Guillermo "Guille" Som

# Localización práctica de aplicaciones

En el número de octubre de **dotNetManía** (nº 19) en la sección *dnm.inicio.fundamentos* estuvimos hablando sobre cómo crear aplicaciones localizables en .NET, en esa ocasión tratamos las clases que .NET Framework pone a nuestra disposición para manejar todo el tema de la localización o globalización de nuestras aplicaciones, en este artículo veremos de forma práctica cómo diseñar un formulario que se adapte a cada referencia cultural que creamos conveniente.

## » Planificar las culturas que nuestra aplicación aceptará

Lo primero que debemos hacer es planificar las diferentes culturas o idiomas y referencias culturales que vamos a aceptar en nuestra aplicación. Esto es importante, ya que siempre debemos crear un interfaz de usuario (*UI User Interface*) que sea genérico, es decir, el que se usará si no existe una referencia cultural contemplada en nuestra aplicación. Por tanto, lo primero que debemos hacer es crear la interfaz gráfica genérica de nuestra aplicación; esto es tan simple como crear un nuevo proyecto (o usar uno existente) y añadir los menús, controles y demás componentes que conformarán el interfaz gráfico de nuestra aplicación. Una vez lo tengamos creado podremos añadir nuevas referencias culturales a nuestra aplicación.Pero es muy importante que primero definamos la interfaz principal de la aplicación, ya que el resto de idiomas se basarán en esta para particularizarla. Tal como indicamos en el artículo anterior, solamente podremos regionalizar componentes de la interfaz gráfica principal, es decir, si no existe en la predeterminada no podremos utilizarlas en las secundarias, aunque lo que si podremos hacer es ocultar las partes que no nos interesen que se muestren en una interfaz localizada y, por supuesto, adaptar el contenido al idioma indicado.

Por tanto, tal como indicamos en el título de esta sección, debemos planificar las diferentes culturas y/o países que nuestra aplicación expondrá, de forma que sepamos qué es lo que debemos traducir y qué es lo que debemos ocultar.

**Guillermo "Guille" Som**
es Microsoft MVP de Visual Basic desde 1997. Es redactor de dotNetManía, miembro de *Ineta Speakers Bureau Latin America*, mentor de *Solid Quality Learning* Iberoamérica y autor del libro *Manual Imprescindible de Visual Basic .NET*.
http://www.elguille.info

## Una aplicación de ejemplo

Lo mejor para ver todo esto es la práctica, además de eso se trata esta parte de *dnm.inicio*: de ejemplos prácticos. Por tanto vamos a crear una aplicación que utilice menús y ciertos controles que utilizarán textos distintos según el idioma que seleccionemos. En este ejemplo vamos a tener en cuenta sólo dos idiomas: el español y el inglés, pero tal como comprobaremos será fácil de ampliar.

Nuestra aplicación va a estar compuesta por dos formularios, el principal y uno de configuración.

El aspecto de los dos formularios es como se muestra en las figuras 1 y 2.

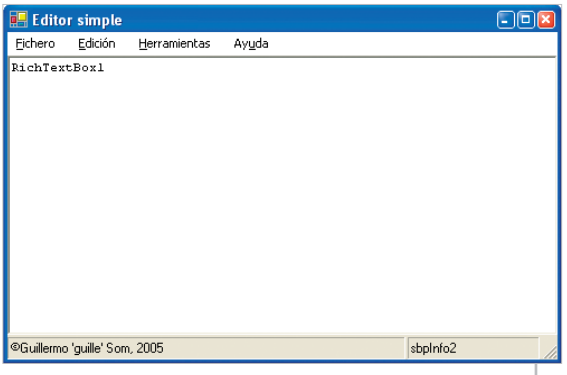


Figura 1. El formulario principal

Como podemos comprobar en la figura 1, vamos a crear un editor de textos; para ello utilizaremos unos menús y en algunas de las opciones de esos menús tendremos asignadas unas teclas de acceso rápido.

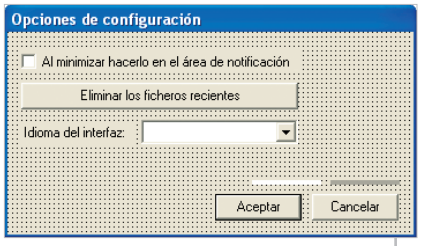


Figura 2. El formulario de configuración (opciones)

```
CultureInfo[] cuis = CultureInfo.GetCultures(CultureTypes.InstalledWin32Cultures);
idiomas = new string[cuis.Length];
int i = 0;
foreach(CultureInfo ci in cuis)
      idiomas[i++] = ci.DisplayName;
```

Fuente 1. Asignar los nombres de los idiomas en la cultura actual

Esas combinaciones de teclas variarán según el idioma y, como es natural, también lo harán los textos de los menús.

En la figura 2 tenemos la ventana de configuración, en la que, entre otras cosas, tenemos una lista desplegable en la que se mostrarán los nombres de los idiomas que actualmente el sistema operativo soporte. Esos nombres de idiomas se mostrarán en el mismo idioma que el que hayamos seleccionado. En la figura 3 podemos ver cómo se mostrarían teniendo seleccionado el idioma español.

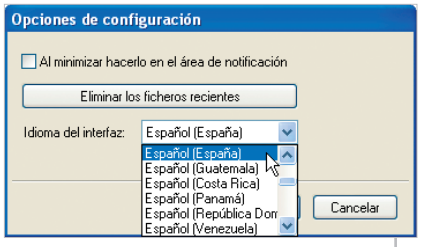


Figura 3. Lista de idiomas soportados por nuestro sistema operativo

Cuando definamos el resto de idiomas, esos nombres se mostrarán en el adecuado, pero no pensemos que tendremos que crear manualmente nuestra propia lista en cada uno de los idiomas soportados, ya que el propio .NET Framework nos da la posibilidad de hacerlo de forma simple, tal como podemos ver en el fuente 1, en el que asignamos a un array los nombres de cada idioma (usando la propiedad `DisplayName`) en el idioma de la cultura que actualmente estemos utilizando.

## Preparar el formulario para que sea localizable

Lo primero que haremos es indicarle a Visual Studio que nuestro formulario será localizable, es decir, que vamos a crear diferentes versiones según los idiomas que queramos soportar. Para ello tendremos que indicar un valor verdadero a la propiedad `Localizable` del formulario, tal como podemos ver en la figura 4.

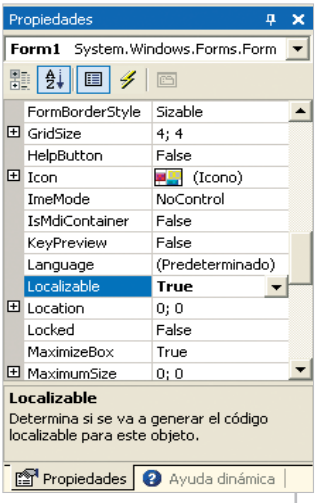


Figura 4. Indicamos que nuestro formulario soportará la localización

Inicialmente la propiedad `Language` la dejaremos tal como está de forma predeterminada, ya que esa propiedad será la que utilizaremos para indicar cada una de las versiones de idiomas de nuestro formulario.

## Personalizar el formulario para un idioma

Ahora vamos a modificar nuestro formulario para que utilice los textos en inglés.

Lo primero que haremos es seleccionar el idioma inglés de la lista desplegable que hay junto a la propiedad `Language`, tal como vemos en la figura 5.

Tal como podemos observar en la figura 5, además de idiomas regionales, también hay uno genérico; eso es así para que podamos personalizar tanto el idioma como la región o país de dicho idioma. En nuestro caso simplemente seleccionaremos "Inglés", ya que utilizaremos los mismos textos para todas las configuraciones de dicho idioma, pero si quisiéramos diferenciar, por ejemplo el inglés de los Estados Unidos con el inglés del Reino Unido, tendríamos que seleccionar uno de ellos, seguir las indicaciones que daremos a continuación y después hacer lo mismo con cada uno de los idiomas que queramos configurar.

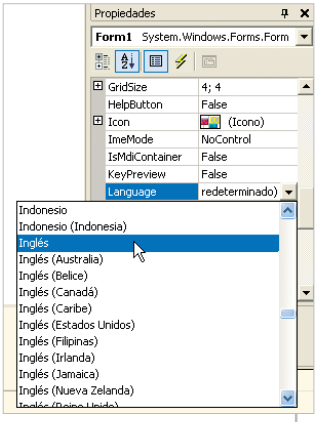


Figura 5. De la lista con los idiomas, seleccionamos el que queremos configurar

## Especificar los textos de cada idioma

Una vez que hemos seleccionado el idioma podemos ir haciendo los cambios que creamos necesarios, por ejemplo, cambiar el texto de los menús del formulario principal, las teclas de acceso rápido, etc.

Una vez que hemos hecho esos cambios, si cambiamos el idioma, tendremos los textos que hayamos indi-

cado. Por ejemplo, en el menú de edición además de los textos, también hemos utilizado diferentes accesos rápidos para algunos de los elementos del menú, en las figuras 6 y 7 podemos ver los menús que se muestran cuando elegimos el idioma español y el inglés respectivamente.


Figura 6. El menú de edición en español

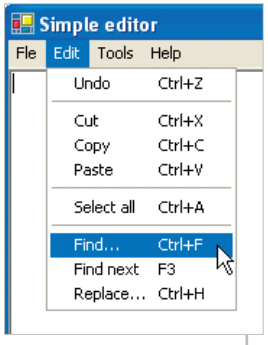


Figura 7. El menú de edición en inglés

> **NOTA**
>
> Tal como tenemos actualmente configurada la aplicación, si elegimos cualquiera de las opciones del idioma inglés, se mostrarán los textos tal como lo hemos configurado al seleccionar "Inglés" en la propiedad **`Language`**. Ya que al no indicar un idioma en particular para el interfaz se usará el genérico. Si quisiéramos distinguir entre las regiones o países, deberíamos seleccionar la combinación idioma/país para asignar los textos a mostrar.

## Cada formulario es independiente en la configuración regional

Una vez que hemos configurado el formulario principal para asignar los textos en inglés, debemos hacer exactamente lo mismo con cada uno de los demás formularios, por tanto, si también queremos localizar el formulario de configuración, tendremos que seguir los pasos indicados anteriormente, es decir: indicar un valor verdadero en la propiedad **`Localizable`**, seleccionar el idioma de la lista desplegable "Language", y modificar los textos de cada uno de los controles. Una vez seguidos todos estos pasos, el formulario "Opciones" tendrá el aspecto mostrado en la figura 8.

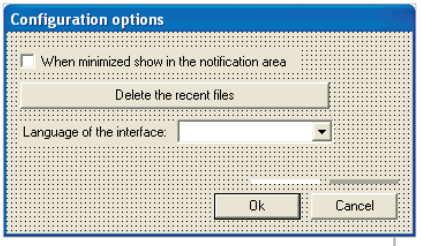


Figura 8. El formulario de configuración localizado al inglés

Como podemos apreciar en la figura 8, el texto de la etiqueta que está junto a la lista desplegable es más extenso que el que teníamos en español (lo podría haber escrito más corto y más correcto, pero así no veríamos esta característica), por tanto, podemos adecuar el tamaño de la etiqueta y desplazar la lista desplegable, además de hacer un poco más ancho el botón de borrar los ficheros recientes (*Delete the recent files*). Estos cambios sólo afectarán al formulario cuando esté seleccionado el idioma inglés, por tanto, cuando estemos usando el idioma predeterminado, todos esos controles seguirán manteniendo el tamaño y posición que tenían originalmente.

## ¿Cómo utiliza Visual Studio cada idioma?

Como podemos comprobar, es fácil crear cada una de las interfaces que usaremos en cada idioma, todo ello debido a que es el propio IDE de Visual Studio el que se encarga de los pormenores de la localización, y como podemos comprobar en los fuentes 2 y 3, en el que tenemos el código generado por el IDE para el botón "Cancelar" del formulario de configuración antes y después de localizar dicho formulario, se tienen en cuenta una serie de propiedades de cada control que hace posible que el interfaz se adapte perfectamente a cada idioma.

Por simplificar, en el fuente 3, no mostramos el código completo de todas las propiedades asignadas dinámicamente, sólo el nombre de la propiedad, para que tengamos una idea de qué propiedades se pueden localizar, que como podemos comprobar son prácticamente todas excepto el nombre del control

Es fácil crear cada una de las interfaces que usaremos en cada idioma, todo ello debido a que es el propio IDE de Visual Studio el que se encarga de los pormenores de la localización

```
//
//btnCancelar
//
this.btnCancelar.DialogResult = System.Windows.Forms.DialogResult.Cancel;
this.btnCancelar.FlatStyle = System.Windows.Forms.FlatStyle.System;
this.btnCancelar.Location = new System.Drawing.Point(264, 140);
this.btnCancelar.Name = "btnCancelar";
this.btnCancelar.TabIndex = 7;
this.btnCancelar.Text = "Cancelar";
this.btnCancelar.Click += new System.EventHandler(this.btnCancelar_Click);
```

Fuente 2. El botón "Cancelar" antes de localizarlo

```
//
//btnCancelar
//
this.btnCancelar.AccessibleDescription = ...;
this.btnCancelar.AccessibleName = ...;
this.btnCancelar.Anchor = ...;
this.btnCancelar.BackgroundImage = ...;
this.btnCancelar.DialogResult = System.Windows.Forms.DialogResult.Cancel;
this.btnCancelar.Dock = ...;
this.btnCancelar.Enabled = ...;
this.btnCancelar.FlatStyle = ...;
this.btnCancelar.Font = ...;
this.btnCancelar.Image = ...;
this.btnCancelar.ImageAlign = ...;
this.btnCancelar.ImageIndex = ...;
this.btnCancelar.ImeMode = ...;
this.btnCancelar.Location = ...;
this.btnCancelar.Name = "btnCancelar";
this.btnCancelar.RightToLeft = ...;
this.btnCancelar.Size = ((System.Drawing.Size)(resources.GetObject("btnCancelar.Size")));
this.btnCancelar.TabIndex = ...;
this.btnCancelar.Text = resources.GetString("btnCancelar.Text");
this.btnCancelar.TextAlign = ...;
this.btnCancelar.Visible = ...;
this.btnCancelar.Click += new System.EventHandler(this.btnCancelar_Click);
```

Fuente 3. El botón "Cancelar" después de localizarlo

y en este caso particular, el valor a devolver cuando se muestre el formulario como un cuadro de diálogo.

Todas las propiedades localizables las obtiene mediante los métodos `GetString` y `GetObject` del objeto `resources`, que inicialmente habrá cargado los recursos del formulario según el idioma actual, o mejor dicho, del idioma (o cultura) que actualmente tenga asignado el objeto `CurrentUICulture` del hilo actual.

## ¿Cuándo se utiliza cada configuración regional?

Cuando nuestra aplicación está en ejecución, es el propio *runtime* de .NET Framework el encargado de seleccionar el idioma adecuado, en realidad lo que hace el CLR (*Common Language Runtime* o motor en tiempo de ejecución de .NET), es asignar el valor adecuado a las propiedades `CurrentUICulture` y `CurrentCulture` del hilo actual, y según esos valores, se utilizarán los recursos que correspondan. Si no existen recursos específicos para la combinación idioma/país, se intentará usar el que haya para ese idioma (el genérico), y en caso de que tampoco existan recursos para ese idioma, se usarán los recursos predeterminados.

El idioma o cultura actual predeterminada es la que está seleccionada en el propio sistema operativo, pero de forma programática podemos cambiarla para nuestra aplicación, ya que esta será la mejor forma de hacer que nuestra aplicación pueda usarse en cualquiera de los idiomas previstos sin necesidad de tener que cambiar la configuración de todo el equipo, porque puede que no nos interese afectar al resto de aplicaciones que se estén ejecutando actualmente.

Como ya comentamos en *dnm.inicio.fundamentos* del pasado mes de octubre, las dos propiedades mencionadas anteriormente son las que nos sirven para trabajar con los diferentes idiomas en nuestras aplicaciones; de hecho asignando el valor adecuado a cada una de esas propiedades conseguiremos que se utilice la referencia cultural que nos interese.

## Cambiar por código el idioma a usar por nuestra aplicación

Recordemos que la propiedad `CurrentCulture` es la usada para saber el formato de la fecha, moneda, separador decimal, etc. Por otro lado, la propiedad `CurrentUICulture` es la que se encargará de indicar que interfaz gráfico hay que usar.

Dicho esto podemos pensar que asignando el valor adecuado a cada una de esas propiedades podremos cambiar el idioma usado en nuestra aplicación. Y estaremos en lo cierto, el problema es que, al menos en lo que respecta al interfaz de usuario, éste no cambia... o casi.

Si probamos con lo que tenemos actualmente, es decir, dos formularios que están preparados para utilizar el idioma inglés y como predeterminado el español, particularmente el formulario principal en el que hemos definido unos menús, veremos que sólo parte del interfaz gráfico de ese formulario se adapta al cambio de idiomas.

Para probar el cambio de idiomas podemos añadir una opción nueva al menú "Herramientas" en la que tengamos otras dos opciones, una para seleccionar el idioma inglés y la otra para seleccionar el idioma español. El código de los eventos `Click` de esas opciones sería el mostrado en el fuente 4.

Pero en realidad esto no hace todo lo que nos gustaría que hiciera. Aunque si mostramos el formulario de "Opciones", veremos que se muestra en el idioma que hayamos indicado,

```
private void mnuHerIdiomaIngles_Click(object sender, System.EventArgs e)
{
      cultura = "en-US";
      Thread.CurrentThread.CurrentCulture = new CultureInfo(cultura);
      Thread.CurrentThread.CurrentUICulture = new CultureInfo(cultura);
}

private void mnuHerIdiomaEspañol_Click(object sender, System.EventArgs e)
{
      cultura = "es-ES";
      Thread.CurrentThread.CurrentCulture = new CultureInfo(cultura);
      Thread.CurrentThread.CurrentUICulture = new CultureInfo(cultura);
}
```

Fuente 4. Código para cambiar la referencia cultural del hilo actual

pero no ocurre lo mismo con el formulario principal.

Y no se muestra porque aunque el idioma del hilo principal de nuestra aplicación haya cambiado, no le hemos dicho que utilice los recursos correspondientes.

En un momento veremos cómo lo podemos hacer, ya que antes hay que puntualizar algo que no debemos olvidar.

Para hacer esta prueba del cambio de idioma (supuestamente) hemos añadido unas nuevas opciones al menú "Herramientas" del formulario principal. Si el idioma seleccionado era el predeterminado, esos mismos menús se han añadido al inglés, pero el texto que mostrarán será el mismo que hemos escrito en el predeterminado, por tanto debemos tener en cuenta esto, si ya tenemos definidos otros idiomas además del predeterminado, y añadimos nuevos elementos al interfaz, ya sean menús o controles, debemos actualizar el texto que se mostrará en esos otros idiomas, ya que si no lo hacemos, al cambiar de idioma se mostrará el texto del predeterminado. En realidad los nuevos elementos del interfaz gráfico los podemos añadir sea cual sea la configuración del idioma del formulario, pero siempre debemos indicar el texto que mostrará en cada uno de los idiomas.

Una vez hecha esta aclaración, que aunque es obvia, es conveniente tenerla en cuenta, veamos cómo podemos hacer que todo el interfaz del formulario cambie para que se muestren los textos (y demás asignaciones) en el idioma seleccionado.

La solución consiste en leer las cadenas de texto del fichero de recursos y asignarlos, en realidad no solo los textos deberíamos asignarlos, sino el resto de propiedades, de esa forma estaría todo como lo dejamos diseñado en el formulario, ¿no?... Ya, resulta largo de hacer, ¿verdad? porque si nos fijamos en el código del fuente 3, para un simple botón, la cantidad de cosas que habría que hacer. Pero si ya tenemos en nuestro código un sitio en el que se hacen esas asignaciones, ¿por qué no lo aprovechamos? Por tanto, si queremos que nuestra aplicación cambie dinámicamente de idioma, es decir, sin tener que cerrarla y volverla a abrir, lo que podemos hacer es una llamada al método que utiliza el diseñador de formularios para la creación de todos los elementos que forman el formulario, el método `InitializeComponent`. Ese método es el que se encarga de leer del fichero de recursos los valores que debe tener: texto a mostrar, estado, posición, tamaño, etc., por tanto es lo que debemos hacer: lo llamamos y tendremos todo en el idioma que hayamos asignado a la propiedad `CurrentUICulture`.

> Cuando compilamos la aplicación, el compilador creará un ensamblado DLL para cada uno de los idiomas definidos, esa librería la guardará en el mismo directorio que el ejecutable, en un directorio con el nombre de la cultura

Dicho esto, podemos cambiar el código mostrado en el fuente 4 para que sea el siguiente:

```
cultura = "<el idioma> ;
Thread.CurrentThread.CurrentCulture =
            new CultureInfo(cultura);
Thread.CurrentThread.CurrentUICulture =
            new CultureInfo(cultura);
InitializeComponent();
```

Pero esto no soluciona el problema. Aunque en nuestra aplicación de ejemplo, en la que no tenemos ningún control que se adapte al idioma salvo los menús sí que "parece" funcionar. Pero si añadimos un botón y asignamos un texto a ese botón para los dos idiomas con los que estamos trabajando nos daremos cuenta que el texto de ese botón no cambia, siempre es el del idioma predeterminado, sin embargo, los menús sí que cambian.

¿Qué es lo que ocurre? Que tenemos controles duplicados en el formulario. La mejor forma de comprobarlo es asignar posiciones distintas a esos controles dependiendo del idioma, de esta forma, al cambiar de idioma veremos tanto los controles de uno como del otro idioma, es más si repetimos el proceso, comprobaremos que cada vez que cambiemos de idioma se añaden nuevos controles al formulario.

¿Cómo lo podemos solucionar? Haciendo una "limpieza" de controles del formulario, de esta forma, al llamar al método `InitilizeComponent` se crearán nuevamente y sólo habrá una copia, funcionando todo como era de esperar.

**NOTA**

En el `.zip` con el código de ejemplo, tenemos un proyecto: `dnm_localizacionUI_1`, (tanto para VB como para C#), en el que podemos comprobar la repetición de controles al cambiar el idioma (*dotnetmania.com*).

La limpieza de controles la podemos hacer llamando al método `Clear` de la

> **NOTA**
>
> En los proyectos **dnm_LocalizacionUI_CS** y **dnm_LocalizacionIU_VB** también se incluyen "extras" que aunque no están comentados en el artículo, pueden serte de utilidad, como es un formulario de búsqueda, una clase para leer y guardar información de configuración sin usar **appSettings** y una función para averiguar la codificación de un fichero, además de otras "cosillas" interesantes.

colección **Controls** que es la colección que contiene todos los controles del formulario, de esta forma, nuestro código de cambio de idioma quedaría así:

```
cultura = "<el idioma>";
Thread.CurrentThread.CurrentCulture =
            new CultureInfo(cultura);
Thread.CurrentThread.CurrentUICulture =
            new CultureInfo(cultura);
this.Controls.Clear();
InitializeComponent();
```

En los proyectos que se incluyen en el ZIP que acompaña a este artículo utilizamos otra forma de saber cuándo cambiar de idioma, ya que ese cambio lo realizamos a través del formulario de configuración, pero básicamente hacemos esto mismo.

## ¿Qué tenemos que distribuir con nuestra aplicación?

O dicho de otra forma ¿dónde almacena el compilador los recursos de cada idioma?

Cuando compilamos la aplicación, el compilador creará un ensamblado DLL para cada uno de los idiomas definidos, esa librería la guardará en el mismo directorio que el ejecutable, en un directorio con el nombre de la cultura. En nuestro ejemplo sólo hemos creado el predeterminado y el inglés genérico, por tanto tendremos una carpeta llamada "**en**" en la que habrá una DLL con el nombre **<nombre del ejecutable>.resources.dll**, por ejemplo: **dnm_localizacionUI_CS.resources.dll**.

Si tuviésemos más idiomas y regiones definidos tendríamos una carpeta para cada uno de esos idiomas/regiones que hayamos creado.

Toda esa estructura de directorios es lo que tendremos que distribuir con nuestra aplicación para que sean operativos todos y cada uno de esos idiomas que hemos configurado.

Prácticamente esto es todo lo que tendremos que hacer para crear nuestras aplicaciones en diferentes idiomas, aunque aún quedan ciertos aspectos que debemos tener en cuenta, tal y como comentamos en el artículo anterior, y es que además del aspecto gráfico de la aplicación debemos preocuparnos del resto de aspectos que hacen diferentes a cada idioma, como es los formatos numéricos o de fechas. Por simplificar, en nuestro editor de textos podríamos tener un nombre de fichero predeterminado cuando creamos uno nuevo, para indicar que aún no le hemos asignado un nombre válido, por ejemplo, en español podemos llamarlo "**Sin título**" y en inglés "**Untitled**", de igual forma, a la hora de mostrar los cuadros de diálogo de abrir o guardar, deberíamos asignar las cadenas adecuadas según el idioma que actualmente esté seleccionado, pero de esos aspectos ya no es el propio compilador el que se encarga, sino que tendremos que hacerlo nosotros por medio de código. Por ejemplo, en el código fuente 5 vemos cómo podríamos asignar a una variable el nombre del fichero "**Sin título**" según el idioma:

```
// Ajustar las cadenas según el idioma seleccionado
// Sólo tendremos en cuenta algunos idiomas
switch(cultura.Substring(0, 2).ToLower())
{
      case "en":
            sinTitulo = "Untitled";
            break;
      //case "es"
      default:
            sinTitulo = "Sin título";
            break;
}
```

Fuente 5. Asignar cadenas según el idioma que estemos usando

Y como esto, algunas otras cosas que no dependan directamente de los controles o componentes que tengamos en los formularios.

Por último decir, que si al cambiar de idioma tenemos otros formularios abiertos, debemos utilizar alguna fórmula para sincronizarlos con la nueva selección, en estos casos lo mejor es crear un método que sea el que se encargue de asignar o actualizar los controles de dicho formulario, en el que haremos algo parecido a lo mostrado anteriormente. Si el formulario no está actualmente abierto no debemos preocuparnos, ya que la próxima vez que lo mostremos, (si lo tenemos también localizado), no tendremos que hacer nada en especial, ya que al crear el formulario se hace una llamada al método **InitialiceComponent** que el IDE de Visual Studio siempre añade a todos los formularios para crear los controles.

Finalmente recordar que como viene siendo habitual en estos artículos de *dnm.inicio*, el código mostrado en el artículo es para C#, pero que en el ZIP con el código tenemos también el código para Visual Basic, y ahora que estamos con la versión final de Visual Studio 2005 en la calle desde el mes pasado, decir que estos proyectos están creados con Visual Studio 2003 pero funcionan igualmente en Visual Studio 2005 incluso en las versiones Express.

¡Feliz localización y próspera globalización!