Aplicaciones multi idioma

Índice de contenido

Variables de sesión....1
Session_Start....1
Session_End (Evento) 1
Global.asax....2
Archivo de configuración....2
Creación de un valor de configuración....3
Utilización del valor de configuración....5
Leer el idioma del navegador....5
Internacionalización con .NET....6
Internacionalización del formulario de trabajo....6
Internacionalización de avisos y descripciones....9
Valores globales....12
Imágenes....13
La Clase ClaseIdioma....15
Incluir la clase....15
Especificar un idioma para un objeto....16

Variables de sesión

ASP.NET proporciona dos eventos que ayudan a administrar las sesiones de usuario: los eventos Session_Start, que se desencadena cuando comienza una nueva sesión, y Session_End, que se desencadena cuando se abandona o expira una sesión.

Session_Start

Puede controlar el evento Session_Start agregando una subrutina denominada Session_Start al archivo Global.asax. La subrutina Session_OnStart se ejecuta al principio de una solicitud si ésta inicia una nueva sesión. Se iniciará una nueva sesión si se crea una solicitud que no contenga ningún valor SessionID o si la propiedad SessionID incluida en la solicitud hace referencia a una sesión expirada.

Puede utilizar el evento Session_Start para inicializar las variables de sesión, así como para realizar el seguimiento de la información relacionada con la sesión.

Session_End (Evento)

Puede controlar el evento Session_End agregando una subrutina denominada Session_OnEnd al archivo Global.asax. La subrutina Session_End se ejecuta cuando se llama al método Abandon o cuando la sesión ha expirado. Una sesión expira cuando el número de minutos especificado por la propiedad Timeout transcurre sin que se haya creado ninguna solicitud para la sesión.

Puede utilizar el evento Session_End para limpiar la información relacionada con la sesión, como la información de un usuario de la que el valor SessionID está realizando el seguimiento en un origen de datos.

Global.asax

Para manejar estos eventos modificaremos los procedimientos **Session_Start** y **Session_End** del archivo **global.asax**.

Si queremos crear un nuevo global.asax pincharemos con el botón derecho en el proyecto, elegiremos Agregar nuevo elemento y de las plantillas instaladas de Visual Studio escogeremos *Clase de Aplicación Global*. Se nos creará un nuevo archivo con los procedimientos que manejan los eventos de sesión y aplicación.

Ejemplo de global.asax que almacena en una variable de aplicación el numero de usuarios conectados.

```
<%@ Application Language="C#" %>

<script runat="server">

 void Application_Start(object sender, EventArgs e)  {
     Application["UsersOnline"] = 0;
 }

 void Session_Start(object sender, EventArgs e) {
    Application.Lock();
    Application["UsersOnline"] = (int)Application["UsersOnline"] + 1;
    Application.UnLock();
 }

 void Session_End(object sender, EventArgs e) {
    Application.Lock();
    Application["UsersOnline"] = (int)Application["UsersOnline"] - 1;
    Application.UnLock();
 }

</script>
```

Para acceder y modificar las variables de sesión utilizaremos de forma análoga a las de aplicación la sintaxis: session["variable"]

Archivo de configuración

Podemos almacenar valores para nuestra aplicación web en el fichero **web.config**. Ahora lo utilizaremos para almacenar pares clave-valor que serán accesibles desde todos los aspx.

La configuración de la aplicación son pares de nombre y valor que representan el texto de valores configurables en la aplicación Web. Utilice la configuración de la aplicación para almacenar información personalizada de configuración de la aplicación, como rutas de acceso a archivos, direcciones URL de servicios Web XML, texto utilizado con frecuencia o cualquier información que desee mantener en una ubicación central y que pueda cambiar fácilmente.

Creación de un valor de configuración

Para ejecutar la Herramienta de Configuración seleccionaremos en el menú **Sitio Web** la opción **Configuración de ASP.NET**. Se nos abrirá el navegador con una página como la de la *ilustración 1*.

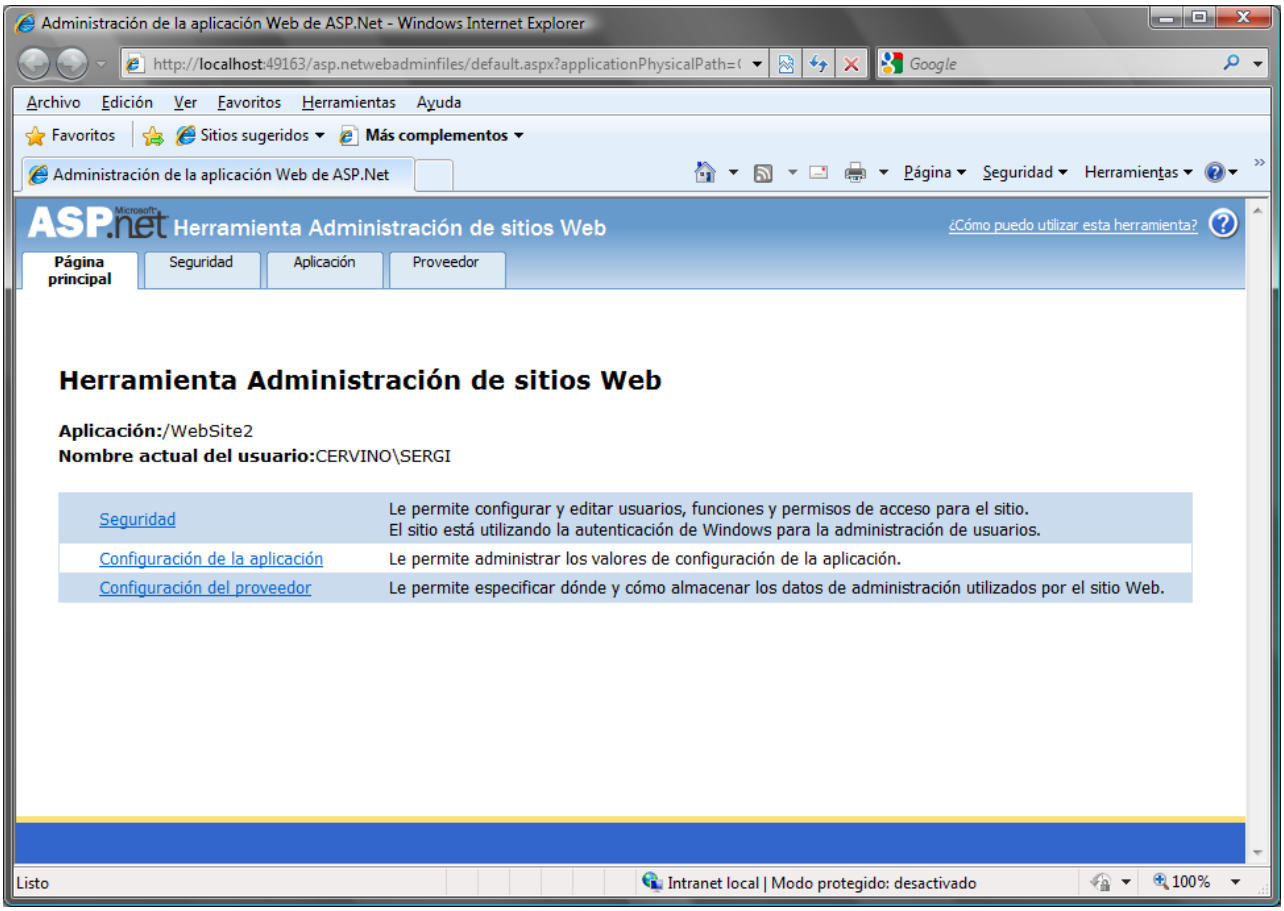

Ilustración 1: Herramienta de administración de sitios Web

Elegimos la pestaña **Aplicación**. (*Ilustración 2*).

Aquí pinchamos la opción **Crear configuración de la aplicación** dentro del cuadro **Configuración de la Aplicación** para agregar una pareja clave-valor o la opción **Administrar configuración de la aplicación** para modificar una existente.

Estableceremos los valores para **Nombre** y **Valor** y pulsaremos el botón **Aceptar**. (*Ilustración 3)*

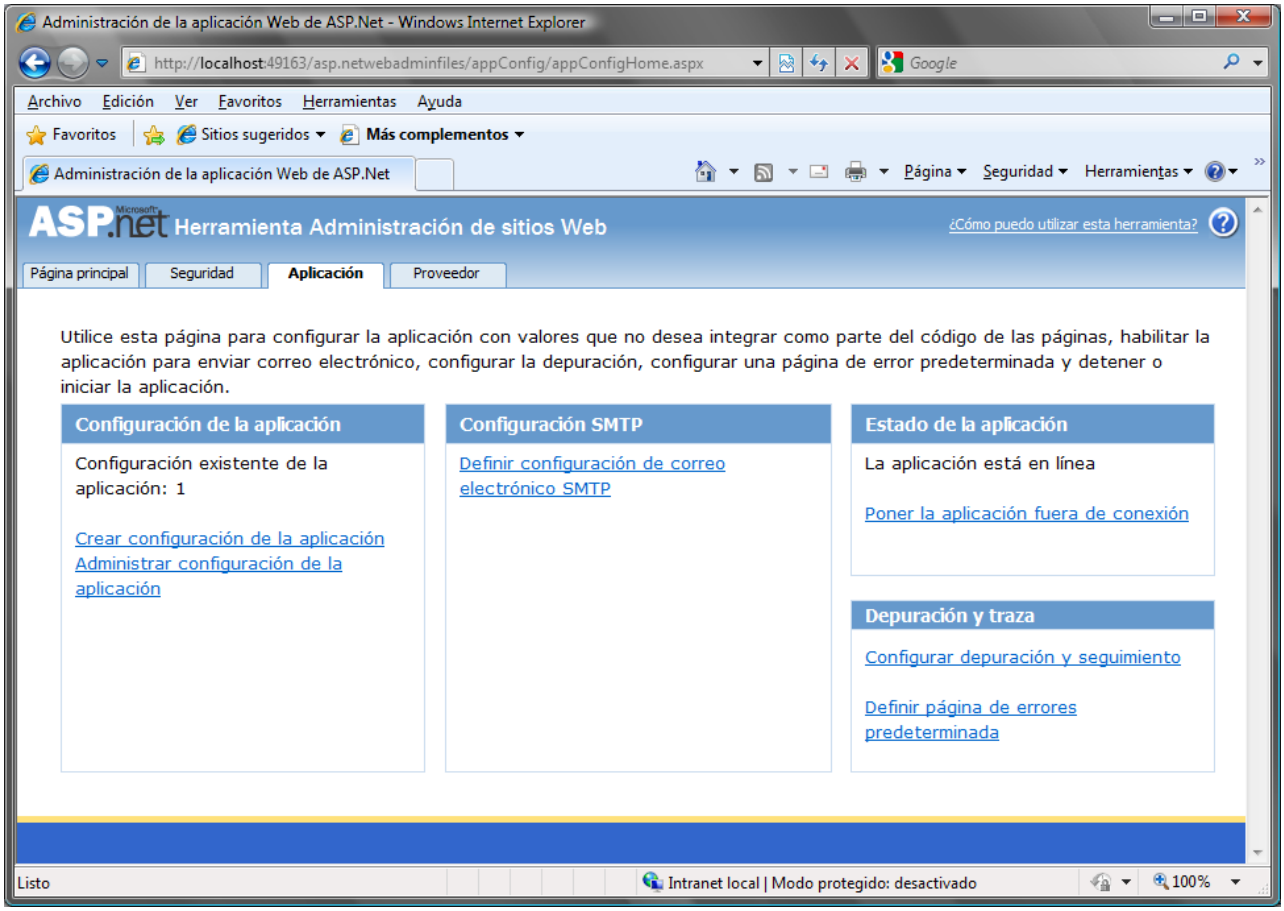

Ilustración 2: Herramienta de administración de sitios Web. Aplicación

En nuestro caso hemos creado un para clave valor con miClave = “Hola Sergi”. Nos aparecerá una ventana de confirmación indicándonos que hemos creado el valor de configuración miClave.

Si queremos ver los valores de configuración creados pulsaremos en **Administrar configuración de la aplicación**. Desde aquí también podemos modificar o eliminar los creados con anterioridad.

Una vez hemos creado los valores que queremos cerramos la Herramienta de Administración de Sitios Web.

Si abrimos desde Visual Studio el fichero web.config veremos que nos ha creado una nueva entrada dentro de <appSettings> para nuestra clave.

```
<appSettings>
  <add key="miClave" value="Hola Sergi" />
</appSettings>
```

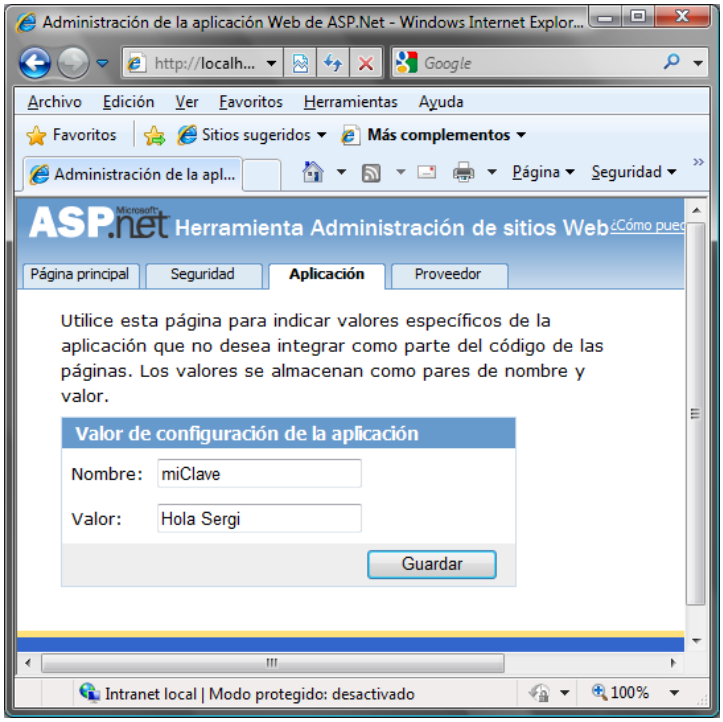

Ilustración 3: Establecer valor de aplicación

Utilización del valor de configuración

Una vez creado el valor, mediante la Herramienta de Administración o modificando el fichero web.config podemos utilizarla en cualquier aspx de nuestra aplicación.

Para ello utilizaremos la clase **ConfigurationManager**, que proporciona acceso a los archivos de configuración. Con el método **AppSettings** accedemos a la sección **appSettings** de web.config.

Por ejemplo para sacar por pantalla el valor de nuestra clave escribiremos:

```
Response.Write (ConfigurationManager.AppSettings["miClave"]);
```

Si queremos cambiar el valor:

```
ConfigurationManager.AppSettings["miClave"] = "Hola Juan";
Response.Write("<br/>" + ConfigurationManager.AppSettings["miClave"]);
```

Leer el idioma del navegador

Para obtener el idioma que tiene configurado el cliente en su navegador utilizaremos la matriz **UserLanguages** del objeto **Request**. Matriz String que contiene los idiomas especificados en el encabezado AcceptLanguage de la solicitud o bien null, si la solicitud de cliente no incluía un encabezado AcceptLanguage

Ejemplo: Si queremos ver todos los idiomas del cliente

```
// Muestra las preferéncias de lenguaje.
string[] types = Request.UserLanguages;
if (types != null) {
   Response.Write("<br />Lenguajes aceptados:");
   foreach (string s in types) {
      Response.Write(String.Concat("<br/>",s));
   }
}
```

Internacionalización con .NET

.NET y en particular Visual Studio nos ofrecen dos métodos muy sencillos para gestionar la internacionalización de las aplicaciones, dependiendo del ámbito que queramos traducir:

- Internacionalización del formulario de trabajo (etiquetas, botones, etc.)
- Textos de avisos o descripciones que pueden aparecer dependiendo de las condiciones de acceso, de las selecciones que hayan hecho, etc.

Internacionalización del formulario de trabajo

Partiendo que disponemos de un formulario muy sencillo Default.aspx con dos etiquetas, dos cajas de texto y un botón (*Ilustración 4*)

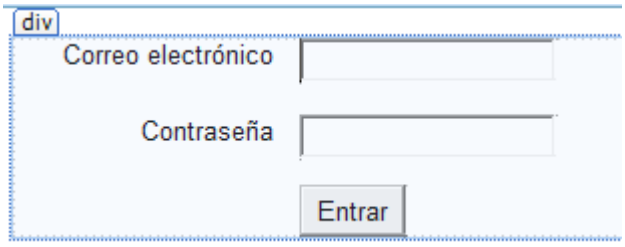

Ilustración 4: Formulario sencillo

Vamos a prepararlo para poder gestionar la internacionalización del formulario.
El primer paso es acceder a la pestaña de **Código** (no se activa la opción de recursos locales desde la pestaña de Diseño) y acceder al menú **Herramientas->Generar recurso local**.

En el Explorador de soluciones podremos ver que aparece dentro de **App_LocalResources** un nuevo fichero llamado Default.aspx.resx. Este archivo de recursos contendrá los textos de la aplicación. (*Ilustración 5*)

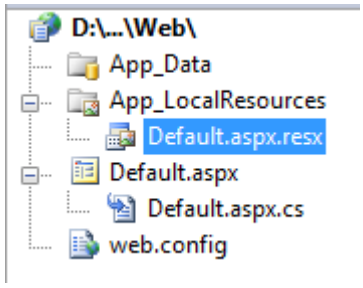

Ilustración 5

Si abrimos el fichero, veremos algo parecido a lo siguiente (*Ilustración 6*)

Nombre	Valor
Button1Resource1.Text	Entrar
Button1Resource1.ToolTip	
lbEmailResource1.Text	Correo electrónico
lbEmailResource1.ToolTip	
lbPasswordResource1.Text	Contraseña
lbPasswordResource1.ToolTip	
PageResource1.Title	
tbEmailResource1.Text	
tbEmailResource1.ToolTip	Correo electrónico de la UA
tbPasswordResource1.Text	
tbPasswordResource1.ToolTip	Contraseña

Ilustración 6: Fichero de recursos

Ahora es el momento de generar un fichero de recursos diferente para cada idioma. Seleccionamos el fichero generado y lo duplicamos (con CTRL+C y CTRL+V) dos veces, uno para cada idioma.*(Ilustración 7).* **El fichero Default.aspx.resx se debe mantener.** *Si no se especifica ningún idioma o si nuestra aplicación no soporta el idioma especificado se utilizará este fichero de recursos.*

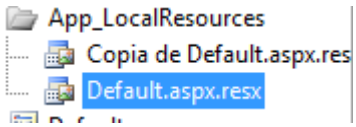

Ilustración 7

Renombramos los ficheros a los idiomas que queramos gestionar, con el siguiente formato: <Nombre del formulario>.<localización>.resx

En nuestro caso las localizaciones son es y ca (para español y valenciano). (*Ilustración 8*)

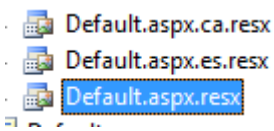

Ilustración 8

Se pueden definir versiones específicas de un idioma, como por ejemplo es-ES o es-AR, aunque lo normal es quedarnos con el idioma general.

Editamos el fichero Default.aspx.ca.resx para ajustar los textos. (*Ilustración 9*)

Nombre	Valor
Button1Resource1.Text	Entra
Button1Resource1.ToolTip	
lbEmailResource1.Text	Adreça electrònica
lbEmailResource1.ToolTip	
lbPasswordResource1.Text	Contrasenya
lbPasswordResource1.ToolTip	
PageResource1.Title	
tbEmailResource1.Text	
tbEmailResource1.ToolTip	Adreça electrònica de la UA
tbPasswordResource1.Text	
tbPasswordResource1.ToolTip	Contrasenya

Ilustración 9: Default.aspx.ca.resx

Ahora vamos a definir el idioma en la cabecera de nuestro formulario.
Incluimos dos referencias en el formulario

```
using System.Globalization;
using System.Threading;
```

Por último tenemos que definir el idioma que deseamos visualizar. Para ello nos apoyamos en un procedimiento que se llama antes de visualizarse el formulario **InitializeCulture**.

Si queremos definir el idioma valenciano, añadiríamos el siguiente código en Default.aspx.cs.

```
 protected override void InitializeCulture()
 {
  Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("ca");
  Thread.CurrentThread.CurrentUICulture = new CultureInfo("ca");

  base.InitializeCulture();
 }
```

El resultado es el siguiente (*Ilustración 10*)

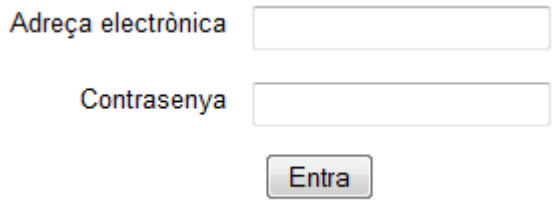

Ilustración 10

Lo normal es que el idioma nos llegue por una variable sesión, o que la leamos de los valores por defecto del navegador (Request.UserLanguages[0]) en caso de no disponer de esta información; tal como hace actualmente la pasarela de la Universidad de Alicante cuando se accede por primera vez.

Al crear un fichero de recursos las cadenas de texto que aparecen se asocian a los controles mediante la inclusión de una propiedad **meta** en la etiqueta.

```
<asp:Image ID="imagenLogo" runat="server"
           meta:resourcekey="imagenLogoResource1" />
```

Internacionalización de avisos y descripciones

1. Si ya disponemos de los recursos resx anteriores, el proceso es muy sencillo, porque simplemente accedemos a ellos y añadimos líneas con Agregar recurso -> Agregar nueva cadena (*Ilustración 11)*

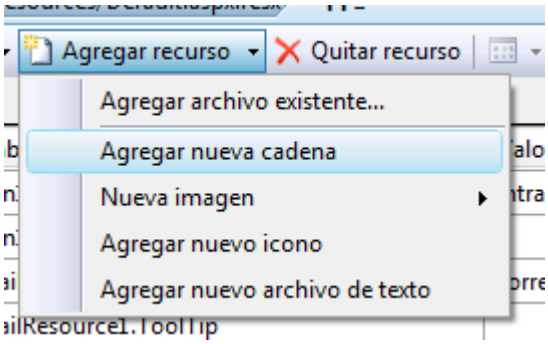

Ilustración 11: Agregar cadena

Añadimos mensaje_identificacion con su descripción en los dos idiomas (*Ilustración 12*)

Ilustración 12

Ahora ya lo podemos usar en cualquier punto de nuestro ASPX. En nuestro caso lo definimos en el evento load de la página (sobre clic sobre el formulario).

```
protected void Page_Load(object sender, EventArgs e)
{
Response.Write("<h1>" + GetLocalResourceObject("mensaje_identificacion") +
"</h1>");
}
```

El resultado sería (*Ilustración 13*)

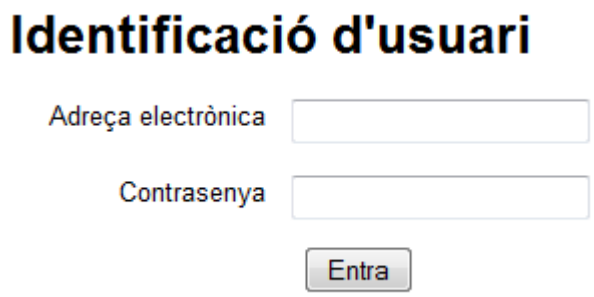

Ilustración 13

2. En caso de no disponer ficheros resx, debemos analizar si tenemos la carpeta App_LocalResources en nuestro proyecto. En caso de no disponerlo lo añadimos pulsando sobre el botón derecho en el Explorador de soluciones y seleccionamos Agregar carpeta ASP.NET -> App_LocalResources (*Ilustración 14*).

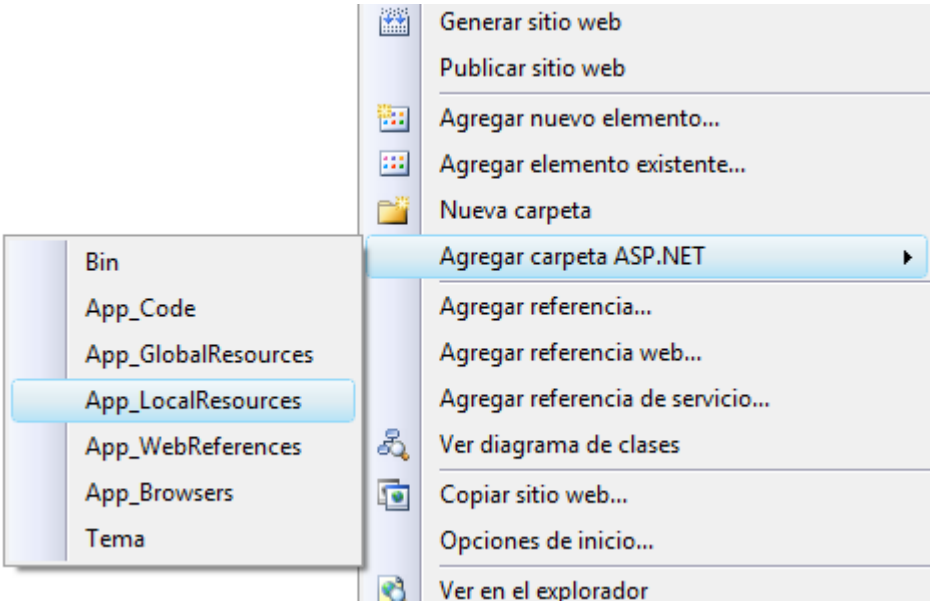

Ilustración 14: Agregar carpeta App_LocalResources

Ahora ya podemos añadir los ficheros de recursos, pulsando el botón derecho sobre App_LocalResources -> Agregar nuevo elemento … (*Ilustración 15*)

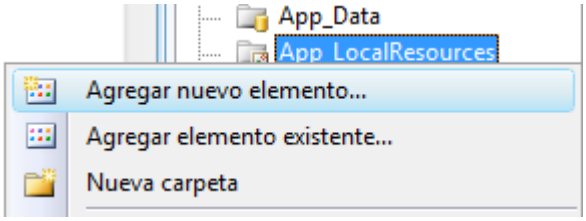

Ilustración 15

Seleccionamos como plantilla Archivo de recursos, y le ponemos como nombre, el nombre del formulario + .resx (*Ilustración 16*)

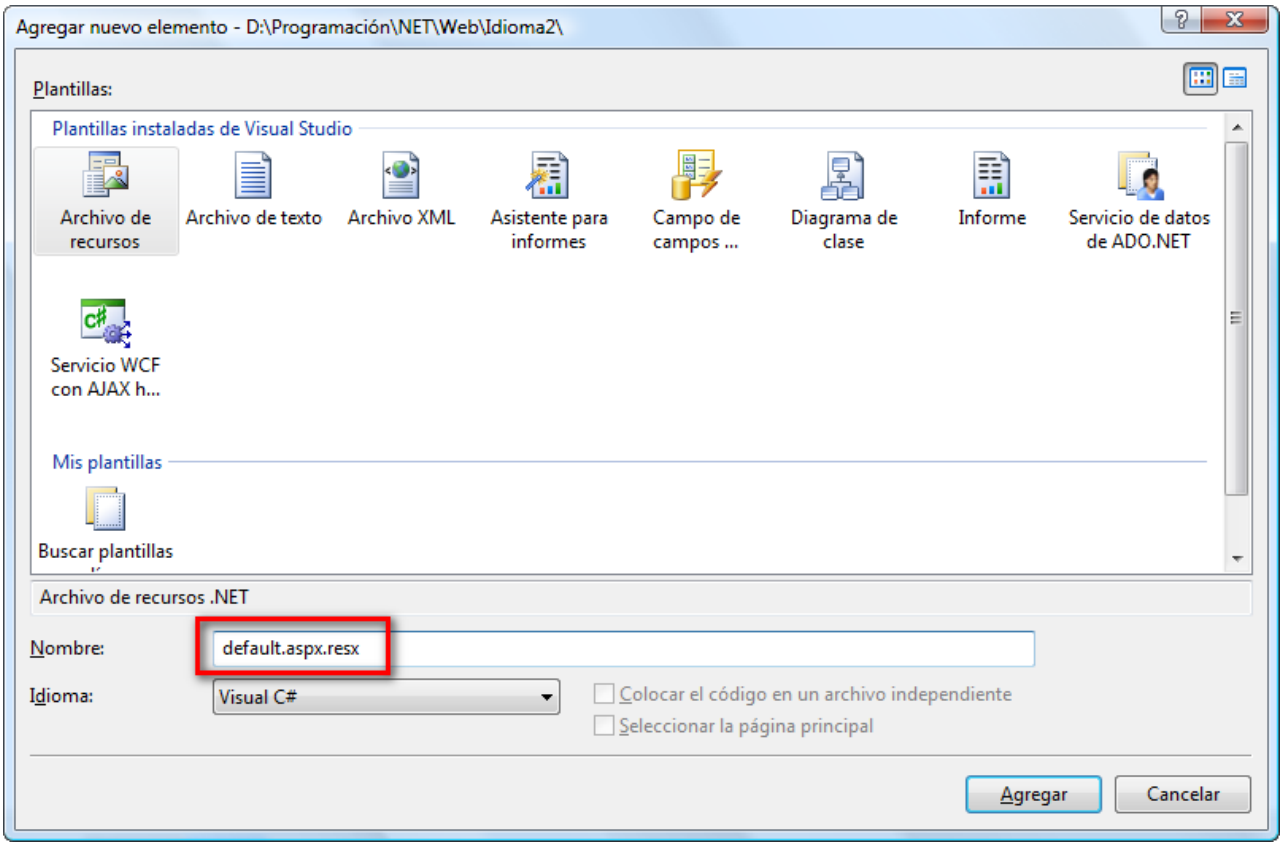

Ilustración 16: Añadir archivo de recursos

Desde este momento podemos añadir cadenas de texto como hemos hecho anteriormente.

Por último duplicaremos el recurso por cada idioma que queramos gestionar, siguiendo el formato anteriormente indicado <Nombre del formulario>.<localización>.resx, y haremos los mismos pasos para poder llamar a las cadenas de texto (Incluimos las dos referencias en el formulario, crear el procedimiento InitializeCulture y usar la función GetLocalResourceObject).

Valores globales

También podemos definir ficheros de recursos globales para nuestra aplicación, que se podrán utilizar desde cualquier aspx de la misma. Por ejemplo para cadenas que se repiten en todas las páginas o elementos comunes como títulos o logotipos.

Para crear recursos globales haremos lo siguiente:

Con el botón derecho pinchamos en el proyecto y elegimos **Agregar carpeta ASP.NET** y de la lista de carpetas **App_GlobalResources**.(*Ilustración 17).*

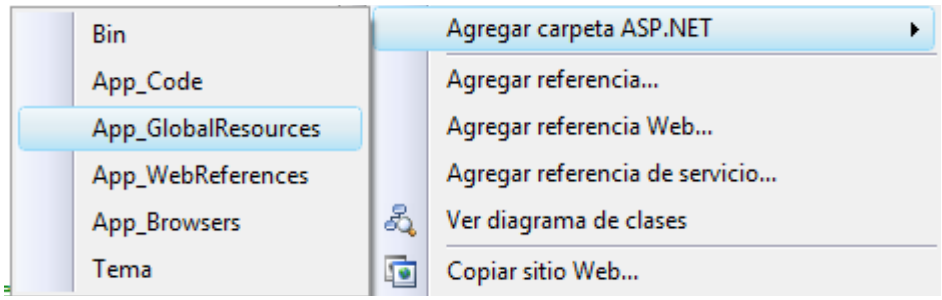

Ilustración 17: Carpeta de recursos globales

Una vez tenemos la carpeta hacemos clic con el botón derecho sobre ella y elegimos **Agregar nuevo elemento**. En la ventana que aparece seleccionamos **Archivo de recursos** le damos un nombre y pulsamos **Agregar**.(*Ilustración 18).*

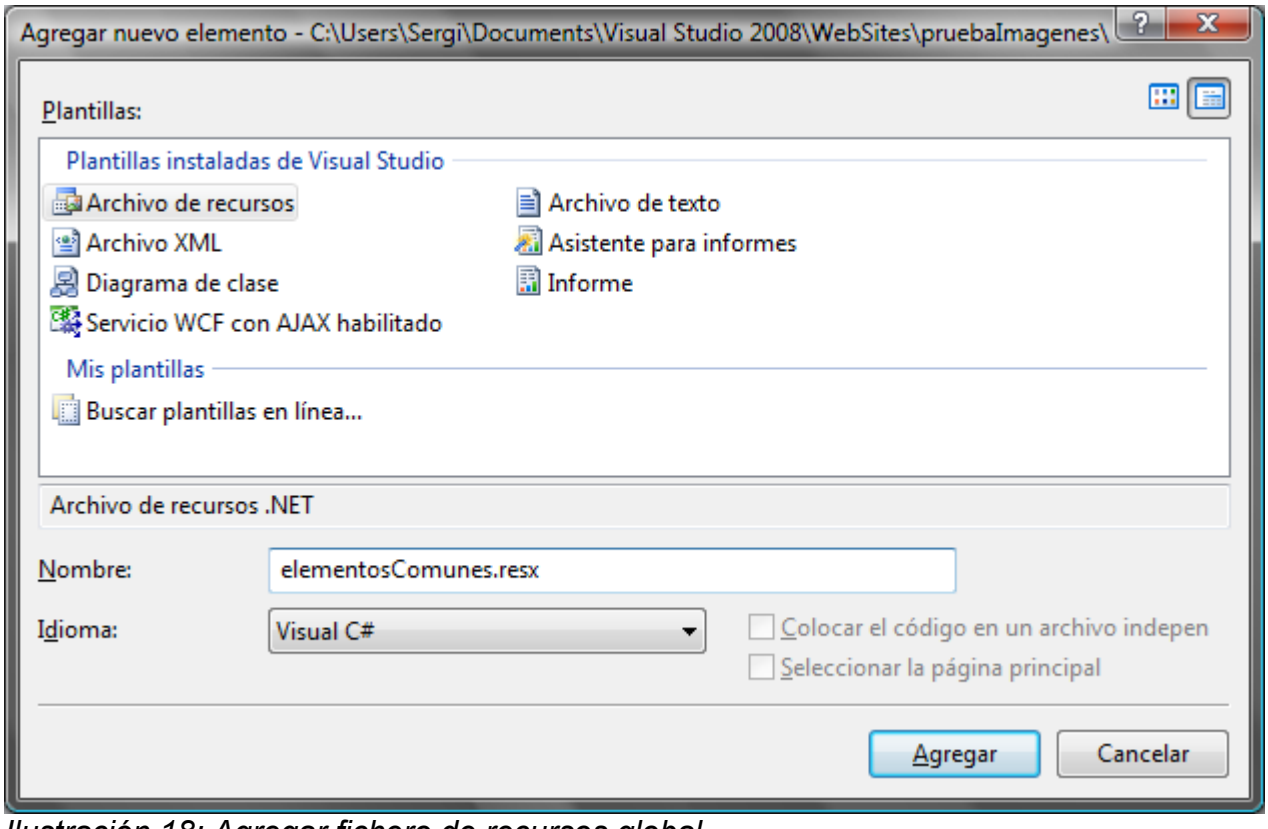

Ilustración 18: Agregar fichero de recursos global

Nos creará el fichero resx bajo la carpeta y se nos abrirá para que introduzcamos datos. Aquí haremos lo mismo que con los recursos locales añadiremos todos los pares Nombre-Valor que necesitemos. (*Ilustración 19*).

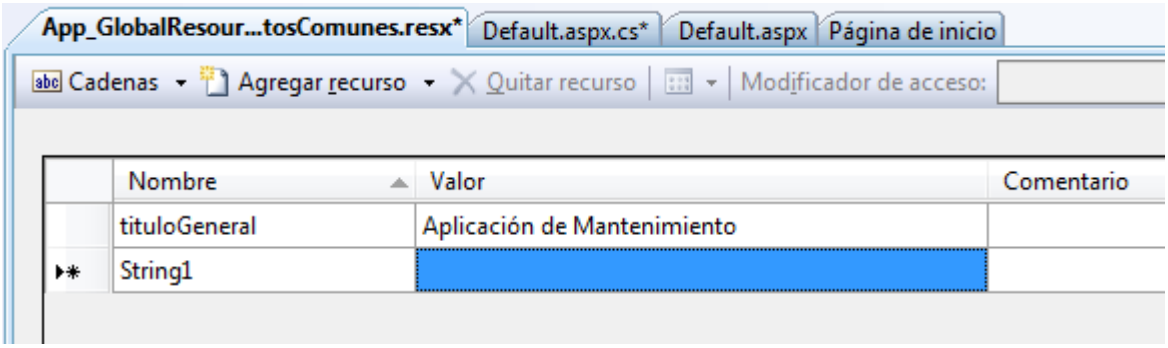

Ilustración 19: Agregar cadena global

Para acceder a estos valores desde el código tendremos dos formas de hacerlo.

1. Mediante el método **GetGlobalResourceObject**, que admite dos parámetros: el nombre del fichero sin extensión y el nombre de la cadena que queremos mostrar.

   ```
   GetGlobalResourceObject("elementosComunes","tituloGeneral")
   ```

2. Mediante el objeto **Resources**. Todos los ficheros globales de recursos aparecen como propiedades de este objeto y las cadenas creadas como propiedades del ultimo. Por lo que para obtener un valor escribiremos Resources.fichero.cadena.

   ```
   Resources.elementosComunes.tituloGeneral
   ```

Una vez tengamos definidas todas las cadenas de nuestra aplicación realizaremos las copias de los ficheros para los diferentes idiomas de la misma forma que con los recursos locales. (Ilustración 20).

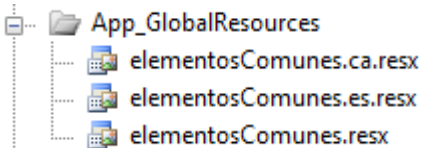

Ilustración 20: Recursos globales

Imágenes

Si tenemos imágenes diferentes según el idioma de la aplicación almacenaremos su Url dentro del un fichero de recursos correspondiente, local o global, para que se muestre una u otra según el idioma.

Los textos de las imágenes como el *texto alternativo* o la descripción estarán en el fichero de recursos local correspondiente a la página.

Nos creamos una carpeta llamada imagenes y dentro de esta una carpeta para cada idioma. Es estas carpetas agregaremos las versiones de la imagen para cada idioma. (*Ilustración 21*).

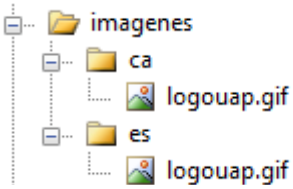

Ilustración 21: Carpeta imágenes

Ahora en los ficheros de recursos correspondientes añadiremos una cadena para guardar la dirección de cada imagen. (*Ilustración 22*).

	Nombre	Valor
	tituloGeneral	Aplicación de Mantenimiento
▶	logo	imagenes/es/logouap.gif
*		

	Nombre	Valor
	tituloGeneral	Aplicació de Manteniment
	logo	imagenes/ca/logouap.gif
*		

Ilustración 22: Dirección de los ficheros de imagen

Ahora al cargar la página asignaremos a la propiedad **ImageUrl** de la imagen la cadena del fichero de recursos.

```
imagenLogo.ImageUrl = Resources.elementosComunes.logo;
```

De esta forma se mostrará la imagen correspondiente al idioma que se esté visualizando.

NOTA: También se pueden crear ficheros de recursos para las páginas maestras, por lo que se podrá aplicar un formato distinto según el idioma.

La Clase Claseldioma

Esta clase estática nos va a servir para trabajar con varios idiomas en las cadenas de información de una clase. Por ejemplo cuando leamos de base de datos un nombre en bilingüe almacenado en dos campos diferentes de una tabla.

Incluir la clase

Primero debemos hacer una referencia a la DLL. Sobre la carpeta Bin, botón derecho, Agregar referencia …

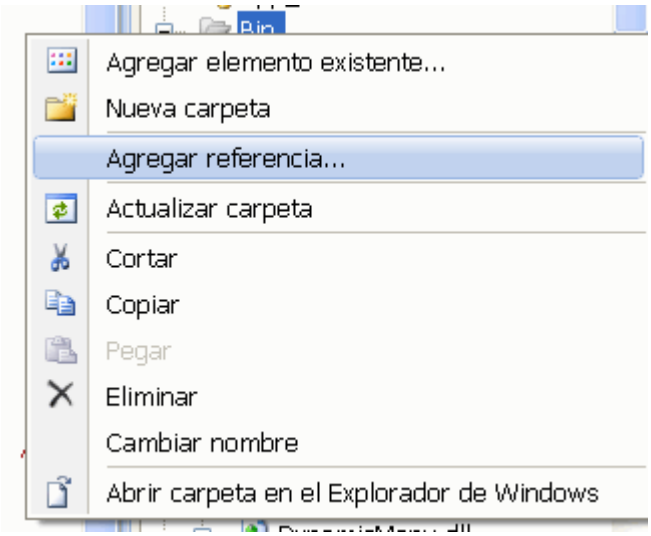

Ilustración 23: Agregar referéncia

Debemos pulsar Examinar y acceder a la carpeta **/toolsnet/DLLs/** y seleccionar **Claseldioma.dll**

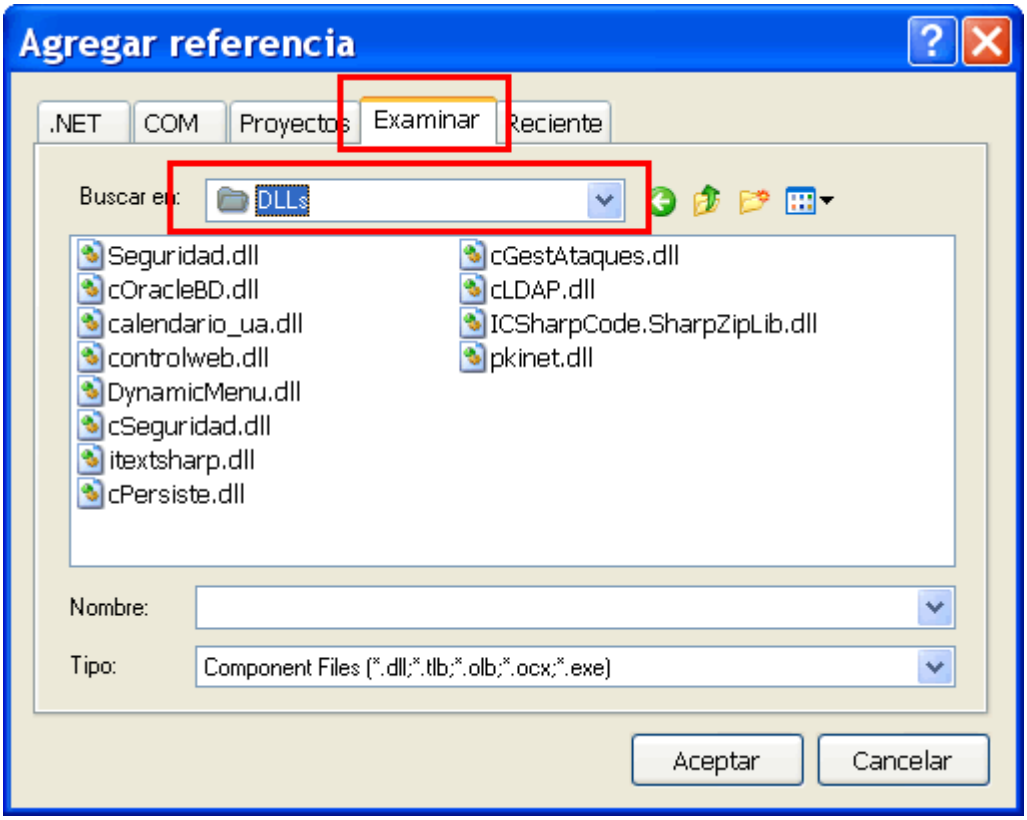

Ilustración 24: Agregar Dll

Ahora debemos hacer referencia desde nuesto ASPX al Namespace en el que se encuentra nuestra clase ("ua").

```
using ua;
```

La **ClaseIdioma** es una clase completamente estática, por lo que no tiene sentido instanciarla.

Especificar un idioma para un objeto

Internamente el idioma es un identificador entero que nos sirve para acceder a las posiciones de un vector donde se almacenan la información según el idioma.

Para obtener este identificador utilizamos el método **ConvierteIdioma**, acepta los descriptores de Campus Virtual (V,C,I), los descriptores estandar ("ca", "es", "en") y los literales definidos para la enumeración ("Valenciano", "Espanyol", "Ingles")

Por ejemplo en una clase me creo una variable miembro para identificar el idioma llamada _idioma. Y en el constructor exijo que se especifique el idioma.

```
public class ClasePrueba
{
    private int _idioma;

      public ClasePrueba(string elIdioma)
      {
        _idioma = ClaseIdioma.ConvierteIdioma(elIdioma);
      }
}
```

Ahora queremos una variable del objeto para almacenar la descripción, que puede estar en cualquier idioma.
Para crearla haremos:

```
private string[] _descripcion = new string[ClaseIdioma.NumeroIdiomas];
```

Si queremos introducir valores utilizaremos la propiedad Idioma que tiene tres identificadores Espanyol, Valenciano e Inglés, que nos devuelve el identificador de cada uno de estos idiomas.

```
        _descripcion[(int) ClaseIdioma.Idioma.Espanyol] = "Buenos dias";
        _descripcion[(int) ClaseIdioma.Idioma.Valenciano] = "Bon dia";
        _descripcion[(int) ClaseIdioma.Idioma.Ingles] = "Good Morning";
```

Para leer el valor, simplemente usamos el identificador de idioma antes creado como indice del vector.

```
  _descripcion[_idioma]
```

Toda la clase junta:

```
public class ClasePrueba
{
    private int _idioma;
    private string[] _descripcion = new string[ClaseIdioma.NumeroIdiomas];

      public ClasePrueba(string elIdioma)
      {
        _idioma = ClaseIdioma.ConvierteIdioma(elIdioma);
        _descripcion[(int) ClaseIdioma.Idioma.Espanyol] = "Buenos dias";
        _descripcion[(int) ClaseIdioma.Idioma.Valenciano] = "Bon dia";
        _descripcion[(int)ClaseIdioma.Idioma.Ingles] = "Good Morning";
      }


    public string Descripcion
    {
        get
        {
            return (_descripcion[_idioma]);
        }
        set
        {
            _descripcion[_idioma] = value;
        }
    }
}
```

Para hacer una consulta en campos virtual en tablas que tiene campos dependientes del idioma podemos usar los métodos **ObtenExtensionCVText** y **ObtenExtensionCVNum** que reciben como argumento el identificador de idioma y devuelven la extensión de texto que se le concatena al nombre del campo.

Por ejemplo, si tenemos dos campos en la tabla localizaciones llamados desc y descval. Podemos hacer la consulta así:

```
"SELECT codloc, descloc" + ClaseIdioma.ObtenExtensionCVText(_idioma) +
" descloc FROM cnet_localizaciones ";
```

Con lo que no tendremos que preocuparnos del idioma, ya que en descloc tendremos siempre el actual.

¡ Y ya está !