Guillermo "Guille" Som

Yo localizo, tú localizas,... ¡localicemos todos!

Es un hecho que en estos tiempos que corren, las aplicaciones que hagamos pueden ser utilizadas por un número o tipo de gente que puede ser de lo más variopinto, todo ello debido a que si dichas aplicaciones las distribuimos en Internet no debemos dejar las puertas cerradas a otras culturas o idiomas. Por ese motivo vamos a ver qué posibilidades nos ofrece .NET para internacionalizar nuestras aplicaciones.

» Globalización, localizabilidad y localización de aplicaciones

Si buscamos en la ayuda de Visual Studio .NET sobre cómo hacer que nuestra aplicación sea una aplicación internacional, es decir, que esté preparada para ser usada por usuarios de diferentes países y culturas, nos encontraremos con los tres conceptos que hemos usado como título de esta sección. Por tanto creemos que es necesario aclarar qué significa cada una de esas palabras y, lo más importante, qué papel juegan a la hora de que nuestras aplicaciones sean "internacionales".

En esa misma fuente de información, define la globalización como:

La globalización es el proceso de diseño y desarrollo de una aplicación que permite el uso de interfaces de usuario localizadas y datos regionales para usuarios de varias referencias culturales.

Por lo que podemos deducir que la globalización es precisamente lo que estamos buscando: la característica que debemos tener en cuenta a la hora de crear aplicaciones internacionales.

Pero para que esa globalización de nuestras aplicaciones sea una realidad debemos tener en mente precisamente esa intención de que sea global. Para conseguirlo debemos tener en cuenta la "localizabilidad", es decir, si nuestra aplicación, o mejor dicho, el código de nuestra aplicación, realmente está preparado para que sea una aplicación "global". Es fácil pretender que nuestra aplicación sea internacional y, como tendremos ocasión de comprobar, lo más fácil será mostrar al usuario una interfaz que se adapte a su idioma (la localización), pero si todo el código que utilicemos no está en consonancia con dicha interfaz, de poco nos servirá.

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

Referencias culturales neutras y específicas

Cuando creamos una aplicación que pueda ser usada por usuarios de diferentes países, no sólo es importante, porque lo es, que los textos mostrados en la interfaz estén en su idioma, sino que el resto de lo que la aplicación haga también esté en consonancia con ese idioma o cultura. Por ejemplo, si nuestra aplicación debe mostrar cifras o fechas, éstas no tendrán el mismo formato para un estadounidense que para un inglés del Reino Unido, aunque ambos utilicen el mismo idioma.

En estos casos hay que diferenciar el idioma o referencia cultural neutra, en nuestro ejemplo el idioma inglés, de la referencia cultural específica en la que debemos indicar el país o región que queremos utilizar o referenciar.

Para indicar esas referencias culturales, se utiliza el formato siguiente: *Idioma-Región/País*. Por ejemplo, `es-ES` indica que estamos usando el idioma español y el país España, si usamos `es-AR` también sería el idioma español, pero de Argentina, lo cual nos sirve para diferenciar los distintos formatos que estas dos referencias culturales puedan tener.

¿Qué aporta .NET en la creación de aplicaciones internacionales?

.NET Framework utiliza ciertas clases y propiedades para que podamos crear de una forma, más o menos fácil, nuestras aplicaciones internacionales. Aunque básicamente trabajaremos con una clase: `CultureInfo`, que nos servirá para saber la cultura o región en la que se está utilizando la aplicación en el momento de ponerla en funcionamiento. Utilizando esa clase podremos

averiguar si es un inglés o un español el que la está utilizando, e incluso si ese inglés es de los Estados Unidos o es de Australia. Por tanto es importante conocerla para que sepamos aplicarla en nuestro código.

La clase CultureInfo

Por medio de esta clase, que se incluye en el espacio de nombres `System.Globalization`, podemos averiguar las características particulares relacionadas con el idioma, país y otras particularidades del usuario actual. `CultureInfo` será la clase con la que, entre otras cosas, daremos formato a fechas, números o la forma de ordenar las cadenas de texto.

Todos estos formatos e información que nos proporciona la clase `CultureInfo` la obtendremos por medio de propiedades, las cuales a su vez pueden devolver objetos con información extra como pueden ser las propiedades `DateTimeFormat` o `NumberFormat`.

Averiguar la configuración cultural actual

La clase `CultureInfo` la podemos usar principalmente de dos formas, una de ellas es utilizando el valor devuelto por la propiedad `CurrentCulture` del hilo (o subproceso) actual, en cuyo caso lo que obtenemos es un objeto con la configuración cultural del usuario actual.

En el fuente 1 tenemos un ejemplo de cómo utilizar la propiedad `CurrentCulture` para obtener el nombre de la cultura actual, el formato corto de fecha y el signo usado como separador decimal.

```
using System;
using System.Globalization;

class Class1
{
  static void Main()
  {
    CultureInfo cui;
    cui = System.Threading.Thread.CurrentThread.CurrentCulture;

    string fmtFecha = cui.DateTimeFormat.ShortDatePattern;
    string signoDecimal = cui.NumberFormat.CurrencyDecimalSeparator;
    string fecha = DateTime.Now.ToString(fmtFecha);

    Console.WriteLine("Cultura actual: {0}", cui.DisplayName);
    Console.WriteLine("Formato de fecha corta: {0}", fecha);
    Console.WriteLine("Separador de decimales: {0}", signoDecimal);
  }
}
```

Fuente 1. Ejemplo de la clase `CultureInfo` usando la referencia cultural actual

Si utilizamos este código con la configuración de español de España y la fecha es el 17 de septiembre, la salida sería la siguiente:

```
Cultura actual: Español (España)
Formato de fecha corta: 17/09/2005
Separador de decimales: ,
```

NOTA

El código utilizado en los fuentes de este artículo es de C#, pero en el ZIP con el código completo se incluye tanto el de C# como el de Visual Basic .NET.

Utilizar una configuración cultural específica

La clase `CultureInfo` admite constructores por medio de los cuales podemos crear un nuevo objeto con la información cultural que nosotros determinemos, en este caso, no se tendrá en cuenta la configuración que tenga nuestro sistema, sino que utilizaremos el objeto creado para acceder a la información cultural que hemos indicado en el constructor.

En el fuente 2 podemos ver cómo crear un objeto con la información cultural de inglés de los Estados Unidos (USA).

```
private static void culturaUSA()
{
  CultureInfo cui = new CultureInfo("en-US");

  string fmtFecha = cui.DateTimeFormat.ShortDatePattern;
  string signoDecimal = cui.NumberFormat.CurrencyDecimalSeparator;
  string fecha = DateTime.Now.ToString(fmtFecha);

  Console.WriteLine("Nombre cultura: {0}", cui.DisplayName);
  Console.WriteLine("Formato de fecha corta: {0}", fecha);
  Console.WriteLine("Separador de decimales: {0}", signoDecimal);
}
```

Fuente 2. Ejemplo de la clase `CultureInfo` indicando una referencia cultural específica

La salida que obtendremos con este código será la siguiente:

```
Nombre cultura: Inglés (Estados Unidos)
Formato de fecha corta: 9/17/2005
Separador de decimales: .
```

Cuando usamos el constructor de la clase `CultureInfo` en el que indicamos una cadena con el nombre de la referencia cultural de la que queremos crear la nueva instancia, siempre debemos indicar el idioma además del país o región, por ejemplo, si modificamos la instanciación de la clase `CultureInfo` que tenemos en el fuente 2, sería erróneo hacerlo de esta forma:

```
CultureInfo cui = new CultureInfo("en");
```

Debido a que no se puede usar una referencia cultural neutra para obtener información de los forma-

tos y demás información proporcionada por la clase `CultureInfo`.

NOTA

Para utilizar de forma normal nuestras aplicaciones, no tenemos que modificar ni instanciar la clase CultureInfo, o el valor que está usando actualmente el sistema, pero en caso de querer hacerlo, con idea de que nuestra aplicación utilice una referencia cultural en particular, debemos asignarlo a la propiedad `CurrentCulture` del hilo actual. En el siguiente ejemplo, `cultura` será una cadena en el formato indicado anteriormente: *idioma-país*, por ejemplo *en-US*:

```
Thread.CurrentThread.CurrentCulture = new
      CultureInfo(cultura);
```

La propiedad CurrentUICulture

Esta propiedad será la que utilizaremos para trabajar con la interfaz gráfica del usuario con una configuración en particular. Esta propiedad es del tipo `CultureInfo`; para obtener el valor que actualmente tenemos lo podemos hacer utilizando tanto la propiedad del mismo nombre de la propiedad `CurrentThread` del hilo actual, o por medio de la propiedad estática `CurrentUICulture` de la clase `CultureInfo`.

En el fuente 3 vemos cómo obtener el valor usando los dos procedimientos indicados.

```
CultureInfo cui;

// Utilizando la propiedad CurrentUICulture del hilo actual
cui = System.Threading.Thread.CurrentThread.CurrentUICulture;
// Utilizando la propiedad CurrentUICulture de la clase CultureInfo
cui = CultureInfo.CurrentUICulture;
```

Fuente 3. Dos formas de acceder al valor de `CurrentUICulture`

Indicar el idioma del interfaz de usuario

De la misma forma que podemos asignar el valor que tendrá el objeto `CultureInfo` del hilo actual, también podemos asignar o indicar qué idioma o región cultural será la que se utilizará para que podamos adaptar la interfaz gráfica de nuestra aplicación, (si es que así lo hemos previsto: ¿recuerda la localizabilidad?), al idioma que el usuario esté usando actualmente o bien, porque, mediante código, lo hayamos indicado nosotros.

Como tendremos ocasión de comprobar en el próximo artículo de `dnm.inicio.taller`, .NET Framework nos proporciona un sistema relativamente sencillo de crear las interfaces gráficas que utilizaremos para cada uno de los idiomas en los que queramos traducir la interfaz gráfica de nuestra aplicación, en esa ocasión también veremos los detalles que tendremos que tener en cuenta para que incluso esa "transformación" la podamos hacer en tiempo de ejecución, es decir, cambiar de idioma mientras estamos utilizando la aplicación, además de guardar la configuración del idioma seleccionado con idea de utilizarlo la próxima vez que iniciemos la aplicación.

NOTA

En caso de que seamos nosotros, por medio de código, los que queramos cambiar el idioma a usar por la aplicación, debemos asegurarnos de que tenemos permisos suficientes para hacerlo, particularmente debemos utilizar la clase `SecurityPermission` indicando el permiso `ControlThread`, ya que este indicador es el que nos permite realizar algunas operaciones avanzadas con los subprocesos. Este valor lo indicaremos en el fichero `AssemblyInfo` como un atributo a aplicar a todo el ensamblado.

```
using System.Security.Permissions;
[assembly: SecurityPermission(SecurityAction.
RequestMinimum, ControlThread=true)]
```

La clase ResourceManager

Esta es la clase con la que podemos acceder a los recursos localizados de la aplicación, como puede ser el texto a mostrar en los distintos controles, e incluso el estado de dichos controles, de forma que tengan, o puedan tener asignados valores diferentes en las propiedades que así lo creamos conveniente; por ejemplo, podemos tener ciertos controles habilitados en un idioma y deshabilitados en otros.

Tal como trabaja .NET Framework a la hora de permitirnos indicar diferentes configuraciones para la interfaz gráfica de cada idioma, siempre existirá una interfaz predeterminada, que será la usada cuando no existe una configuración particular de algún idioma, y si indicamos otros idiomas, éstos siempre tendrán que usar los controles existentes en dicha configuración genérica, pero lo que no se nos permitirá será añadir nuevos elementos que sólo estén en algunas configuraciones regionales, ya que esos elementos "localizables" siempre deberán existir en la interfaz gráfica predeterminada. De estos detalles nos ocuparemos próximamente en la parte práctica de esta sección.

Como hemos podido comprobar, la globalización de aplicaciones en .NET es algo que, al menos teóricamente, es fácil de implementar; y como veremos en el próximo taller de `dnm.inicio`, podremos conseguir que nuestra aplicación utilice varias referencias culturales, algunas de ellas para un mismo idioma, pero para un país o región diferentes.

Pero de todo esto nos encargaremos en otro artículo, en el que abordaremos de forma más detallada todos los aspectos que debemos tener en cuenta para crear diferentes versiones de nuestra aplicación para los idiomas y países que así lo estimemos oportuno.