



# ASP.NET 3.5

OPTIMIZACION Y  
SERVICIOS WEB

## **OPTIMIZACION Y SERVICIOS WEB**



**DISTRIBUIDO POR:**

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

## **OPTIMIZACION Y SERVICIOS WEB**

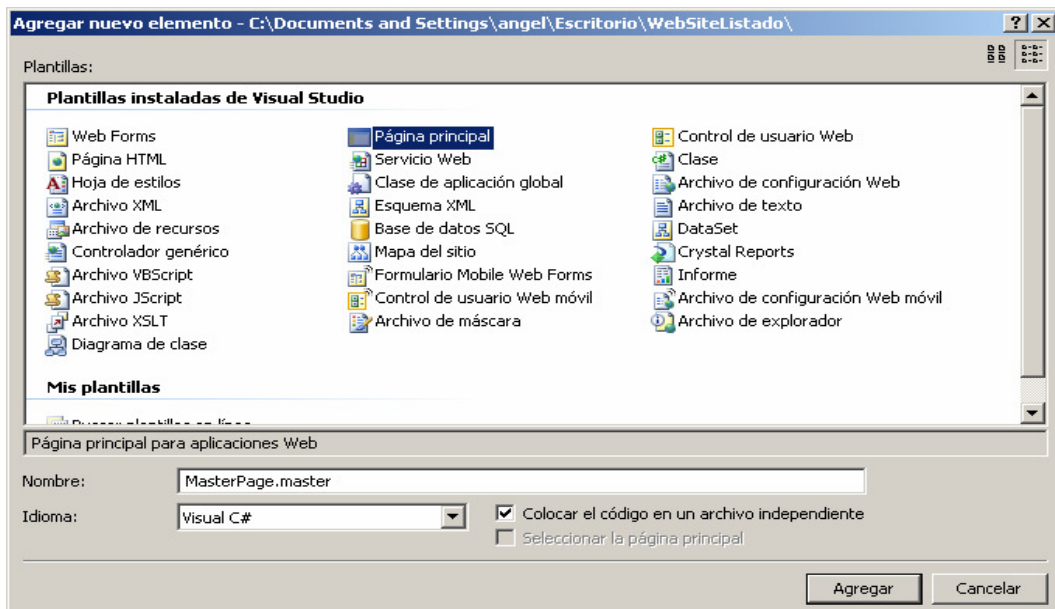
## Páginas Maestras

Una página maestra proporciona un código HTML provisto de controles ASP.NET y código propio, que aparece a modo de plantilla en todas las páginas que conforman una aplicación Web.

Las páginas maestras suelen emplearse para contener elementos tales como logotipos, o un menú de opciones; de manera que se muestre siempre en conjunto con el contenido de todas las demás páginas de la aplicación Web.

### 1.1.- CREACION DE UNA PAGINA MAESTRA

Puede añadirse una página maestra a la aplicación Web seleccionando la opción “Página principal” en el formulario “Agregar Nuevo Elemento”.



El nombre que el Visual Studio asigna a la página maestra por defecto es siempre “MasterPage”. A diferencia de una página ASP.NET convencional, la extensión de las páginas maestras es (.master), y no (.aspx). La página maestra según es creada muestra el siguiente código de diseño:

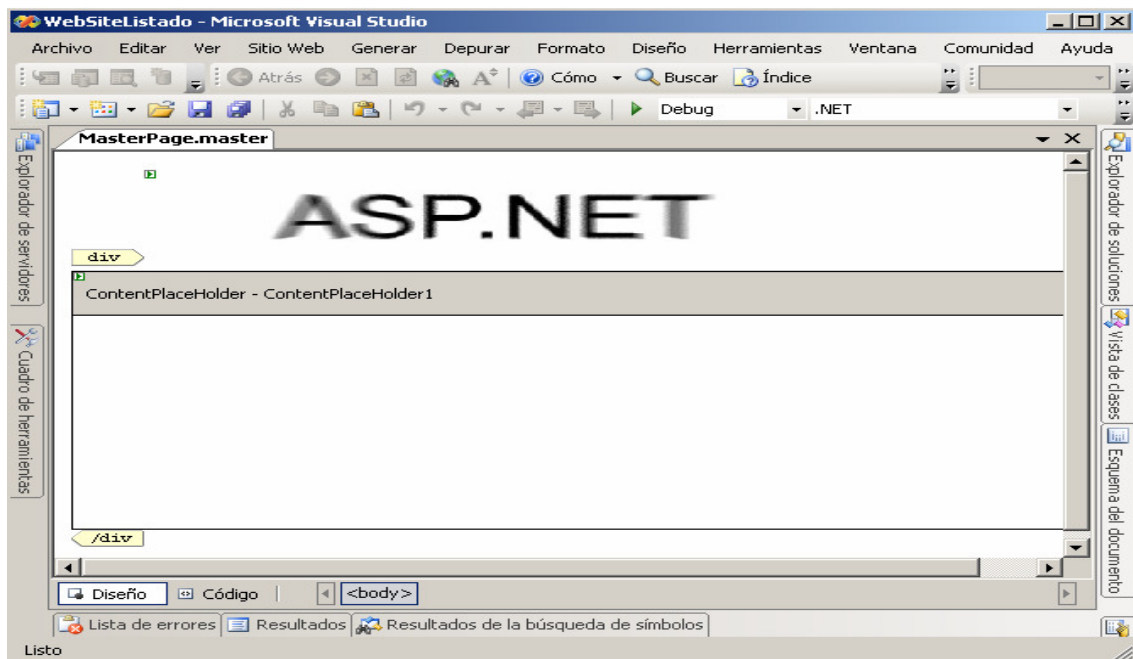
```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<!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></title>
<asp:ContentPlaceHolder id="head" runat="server">
</asp:ContentPlaceHolder>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
</div>
</form>
</body>
</html>
```

# OPTIMIZACION Y SERVICIOS WEB

Todas las páginas maestras constan de una zona de contenido propio, y otra reservada por un control *contentPlaceholder* para el contenido de las demás páginas de la aplicación Web.



*Aspecto de la ventana de diseño de una página maestra*

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<!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></title>
  <asp:ContentPlaceholder id="head" runat="server">
</asp:ContentPlaceholder>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:Image ID="Image1" runat="server" Height="79px" ImageUrl="~/images/title.jpg"
        Width="422px" />
      <asp:ContentPlaceholder id="ContentPlaceholder1" runat="server">
</asp:ContentPlaceholder>
    </div>
  </form>
</body>
</html>
```

En el ejemplo la página maestra muestra un control imagen añadido en su zona de contenido propio encima del control *contentPlaceholder*.

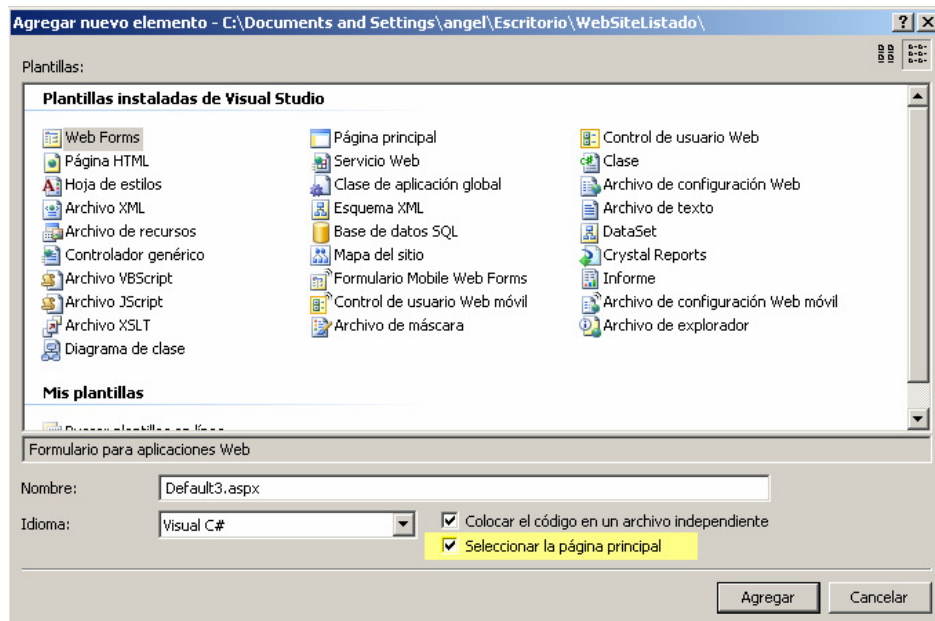
La página maestra consta básicamente de un formulario web, que contiene a su vez el control imagen añadido, y el control *contentPlaceholder*. El contenido del resto de páginas que conforman la aplicación Web se mostrarán dentro del área del control *contentPlaceholder* formando así parte del diseño de la propia página maestra.

El valor del atributo *Id* del control *contentPlaceholder* es bastante importante puesto que aparecerá en el código de diseño del resto de páginas de la aplicación Web.

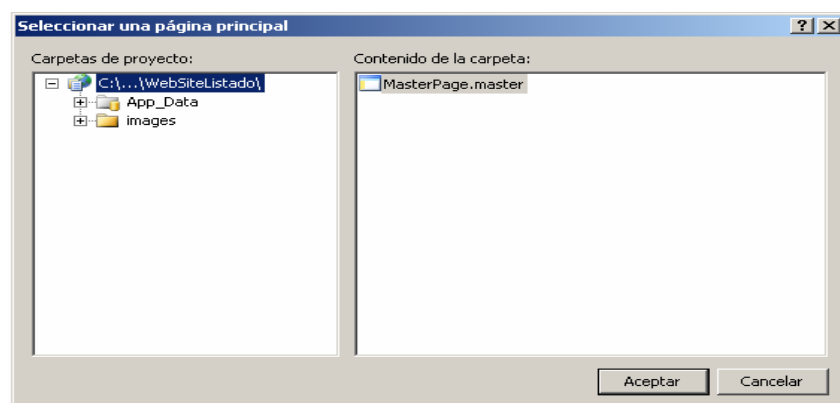
# OPTIMIZACION Y SERVICIOS WEB

## 1.2.- CREACION DE PAGINAS CONTENIDO

Para crear una página contenido que utilice la página maestra, el procedimiento es exactamente el mismo que para crear una página ASP.NET convencional.



La única diferencia es que debemos marcar la casilla de verificación con el título “Seleccionar la página principal”. Al marcar esta opción y tras pulsar el botón “agregar” nos aparecerá la ventana de selección de página maestra.



Seleccionamos la página maestra y pulsamos “Aceptar”. De este modo se creará la nueva página como “contenido” de la página maestra seleccionada. Una vez creada, si observamos su código de diseño se pueden apreciar ciertas diferencias con el código de una página convencional:

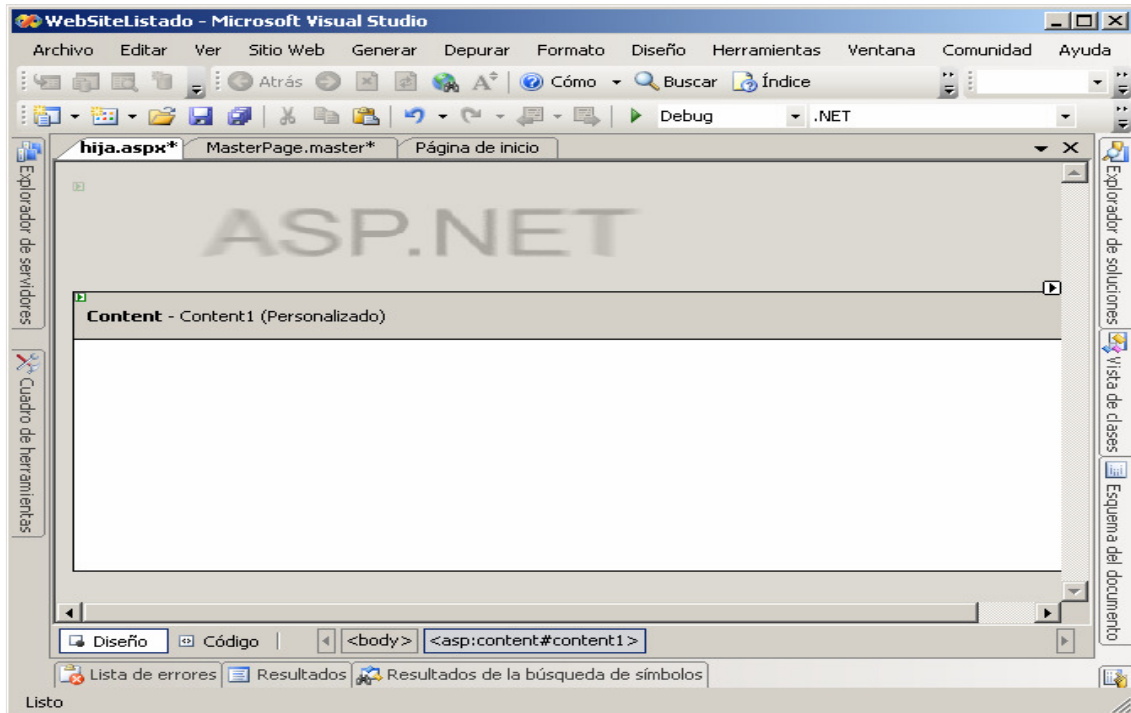
```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
```

La etiqueta Page contiene el atributo *MasterPageFile* con el valor de la URL de la página maestra correspondiente. Esto asocia la página con su página maestra para que se muestren juntas.

## OPTIMIZACION Y SERVICIOS WEB

Otra diferencia es que en vez de aparecer el clásico control formulario con el atributo `runat="server"`, aparece en su lugar un control *Content* cuyo atributo *ContentPlaceHolder* tiene por valor el correspondiente al atributo *Id* del control *ContentPlaceHolder* presente en la página maestra.

Si vemos la vista de diseño de la página contenido observaremos lo siguiente:



El diseño de la página contenido aparece dentro del área del control *Content*, que se corresponde con el área del control *ContentPlaceHolder* presente en la página maestra. El diseño de la página maestra aparece en gris indicando que no puede ser modificado desde el diseño de la página contenido. Así mismo, sólo podemos añadir componentes a la página contenida dentro del área del control *Content*.

Supóngase que añadimos tres controles de caja de texto:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
  <asp:Button ID="Button1" runat="server" Text="Button" />
</asp:Content>
```

Los controles de la página contenido se disponen dentro del control *Content*.

A nivel de cliente, no existe ninguna diferencia apreciable entre la respuesta enviada por una página convencional de ASP.NET, y el generado por una página contenida en una página maestra. Los controles de ambas se generan entremezclados como si de una sola página se tratase en un único código HTML.



# OPTIMIZACION Y SERVICIOS WEB

## 1.3.- CONFIGURACION DE PAGINA MAESTRA

Existen tres maneras de asociar una página ASP.NET con una página maestra. La forma más normal es la inserción del atributo Master a la etiqueta Page en el código de diseño de la página ASP.NET.

Esta asociación también puede llevarse a cabo mediante código en tiempo de ejecución, asociando la URL de la página maestra a la propiedad *MasterPageFile* de la página dentro del evento PreInit.

```
Protected void Page_PreInit(ByVal sender As Object, ByVal e As EventArgs)
{
    This.MasterPageFile = "~/otc.master"
}
```

Finalmente, es posible asociar una página maestra a todas las páginas de una aplicación Web empleando el fichero web.config. Para ello se debe añadir una etiqueta *page* con el atributo *master* indicando la URL de la página maestra a emplear.

```
<configuration>
  <system.web>
    <pages master="otc.master" />
  </system.web>
</configuration>
```

## 1.4.- EJECUCION DE PAGINA MAESTRA

Cuando un servidor Web recibe una petición de una página ASP.NET asociada a una página maestra, ambas son ejecutadas para producir una única página de respuesta para el usuario.

Como ya se ha visto; el diseño de una página maestra contiene un formulario web con sus controles propios y un control *ContentPlaceholder* al menos en su interior. Las páginas contenido poseen un control *Content* que contiene a su vez el resto de controles propios de la página, en este caso; un control label con el texto "HOLA MUNDO".

CODIGO PAGINA MAESTRA	CODIGO PAGINA CONTENIDO
<pre>&lt;%@ Master Language="VB" %&gt; &lt;html xmlns="http://www.w3.org/1999/xhtml"&gt; &lt;head runat="server"&gt;   &lt;title&gt;Untitled Page&lt;/title&gt; &lt;/head&gt; &lt;body&gt;   &lt;form id="form1" runat="server"&gt;     &lt;div&gt;       &lt;asp:ContentPlaceholder ID="ContentPlaceholder1" runat="server"&gt;       &lt;/asp:ContentPlaceholder&gt;     &lt;/div&gt;   &lt;/form&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>&lt;%@ Page Language="C#" MasterPageFile="~/Master1.master" AutoEventWireup="true" Title="Untitled Page"%&gt; &lt;asp:Content ID="Content1" Runat="Server" ContentPlaceHolderID="ContentPlaceholder1" &gt;   &lt;asp:Label ID="Label1" runat="server" Text="Hola mundo"/&gt; &lt;/asp:Content&gt;</pre>

El procesamiento de la página contenido asociada a la página maestra es equivalente al de cualquier página normal, pero con la diferencia de que al procesarse el control Content se procesa el código de la página maestra asociada primero, y una vez completo; se procesa el contenido del propio control Content.

# OPTIMIZACION Y SERVICIOS WEB

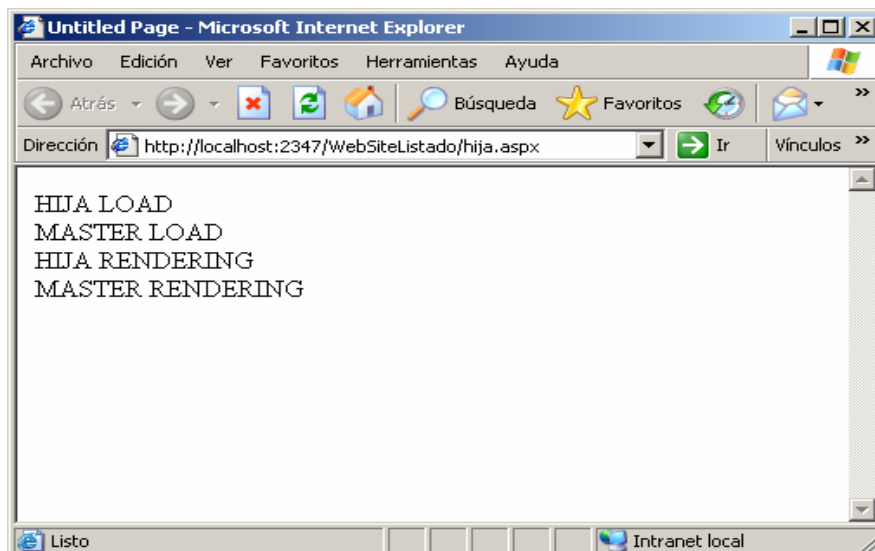
Esto se puede comprobar fácilmente editando los eventos Load, y PreRender de la página maestra y de una página Contenido en una aplicación Web:

```
public partial class MasterPage : System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e) {
        Response.Write("MASTER LOAD<br />");
    }
    protected void Page_PreRender(object sender, EventArgs e) {
        Response.Write("MASTER RENDERING<br />");
    }
}
```

Edición de los eventos en el código de servidor de la página contenido:

```
public partial class hija : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e) {
        Response.Write("HIJA LOAD<br />");
    }
    protected void Page_PreRender(object sender, EventArgs e) {
        Response.Write("HIJA RENDERING<br />");
    }
}
```

El resultado al solicitarse la página contenido, sería el siguiente:



Como puede verse, en algunos casos el mismo evento se desencadena tanto en la página principal como en la de contenido. Por ejemplo, tanto la página maestra como la página contenido desencadenan los eventos Init y Load.

A continuación se muestra la secuencia en la que se desencadenan los eventos cuando una página principal se combina con una de contenido:

- Evento **PreInit** de la página de contenido.
- Evento **Init** de los controles de la página maestra.
- Evento **Init** de los controles de la página de contenido.
- Evento **Init** de la página maestra.
- Evento **Init** de la página de contenido.
- Evento **Load** de la página de contenido.
- Evento **Load** de la página maestra.
- Evento **Load** de los controles de la página maestra.
- Evento **Load** de los controles de la página de contenido.
- Evento **PreRender** de la página de contenido.
- Evento **PreRender** de la página maestra.
- Evento **PreRender** de los controles de la página maestra.
- Evento **PreRender** de los controles de la página de contenido.

# OPTIMIZACION Y SERVICIOS WEB

La regla general sobre cómo se desencadenan los eventos entre una página maestra y la de contenido es que los de eventos de inicialización ( Init, PreInit ) se activan desde el control más interno al más externo, mientras que el resto de los eventos se activan justo en sentido inverso. Hay que tener en cuenta la página maestra se procesa como un control de la página de contenido aunque visualmente sea justo al revés.

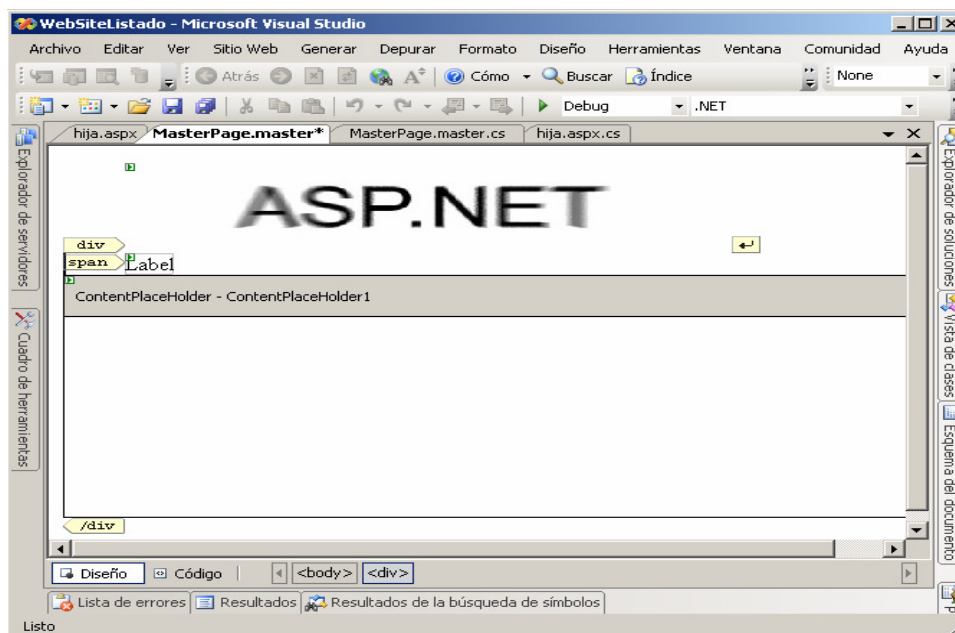
## 1.5.- MANIPULADO DE PAGINA MAESTRA DESDE PAGINA CONTENIDO

En algunas ocasiones puede ser necesario que en respuesta a ciertos eventos sucedidos en los controles de una página contenido, sea necesario modificar algún control de la página maestra con el fin de modificar su aspecto.

Para ello, lo primero es definir propiedades o métodos públicos que puedan ser accedidos desde fuera de la clase que conforma la página maestra.

Por ejemplo, supóngase que en la página maestra tenemos definido un control Web Label de nombre Label1. Deseamos poder modificar el texto mostrado por el control al pulsar un botón en una página contenido.

Dado que los controles, por defecto, son protegidos y son invisibles desde fuera de la clase de la página, será necesario crear una propiedad pública que permita modificar la propiedad Text del control desde fuera:



```
public partial class MasterPage : System.Web.UI.MasterPage
{
    public string title
    {
        get { return this.Label1.Text; }
        set { this.Label1.Text = value; }
    }
}
```

Una vez hecho ésto, el objetivo es poder acceder a la propiedad title de la clase MasterPage desde el código de la clase Contenido.

## OPTIMIZACION Y SERVICIOS WEB

En el código de la página contenido.aspx, implementamos el código del evento Click correspondiente al botón:

```
public partial class hija : System.Web.UI.Page
{
    protected void Button1_Click(object sender, EventArgs e)
    {
        this.Master.title = "PULSADO";
    }
}
```

En el código anterior se emplea la propiedad *Master* de la clase Page para obtener la instancia de la página maestra a la que deseamos acceder. Una vez obtenida la instancia de la página maestra, accedemos a la propiedad pública title y le asignamos el valor deseado.

Para que el IntelliSense del Visual Studio reconozca adecuadamente las propiedades de la clase que representa la página maestra al escribir el código, se recomienda añadir la siguiente directiva a la parte superior del código de diseño de la página contenido:

```
<%@ MasterType TypeName="MasterPage" %>
```

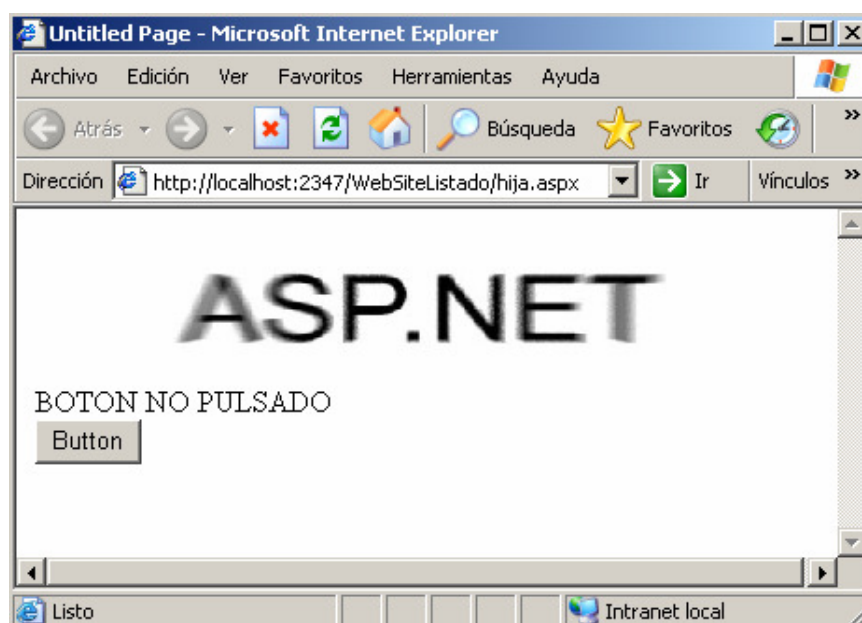
De modo que el código de diseño de la página contenido quedaría:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
CodeFile="hija.aspx.cs" Inherits="hija" Title="Untitled Page" %>
<%@ MasterType TypeName="MasterPage" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <div>
        <asp:Button ID="Button1" runat="server" Text="Button" onClick="Button1_Click" />
    </div>
</asp:Content>
```

Y finalmente, recompilamos el sitio Web:

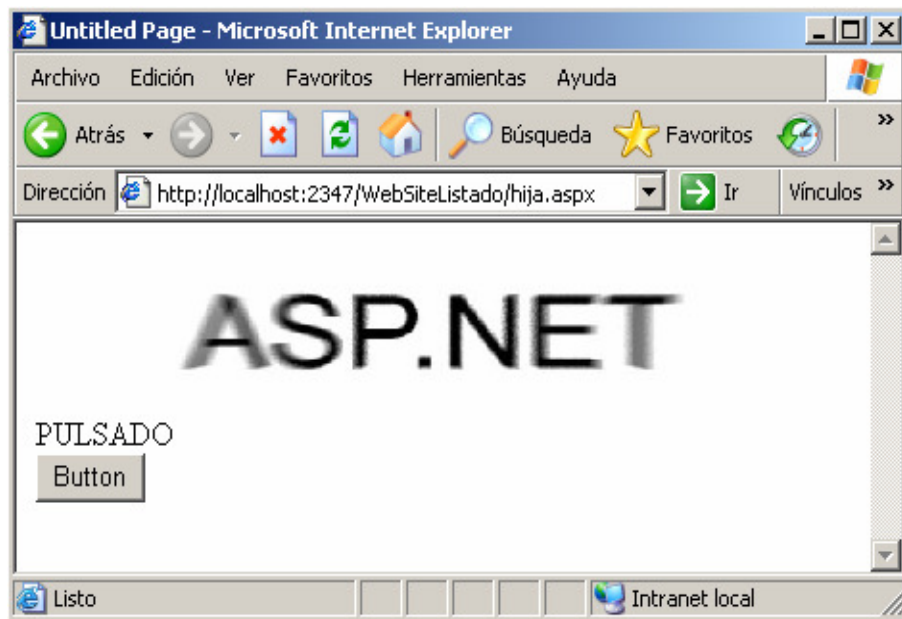
**Generar → Generar de nuevo el Sitio Web.**

Si ahora ejecutamos la aplicación Web:



## OPTIMIZACION Y SERVICIOS WEB

Si ahora pulsamos el botón ( que recordemos realmente pertenece a la página contenido ), provocaremos la recarga de la página.



Ahora obtenemos la página con el botón Label ( que recordemos pertenece a la página maestra ) mostrando el valor indicado en el código del evento del botón en la página contenido.

### **1.6.- MANIPULAR PAGINA CONTENIDO DESDE PAGINA MAESTRA**

Existen otras ocasiones en las se desea que un evento en la página maestra produzca un efecto en la página contenido.

Una forma de conseguirlo es acceder a los controles de la página de contenido desde los eventos de los controles de la propia página maestra. Para ello podemos emplear el método *FindControl*.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label label;
    label = this.ContentPlaceHolder1.FindControl("Label1") as Label;
    label.Visible = false;
}
```

En el código mostrado se accede al control Label1 declarado en la página de contenido actual para hacerlo invisible.

El método *FindControl* nos permite obtener la instancia a un control Web contenido dentro de un control contenedor a partir de su ID. Dado que los controles de la página contenido aparecen vinculados al control ContentPlaceHolder de la página maestra, es este control sobre el que debe buscarse los controles.

A partir de la instancia obtenida es posible acceder a las propiedades del control tal y como si estuviese definida en la propia página maestra y modificar sus propiedades como se desee.

## OPTIMIZACION Y SERVICIOS WEB

Otra forma de conseguir lo mismo es crear un evento personalizado en la página maestra que es disparado cuando se produce una evento sobre alguno de sus controles. Dicho evento debe ser capturado y atendido por las páginas contenido, las cuales se suscribirán a él y aportan el código de atención al mismo.

Supongamos por ejemplo, que tenemos una página maestra con una especie de barra de herramientas formada por múltiples botones. Cada uno de los botones desempeña una acción diferente al ser pulsado, pero dicha acción depende directamente de la página contenido mostrada en el momento. Ante ésta situación, no puede ser la página maestra quien implemente las acciones, si no las contenido.

Lo primero que debemos hacer es crear el evento y delegado correspondiente en el código de la página maestra:

```
public delegate void botonPulsado( int numBoton );  
public event botonPulsado MasterClick;
```

El delegado botonPulsado expone un parámetro de tipo entero. Este parámetro es necesario para conocer en la página contenido cuál fue el botón pulsado en la página maestra. El delegado es utilizado por el evento público MasterClick que debe ser atendido por las páginas contenido.

A continuación, es necesario programar el disparo del evento desde los eventos Click de los diferentes botones presentes en la página maestra.

```
protected void ButtonAnterior_Click(object sender, EventArgs e)  
{  
    MasterClick(0);  
}  
protected void ButtonSiguiente_Click(object sender, EventArgs e)  
{  
    MasterClick(1);  
}  
protected void Button1_Imprimir(object sender, EventArgs e)  
{  
    MasterClick(2);  
}
```

Una vez hecho esto, ya tenemos la página maestra preparada para provocar un evento propio llamado MasterClick, que envía un parámetro de tipo entero indicando el botón pulsado en la misma. Ahora es necesario que las páginas contenido, se suscriban y atiendan este evento:

```
protected void Page_Load(object sender, EventArgs e)  
{  
    this.Master.MasterClick += new MasterPage.botonPulsado(Master_MasterClick);  
}  
void Master_MasterClick(int num)  
{  
    switch (num)  
    {  
        case 0:  
            break;  
        case 1:  
            break;  
        case 2:  
            break;  
        ...  
    }  
}
```

## **OPTIMIZACION Y SERVICIOS WEB**

La suscripción al evento de la página maestra se lleva acabo en el evento Load de cada página contenido, empleando la propiedad Master y asociando una método de respuesta al evento de la página maestra. De este modo, cada una de las páginas contenido de la aplicación Web debe definir un código propio para atender el evento MasterClick de la página maestra.

La atención del evento puede suponer cambiar un valor de algún control de la propia página contenido, o un redireccionamiento a otra página... etc. En cualquier caso, la acción dependerá de la página contenido a pesar de originarse en la página maestra.

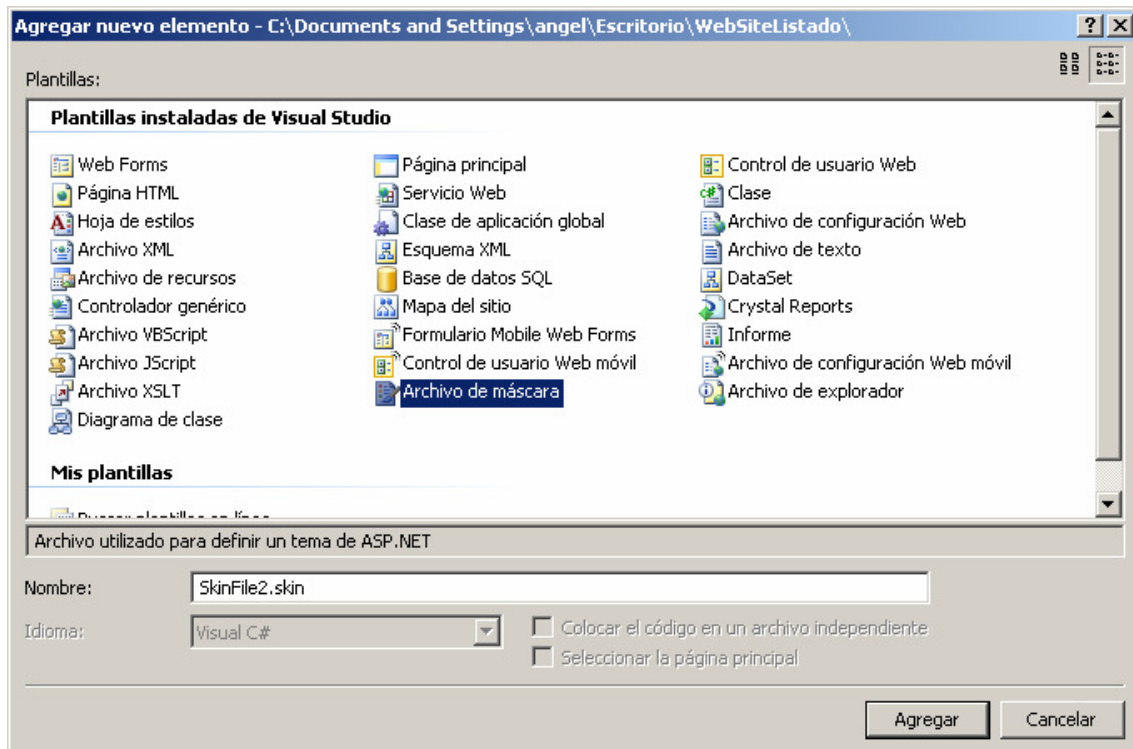




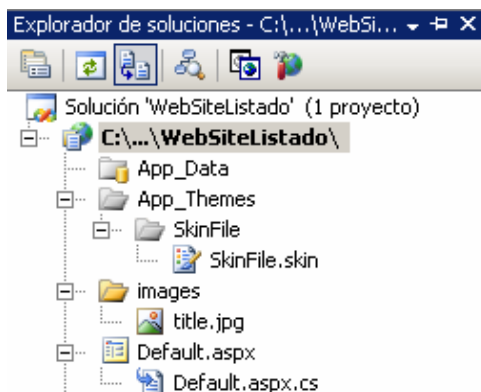
## *Temas*

Los temas son archivos que definen el aspecto de los controles Web de una o varias páginas de una aplicación Web. Estos archivos de extensión .skin pueden a su vez estar formados por hojas de estilo CSS que definen el aspecto de las páginas.

La creación de una tema puede hacerse en Visual Studio 2005 seleccionando la opción “Agregar nuevo elemento...”, y seleccionando a continuación la plantilla “archivo de máscara”.



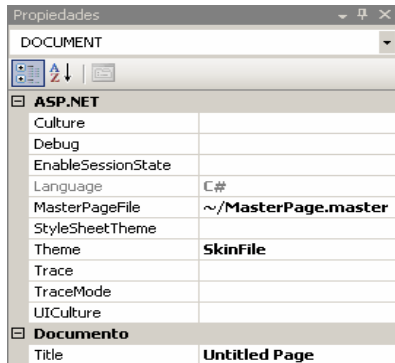
Los archivos de temas deben almacenarse en una carpeta del sitio Web denominada “App\_Themes”. Si ésta no existe, el IDE del Visual Studio la crea cuando se agrega el primer tema a la aplicación Web.



La ventana del explorador de soluciones de la aplicación Web muestra la carpeta App\_Themes, junto con el archivo de tema por defecto: SkinFile.skin.

## OPTIMIZACION Y SERVICIOS WEB

Los temas una vez creados se pueden aplicar a los controles presentes en las páginas de la aplicación Web mediante el uso de los atributos *Theme* o *StyleSheetTheme* de la etiqueta Page en el código de diseño de cada página:



```
<%@ Page Theme="SkinFile"
Language="C#"
AutoEventWireup="true"
CodeFile="hija.aspx.cs"
Inherits="hija"
Title="Untitled Page" %>
```

Esta propiedad puede fijarse a través de la ventana de propiedades de la propia página señalando la etiqueta @Page.

No obstante, si se desea aplicar un fichero de tema a todas las páginas de una aplicación Web, lo más cómodo es modificar el archivo web.config añadiendo la etiqueta <pages> con el atributo *theme* indicando el nombre del archivo de temas dentro del bloque <system.web>:

```
<configuration>
  <system.web>
    <pages theme="skinFile" />
  </system.web>
</configuration>
```

Al crear un nuevo archivo de tema, su contenido por defecto es el siguiente:

```
<%--
Plantilla de máscara predeterminada. Las siguientes máscaras se proporcionan sólo como ejemplos.

1. Máscara de control con nombre. SkinId debería definirse de forma única porque
no se permite que haya varios SkinId por tipo de control en el mismo tema.

<asp:GridView runat="server" SkinId="gridviewSkin" BackColor="white" >
  <AlternatingRowStyle BackColor="Blue" />
</asp:GridView>

2. Máscara predeterminada. Skinid no está definido sólo se permite
una máscara de control por tipo de control en el mismo tema.

<asp:Image runat="server" ImageUrl="~/images/image1.jpg" />
--%>
```

En el código se muestran las dos formas de crear temas.

**Tema por defecto:** Se crea un tema asociado a un control Web sin definir ningún identificador. Este tema pasa a ser el tema por defecto para a todos los controles del mismo tipo. Esto no permite definir más temas para ese control.

Ejemplo: Supongamos la siguiente línea de código en el fichero de tema de la aplicación Web.

```
<asp:Button runat="server" BackColor="red"/>
```

Esta línea define un tema a aplicar a todos los controles Web botón de las páginas ASP.NET asociadas con el tema, según el cual tendrán el color de fondo rojo.

## OPTIMIZACION Y SERVICIOS WEB

**Temas Identificados:** Se crea un tema que se aplica a un determinado control bajo un determinado identificador. Esto permite poder definir varios temas para un mismo control Web pero con diferentes identificadores.

Ejemplo: Supongamos que tenemos la siguiente línea de código en el fichero de tema de la aplicación Web.

```
<asp:Button runat="server" SkinId="botonRojo" BackColor="red"/>
```

Esta línea define un tema para los controles Web de tipo Boton con el identificador "botonRojo".

Comportamiento	
CausesValidation	True
CommandArgument	
CommandName	
Enabled	True
EnableTheming	True
EnableViewState	True
OnClientClick	
PostBackUrl	
SkinID	<b>botonRojo</b>
ToolTip	
UseSubmitBehavior	True
ValidationGroup	
Visible	True

Si deseamos que este tema se aplica a un botón concreto de una página ASP.NET de la aplicación Web, debe indicarse el identificador del tema en el atributo SkinId del control.

Esto se puede hacer bien directamente en el código de diseño del botón, o bien a través de la ventana de propiedades.

El código de diseño del botón quedaría:

```
<asp:Button ID="Button1" runat="server"
Text="Button"
SkinID="botonRojo" />
```

Como puede observarse la declaración de los temas consiste en codificar versiones ya configuradas de controles Web. Estos hacen de máscara para los controles Web de las páginas ASP.NET.



# ***Control de Caché en ASP.NET***

Existen múltiples maneras de mejorar el rendimiento de las aplicación Web en ASP.NET. Una manera es mediante el uso de la memoria caché.

Emplear la memoria caché consiste en almacenar los datos más frecuentemente solicitados en la memoria del servidor, de forma que la próxima vez que se solicite éstos se obtengan más desde la memoria más rápidamente que volviendo a generarse la información. Esto puede aprovecharse para mejorar el rendimiento, especialmente en el caso del contenido generado dinámicamente como son las páginas Web ASP.NET.

La mayoría de los navegadores Web existentes actualmente, almacenan en una memoria caché las páginas visitadas, de forma que, si se vuelve a solicitar poco después la misma página ésta se carga desde la memoria caché en vez de solicitarla nuevamente por internet. La mayoría de los sistemas operativos actuales emplean la misma técnica para almacenar los datos más frecuentes y así agilizar su recuperación y uso. En el caso de los servidores Web con ASP.NET se puede emplear la memoria caché del framework .NET en el lado del servidor.

En todos los tipos de caché disponibles en ASP.NET la información se almacena en una zona de la memoria del servidor directamente gestionada por ASP.NET. Las siguientes solicitudes de las informaciones almacenadas se recuperarán de la caché en vez de volver a generarse o volverse a solicitar a sus respectivo origen.

Si la información en la caché caduca, o su origen es modificado, o se detecta una modificación en algún elemento del que depende dicha información; la caché es automáticamente invalidada. Por consiguiente, la siguiente petición hará que la información se vuelva a generar o solicitar a su respectivo origen.

Las causas de la invalidación de un dato almacenado en la caché dependen directamente del tipo de caché que estemos empleando.

### **3.1.- TIPOS DE CACHE EN ASP.NET**

#### **3.1.1.- Caché de Clase:**

Una página o servicio Web se compilan al solicitarse, generando un ensamblado precompilado que se ejecuta mucho más rápido la siguiente vez que se solicita la misma página al servidor.

La máquina de ejecución de .NET ( CLR ), vigila constantemente posibles cambios en el código de las páginas ASP.NET. Si se produce algún cambio en el código fuente de una página, el ensamblado se invalida, y la CLR vuelve a compilar su código y generar un nuevo ensamblado la próxima vez que sea solicitada. Todo este proceso funciona automáticamente sin intervención del usuario ni del programador Web.

## OPTIMIZACION Y SERVICIOS WEB

### 3.1.2.- Caché de Configuración:

La información de configuración de la aplicación se almacena en archivos de configuración como es el caso del web.config. Cuando se inicia la aplicación Web ( éste es, la primera vez que se solicita una página contra el servidor web ), debe cargarse toda la información de configuración y eso puede tardar unos cuantos segundos dependiendo de la complejidad y extensión del archivo de configuración.

El almacenamiento de esta información una vez cargada en la memoria caché de configuración es automática, y permite atender más rápidamente las demás peticiones. No obstante; si el fichero de configuración ( web.config ) es modificado, se invalidan entonces los datos de caché y se vuelven a cargar desde el fichero.

### 3.1.3.- Caché de Datos:

El almacenamiento en la caché de datos de una base de datos es una de las técnicas que mejoran más el rendimiento. Los controles orígenes de datos están diseñados específicamente para emplear la caché de datos.

La caché de datos almacena siempre información recuperada de una base de datos. Mientras la caché no caduque, las solicitudes de información se recuperan desde la caché en vez de volverla a solicitar a la base de datos. Si la caché caduca, los datos se recuperan automáticamente desde la base de datos nuevamente.

Los controles orígenes de datos heredan todos de la clase abstracta *DataSourceControl*. Entre estos controles podemos encontrar las clases *SqlDataSource*, *AccessDataSource*, *ObjectDataSource*... etc. Todas estos controles poseen una serie de propiedades destinadas al manejo de la caché de datos:

<b>CacheDuration</b>	Valor entero que expresa la cantidad de tiempo medida en segundos en el que se considera válidos los datos.
<b>CacheExpirationPolicy</b>	Valor del enumerador <i>DataSourceCacheExpiry</i> que indica cómo se mide el tiempo de validez de los datos. El valor <i>Absolute</i> indica desde el momento de la carga de los datos. El valor <i>Sliding</i> indica desde el último acceso a los datos en la caché.
<b>EnableCaching</b>	Valor booleano que indica si está activo el uso de la caché de datos para el control.

Estas propiedades pueden configurarse empleando la ventana de propiedades correspondiente al control de origen de datos seleccionado.

# OPTIMIZACION Y SERVICIOS WEB

Estas propiedades se ven reflejadas en el código de diseño del control `SqlDataSource`:

```
<asp:SqlDataSource ID="SqlDataSource1"
    runat="server"
    ConnectionString="<%$ ConnectionStrings:libreriaConnectionString %>"
    SelectCommand="SELECT [usuario], [password], [email], [telefono], [id_usuario] FROM
[clientes]"
    CacheDuration="60"
    EnableCaching="True">
</asp:SqlDataSource>
```

En el ejemplo se ha configurado el control origen de datos `SqlDataSource` para que almacene la información recuperada en la caché de datos automáticamente hasta pasado un minuto. La medición de este tiempo depende directamente del valor de la propiedad `CacheExpirationPolicy`. Por defecto el valor de esta propiedad es `DataSourceCacheExpiry.Absolute`, con lo que la caché caducará transcurridos los segundos indicados en la propiedad `CacheDuration` se hayan accedido a los datos durante ese tiempo o no.

## 3.1.4.- Caché de Resultados

La caché de resultados consiste en el almacenamiento de las página o parte de ellas que se envían a los clientes como resultado de una petición. Este proceso no se lleva acabo automáticamente; es preciso activarlo añadiendo la etiqueta `OutputCache` en la página en cuestión.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default2.aspx.cs" Inherits="Default2" %>
<%@ OutputCache Duration="60" VaryByParam="*" %>
<!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>Página sin título</title>
```

El código de ejemplo anterior muestra las primera líneas del código de diseño de una página ASP.NET que se almacena en la caché de resultados durante un tiempo equivalente a 60 segundos. Durante ese tiempo, las solicitudes de la página se responderán empleando la información almacenada en la caché. Una vez transcurrido ese tiempo, la caché caduca, y la respuesta vuelve a generarse cuando llega la siguiente solicitud.

El atributo `Duration` determina el tiempo máximo en segundos que se mantiene la información en la caché antes que se considere caducada. El parámetro `VaryByParam` permite que se almacene en la caché diferentes respuestas generadas a partir de la misma página dependiendo su valor.

Si el valor es `"none"`, sólo se almacena una versión de la página en la caché de resultados, y se devuelve esa versión independientemente de los valores que puedan ser enviados a la página en la solicitud mediante GET o POST.

Si el valor es `"*"`, se guarda en la caché de resultados una versión distinta de la página por cada combinación de valores recibidos en la solicitud mediante GET o POST.

# OPTIMIZACION Y SERVICIOS WEB

## 3.1.5.- Caché de Objetos

La caché de objetos puede utilizarse para almacenar en ella cualquier objeto que se desee almacenar. El objeto puede ser de cualquier tipo: un tipo de dato, un control Web, un conjunto de datos... etc. La caché de objetos se almacena en la memoria del servidor igual que los demás tipos por lo que se debe ser cauteloso con la información que se almacena en ella. Un uso abusivo podría reducir peligrosamente la memoria del servidor provocando una reducción del rendimiento.

La clase Cache representa la caché de objetos de ASP.NET. Cuando se inicia la aplicación web, automáticamente se crea una instancia de esta clase para cada aplicación. Esta clase posee métodos muy semejantes a los del estado de sesión y aplicación. La información se almacena en una colección de elementos clave-valor. El valor representa la información y la clave un valor distintivo que permite recuperarlo posteriormente.

Supóngase el siguiente ejemplo:

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

    private void createDataGrid()
    {
        DataSet dgrid;
        dGrid = (DataSet)Cache["DataGrid"];
        if (dgrid == null)
        {
            dgrid = getDataSet();
            Cache["DataGrid"] = dgrid;
            Response.Write("Datos desde la base de datos");
        }
        else
        {
            Response.Write("Datos recuperados de la caché");
        }
        GridView1.DataSource = dgrid.Tables[0];
        GridView1.DataBind();
    }

    private DataSet getDataSet()
    {
        // Código de conexión y consulta a la base de datos.
    }

    protected void BtnLimpiar_Click(object sender, EventArgs e)
    {
        Cache.Remove["DataGrid"];
        createDataGrid();
    }
}
```

El método CreateDataGrid se invoca cada vez que se solicita la página. Para ello se le llama desde el evento Page\_Load de la página.

En el método CreateDataGrid se recupera un objeto DataSet del objeto Caché presente como una propiedad de la propia página. Al igual que con las propiedades Session y Application, el elemento retornado es de tipo Object y debe convertirse explícitamente al tipo de dato adecuado. En este caso un DataSet.

```
dGrid = (DataSet)Cache["DataGrid"];
```



## OPTIMIZACION Y SERVICIOS WEB

Después debe comprobarse si el objeto Caché a retornado o no algún valor. Si no se ha almacenado ningún valor en la caché con la clave indicada ( en este caso "DataGrid" ), o ha expirado; la variable de instancia empleada dGrid no apuntará a ningún objeto valiendo null.

```
if (ddrid == null)
```

Si la variable de instancia apunta a un objeto indica que se almacenó previamente con la clave empleada y que no había caducado aún en el momento de recuperarlo. En caso contrario, se invoca al método `getDataSet` para que se recuperen la información desde la base de datos, se cargue en la variable de instancia dGrid, y finalmente se almacene ésta dentro de la caché con la clave "DataGrid" para ser recuperada posteriormente.

```
dgrid = getDataSet();  
Cache["DataGrid"] = dgrid;
```

En cualquier caso, bien sea recuperándolo desde la caché de objetos, o bien volviéndolo a cargar desde la base de datos; el control GridView es vinculado a la instancia de Dataset dGrid.

En este ejemplo, no hay manera de que el objeto DataSet almacenado en la caché de objetos llegue a caducar, a menos que el servidor se quede sin memoria y sea eliminado para reservas más.

Debe tenerse en cuenta que la caché de datos es común a toda la aplicación independientemente de las sesiones, tal y como sucede con los datos almacenados en el objeto Application.

En el ejemplo anterior se añade un botón ( `btnLimpiar` ) cuya función es refrescar la caché de datos. El controlador de eventos de ese botón invoca al método `remove` del objeto caché eliminando el dato indicado por la clave de la caché de datos. A continuación se invoca el método `createDataGrid` que vuelve a recuperar el objeto DataSet de la caché. Al acabar de ser eliminada, los datos se vuelven a recuperar forzosamente desde la base de datos y se almacenan en la caché de datos refrescados.

```
protected void BtnLimpiar_Click(object sender, EventArgs e)  
{  
    Cache.Remove["DataGrid"];  
    createDataGrid();  
}
```

# OPTIMIZACION Y SERVICIOS WEB

## 3.2.- CONTROL DE CACHE DE OBJETOS

La clase caché además de permitir agregar y recuperar valores en función de claves, permite establecer dependencias, gestionar la caducidad de los datos agregados, y controlar la recuperación de la memoria reservada para los elementos almacenados en la caché.

Estas funcionalidades se controlan mediante el uso de los métodos Add e Insert. Ambos métodos tienen la misma funcionalidad con la diferencia de que Add requiere todos los parámetros, e Insert posee múltiples sobrecargas para no tener que indicarlos todos.

```
cache.Add/Insert( keyName, keyValue, Dependencies,  
                 AbsoluteExpiration,  
                 SlidingExpiration,  
                 Priority,  
                 CacheItemRemovedCallback )
```

Los campos clave ( *keyName* ) y valor ( *keyValue* ) representan la información que se almacena en la caché de datos.

Las Dependencias son instancias de la clase *cacheDependency* que establecen una relación entre un dato almacenado en la caché y otro objeto, fichero o momento temporal. Si se alcanza ese punto en el tiempo o el objeto es modificado o desaparece, el elemento dependiente almacenado en la caché caduca y es eliminado.

El objeto externo que controla la dependencia puede ser un archivo, un directorio, una matriz de ambos, o incluso otro dato almacenado también en la caché de objetos. El punto de tiempo designado puede ser absoluto o relativo.

### 3.2.1.- Objeto dependiente de cambio en un archivo

En el ejemplo de código siguiente, el objeto DataSet se recupera a partir de un fichero XML. Si el fichero XML cambia, el objeto de la caché se vuelve obsoleto e inútil. Por lo tanto, es necesario establecer una dependencia tal que si el fichero XML es modificado, el objeto DataSet almacenado en la caché caduca y es eliminado para ser sustituido por otro DataSet cargado desde el fichero XML con los datos actualizados.

```
private void createDataGrid()  
{  
    DataSet dgrid;  
    dGrid = (DataSet)Cache["DataGrid"];  
    if (dgrid == null)  
    {  
        dgrid = getDataSet();  
        CacheDependency fileDepends = new CacheDependency(Server.MapPath("datos.xml");  
        Cache.Insert("DataGrid", dgrid, fileDepends);  
        Response.Write("Datos desde la base de datos");  
    }  
    else  
    {  
        Response.Write("Datos recuperados de la caché");  
    }  
    GridView1.DataSource = dgrid.Tables[0];  
    GridView1.DataBind();  
}  
  
private DataSet getDataSet()  
{  
    // Código de recuperación de datos del fichero XML "datos.xml".  
}
```

## OPTIMIZACION Y SERVICIOS WEB

El objeto de la clase *CacheDependency* se encarga de establecer una dependencia entre el datos almacenado en la caché ( el DataSet dgrid ), y el fichero XML cuya ruta en el servidor de datos se indica empleando el método MapPath del objeto Server. ( "datos.xml").

### 3.2.2.- Objeto dependiente de otro objeto en la caché

Un objeto almacenado en la caché puede depender de uno o más objetos almacenados también en la caché. El objeto caducará y será eliminado de la cache si alguno de los objetos de los que depende sufre algún cambio. Estos cambios incluyen la modificación de su valor o la eliminación del mismo.

En el ejemplo siguiente se muestra la inserción del objeto dset de los ejemplos anterior en la caché de objetos, pero añadiendo una dependencia con otros tres objetos almacenados en la caché con los valores clave "dato0", "dato1" y "dato2".

```
private void createDataGrid()
{
    DataSet dgrid;
    dGrid = (DataSet)Cache["DataGrid"];
    if (dgrid == null)
    {
        dgrid = getDataSet();
        String[] cacheDependsArray = { "dato0", "dato1", "dato2" };
        String[] fileDependsArray = { Server.MapPath("datos.xml") };
        CacheDependency Depends = new CacheDependency( fileDependsArray, cacheDependsArray );
        Cache.Insert("DataGrid", dgrid, Depends );
        Response.Write("Datos desde la base de datos");
    }
    else
    {
        Response.Write("Datos recuperados de la caché");
    }
    GridView1.DataSource = dgrid.Tables[0];
    GridView1.DataBind();
}
```

En este caso se emplea una sobrecarga diferente del constructor de la clase *CacheDependency*. El constructor exige dos arrays de strings, el primero con las rutas de ficheros, y la segunda con claves de valores.

Si los valores de los objetos en la caché bajos las claves "dato0", "dato1", o "dato2" cambian o son los eliminados; el objeto "DataGrid" caducaría en el momento y resultaría eliminado automáticamente de la caché de objetos.

### 3.2.3.- Objeto dependiente del tiempo

Los objetos almacenados en la caché pueden recibir una dependencia basada en el transcurso del tiempo. Los parámetros que controlan el tiempo son *AbsoluteExpiration* y *SlidingExpiration*; ambos presentes en los métodos Add e Insert del objeto Cache.

El parámetro *AbsoluteExpiration* es de tipo DateTime y define un tiempo de caducidad basado en una fecha concreta, por ejemplo: el 31 de Diciembre del 2009.

```
DateTime expireDate = new DateTime( 2009, 12, 31, 0, 0, 0 );
Cache.Insert("DataGrid", dgrid, null, expireDate, Cache.NoSlidingExpiration );
```

## OPTIMIZACION Y SERVICIOS WEB

Por otro lado, también es posible definir un tiempo de caducidad basado en el momento del ingreso del dato en la caché. Por ejemplo; transcurridos 30 minutos tras la inserción.

```
Cache.Insert("DataGrid", dgrid, null, DateTime.Now.AddMinutes(30),  
Cache.NoSlidingExpiration );
```

El parámetro *SlidingExpiration* es de tipo *TimeSpan*. Este parámetro especifica un intervalo de tiempo entre el momento en el que se accede por última vez a un objeto almacenado en la caché y cuando caduca. Por ejemplo, se puede insertar un objeto que caduca si no se accede a él durante un periodo superior a 30 segundos. Si antes de que transcurra ese lapso de tiempo se accede a la información, el reloj de caducidad se reinicia y el objeto continua en la caché otros 30 segundos antes de caducar a menos que vuelve a ser solicitado.

```
Cache.Insert("DataGrid", dgrid, null, Cache.NoAbsoluteExpiration,  
TimeSpan.FromSeconds(30));
```

### 3.2.4.- Control de prioridad de eliminacion de objetos obsoletos.

Los objetos almacenados en la caché de datos pueden ser eliminados si la memoria del servidor es inferior a un determinado límite. La eliminación de los objetos presente en la caché de datos en estas circunstancias se gestiona en función del valor del parametro *priority*.

Aquellos objetos con niveles de prioridad más bajos serán los primeros eliminarse en caso de necesidad de memoria. Los valores admitidos para este parámetro son los pertenecientes al enumerado *CacheItemPriority*:

<b>Low</b>	Los elementos de la memoria caché con este nivel de prioridad son los que más posibilidades tienen de ser eliminados de la memoria caché cuando el servidor libera la memoria del sistema.
<b>BelowNormal</b>	Los elementos de la memoria caché con este nivel de prioridad tienen más posibilidad de ser eliminados cuando el servidor libera la memoria del sistema que aquéllos que tengan asignada una prioridad <b>Normal</b> .
<b>Normal</b>	Los elementos de la memoria caché con este nivel de prioridad podrán ser eliminados de la memoria caché cuando el servidor libere la memoria del sistema sólo después de eliminarse los elementos con la prioridad <b>Low</b> o <b>BelowNormal</b> . Éste es el valor predeterminado.
<b>AboveNormal</b>	Los elementos de la memoria caché con este nivel de prioridad tienen menos posibilidades de ser eliminados cuando el servidor libera la memoria del sistema que aquéllos que tengan asignada una prioridad <b>Normal</b> .
<b>High</b>	Los elementos de la memoria caché con este nivel de prioridad son los que menos posibilidades tienen de ser eliminados de la memoria caché cuando el servidor libera la memoria del sistema.
<b>NotRemovable</b>	Los elementos de la memoria caché con este nivel de prioridad no se eliminarán de la memoria caché cuando el servidor libere la memoria del sistema. Sin embargo, los elementos con este nivel de prioridad se quitan junto con otros elementos en función de la fecha de caducidad absoluta o variable del elemento.
<b>Default</b>	El valor predeterminado para la prioridad de un elemento de la memoria caché es <b>Normal</b> .

### 3.2.5.- Detección de caducidad.

Cuando un elemento es eliminado de la caché se produce un evento *CacheItemRemoveCallBack*. Este evento puede ser aprovechado para implementar algún código que deba ejecutarse al caducar un determinado elemento de la caché.

El parámetro *CacheItemRemovedCallBack* hace referencia a un delegado que determina los parámetros de entrada y salida del método a invocar cuando se produzca el evento de caducidad.

## Metadatos

### 4.1.- CONCEPTO DE METADATOS

La etiqueta <META> de HTML permite introducir en el código de la página información sobre su contenido, el autor, definir palabras claves... etc. Las palabras clave ( keywords ) son normalmente empleadas por los motores de búsqueda de sitios como Google, Yahoo.. etc; para crear sus índices. Los Metas más importantes son los “*Meta Keywords*” y los “*Meta Description*”.

Los “*Meta Keywords*” se emplean para incluir palabras que describan el contenido de la página cara a los motores de búsqueda. Los “*Meta Description*” contienen la descripción del contenido de la página.

- **Meta Keywords:**

<META NAME="keywords" CONTENT="frutas, manzana, naranjas, pera">.

Los "keywords" son palabras claves que describen el contenido de tu página web.

- **Meta Description:**

<META NAME="description" CONTENT="Ventas de frutas frescas a través de internet.">.

La descripción de tu página es muy importante. Debes saber que la mayoría de los motores de búsqueda solo toman los primeros 150 caracteres de la descripción. Así que debes colocar una descripción bastante concisa y efectiva.

Existen muchos otros tipos de Metas con otros propósitos:

- **Meta Redirect.**

<META HTTP-EQUIV="refresh" CONTENT="10; url=http://www.ecodig.com">

Con este meta podemos redireccionar al usuario de una pagina a otra. Por ejemplo: En esta META de arriba, al cabo de 10 segundos el usuario será trasladado a [www.ecodig.com](http://www.ecodig.com).

- **Meta Refresh**

<META HTTP-EQUIV="refresh" CONTENT="600">

Recarga la página periódicamente. El número 600 indica los segundos que se tardará en recargar la página.

- **Meta Window-target**

<META HTTP-EQUIV="window-target" CONTENT="\_top">

Se usa para mantener la página web fuera del marco (frame).

- **Meta Author**

<META NAME="Author" CONTENT="Jaime Olmo ">

- **Meta Date**

<META NAME="Date" CONTENT="May 17, 2002">

## OPTIMIZACION Y SERVICIOS WEB

- **Meta Copyright**

```
<META NAME="Copyright" CONTENT="2000-2003 Ecodig. Todos los derechos reservados.">
```

- **Meta Expiration**

```
<META HTTP-EQUIV="expires" CONTENT="thu,31 DEC 2002 00:04:00 EST">
```

El navegador carga la página directamente del caché. Una vez expire la fecha el navegador carga la página desde el servidor.

```
<META HTTP-EQUIV="expires" CONTENT="0">
```

Recarga la página web directamente al servidor.

- **META Cache-Control**

```
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
```

Evita que la página web sea cargada al cache por el servidor o el navegador.

- **META Content-Type**

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
```

- **META Content-Style-Type**

```
<META HTTP-EQUIV="Content-Style-Type" CONTENT="text/css">
```

- **META Robots:**

Los "Robots" son programas usados por los motores de búsqueda para indexar las páginas de un sitio web. Si deseas que todas las páginas sean indexadas utiliza el siguiente meta:

```
<META NAME="robots" CONTENT="all">
```

Pero si no lo deseas, puedes:

```
<META NAME="robots" CONTENT="noindex,nofollow">
```

No añades la página al motor de búsqueda. También puedes usar

```
<META NAME="robots" CONTENT="none"> )
```

No añades la página inicial pero el robot continúa el orden de los enlaces de tu sitio.

```
<META NAME="robots" CONTENT="index,nofollow">
```

Solo indexa la página inicial. Esta es la opción por defecto y la más utilizada.

```
<META NAME="robots" CONTENT="index,follow">
```

# OPTIMIZACION Y SERVICIOS WEB

En ASP.NET los Metas son representados por objetos de la clase *HtmlMeta* definida en el espacio de nombres *System.Web.UI.HTMLControls*. Los clase *HtmlMeta* define dos propiedades: *Name* y *Content*, que describen el tipo de Meta que va a generar y el valor contenido.

```
HtmlMeta keys = new HtmlMeta();
keys.Name = "keywords";
keys.Content = "simpsons, krosty, payaso, mou";
```

Para que los objetos Meta aparezcan en la página resultado deben añadirse a la colección de elementos de la cabecera de la página. La cabecera de la página es un objeto de la clase ***HtmlHead*** accesible a través de la propiedad *Header* de la página.

```
Page.Header.Controls.Add(keys);
```

Código de página de ejemplo:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="Default6" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
    </div>
  </form>
</body>
</html>
```

Código en el lado: Evento Load de la página.

```
protected void Page_Load(object sender, EventArgs e)
{
  //creamos objetos para las meta tags
  HtmlMeta keys = new HtmlMeta();
  HtmlMeta description = new HtmlMeta();

  keys.Name = "keywords";
  keys.Content = "simpsons, krosty, payaso, mou";

  description.Name = "Description";
  description.Content = "Un sitio de los simpson muy chiquito";

  //Agregamos los objetos metatags a la colección de encabezados
  this.Header.Controls.Add(keys);
  this.Header.Controls.Add(description);

  //Cambiamos el título de la pagina
  this.Title = "Simpsons";
}
```

La página HTML resultante quedaría así:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Simpsons</title>
<meta name="keywords" content="simpsons, krosty, payaso, mou" />
<meta name="Description" content="Un sitio de los simpson muy chiquito" />
</head>
<body>
...
</body>
</html>
```

# OPTIMIZACION Y SERVICIOS WEB

## 4.2.- GENERACION DINAMICA DE METADATOS

Normalmente son las páginas maestras las que contienen el código HTML correspondiente a la cabecera *<head>* incluyendo las etiquetas del título de la página *<title>*, los scripts y uno o más campos de metadatos. *<meta>*. Sin embargo, existen ocasiones en las que puede ser necesario reemplazar estos elementos por otros contenidos en las páginas de contenido.

Un ejemplo es el campo de título, que a pesar de estar definido en el código de diseño de la página maestra puede depender de la página contenido. En este caso concreto, la página contenido puede reemplazar el título de la cabecera de la página maestra incluyendo el título deseado en el atributo *title* de su directiva *Page*.

```
<%@ Page Language="VB" MasterPageFile="~/Master1.master"
      AutoEventWireup="true" Title="Home"
%>
```

Si se desea añadir scripts o metadatos mediante código desde una página de contenido puede emplearse la propiedad *Header* de la clase *Page*.

La propiedad *Header* hace referencia a un objeto *HttpHeader* (*System.Web.UI.HtmlControls.HtmlHeader*), que encapsula todos los controles que se generan en la cabecera de la página respuesta. La clase *HttpHeader* posee una colección con los controles a mostrar en la cabecera. Esta colección es accesible y modificable a través de la propiedad *Controls*.

Ejemplos:

```
Protected Sub Page_Load(ByVal sender As Object, _
                        ByVal e As System.EventArgs)

    Dim metaTag As New HtmlMeta
    metaTag.HttpEquiv = "Refresh"
    metaTag.Content = "2;URL=http://www.dotnet.com"
    Header.Controls.Add(metaTag)
End Sub
```

También es posible añadir enlace a hojas de estilos ( *.css* ).

```
Protected Sub Page_Load(ByVal sender As Object, _
                        ByVal e As System.EventArgs)

    Dim cssLink As New HtmlLink()
    cssLink.Href = "~/styles.css"
    cssLink.Attributes.Add("rel", "stylesheet")
    cssLink.Attributes.Add("type", "text/css")
    Header.Controls.Add(cssLink)
End Sub
```

Otra forma de modificar la cabecera de la página maestra desde las página de contenido es emplear los controles *ContentPlaceHolder*. Estos controles definen un área de sustitución en el diseño de la página maestra, que es sustituido por el contenido del control *Content* de las página contenido al generarse la página de respuesta.



## OPTIMIZACION Y SERVICIOS WEB

De igual manera que se define un ContentPlaceholder en el cuerpo del diseño de la página maestra; se puede definir otro dentro del código de la cabecera.

```
<head runat="server">
  <title>Untitled Page</title>
  <asp:ContentPlaceholder id="headerPlaceHolder" runat="server" />
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ContentPlaceholder ID="ContentPlaceholder1" runat="server">
      </asp:ContentPlaceholder>
    </div>
    <asp:Label runat="server" ID="PageFooter" Text="Default footer text" />
  </form>
</body>
</html>
```

Con el código indicado arriba, cualquier página de contenido puede añadir controles a la cabecera de la página maestra empleando un control Content conectado al control ContentPlaceholder definido en la cabecera de la página maestra:

```
<asp:Content ID="HeaderContent" runat="server"
  ContentPlaceHolderID="headerPlaceHolder">
  <link rel="stylesheet" type="text/css" href="customstyles.css" />
</asp:Content>

<asp:Content ID="Content1" Runat="Server"
  ContentPlaceHolderID="ContentPlaceholder1" >
  <asp:Label ID="Label1" runat="server" Text="Hello, World"/>
</asp:Content>
```

Recordar que cada control Content queda ligado a un control ContentPlaceholder al coincidir el valor de su atributo ContentPlaceHolderID con el del atributo ID del control ContentPlaceholder correspondiente.

Las páginas de contenido no tienen obligación de poseer un control Content por cada ContentPlaceholder existente en la página maestra. En tal caso, la página maestra mostrará el código por defecto incluido en el interior del ContentPlaceholder.

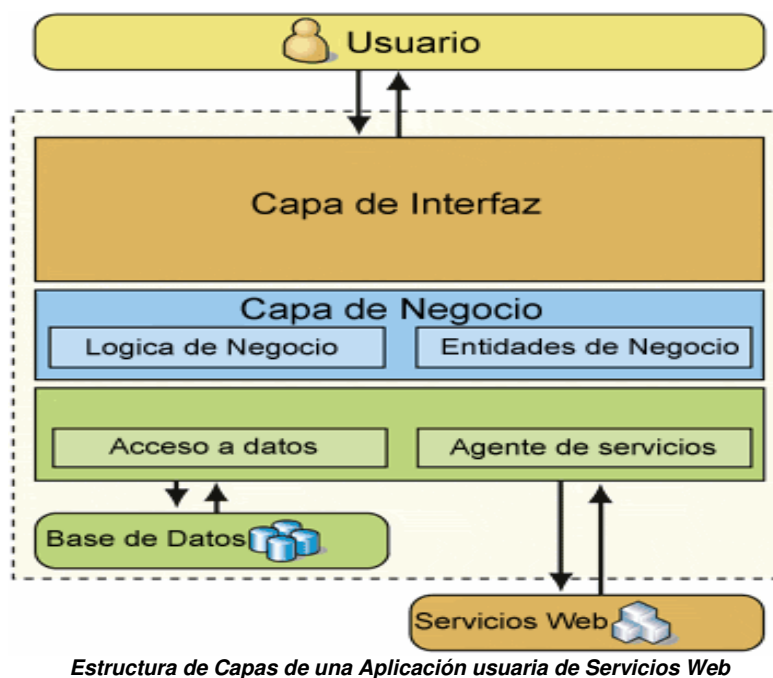
En el Visual Studio 2008, el diseño por defecto de las páginas maestras y las páginas contenido contiene ya los controles ContentPlacerHolder y Content correspondientes para el cuerpo y cabecera del diseño de las páginas.



# Servicios Web

Un servicio Web es básicamente un componente de software accesible remotamente mediante internet que puede ser utilizado por una o varias aplicaciones a la vez.

Los servicios Web se basan en un modelo de aplicación cliente y servidor de servicios Web conectados a internet, o a una red local. El formato de intercambio de información entre ambos se lleva a cabo mediante mensajes descriptivos en SOAP basados a su vez en XML. Esto hace independientes a ambas partes de las diferencias entre sistemas operativos, lenguajes o modelos de objetos empleados en su programación.



*Estructura de Capas de una Aplicación usuaria de Servicios Web*

### 5.1.- PROTOCOLOS

Los servicios Web se basan en varios protocolos para su funcionamiento:

- **HTTP:** ( *HyperText Transfer Protocol* ): Protocolo de transporte empleado extensivamente en internet que permita la comunicación entre el servicio Web y la aplicación cliente.
- **XML** ( *eXtensible Markup Language* ): La información entre la aplicación que emplea un servicio ( *cliente de servicio* ), y el servicio Web se codifica empleando mensajes en XML. El XML es un language multiplataforma que permite la descripción de información de forma estructurada y jerárquica. Es un lenguaje reconocido mundialmente, simple en sintaxis, e independiente del protocolo de transporte empleado.
- **WDSL** ( *Web Services Definition Service* ): Lenguaje basado a su vez en XML que representa un esquema descriptivo de los métodos y parámetros que expone un servicio Web. Un documento WSDL proporciona la información necesaria para

# OPTIMIZACION Y SERVICIOS WEB

permitir que una aplicación cliente pueda utilizar ( *también se dice consumir* ) el servicio Web.

- **SOAP** ( *Simple Object Access Protocol* ): Este protocolo permite que la comunicación entre el código de la aplicación cliente, y el código del servicio Web empleando “mensajes SOAP”. Estos mensajes se codifican a su vez empleando XML.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <!--Optional header information goes here. -->
    <To>Scott</To>
    <From>Suzanne</From>
  </soap:Header>
  <soap:Body>
    <!--Message goes here. -->
    Please pick up some milk on your way home from work.
  </soap:Body>
</soap:Envelope>
```

*Ejemplo de un paquete SOAP*

- **UDDI** ( *Universal Description Discovery & Integration* ): UDDI es un registro público diseñado para almacenar de forma estructurada información sobre empresas y los servicios que éstas ofrecen.

Los protocolos que intervienen en el funcionamiento de los servicios web pueden organizarse en las siguientes categorías según su objetivo:

Discovery <b>UDDI, DISCO</b>	<b>Protocolos de Descubrimiento</b> UDDI
Description <b>WSDL, XML Schema</b>	<b>Descripción</b> WSDL, Esquema XML, Docs
Message Format <b>SOAP</b>	<b>Formato de Mensaje</b> SOAP
Encoding <b>XML</b>	<b>Codificación</b> XML
Transport <b>HTTP, SMTP</b>	<b>Transporte</b> HTTP

## 5.2.- REQUISITOS DE UN SERVICIO WEB

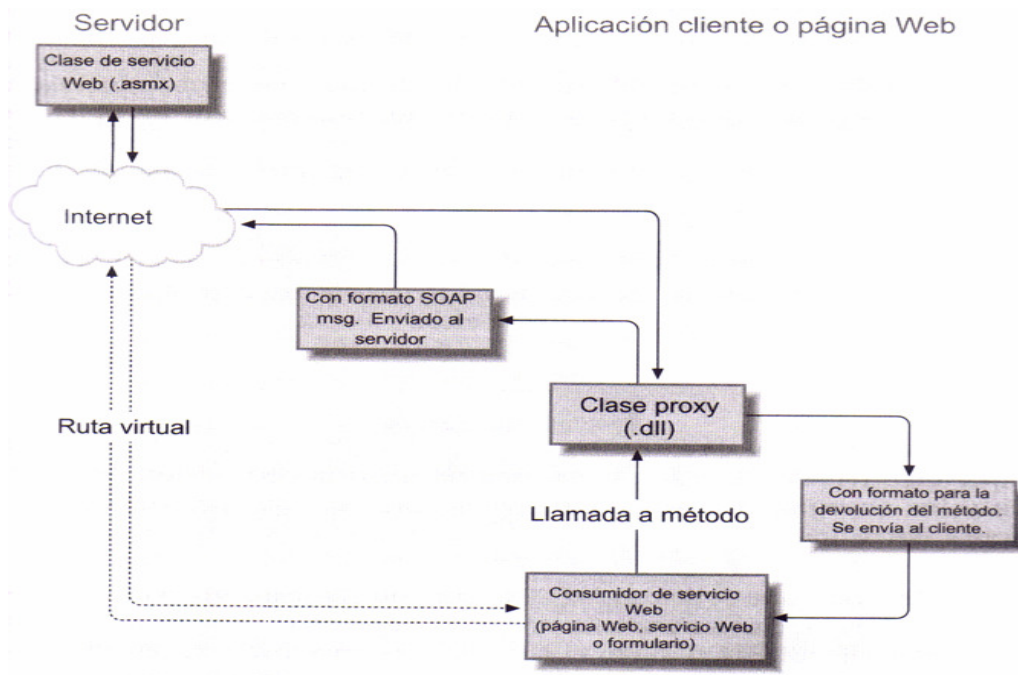
- **Interoperabilidad:** Un servicio Web debe ser independiente de la plataforma, y poder se empleado desde diferentes plataformas y lenguajes a los empleados en su creación.
- **Funcionamiento Remoto:** Un servicio Web debe poder ser invocado remotamente por múltiples aplicaciones clientes al mismo tiempo.
- **Tipado compatible:** Los tipos de datos empleados en los parámetros de los servicios Webs deben ser lo más compatibles posible con los tipos de la mayoría de los lenguajes existentes.
- **Soporte independiente del lenguaje e infraestructura:** La utilización de un servicio Web no debe ir ligada a un lenguaje o infraestructura determinadas. La

# OPTIMIZACION Y SERVICIOS WEB

aplicación usuaria debe poder estar escrita en cualquier lenguaje y el servicio operar bajo los protocolos estandar ( XML, WSDL, SOAP, UDDI ).

## 5.3.- FUNCIONAMIENTO DE SERVICIOS WEB

En la siguiente figura se muestra de forma esquemática la lógica de funcionamiento de un servicio Web en .NET.



La pieza clave es el consumidor de servicio web que es la aplicación que hace la llamada a los métodos del servicio Web. Estas llamadas son transparentes para el código, esto es; la llamada se hace de igual manera que si dichos métodos formasen parte de la misma.

Sin embargo, la invocación de los métodos del servicio Web se reconducen a través de una clase proxy que hace de intermediaria. Su misión es enviar al servicio Web la llamada a un método junto con sus parámetros asociados, y devolver a la aplicación los resultados devueltos. La comunicación de la clase proxy con el servidor Web se hace mediante mensajes SOAP codificados en XML. La solicitud y la respuesta se transmiten empleando el protocolo HTTP.

La aplicación consumidora de un servicio Web puede ser una aplicación de windows, o una aplicación Web escrita en ASP.NET. La clase proxy se compila junto con la aplicación de forma automática mediante el IDE del Visual Studio 2005.

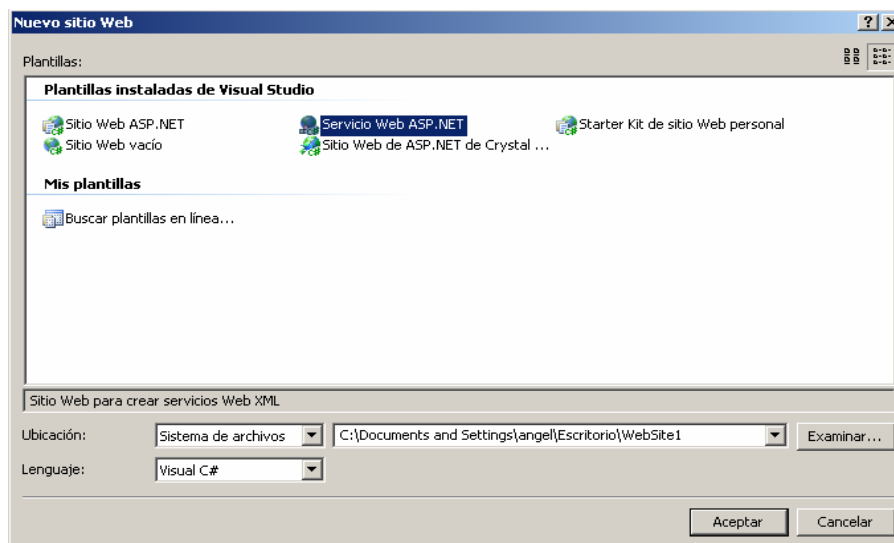
# OPTIMIZACION Y SERVICIOS WEB

## 5.4.- CREACION DE UN SERVICIO WEB

Lo primero que hay que tener claro a la hora de crear un servicio Web es que se crea dentro de un sitio Web. A diferencia de las páginas Web; los servicios Web carecen de interfaz de usuario y de componentes visuales.

Las páginas Web se crean para la interacción con un usuario a través de un navegador Web, pero los servicios Web se crean para servir a otras aplicaciones ofreciendo métodos que éstas pueden invocar remotamente.

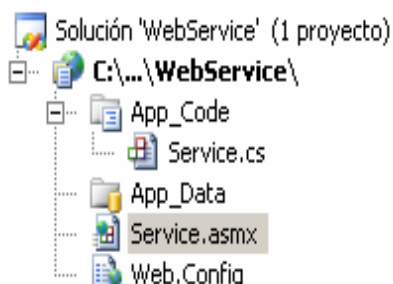
**Archivo → Nuevo → Sitio Web**



- Lenguaje: Visual C#
- Plantilla tipo de proyecto: Servicio Web ASP.NET.
- Ubicación: Sistema de archivos.

Los servicios Web al igual que un proyecto de sitio Web convencional puede generarse en el disco duro del equipo local ( Sistema de archivos ), o bien generarse en un servidor remoto empleando los protocolos FTP o HTTP.

Al crear un sitio Web en Visual Studio 2005 se crea una estructura con los siguientes archivos:



**Service.asmx:** Archivo de acceso al servicio Web para las aplicaciones consumidoras.

**Service.cs:** Archivo de código oculto que contiene el código de los métodos que conforman el servicio Web. Este archivo debe encontrarse obligatoriamente dentro de la carpeta App\_Code.

**Web.Config:** Fichero de configuración del servicio Web.

## OPTIMIZACION Y SERVICIOS WEB

Al crear un proyecto de servicio Web con el Visual Studio 2005, éste se crea con un código predeterminado:

Código predeterminado del archivo de acceso ( .asmx ):

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/Service.cs" Class="Service" %>
```

El código es semejante a la primera línea del archivo de diseño de una página Web ASP.NET. La primera diferencia está en el uso de la directiva *WebService* en vez de *Page*. El atributo *Language* indica el lenguaje en el que está codificado el servicio Web. El atributo *Class* indica el nombre de la clase que implementa el propio servicio Web. El atributo *CodeBehind* especifica la ruta relativa del fichero de código correspondiente.

Código predeterminado del archivo de código ( .cs ):

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {}

    [WebMethod]
    public string HelloWorld() {
        return "Hola a todos";
    }
}
```

El código de un servicio Web consiste en una clase que hereda de la clase *WebService* ( *System.Web.Services.WebService* ), de igual manera que una página de ASP.NET hereda a su vez de la clase *Page* ( *System.Web.UI.Page* ).

La derivación de la clase *WebService* confiere al servicio Web acceso a varios objetos comunes de ASP.NET, tales como *Application*, *Session*, *Context* y *User*. El objeto *User* permite autenticar el acceso al servicio Web, mientras que *Context* facilita información sobre la solicitud HTTP recibida por el servicio Web.

También es importante destacar la presencia de tres atributos insertados en el código: *WebService*, *WebServiceBinding*, y *WebMethod*. Estos atributos poseen una serie de propiedades que permiten configurar tanto el servicio Web, como los métodos que lo componen.

# OPTIMIZACION Y SERVICIOS WEB

## 5.4.1.- Atributos descriptores

- El atributo [WebService]

Este atributo permite agregar información adicional a un servicio Web. Sus propiedades son las siguientes:

- **Description:** Esta propiedad permite asignar un texto descriptivo al servicio Web. Esta información se mostrará en la página de ayuda del servicio Web.
- **Name:** Indica el nombre del servicio Web. Por defecto; el nombre de un servicio Web se corresponde con el nombre de la clase que lo implementa. Este nombre está disponible para cualquier aplicación que vaya a consumir el servicio Web.
- **Namespace:** Permite especificar un espacio de nombres asociado al servicio Web reconocido por el identificador de recursos universal ( URI ) que lo identifica de forma única.

Cada servicio Web es descrito por un documento WSDL definido en XML. Sin embargo; para que una aplicación pueda identificar a un servicio Web de forma inequívoca, es necesario que tenga un espacio de nombres único asociado.

El espacio de nombres por defecto es: <http://tempuri.org>.

Generalmente se define un nuevo espacio de nombres usando un nombre único, como por ejemplo el sitio Web de una empresa. No obstante; este valor no requiere forzosamente una URL bien definida.

- El atributo [WebMethod]

Este atributo permite determinar qué métodos de la clase que conforma el Servicio Web pueden invocarse remotamente. Para ello, los métodos deben declararse públicos y poseer el atributo *WebMethod*.

El atributo *WebMethod* posee una serie de propiedades que permiten configurar el comportamiento del método Web especificado:

- **BufferResponse:** Por defecto ASP.NET almacena en memoria intermedia el retorno de un método antes de enviarlo al cliente. Sin embargo; si el valor de retorno es muy grande puede ser necesario desactivarlo:

```
[webMethod(BufferResponse = false)]
```

- **CacheDuration:** Los servicios Web al igual que las páginas Web pueden almacenar en la caché del servidor los resultados. De este modo; si una aplicación invoca a un método con los mismos parámetros de otra llamada ya realizada; se devolverá la respuesta almacenada en la caché en vez de ejecutar el método otra vez. Esto puede mejorar bastante el rendimiento en el caso de operaciones costosas en tiempo. Este atributo define el número de segundos de validez de los resultados almacenados. Por defecto el valor es 0, lo que equivale a invalidar la caché.

```
[webMethod(CacheDuration = 30)]
```



# OPTIMIZACION Y SERVICIOS WEB

- **Description:** Este atributo permite adjuntar una cadena descriptiva a un método Web. Esta cadena aparecerá en la página de información del servicio Web al acceder al mismo mediante un navegador Web.

```
[WebMethod(Description="xxxxxxxxxxxxxxxxxxxxx")]
```

- **EnableSession:** Permite habilitar el uso del estado de sesión en el método Web. Si el valor es "true" se puede acceder al estado de sesión mediante el objeto Session heredado de WebService.

```
[WebMethod(EnableSession=true)]
public int getNumLlamadas()
{
    if (Session["Counter"] == null)
    {
        Session.Add("Counter", 0);
    } else {
        Session["Counter"] = Convert.ToInt32(Session["Counter"]) + 1;
    }
    return Convert.ToInt32(Session["Counter"]);
}
```

El estado de sesión en los servicios Web funciona exclusivamente mediante Cookies. Si éstas no están permitidas el estado de sesión no funcionará.

- **MessageName:** Los servicios Web no permiten la sobrecarga de métodos. Esta propiedad permite asignar un alias único a cada versión sobrecargada de un método de manera que puedan ser diferenciados remotamente.

```
[WebMethod(MessageName="Suma de enteros")]
public int getSuma(int a, int b) {
    return a + b;
}
[WebMethod(MessageName="Suma de reales")]
public double getSuma(Double a, Double b) {
    return a + b;
}
```

Supongamos que creamos un servicio Web que exponga dos métodos que permiten calcular la suma y resta de dos valores enteros y retornar el resultado correspondiente:

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

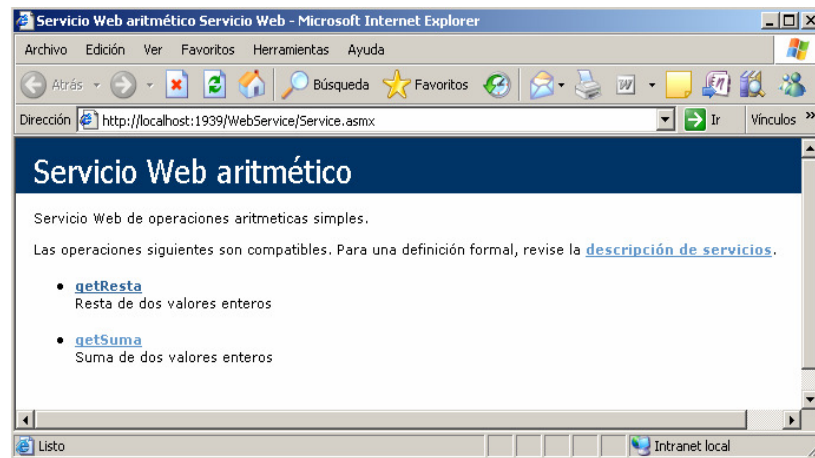
[WebService(Namespace = "http://www.cipsa.net/",
    Description="Servicio Web de operaciones aritmeticas simples.",
    Name="Servicio Web aritmético")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {
        //Eliminar la marca de comentario de la línea siguiente si utiliza los componentes diseñados
        //InitializeComponent();
    }

    [WebMethod(Description="Suma de dos valores enteros")]
    public int getSuma(int a, int b)
    {
        return a + b;
    }

    [WebMethod(Description = "Resta de dos valores enteros")]
    public double getResta(int a, int b)
    {
        return a - b;
    }
}
```

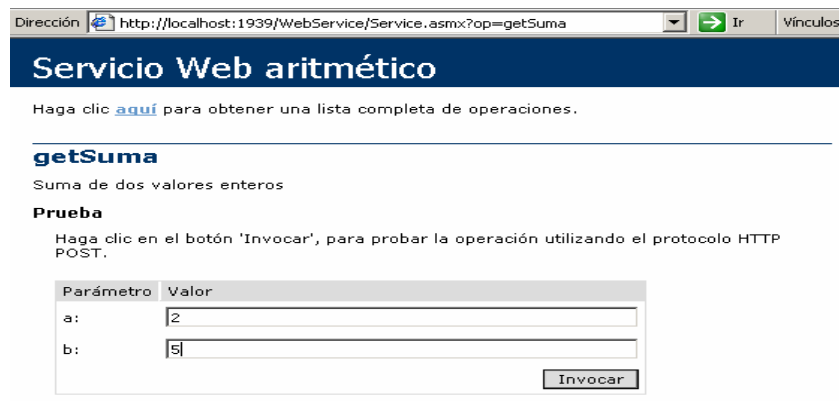
# OPTIMIZACION Y SERVICIOS WEB

Si ahora ejecutamos el proyecto desde el Visual Studio obtendremos la página de ayuda correspondiente al servicio Web. Esta página es la respuesta que ofrece el servicio Web al acceder a él desde un navegador:



Esta página sólo sirve como ayuda para observar los métodos que expone públicamente el servicio Web. Si pulsamos sobre el hipervínculo “descripcion de servicios” se nos mostrará el contenido del archivo descriptor de servicios del servicio Web. ( archivo WSDL ).

Cada uno de los métodos muestra la descripción indicada en la propiedad Description su atributo WebMethod, y se representa como un hipervínculo que permite conocer sus parámetros de entrada y salida:



La página de cada método se compone de un pequeño formulario con tantos campos como parámetros de entrada requiera el método seleccionado. Al pulsar el botón invocar se ejecuta el método y se muestra el código de la respuesta HTTP que devolvería:

```
<?xml version="1.0" encoding="utf-8" ?>
<int xmlns="http://www.cipsa.net/">7</int>
```

Este código es recogido por la clase proxy de la aplicación consumidora del servicio Web, de donde extrae el valor resultado y lo devuelve como retorno al código de la aplicación.

## OPTIMIZACION Y SERVICIOS WEB

### 5.4.2.- Creación de documento de descubrimiento

Tras crear un servicio Web, los programadores que deseen emplearlo en sus aplicaciones necesitarán conocer tres datos:

- 1.- Qué servicios Web están disponibles?
- 2.- Qué métodos ofrecen?
- 3.- Qué parámetros de entrada y salida tienen?

Este proceso se denomina descubrimiento y es un proceso opcional si el desarrollador de la aplicación consumidora conoce de antemano la URL del archivo del servicio Web ( .asmx ). En tal caso, no es necesario hacer el descubrimiento.

El documento de descubrimiento es un archivo (.disco) que contiene referencia a descripciones de los recursos expuestos por el servicio Web. Este archivo puede crearse automáticamente empleando la aplicación “disco” presente en el Visual Studio 2005.

Supongamos que tenemos nuestro servicio Web disponible en la URL:

<http://localhost/services/service.asmx>

Para generar el archivo de recursos deberíamos abrir la aplicación “Símbolo de sistema de Visual Studio 2005”, e introducir el siguiente comando:

```
Disco /out:c:\discovery\ http://localhost/services/service.asmx
```

Este comando generará dentro de la carpeta discovery en el disco duro local los siguientes archivos:

`service.disco` → Archivo de descubrimiento.

`service.wsdl` → Archivo descriptor de recursos del servicio Web.

`results.discomap` → Archivo de referencias a los otros dos archivos.

El archivo de descubrimiento debe estar situado en la raíz del sitio Web que contiene el servicio Web para permitir su descubrimiento y que las aplicaciones consumidoras puedan generar sus clases proxy de comunicación.

# OPTIMIZACION Y SERVICIOS WEB

## 5.5.- CREACION DE APLICACIÓN CONSUMIDORA

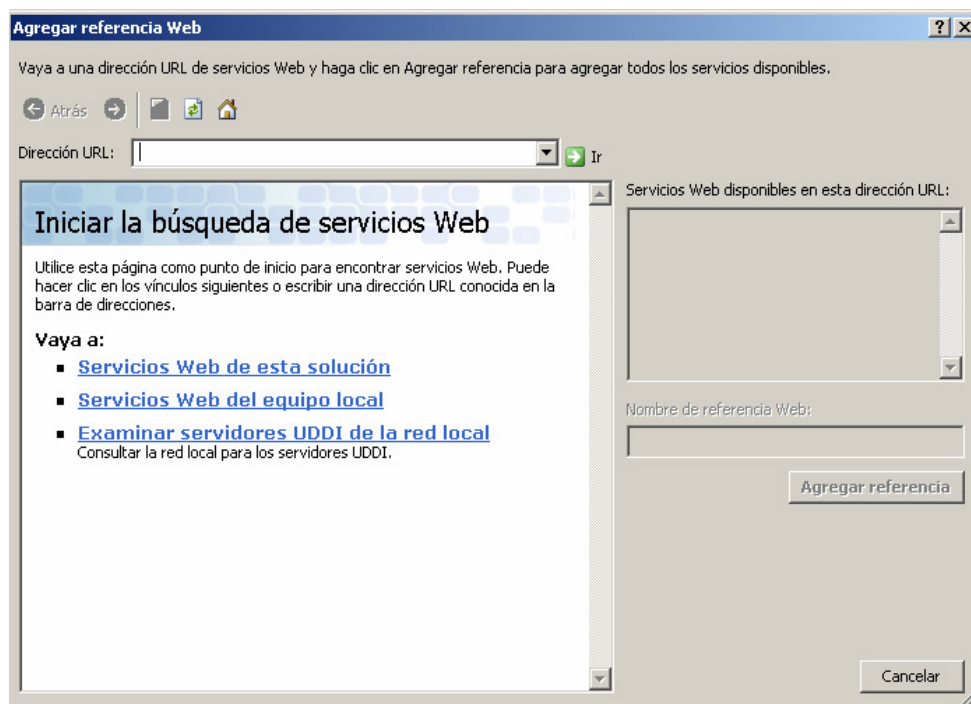
Una aplicación puede emplear servicios Web para su funcionamiento. Se dice entonces que la aplicación es consumidora del servicio Web. El desarrollador debe crear la aplicación consumidora de modo que busque el servicio Web, descubra los métodos que contiene, y genere una clase proxy que haga de intermediaria entre la aplicación y el servicio.

Una vez que se ha generado, compilado y referenciado en la aplicación la clase proxy, la aplicación web invocar los métodos del servicio Web tal como si formasen parte del propio código de la aplicación. Mientras el servicio Web no altere el nombre, número, o tipo de los parámetros de entrada y del de retorno de los métodos expuestos; las clases proxy de las aplicaciones consumidoras seguirán funcionando correctamente. En el caso de añadir nuevos métodos al servicio Web, éstos no podrán ser invocados hasta que no se regenere la clase proxy otra vez.

### 5.5.1.- Consumo de un servicio Web

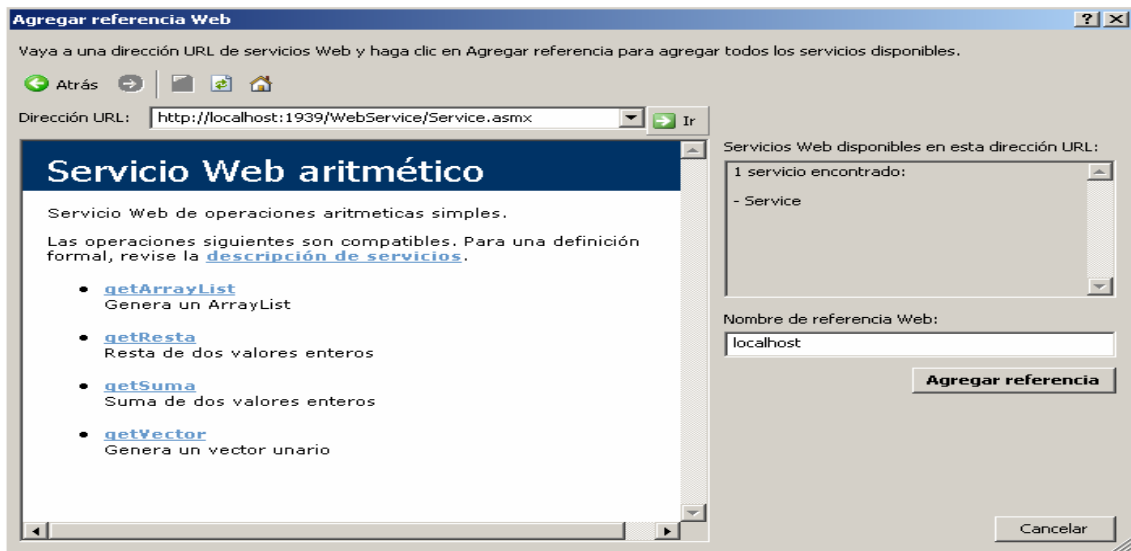
Si deseamos que una aplicación Web consuma un servicio Web, lo primero es establecer una referencia al mismo:

#### ***Sitio Web → Agregar Referencia Web...***

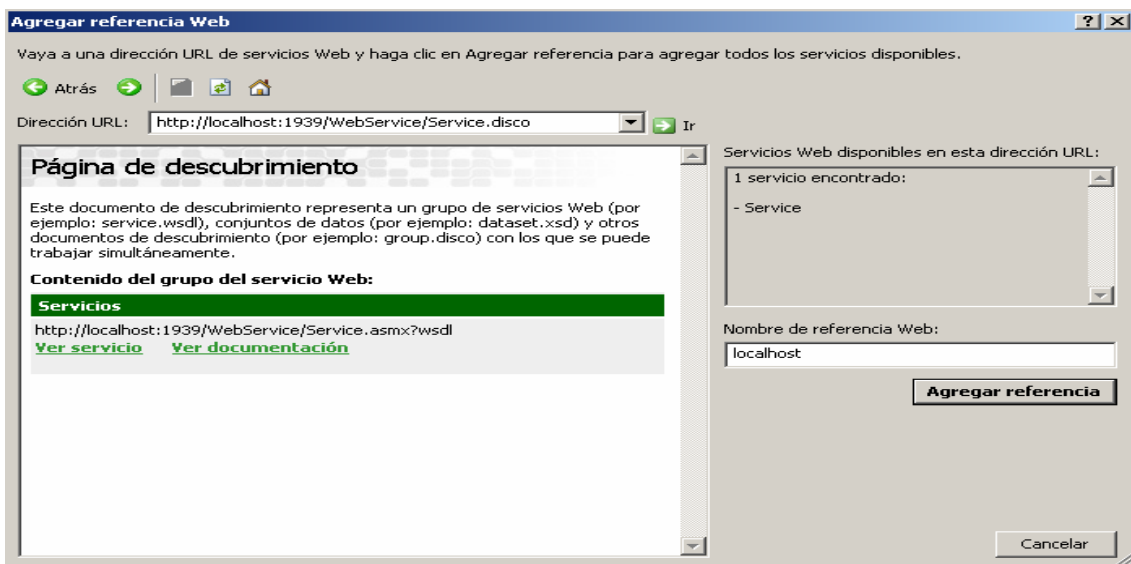


Si el servicio Web al que se desea acceder está en un equipo remoto, debe introducirse la URL del archivo del servicio Web ( .asmx ), o al archivo de descubrimiento ( .disco ), en el campo “Dirección URL” y pulsar el botón IR.

# OPTIMIZACION Y SERVICIOS WEB



*Pantalla de referencia a servicio Web indicando URL de archivo de servicio .asmx*



*Pantalla de referencia a sitio Web indicando URL de archivo de descubrimiento .disco*

Mediante estas pantallas se puede consultar los métodos expuestos por el servicio Web y los parámetros requeridos por cada uno.

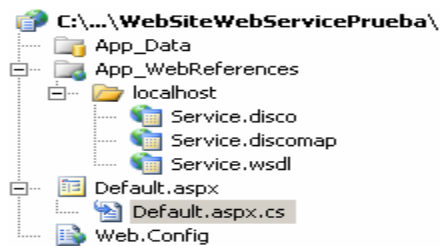
En el caso de que deseemos probar el servicio Web dentro del propio Visual Studio, podemos seguir los siguientes pasos:

- 1.- Se crea un sitio Web de aplicación ASP.NET dentro de la propia solución.
- 2.- Se establece referencia con el sitio Web de la propia solución, seleccionando la opción "Servicios Web de esta solución" en la ventana Agregar Referencia Web.

El valor del campo "Nombre de referencia Web" identifica el espacio de nombres en el que se inscribirán las clases expuestas en el servicio Web y sus métodos correspondientes dentro del código de la aplicación consumidora.

## OPTIMIZACION Y SERVICIOS WEB

Al agregar una referencia Web, se crea automáticamente en el sitio Web de la aplicación la carpeta App\_WebReferences.



En la ventana del explorador de soluciones, podemos apreciar como dentro de la carpeta App\_WebReferences, aparece a su vez una subcarpeta con el nombre de la referencia al servicio Web y los tres archivos descriptores del mismo.

Al agregar la referencia Web se genera también la clase oculta que hace de proxy entre aplicación y servicio. La clase proxy es generada y administrada de forma oculta por el Visual Studio, permitiendo que las clases y métodos expuestos por el servicio Web puedan invocarse como si fueran locales al código de la aplicación. Ej:

```
public partial class _Default : System.Web.UI.Page
{
    localhost.ServicioWebdecalculoaritmético proxy = new localhost.ServicioWebdecalculoaritmético();

    protected void Page_Load(object sender, EventArgs e) { }

    protected void btnSumar_Click(object sender, EventArgs e)
    {
        int valorA, valorB;
        int valorResultado = 0;

        Page.Validate();
        if (Page.IsValid) {
            valorA = Convert.ToInt32(this.txtA.Text);
            valorB = Convert.ToInt32(this.txtB.Text);
            valorResultado = proxy.getSuma(valorA, valorB);
            this.lblResult.Text = valorResultado.ToString();
        }
    }
}
```

En el ejemplo; se crea un objeto proxy de la clase localhost.ServicioWebcalculoaritmético. La clase pertenece al espacio de nombres "localhost" que coincide con el nombre de la referencia al servicio Web. La clase ServicioWebcalculoaritmético se corresponde con la clase Service declarada en el servicio Web. El nombre que adquiere dicha clase en la aplicación consumidora se debe al valor de la propiedad Name del Atributo WebService:

Código asociado al servicio Web:

```
[WebService(Namespace = "http://www.cipsa.net/",
Description="Servicio web de operaciones aritmeticas simples.",
Name="Servicio Web de calculo aritmético")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

    }

    [WebMethod(Description="Suma de dos valores enteros")]
    public int getSuma(int a, int b)
    {
        return a + b;
    }
}
```

Código asociado al cliente Web:

```
localhost.ServicioWebdecalculoaritmético proxy = new localhost.ServicioWebdecalculoaritmético();
valorResultado = proxy.getSuma(valorA, valorB);
```

Si publicamos la aplicación Web consumidora, junto con los ensamblados correspondientes al código de las páginas ASP.NET; Visual Studio genera un ensamblado de nombre App\_WebReferences.dll resultado de la compilación de la clase proxy.

# OPTIMIZACION Y SERVICIOS WEB

## 5.5.2.- Consideraciones sobre tipos de datos

Los servicios Web pueden usar cualquier tipo de dato primitivo compatible con la especificación común de tipos de .NET ( CLR ).

Nombre de la clase	Tipo de dato en VB.NET	Tipo de dato en C#	Descripción
Byte	Byte	Byte	Entero sin signo de 8 bit.
Int16	Short	short	Entero con signo de 16 bit.
Int32	Integer	int	Entero con signo de 32 bit.
Int64	Long	long	Entero con signo de 64 bit.
Single	Single	float	Numero con coma flotante de precisión simple, de 32 bit.
Double	Double	double	Numero con coma flotante de precisión doble, de 64 bit.
Boolean	Boolean	bool	Valor logico
Char	Char	char	Carácter unicode de 16 bit.
Decimal	Decimal	decimal	Valor decimal de 96 bit.
String	String	string	Cadena de caracteres.

- **Uso de Matrices.**

El uso de una matriz de un tipo de elemento primitivo, como por ejemplo; una matriz de enteros, de cadenas, o de números decimales de doble precisión se transmitirá a la aplicación consumidora tal cual:

```
[WebMethod(Description = "Generacion de array de 5 enteros")]
public int[] generarArray()
{
    int[] x = { 1,2,3,4,5 };
    return x;
}
```

El código XML retornado por el servicio web seria:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfInt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://cipsa.com/">
  <int>1</int>
  <int>2</int>
  <int>3</int>
  <int>4</int>
  <int>5</int>
</ArrayOfInt>
```

*Código XML de respuesta de un método que retorna un array de 5 elementos enteros*

El método correspondiente en la clase proxy de la aplicación consumidora mostrará como retorno un array de enteros.

Si el número de elementos es desconocido o dinámico se deberá emplear mejor una colección ArrayList. Si se emplea un ArrayList en un servicio Web, se convertirá en una matriz de objetos en la descripción del servicio Web. El proxy cliente devolverá una matriz de objetos.

```
[WebMethod(Description = "Generador de ArrayList")]
public System.Collections.ArrayList generateArrayList()
{
    ArrayList collection = new ArrayList();
    collection.Add(1);
    collection.Add(2);
    collection.Add(3);
    return collection;
}
```

# OPTIMIZACION Y SERVICIOS WEB

El código XML retornado por el servicio web sería:

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfAnyType xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://cipsa.com/">
  <anyType xsi:type="xsd:int">1</anyType>
  <anyType xsi:type="xsd:int">2</anyType>
  <anyType xsi:type="xsd:int">3</anyType>
</ArrayOfAnyType>
```

*Código XML de respuesta de un ArrayList con cinco elementos.*

En este caso, la clase proxy cliente mostrará el retorno de la función como una matriz de objetos.

- **Uso de clases y estructuras.**

Los servicios Web pueden usar también clases y estructuras definidas por el usuario como parámetros o tipos de retorno, siempre que se cumplan las siguientes condiciones:

- Todas las variables deben ser de tipos primitivos, o matrices de tipos primitivos.
- Todas las variables de la clase deben ser públicas, o poseer propiedades públicas que implementen los accesores get/set.

Sea:

```
public class Service : System.Web.Services.WebService
{
    public class Coordinate
    {
        private int x;           // Atributos privados
        private int y;

        public Coordinate() { }   // Constructor predeterminado

        public coordinate(int _x, int _y) {
            x = _x;
            y = _y;
        }

        public int coordinatex   // Propiedad publica enmascara x
        {
            get { return x; }
            set { x = value; }
        }

        public int coordinatex   // Propiedad publica enmascara y
        {
            get { return y; }
            set { y = value; }
        }
    }

    [WebMethod(Description = "Generador de coordenadas.")]
    public coordinate generateCoordinate(int _x, int _y)
    {
        return new coordinate(_x, _y);
    }
}
```

El código XML retornado por la llamada al método web sería:

```
<?xml version="1.0" encoding="utf-8"?>
<coordinate xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://cipsa.com/">
  <Coordinatex>0</Coordinatex>
  <Coordinatex>9</Coordinatex>
</coordinate>
```

*Código XML de respuesta de una instancia de la clase Coordinate*



# OPTIMIZACION Y SERVICIOS WEB

- **Uso de DataSet**

Un objeto DataSet es el único tipo almacén de datos ADO.NET que puede ser devuelto por un servicio Web. Esto es debido a que ADO.NET representa internamente los DataSet mediante XML.

Sea:

```
[WebMethod(Description = "Generador de DataSets.")]
public System.Data.DataSet generateDataset()
{
    DataSet dset = new DataSet();
    DataTable dtable = new DataTable("CLIENTES");
    dtable.Columns.Add(new DataColumn("ID", typeof(System.Int32)));
    dtable.Columns.Add(new DataColumn("NAME", typeof(System.String)));
    DataRow drow = dtable.NewRow();
    drow["ID"] = 1;
    drow["NAME"] = "roger";
    dtable.Rows.Add(drow);
    drow = dtable.NewRow();
    drow["ID"] = 2;
    drow["NAME"] = "petrov";
    dtable.Rows.Add(drow);
    dset.Tables.Add(dtable);
    dset.AcceptChanges();
    return dset;
}
```

El método Web retorna un DataSet de una sola Tabla CLIENTES, provista de dos columnas "ID" de tipo int, y "NAME" de tipo String, con dos registros de datos en su interior.

El código XML retornado por la llamada al método web sería:

```
<?xml version="1.0" encoding="utf-8"?>
<DataSet xmlns="http://cipsa.com/">
  <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true"
      msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="CLIENTES">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ID" type="xs:int" minOccurs="0" />
                <xs:element name="NAME" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
    <NewDataSet xmlns="">
      <CLIENTES diffgr:id="CLIENTES1" msdata:rowOrder="0">
        <ID>1</ID>
        <NAME>roger</NAME>
      </CLIENTES>
      <CLIENTES diffgr:id="CLIENTES2" msdata:rowOrder="1">
        <ID>2</ID>
        <NAME>petrov</NAME>
      </CLIENTES>
    </NewDataSet>
  </diffgr:diffgram>
</DataSet>
```

*Código XML de respuesta con una instancia de DataSet.*

El proxy cliente devolverá un DataSet idéntico al enviado por el método Web con los mismos datos en su interior.

## OPTIMIZACION Y SERVICIOS WEB

### 5.5.3.- Consumo asíncrono de servicio Web.

Cuando se llama a un método Web de forma síncrona; la aplicación se queda esperando a recibir la respuesta del método invocado. Mientras el método llamado no tarde demasiado en procesar la petición y no haya demasiado retardo en la red para la transmisión y recepción de los mensajes XML; el tiempo de espera en la aplicación no será un problema. Sin embargo, en situaciones en las que el método puede tardar mucho en procesarse, o bien el retardo de red puede ser importante; el retraso puede afectar al rendimiento de la aplicación considerablemente. Esto es más claro en los casos en los que las llamadas a los métodos conllevan un viaje de ida y vuelta a través de internet, donde los retardos de red son considerables.

Una solución es emplear llamadas asíncronas. En este caso, la aplicación invoca al método del servicio Web sin esperar respuesta. Cuando el método web termina, se invoca a un método de “retrollamada” ( *callback* ) de donde la aplicación recupera los datos retornados y continua la ejecución sin espera alguna:

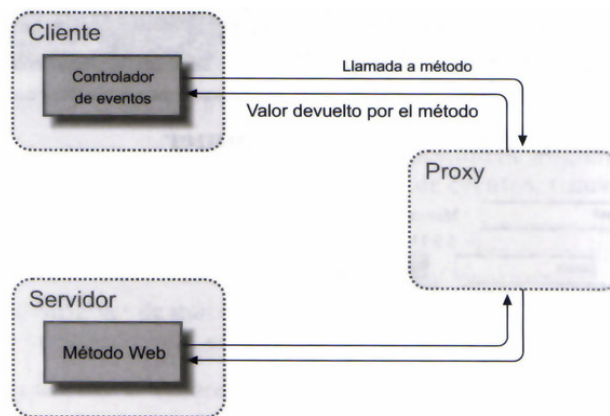


Figura 16.7. Llamadas a método síncronas.

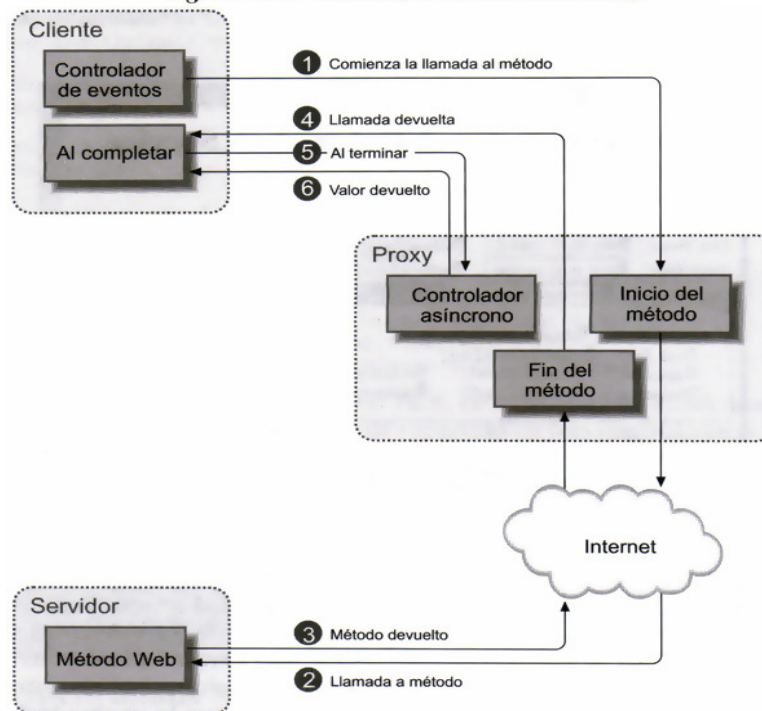


Figura 16.8. Llamadas a método asíncronas.

# OPTIMIZACION Y SERVICIOS WEB

## 5.5.3.1.- Llamadas asíncronas con BeginXXX / EndXXX

Las llamadas asíncronas emplean el patrón típico en .NET asíncrono mediante delegados. En este caso, el delegado de retrollamada empleado para informar a la aplicación del fin de proceso de un método Web está definido como:

```
public delegate void AsyncCallback ( IAsyncResult ar )
```

El parámetro *IAsyncResult*, es el que permite recuperar la información de la llamada originaria, y de ahí; el resultado del método invocado asíncronamente.

Para realizar una llamada asíncrona a un método Web y obtener después las notificación de finalización; debe proporcionarse un método de retrollamada que cumpla con el delegado *AsyncCallback*, e invocar al método mediante *BeginXXX*, donde XXX se corresponde con el nombre del método definido en el servicio web al que deseamos llamar asíncronamente. La clase proxy con el servicio Web contiene un método tipo *BeginXXX* y *EndXXX* por cada método del mismo cuando se le indique que genere métodos de llamada asíncrona.

Supongamos que tenemos el siguiente servicio web:

```
WebService(Namespace = "http://cipsa.com/")
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {}

    [WebMethod(Description="Operacion de suma")]
    public int Sumar(int a, int b)
    {
        return a + b;
    }
}
```

El servicio Web expone un método llamado sumar. Si deseamos invocar a este método de forma asíncrona desde una aplicación, deberíamos emplear el método *BeginSumar* de la clase proxy.

Ejemplo:

```
class Program
{
    // Método de retrollamada de la invocación a BeginSumar.
    public static void onSumarCompleted(IAsyncResult ar)
    {
        ServiceSoapClient proxy = ar.AsyncState as ServiceSoapClient;
        int result = proxy.EndSumar(ar);
        Console.WriteLine("LA SUMA ES {0}", result.ToString());
    }

    static void Main(string[] args)
    {
        // Instancia a proxy de servicio Web
        ServiceSoapClient proxy = new ServiceSoapClient();
        // Llamada asíncrona a método Sumar del servicio Web.
        // El método de retrollamada es onSumar
        proxy.BeginSumar(2, 2, onSumarCompleted, proxy);
        Console.WriteLine("Pulsa una tecla para continuar...");
        Console.ReadLine();
    }
}
```

El método *BeginSumar* recibe como parámetros iniciales los dos correspondientes a la definición del propio método, seguido de una referencia al método de retrollamada, y a la propia instancia del proxy con el servicio Web.

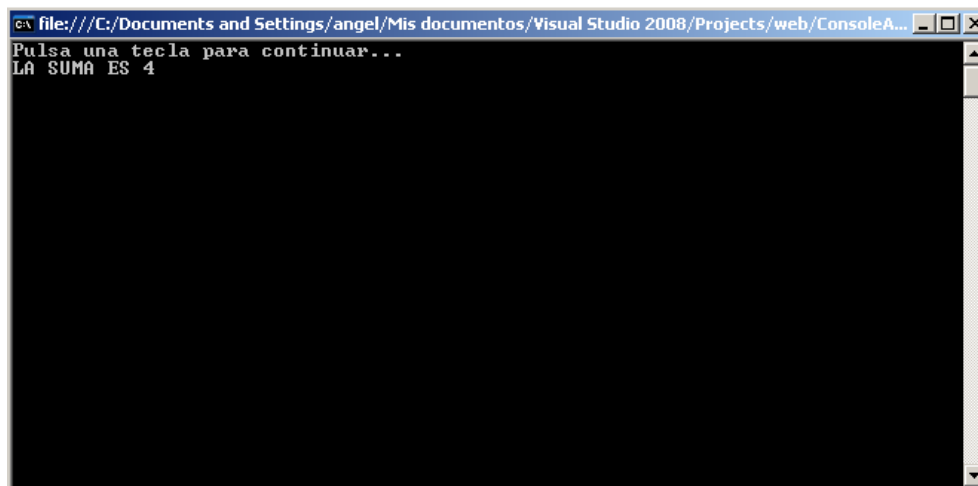
## OPTIMIZACION Y SERVICIOS WEB

El método de retrollamada ( *onSumar* ), debe cumplir con el delegado *AsyncCallback*. Recibe un único parámetro de tipo *IAsyncResult* y retorna void.

```
// Método de retrollamada de la invocación a BeginSumar.
public static void onSumar(IAsyncResult ar)
{
    ServiceSoapClient proxy = ar.AsyncState as ServiceSoapClient;
    int result = proxy.EndSumar(ar);
    Console.WriteLine("LA SUMA ES {0}", result.ToString());
}
```

El parámetro *IAsyncResult* recibido en el método de retrollamada debe ser pasado como parámetro al método de finalización correspondiente en el proxy; *EndXXX*, donde XXX es el nombre del método invocado. Este método retornara el valor correspondiente al retorno del método web invocado. En el caso del ejemplo, el método *EndSumar* retorna un valor de tipo entero que se corresponde con el valor de retorno el método web *Sumar*.

Si ejecutamos el código de ejemplo mostrado; obtendríamos lo siguiente:



### 5.5.3.2.- Llamadas asíncronas con XXXAsync y eventos XXXCompleted .

La clase Proxy también genera métodos XXXXAsync por cada uno de los métodos web expuestos, así como eventos XXXXCompleted para indicar el fin de su proceso.

Para emplear éstos métodos para invocar asíncronamente a un método web, lo primero es asignar un método de respuesta al evento correspondiente. Si, por ejemplo, deseamos invocar asíncronamente al método *Sumar* del servicio web anterior; deberíamos asignar primero un método de respuesta al evento *SumarCompleted*.

El método de respuesta debe cumplir con el delegado del evento *EventHandler<XXXCompletedEventArgs>*, que exige un método de retrollamada sin parámetro de retorno y con dos parámetros de entrada:

```
void proxy_XXXXCompleted(object sender, XXXXCompletedEventArgs e)
```

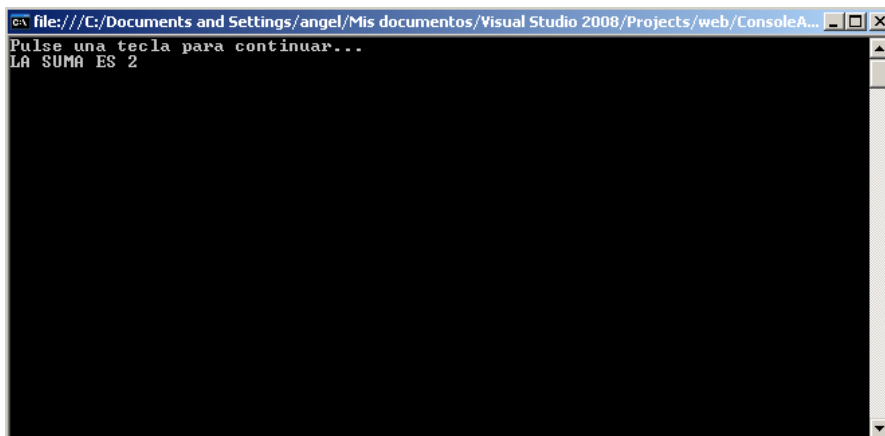
## OPTIMIZACION Y SERVICIOS WEB

Una vez asignado el método de atención al evento correspondiente, se puede invocar al método de la instancia del proxy correspondiente: SumarAsync.

```
// Estilo VS2008
static void Main(string[] args)
{
    ServiceSoapClient proxy = new ServiceSoapClient();
    // Asignación del método proxy_SumarCompleted al evento SumarCompleted de
    // la instancia proxy con el servicio web.
    proxy.SumarCompleted += new
EventHandler<SumarCompletedEventArgs>(proxy_SumarCompleted);
    // Llamada asíncrona al método web
    proxy.SumarAsync(1, 1);
    Console.WriteLine("Pulse una tecla para continuar...");
    Console.ReadLine();
}

// Método de recepción del evento SumarCompleted.
static void proxy_SumarCompleted(object sender, SumarCompletedEventArgs e)
{
    Console.WriteLine("LA SUMA ES {0}", e.Result);
}
```

Para recoger el valor retornado por el método, se debe emplear la propiedad result del parámetro 'e'; que en este caso se corresponde con el valor de retorno de tipo entero del método Sumar. Si ejecutamos el código del ejemplo expuesto obtenemos lo siguiente:

A screenshot of a Windows console window. The title bar shows the file path: file:///C:/Documents and Settings/angel/Mis documentos/Visual Studio 2008/Projects/web/ConsoleA... The console output consists of two lines: "Pulse una tecla para continuar..." followed by "LA SUMA ES 2". The cursor is positioned at the end of the second line.

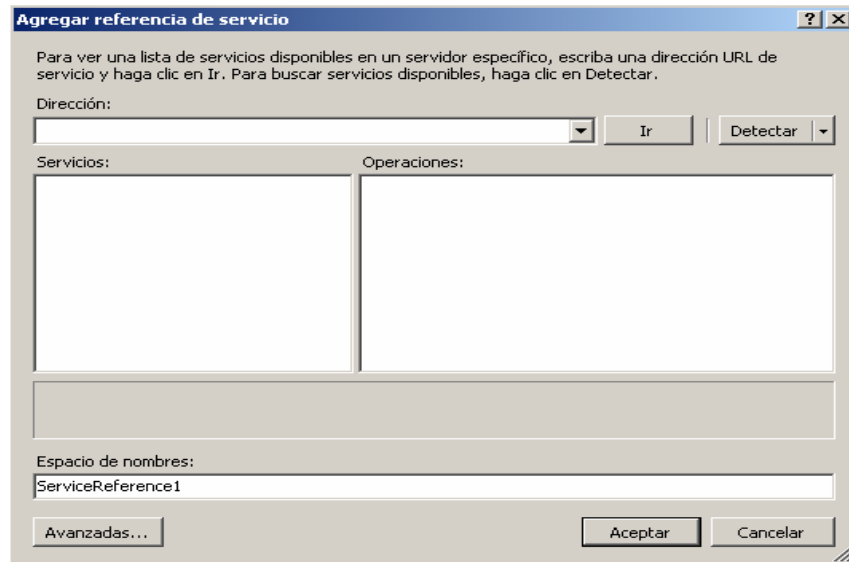
```
file:///C:/Documents and Settings/angel/Mis documentos/Visual Studio 2008/Projects/web/ConsoleA...
Pulse una tecla para continuar...
LA SUMA ES 2
```

# OPTIMIZACION Y SERVICIOS WEB

## 5.6.- AGREGAR REFERENCIAS A SERVICIOS WEB EN VS2008

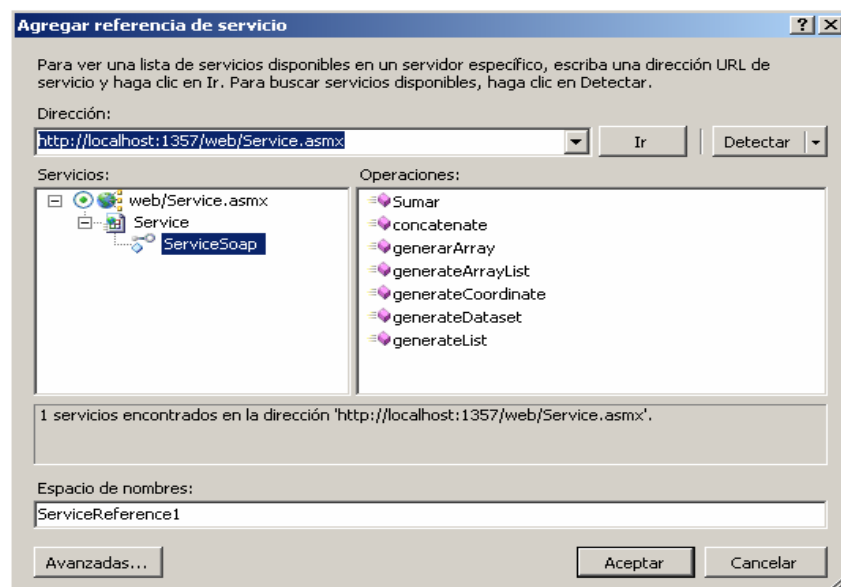
En el Visual Studio 2008, si deseamos que una aplicación consuma un servicio Web, lo primero es establecer una referencia Web al mismo:

**Proyecto → Agregar Referencia de Servicio....**



Si conocemos la dirección del servicio Web podemos ponerla en el campo de texto bajo la etiqueta dirección, y pulsar el botón “Ir”. La dirección puede ser cualquier URL hacia el archivo .asmx, o .disco de un servicio web accesible. Otra opción es pulsar el botón “Detectar”; que permite detectar servicios web dentro de la propia solución. Esto es útil si estamos desarrollando servicio web y aplicación consumidora dentro del entorno de una misma solución.

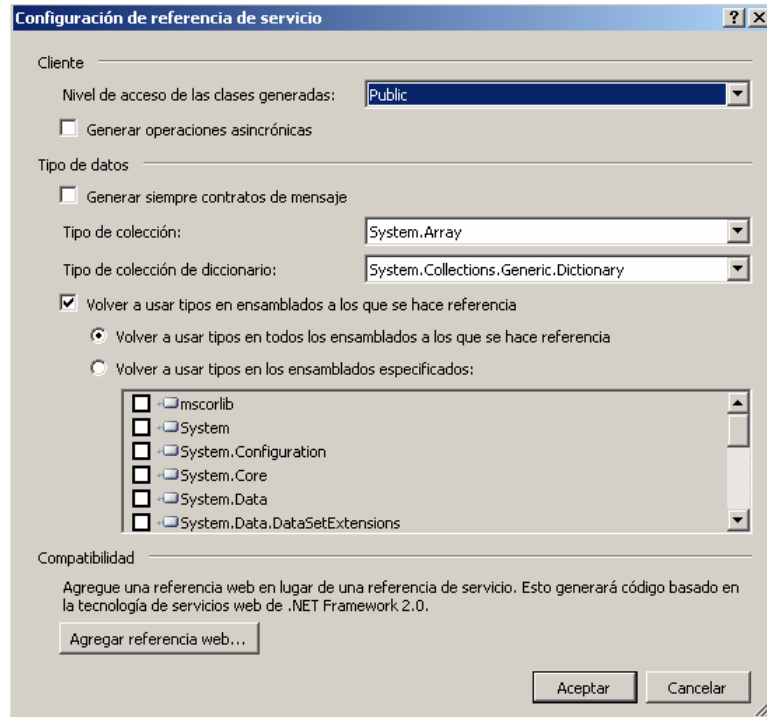
Una vez que el servicio web seleccionado aparece en la ventana de servicios, es posible examinar los métodos que expone fijándonos en la ventana de operaciones:



## OPTIMIZACION Y SERVICIOS WEB

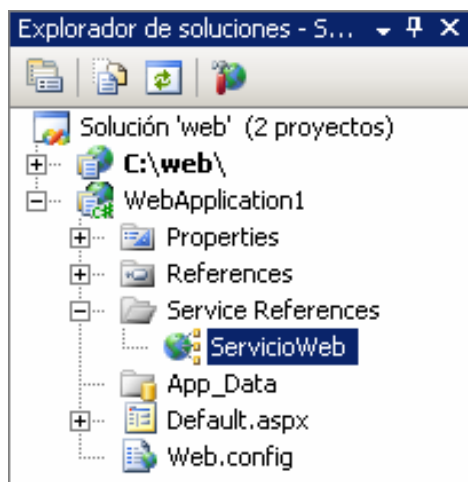
La caja de texto “*Espacio de nombres*” indica el nombre del namespace en el que se creará la clase proxy con el servicio web.

Si pulsamos el botón “*Avanzadas...*”, accedemos al cuadro de dialogo de configuración de referencia de servicio:



En esta ventana se puede seleccionar el nivel de acceso que tendrán las clases secundarias que se generen junto con la clase proxy. La casilla de verificación “*Generar opciones asíncronas*”, es especialmente importante puesto que permite indicar si se deben generar los métodos y delegados necesarios en la clase proxy para permitir la llamada asíncrona a los métodos del servicio web.

Si pulsamos el botón Aceptar se añadirá la referencia al servicio web al proyecto:

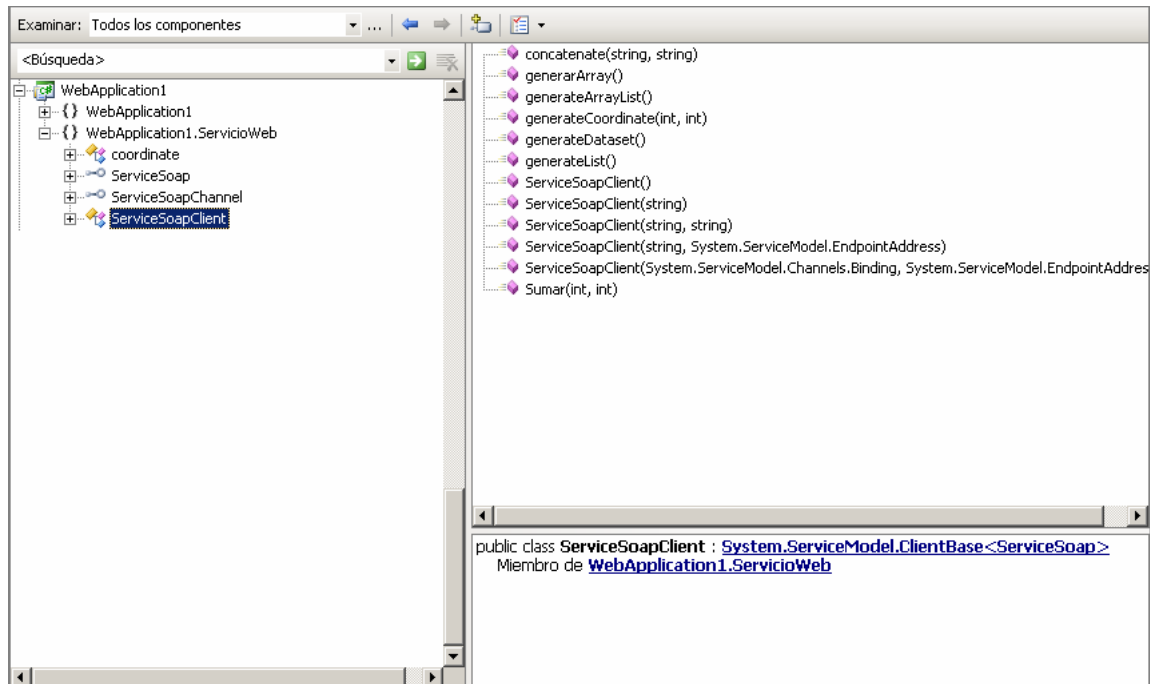


En la ventana del explorador de soluciones del proyecto se puede observar la creación de una nueva carpeta: “*Service References*”.

Esa carpeta contiene la referencia al servicio Web, cuyo nombre coincide con el del namespace indicado al agregar la referencia al proyecto.

Si hacemos doble click sobre la referencia se nos abrirá la ventana del examinador de objetos, que permite observar todos los objetos y métodos a los que tiene acceso ahora la aplicación.

# OPTIMIZACION Y SERVICIOS WEB



Ventana del examinador de objetos con la estructura de la clase proxy recién generada:

Dentro del espacio de nombres indicado: “*ServicioWeb*”, se encuentran todas las clases secundarias definidas en el servicio web, así como la clase proxy principal: “*XXXSoapClient*”; donde *XXX* se corresponde con el nombre del servicio Web.

Si seleccionamos la clase proxy ( *ServiceSoapCliente* en este caso ), podremos examinar todos sus métodos, muchos de los cuales son reflejo de los métodos expuestos por el servicio web al que representa en la aplicación.

Para referirnos a los métodos del servicio Web e invocarlos debemos crear primero una instancia de la clase proxy, y después invocar los métodos deseados sobre ella:

```
ServiceSoapClient proxy = new ServiceSoapClient();  
int resultado = proxy.Sumar(2, 3);
```



# OPTIMIZACION Y SERVICIOS WEB

## EJERCICIOS

1. Crea un servicio Web que implemente las siguientes operaciones aritméticas:

```
int sumar( int a, int b)
int media( int[] valores )
long potencia( int base, int exponente )
int cociente( int dividendo, int divisor )
int resto( int dividendo, int divisor )
```

2. Crea un servicio Web de validaciones que exponga los siguientes métodos.

```
boolean comprobarFecha( string dato )
boolean comprobarDNI( string dato )
boolean comprobarMail( string dato )
```

3. Crea un servicio Web que implemente las siguientes clases: Circulo, Cuadrado y Triángulo. El servicio web debe exponer los siguientes métodos:

```
double calcular_areas
double calcular_perimetro
```

4. Crea un servicio web que exponga los siguientes métodos:

- Una función que reciba un array de 10 valores y retorne la suma de todos sus valores
- Una función que dado un valor entero retorne un array con los diez primeros resultados de su tabla de multiplicar.
- Una función que reciba un array de 30 valores y retorne la suma de todos ellos, y otra que retorne el producto.
- Una función que dado un parámetro de entrada numérico mayor que cero, retorne un array con los “n” primeros términos de la serie de Fibonacci. En esta serie los dos primeros números son 1, y el resto se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21...
- Una función que dada una matriz de valores y un número entero, multiplique por ese número todos los valores del array retornándolo.
- Una función que reciba un array, y retorne otro distinto con los mismos elementos pero ordenados de mayor a menor.
- Una función que dado un valor y un array retorne un valor booleano indicando si el valor se encuentra o no en el array.

5. Crear un servicio web con las siguientes funciones:

```
int getMayor( int[] datos )
int getMenor( int[] datos )
bool existeValor( int valor )
int getValor( int valor )
int[] ordenar( int[] datos )
```

6. Crear un servicio Web de geometría que defina una clase llamada vector. La clase debe poseer únicamente dos atributos de tipo entero: x e y. El servicio web debe además exponer los siguientes métodos de operaciones con vectores.

```
double calcularDistancia( vector a,vector b )
vector sumarVectores( vector a, vector b )
double calcularAngulo( vector a, vector b )
vector calcularModulo( vector a )
vector obtenerUnitario( vector a )
vector obtenerVector( double angulo )
```

## OPTIMIZACION Y SERVICIOS WEB

7. Crea un servicio web de procesamiento de expresiones matemáticas. El servicio debe exponer un método capaz de recibir una expresión matemática en una cadena de texto y retornar su resultado.

La expresión matemática sólo puede contener operadores aritméticos de suma, resta, producto y división. Los valores operados deben ser de tipo entero exclusivamente. La cadena debe terminar con el símbolo '=' para considerarse válida.

El servicio web debe exponer los siguientes métodos.

```
boolean esvalido( string expresion )
```

Examina si la expresión es válida retornando TRUE en caso afirmativo y FALSE en caso contrario. La expresión se considera válida si cumple con el siguiente esquema.

`<valor>[<operador><valor>]...[<operador><valor>] '='`

Los operadores válidos permitidos son '+', '-', 'x', '/'. Los valores deben ser numéricos enteros o decimales empleando la coma como separador decimal. La expresión debe terminar con el carácter '=' para considerarse bien construida.

La expresión no se considera válida si contiene una división por cero.

```
double calcular( string expresion )
```

Este método procesa la expresión matemática retornando el valor resultante de la misma.

8. Crea un servicio web de traducción de palabras del español al ingles. Para simplificar la implementación, el servicio contendrá con una colección interna con 10 palabras en español e ingles. El servicio web debe exponer los siguientes métodos:

```
string trad_Ingles( string palabra_esp )  
string trad_Espa( string palabra_ing )
```

9. Crea un servicio Web encargado de generar números aleatorios. El servicio debe exponer los siguientes métodos.

```
int generarRandom( int max, int min );
```

Genera un valor aleatorio entero entre un máximo y un mínimo.

```
int[] generarRandom( int max, int min, int count );
```

Genera un número determinado de valores aleatorios enteros entre un máximo y un mínimo.

## OPTIMIZACION Y SERVICIOS WEB

10. Crear un servicio web de información horaria para sincronizar diversas aplicaciones distribuidas por el mundo en las siguientes ciudades.

Ciudad	Zona Horaria
Wellington	GMT + 12 h
New-Caledonia	GMT + 11 h
Sydney Eastern	GMT + 10 h
Vladivostok	GMT + 10 h
Seúl	GMT + 9 h
Tokyo	GMT + 9 h
Hongkong	GMT + 8 h
Beijing	GMT + 8 h
Bangkok	GMT + 7 h
Almaty	GMT + 6 h
Calcuta	GMT + 5.5h
Karachi	GMT + 5 h
Mauritius	GMT + 4 h
Bagdad	GMT + 3 h
Moscú	GMT + 3 h
Estambul	GMT + 2 h
Cairo	GMT + 2 h
Berlin	GMT + 1 h
Madrid	GMT + 1 h
París	GMT + 1 h
Roma	GMT + 1 h
Dakar	GMT
Lisboa	GMT
Londres	GMT
Reykjavik	GMT
Azores	GMT - 1 h
Buenos Aires	GMT - 3 h
Río de Janeiro	GMT - 3 h
Caracas	GMT - 4 h
Santiago de Chile	GMT - 4 h
New York	GMT - 5 h
Washington	GMT - 5 h
Chicago	GMT - 6 h
México DF	GMT - 6 h
Denver Mountain	GMT - 7 h
Los Ángeles	GMT - 8 h
San Francisco	GMT - 8 h
Anchorage	GMT - 9 h
Fairbanks	GMT - 9 h
Honolulu	GMT - 10 h
West Samoa	GMT - 11 h

Los métodos que debe implementar el servicio web deben ser los siguientes:

<code>Datetime getDatetimeCentral()</code>	→ Devuelve la fecha y hora en GMT
<code>String getGMT( String ciudad )</code>	→ Devuelve "GMT+X", o "NULL"
<code>DateTime getDatetimeGMT( String ciudad )</code>	→ retorna DateTime o null.

Por compatibilidad con otros sistemas que no poseen la estructura `Datetime`; se ha decidido añadir dos métodos adicionales que sustituyen la instancia de retorno `Datetime` por strings con el siguiente formato:

`dd/mm/aa-hh:mm:ss:msg`

Los métodos deben tener la siguiente signature:

```
string getDatetimeStringCentral()
string getDatetimeStringGMT( String ciudad )
```

## OPTIMIZACION Y SERVICIOS WEB

11. Crear un servicio web de identificación de personas. El servicio Web deberá contener un listado interno de diez personas con sus nombres y contraseñas.

```
bool identificar( string user, string pass )
```

12. Modificar el servicio web de identificación de personas para que admita modificaciones.

```
int alta( string user_admin, string pass_admin, string user, string pass )  
int modificarClave( string user, string pass_viejo, string pass_nuevo )  
int baja( string user_admin, string pass_admin, string user_baja )  
int modificarAdmin( string admin, string pass_viejo, string pass_nuevo )
```

13. Modificar el servicio web para que registre las peticiones de identificación, y permita obtenerlos posteriormente mediante el método:

```
DataSet getRegistro( datetime inicio, datetime fin )
```

# OPTIMIZACION Y SERVICIOS WEB

## PRACTICA

Crea una plataforma web de localización de taxis que se emplea para tener localizados los taxis de una gran ciudad.

El sistema ha sido simplificado dividiendo la ciudad en una rejilla de modo que la posición de cada vehículo se determina mediante dos coordenadas decimales ( X e Y ) comprendidas entre 0 y 10. La información se almacena en una base de datos con las siguientes tablas:

```
TAXIS( matricula, propietario )
POSICIONES( tiempo, matricula, x, y )
```

El sistema se compone de dos partes:

1.- El servicio Web que provee acceso a la base de datos tanto a la aplicación Web como a los terminales instalados en los vehículos exponiendo los siguientes métodos:

- String[] obtenerTaxis():

Devuelve una matriz de String con los datos de todos los taxis en servicio. La información de cada taxi se codifica en cada String con el siguiente formato: **<matricula>#<propietario>**

- String obtenerTaxiMatricula( String propietario ):

Devuelve una cadena con los datos del taxi que se corresponde con la matrícula indicada con el formato: **<matricula>#<propietario>**. En caso de no encontrarse ningún taxi se devuelve la cadena vacía.

- String obtenerTaxiPropietario( String matrícula )

Equivalente al anterior pero buscando por matrícula.

- String[] obtenerHistorico( String matrícula, String inicio, String final )

Devuelve todas las posiciones del vehículo con la matrícula indicada entre las fechas indicadas. Cada posición se codifica en un String mediante el siguiente formato: **<fecha>#<x>#<y>**

- String obtenerUltimaPosicion( String matrícula )

Devuelve la última posición registrada del taxi con la matrícula indicada. La información de la posición se codifica con el formato: **<fecha>#<x>#<y>**

- String obtenerTaxiCercano( float x, float y )

Devuelve únicamente la matrícula correspondiente al taxi cuya última posición es la más cercana al punto de las coordenadas dadas.

# OPTIMIZACION Y SERVICIOS WEB

## Implementación

Primero debe implementarse el servicio Web con todas las funciones indicadas comprobando su correcto funcionamiento.

El servicio Web debe acceder a una base de datos situada en un servidor de Microsoft SQL Server con los siguientes datos:

Origen de datos: Microsoft SQL server (sqlClient)

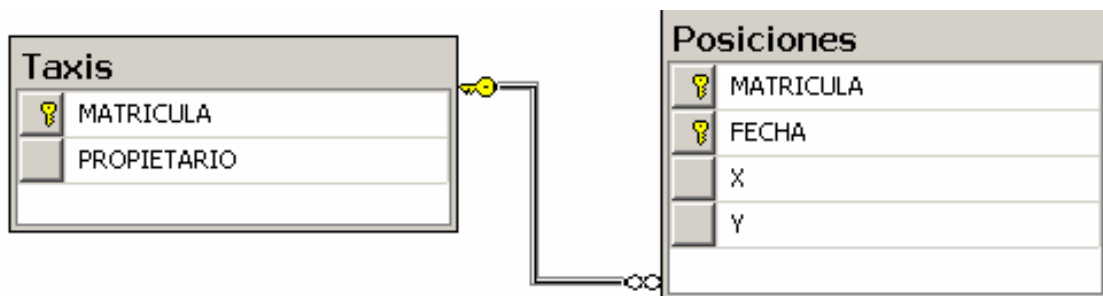
Servidor: WEBSERVER

Usuario: alumno

Clave: alumno

Base de datos: flota.

La estructura de la base de datos es la siguiente:



La base de datos consta además con dos procedimientos almacenados que deben emplearse:

- TAXI\_CERCANO: Devuelve la matrícula del taxi más cercano a la posición de un usuario. Requiere parámetros: px y py de tipo real con las coordenadas del usuario.

Código del procedimiento almacenado:

```
CREATE PROCEDURE [dbo].[TAXI_CERCANO]
-- Add the parameters for the stored procedure here
    @PX AS REAL,
    @PY AS REAL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT TOP 1(MATRICULA), SQRT(POWER(X-@PX, 2) + POWER(Y-@PY, 2 )) AS DISTANCIA
    FROM POSICIONES
    WHERE FECHA = (
        select max(fecha) from posiciones
    )
    ORDER BY DISTANCIA
END
```

## OPTIMIZACION Y SERVICIOS WEB

- **ULTIMA\_POSICION:** Devuelve los valores la matrícula, hora, y coordenadas X e Y de la última posición del taxi indicado. Requiere parámetro matrícula de tipo cadena con la matrícula del taxi.

Código del procedimiento.

```
CREATE PROCEDURE [dbo].[ULTIMA_POSICION]
-- Add the parameters for the stored procedure here
    @MATRICULA NVARCHAR(10)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    SELECT FECHA, X, Y
    FROM POSICIONES
    WHERE MATRICULA like @MATRICULA AND
           FECHA = (
               select TOP 1(FECHA)
               from posiciones
               where matricula like @MATRICULA
               order by FECHA DESC
           )
END
```

Puede comprobarse el funcionamiento del servicio web ya implementado como ejemplo accediendo a la dirección:

<http://webserver:8080/webflotas/>

Una vez terminada la implementación del servicio Web, debe procederse a la implementación de la aplicación Web correspondiente:

Esta debe constar de una página maestra con las siguientes opciones:

- **TAXIS** -> Muestra una subpágina subpágina en la que puede obtenerse la matrícula de un taxi dado el nombre del propietario.
- **HISTORICOS** → Muestra en una subpágina la lista completa de posiciones en las que ha estado un taxi dada su matrícula y una fecha.
- **SERVICIO** → Muestra una subpágina en la que introduciendo una posición en la ciudad determinada por sus coordenadas X e Y devuelve la matrícula del taxi más cercano.

La aplicación Web debe implementarse de modo que acceda a los datos únicamente utilizando una referencia al servicio Web anteriormente indicado. No se permite el acceso directo a la base de datos desde la aplicación Web.

*\* Nota: La base de datos contiene información referente únicamente al mes de Enero de 2011. (1/1/2011 – 31/1/2011 ).*