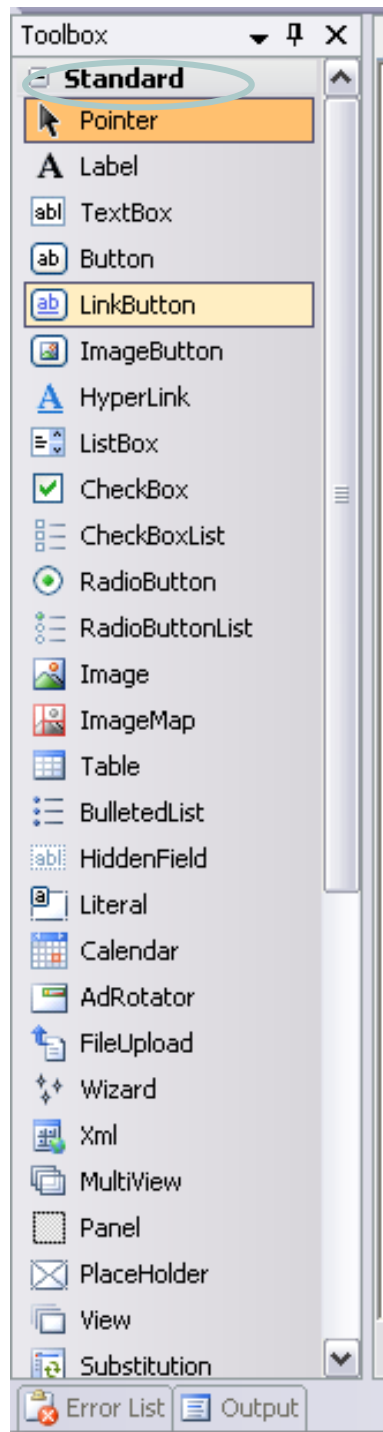
- 1. HTML Controls
- 2. standard controls (web)
- 3. Validation controls
- 4. login controls
- 5. navigation controls
- 6. Webparts controls
- 7. data controls
- 8. user controls

## Main standard controls

Label	It is used for displaying dynamic text (changing its properties in the server code)
Hyperlink	Shows a link to other page.
TextBox	Input of text. Property Textmode values: Single, Multiline or Password.
Image	It is used to display an image in the web page.
Button	It is used in a web page for creating a submit type control (OnClick event) or a command type button (OnCommand event).
LinkButton	It provides a link appearance but it has button functionality
ImageButton	Shows an image that manages click events.
Checkbox	Used to add checkbox elements to a page.
RadioButton	Used to add individual radio button elements to a page.

# Main standard controls

DropDownList	Provides a good way to select multiple elements in a small space. Each element is created with a ListItem control.
ListBox	SelectionMode property: Single, Multiple. Each element is created with a ListItem control.
CheckBoxList	It allows presenting a set of options from which the user can select several elements.
RadioButtonList	It allows creating a list of radio buttons from which the user can only select one element.
Panel	It is a container for other controls.
Table, TableRow, TableCell	We can dynamically create a table (using code).



## Standard controls

- AdRotator
- Placeholder
- ImageMap
- BulletedList
- HiddenField
- FileUpload
- Wizard
- Xml
- MultiView
- Substitution...

# TextBox

## Página.aspx

```
<form id="form1" runat="server">
<div>
<p>
    Username: <asp:TextBox id="userTextBox"
        TextMode="SingleLine"
        Columns="30" runat="server" />
</p>
<p>
    Password: <asp:TextBox id="passwordTextBox"
        TextMode="Password" Columns="30"
        runat="server" />
</p>
<p>
    Comments: <asp:TextBox id="commentsTextBox"
        TextMode="MultiLine" Columns="30" Rows="10"
        runat="server" />
</p>
</div>
</form>
```

Username:

Password:

Comments:

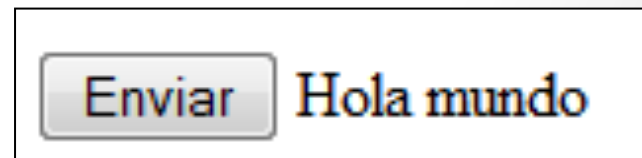
# Button

## Página.aspx

```
<form id="form1" runat="server">  
<asp:Button id="BotonEnviar" Text="Enviar"  
    runat="server" OnClick="WriteText" />  
<asp:Label id="Label1" runat="server" />  
</form>
```

## Página.aspx.cs

```
protected void WriteText(object sender, EventArgs  
e)  
{  
    Label1.Text = "Hola mundo";  
}
```



# ImageButton

## Página.aspx

```
<form id="form1" runat="server">
  <div>
    <asp:ImageButton id="BotonImagen" ImageUrl="~/
    garfield.gif"
    runat="server" OnClick="WriteText" />
    <asp:Label id="Label4" runat="server" /> </form>
```

## Página.aspx.cs

```
protected void WriteText(object sender, ImageClickEventArgs e)
{
    Label4.Text = "Coordenadas:" + e.X + "," + e.Y;
}
```



# Panel

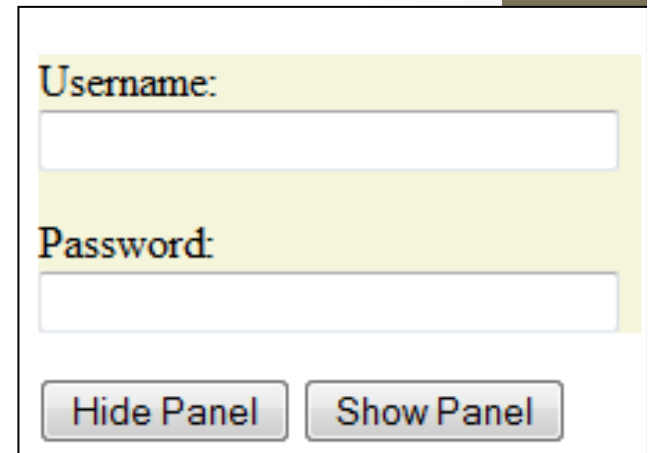
## Página.aspx

```
<form id="form1" runat="server">
  <asp:Panel id="myPanel" BackColor="Beige" Width="220" runat="server">
    <p>Username: <asp:TextBox id="usernameTextBox" Columns="30"
      runat="server" /></p>
    <p>Password: <asp:TextBox id="TextBox1" TextMode="Password"
      Columns="30" runat="server" /></p>
  </asp:Panel>
  <asp:Button id="hideButton" Text="Hide Panel" OnClick="HidePanel"
    runat="server" />
  <asp:Button id="showButton" Text="Show Panel" OnClick="ShowPanel"
    runat="server" />
</form>
```

## Página.aspx.cs

```
protected void HidePanel(object sender, EventArgs e)
{ myPanel.Visible = false; }
```

```
protected void ShowPanel(object sender, EventArgs e)
{ myPanel.Visible = true; }
```

A screenshot of a web application interface. It features a light yellow rectangular panel with a thin border. Inside the panel, the text "Username:" is followed by a text input field. Below that, the text "Password:" is followed by a password input field (indicated by a small eye icon). At the bottom of the panel, there are two buttons: "Hide Panel" and "Show Panel". The background of the page is a light gray gradient with a dark gray vertical bar on the right side.



# 3

## Events in the server controls

# Events model

- Unlike events in desktop applications, ASP.NET server-control events are raised as well as handled on the server. When a Web request communicates a client-side action to the server, a control can raise events on the server in response to the client action.
- The event is handled by the page or by its child controls, and ASP.NET sends a response back to the client. This results in a user experience similar to that of a desktop application.

# Connecting events to methods

- An event is a message sent by an object to signal the occurrence of an action. The action could be caused by user interaction, such as a mouse click, or it could be triggered by some other program logic.
- To consume an event in an application, you must provide an event handler (an event-handling method) that executes program logic in response to the event and register the event handler with the event source. This process is referred to as event wiring.
- The connection of an event and a method (handler) is done using **delegates**

# Events and delegates

- The object that raises (triggers) the event is called the event sender. The object that captures the event and responds to it is called the event receiver.
- In event communication, the event sender class does not know which object or method will receive (handle) the events it raises. **What is needed is an intermediary (or pointer-like mechanism) between the source and the receiver.**
- The .NET Framework defines a special type (**Delegate**) that provides the functionality of a [function pointer](#).

# Events and delegates

- A delegate is a class that can hold a reference to a method.
- Unlike other classes, a delegate class has a signature, and it can hold references only to methods that **match its signature**.
- A delegate is thus equivalent to **a type-safe function pointer or a callback**.
- By convention, event delegates in the .NET Framework have two parameters, the source that raised the event and the data for the event.

# Events and delegates

- Custom event delegates are needed only when an event generates event data. Many events, including some user-interface events such as mouse clicks, do not generate event data.
- In such situations, the event delegate provided in the class library for the no-data event, `System.EventHandler`, is adequate.

`delegate void EventHandler(object sender, EventArgs e);`

- Event delegates are multicast, which means that they can hold references to more than one event handling method. In the Web ASP.NET pages, we do not need to explicitly codify delegates if the control is created in the aspx page.

# Event handlers

- The prototype of the event handler methods have to match with the prototype of the *EventHandler delegate*.
- Therefore, the event handlers in ASPnet return *void* and have **two parameters**:
  - *Object triggering the event*
  - *Event arguments: specific information of the event (EventArgs or derived type)*
- For instance, for a ImageButton control, the second argument is of the type ImageClickEventArgs, which include information about the coordinates where they user has clicked..

# Associate the handler to the control

- Manually

## Archivo Default.aspx

```
<asp:Button ID="Button1" runat="server"
onclick="EventoClick" Text="Button" />
```

## Archivo Default.aspx.cs

```
protected void EventoClick(object sender,
EventArgs e)
{ }
```

- Double click on a control at design mode (default event)

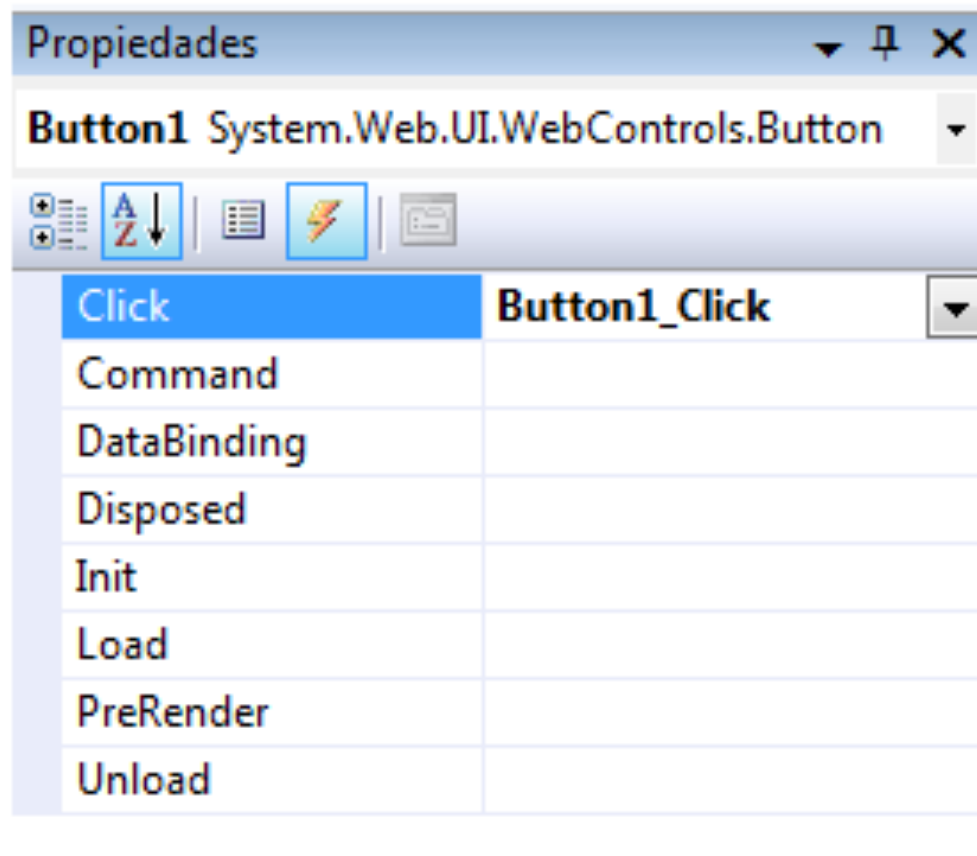
```
protected void Button1_Click(object sender,
EventArgs e)
```

When compiling the page, ASP.NET looks for a method called EventoClick and confirms that has the proper signature (2 arguments, **Object** and EventArgs type). Then, ASP.NET automatically links the event to the method



# Associate the handler to the control

- Select the event in the properties window (and associate a method name or doble click)



# Creating an event handler at execution time with Visual C#

- Normally this is done when we want to create controls programatically.
- For this purpose, **we have to instantiate the delegate EventHandler**, to this instance we will pass the address of the method to be linked.
- Next, the delegate object has to be added to the list of methods called when the event is triggered.
- In the next code we see how to link the **Click** event of the Button1 to a method called myEventHandler:

```
Button1.Click += new System.EventHandler  
(this.myEventHandler);
```

# Example

```
Button b = new Button;  
b.Text = "Click";  
b.Click += new System.EventHandler(ButtonClick);  
Placeholder1.Controls.Add(b);
```

# Types of events

- **POSTBACK events.** These are the events triggered by controls that do a post to the server in as immediate way for processing the Web Form.
  - When sending a form pressing the accept buttonn. This controls are: Button, Link Button and Image Button.
- **Cached events (NO-POSTBACK).** These events are triggered in the view and will be processed in the server when the data is sent by means of a postback event.
  - For instance, when selecting an element of a list, the status of it changes and then in the server we can check that status. Cached events are: TextBox, DropDownList, CheckBox...

# postback vs no-postback Events

- A page is loaded after every request: this is known as a **PostBack**.
- **PostBack Events:**
  - Cause that the information of the form is sent immediately to the server.
- **no-PostBack (or cached) Events:**
  - The information is sent in the next postback event.
  - Events are stored in a queue on the client until a postback event occurs.

## postback vs no-postback Events (II)

- Button, Link Button and Image Button cause **postback events**.
- TextBox, DropDownList, ListBox, RadioButton and CheckBox, cause **cached events**.
  - However, we can override this behaviour in the controls for doing postback events, changing the property **AutoPostBack to true**.

# Connecting several events with a unique event handler

```
<asp:Button ID="Button1" onclick="Button_Click" runat="server"
  Text="Button1" />
<br />
<asp:Button ID="Button2" onclick="Button_Click" runat="server"
  Text="Button2" />
```

- **Which control triggered the event?**

```
private void Button_Click(object sender, System.EventArgs e)
{
    Button b = (Button) sender;
    Label1.Text = b.ID;
}
```

We can see the event handler method for the **Click** event of a **Button** control called by several buttons. The method shows the **ID** property of the button which triggered the event.

# Page events

- The ASP.NET pages trigger life cycle events such as **Init**, **Load**, **PreRender** and others.
- By default, the page events are connected to methods using the convention of nomenclature **Page\_Eventname**
  - For instance, for managing the **Load** event of the page, we can create a method called **Page\_Load**.
  - At compilation time, ASP.NET will look for the methods based on this convention of nomenclature and will automatically connect the event and the method.
- We can use the convention **Page\_Eventname** for any event of the **Page** class.



# Page events

- The ASP.NET pages automatically connect the page events to the methods with the name **Page\_event**.
- This automatic connection is configured by the attribute **AutoEventWireup** of the @Page directive, which value is set by default to **true**.
- If we set **AutoEventWireup** to **false**, the page will not automatically search the methods with the name **Page\_event**. Therefore, we can create the methods with any name and connect them to the pages events explicitly.

# IsPostBack Property

- A page is loaded after every request: known as **PostBack** .
- The Page\_Load event is triggered in every request.
- **Page.IsPostBack** allows us to do conditional code.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
        Response.Write("<br>Page has been posted back.");
    }
}
```

**Important:** The event handlers for page events do not require any argument. These events (such as Load) can accept the 2 standard arguments , but they will have no value.

# 4

Look and feel  
with CSS

# What is CSS?

- CSS is a tool that allows defining the presentation of a document
- It allows creating a set of styles in a single location
- The pages to which we apply the same style sheet will have the same fonts, colors and size
- They provide a homogeneous stetic to all the pages of a website

# Style sheet types

- External
  - The style rules are placed in a single external style sheet
  - This style sheet is linked with all the pages that are going to have the same appearance
- In a web form:
  - `<link rel="Stylesheet" type="text/css" href="~/hoja.css" />`

It is located inside the HEAD element

# Style sheet types

- Internal

- The style sheet is incrustated inside a document
- The style rules are defined inside the tags:

```
<style type="text/css">  
a { background-color: #ff9;  
    color: #00f; }  
</style>
```

- Problem: We have to rewrite it in every page

It is located inside the HEAD element

# Style sheet types

- In-line
  - Allow assigning an style to a certain element using the **style** attribute

```
<a href="/" style="background-color: #ff9;  
color: #00f;"> Home</a>
```

# Style selector

- In the case of using external or internal style sheets, each style rule has a selector
- A selector can be the type of element to which we are going to apply the style

- Example:

```
a {  
    background-color: #ff9;  
    color: #00f;  
}
```

- In ASP.net there are 2 types of selectors:
  - Types of elements and classes.



# Element selector

- The style is applied to all the elements of the same type

```
h2 { color: #369;}
```

- This code changes the color of all the second-level headers

# Class selector

- It is used when we assign a certain class to an element

```
<p class="pageinfo">
```

```
    Copyright 2010
```

```
</p>
```

- The style sheet will contain a set of features for each class

```
.pageinfo
```

```
{
```

```
    font-family: Arial;
```

```
    font-size: x-small;
```

```
}
```

Class selectors are preceded by "."

# Style properties

- **Font:** font size, type, color...
- **Background:** background color or image...
- **Block:** space between paragraphd, lines, words...
- **Box:** personalize tables, colors, borders...
- **Border:** draws boders around different elements
- ...

# CssClass

- Associate a selector of type class in ASP.NET Webforms:

```
<head runat="server">
```

```
<title>Probando CSS</title>
```

```
<link rel="Stylesheet" type="text/css" href="hoja.css" />
```

```
</head>
```

```
<asp:TextBox ID="TextBox1" CssClass="textbox"  
  runat="server" />
```

hoja.css

```
.textbox { font-family:  
Arial; background-color:  
#0099FF; border: 1px  
solid }
```

# Layout with CSS

- ***Giving layout to a web page is passing the design to HTML code***, locating each element (a header, a menu, etc.).
- Before, we only could do it using **HTML tables** (<table>), but this has many limitations and disadvantages:
  - Data tabulation,
  - Non flexible design, we need to redo it if we want to redistribute the elements.
  - Each browser renders in a different way the HTML tables
  - It occupies more bandwidth and space.
  - Google does not index in the same way the pages structured with tables.
- That is why the layout technique evolved for not using tables, but layers (**DIVs**) **and we define their style using CSS.**

# Divs

- The layers, layouts or divs are the same thing with different name.
- We can imagine them as ***containers or blocks where introducing what we want inside(images, text, animations another block, or everything at the same time) and we assign them a width, height and position to get the page structure we want.***

# Formatting a DIV

- To give format to a DIV we need to identify it in some way, we use the **ID attribute**, on it we will put the name for identifying it in the style sheet:
- `<div id="capa1">¡Esta es mi primer capa!</div>`

```
#capa1{  
  width:210px;  
  height:300px;  
  background-color:green;  
}
```



¡Esta es mi primer capa!

# Class or id?

- The difference between a class (.) and a block (#) is that the block is unique in the page, if we create an style of the block "**#busqueda**" to show a search method, we cannot use it again in the same page, but if we convert it to a class "**.busqueda**" we can use it as many times we want.



# Layout properties

```
<div style = "display: table; margin-left: auto; margin-right: auto;
    width: 500px;">
  <div style = "height: 45px; width: 500px;"></div>
  <div style = "float: left; height: 75px; width: 150px;"></div>
  <div style = "float: right; height: 75px; width: 350px;"></div>
  <div style = "clear: both; height: 35px; width: 500px;"></div>
</div>
```

- The DIV elements can be centered using the attributes **margin-left: auto; margin-right: auto;**
- **The float property** adjusts the elements to the indicated margin (when we have adjacent layers).
- Sometimes you will need have a class or block with no adjacent layers, for that purpose we have the CSS property ***Clear***. (e.g. footer). This property is used together with *float* and allows to **avoid that a layer is positioned next to it**

# Attention

- In the classic structures with tables, we can use size using percentages.
- Here we can also use them, however we have to check it since the result can be not as expected.

# Layout with CSS links

- <http://www.comocrearunsitioweb.com/conceptos-basicos-css>
- <http://www.webexperto.com/articulos/art/232/por-que-maquetar-con-estandares/>
- <http://www.desarrolloweb.com/manuales/manual-css-hojas-de-estilo.html>
- <http://www.comocrearunsitioweb.com/maquetando-pagina-web-css>

# 5

## Navigation in webforms

# Traditional navigation

- In HTML:
  - Element `<a>` and href attribute
  - `<a runat="server" href="Login.aspx">Regístrate aquí</a>`
  - The page Login.aspx has to be in the same directory as the one that contains the link
  - After clicking the link the page Login.aspx is shown
- In ASPx:
  - `<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Login.aspx">Regístrate aquí</asp:HyperLink>`

# Different ways of browsing

- **Hyperlink control**
  - Navigates to other page
- **Response.Redirect Method**
  - Navigates to other page by means of code. (Equivalent to navigating through a hyperlink)

# Hyperlinks and redirection

- Hyperlinks react to a click event showing the specified page in the NavigateURL property of the control.
- If you want to handle a click in the code, you have to use the events of a LinkButton or ImageButton and use `Response.Redirect("SiguientePagina.aspx");`
- We can also pass parameters to the next page.

# Passing paremeters

- Passing parameters in the URL.
  - Stores the data in the QueryString collection
  - Multiple parameters separated by &

```
int valor = 22;
```

```
int valor1 = 25;
```

```
Response.Redirect("Default4.aspx?par1=" + valor);
```

```
Response.Redirect("Default4.aspx?par1=" + valor + "&par2=" +  
valor1);
```

<http://localhost:49999/Default4.aspx?par1=22&par2=25>



# Passing parameters

- QueryString limitations
  - The characters used have to be allowed in a URL
  - The information is visible to the users in the navigation bar
  - The users can modify the information causing unexpected errors
  - Many browsers have a limit for the URL length

# Passing parameters

- Special characters:
    - & (for separating multiple query strings)
    - + (alternative for representing a space)
    - # (specifies a markup in a webpage)
  - Solution:
    - Use the methods of the `HttpServerUtility` class for encoding the data
- ```
Response.Redirect("WebForm2.aspx?par1="+  
Server.UrlEncode(" &hola "));
```

# Passing paremters

- The UriEncode method substitutes the special characters by scape sequences

`http://localhost:49999/WebForm2.aspx?par1=+%26hola+`

- When getting the encoded data using the method UriEncode with QueryString is automatic in ASP.NET (we do not need a method for decoding the data)

# Request Object

- **Request**: provides information about the HTTP request of the client for the current loaded page.
  - Information about the client
    - (Request.Browser, Request.Browser.IsMobileDevice, Request.Browser.Id)
  - Cookies (Request.Cookies)
  - Parameters passed to the page:

```
if (Request.QueryString["par1"] != "")  
    Label1.Text = Request.QueryString["par1"];
```