# Serialización o persistencia de objetos (I)

**Luis Miguel Blanco Ancos**

En los oscuros tiempos...

...de la programación en modo comando, cuando los lóbregos interfaces basados en caracteres dominaban el orbe del desarrollo, los programadores armados con sus rudimentarios compiladores y editores se veían abocados a la producción de incontables líneas de código (ahora también, pero así le damos un poco de ambiente a la cosa).

En aquellos sombríos días, las técnicas de codificación poco flexibles, requerían que los programadores tuvieran que repetir procesos de características similares en su código. Y el programador pensó que esto era malo.

Pero una incipiente luz se abrió camino en aquellos tenebrosos días, llegó la época de la OOP, para ayudar a los sufridos programadores en su penosa tarea de codificar. La liberación y la iluminación del código se instauraron, y el tiempo de los interfaces gráficos dominó sobre la Tierra. Y el programador pensó que esto era bueno.

A pesar de esta época de prosperidad, los sufridos programadores se hallaron ante un nuevo designio que les impedía elaborar su preciado código con el acabado que otros gremios de aquellas remotas edades conseguían para sí de sus mercaderías. Si bien disponían de la capacidad de crear objetos, dichos objetos no podían ser mantenidos en un estado de animación suspendida una vez finalizada la ejecución del programa, para poder así ser devueltos a la vida por el mismo u otro programa que necesitar de sus favores hubiere. Y el programador pensó que esto era malo.

Mas no desesperemos, una nueva era de ilustración ha llegado, el artesano del código dispone ahora en sus manos de una nueva herramienta capaz de devolver a la vida un objeto destruido en el fragor de la batalla de la ejecución de un programa. Dicha herramienta es la plataforma .NET, y la serialización es nuestro arma. Y el programador pensó...

... debido a lo relativamente reciente de la tecnología, invitamos al programador a leer este artículo y así que pueda pensar libremente.

Persistencia, el Ave Fénix del código

# Serialización o persistencia de objetos (I)

Cuando un objeto que se encuentra en ejecución durante el transcurso del funcionamiento de un programa es destruido, bien porque el código lo hace de forma explícita, bien porque la variable que contiene la referencia al objeto pierde ámbito, o por cualquier otro motivo, toda la información que dicho objeto almacenaba en cuanto a los valores de sus propiedades también se pierde, dado que el soporte volátil del objeto, la memoria, no permite guardar de forma permanente el objeto.

Para solucionar este inconveniente, y como ya habrá podido intuir el lector en la introducción estilo radionovela de este artículo, la plataforma .NET nos proporciona la persistencia de objetos, una técnica que nos permite tomar el estado de un objeto en un determinado momento de su ejecución o periodo de vida, y volcarlo a un soporte permanente como el disco, o transmitirlo de forma remota a otra ubicación.

La persistencia es una característica de gran importancia dentro de la tecnología .NET, ya que, sin ir más lejos, uno de los componentes más importantes y novedosos de esta plataforma como son los servicios Web, hacen uso intensivo de ella.

En el presente artículo realizaremos una descripción de los aspectos principales de la persistencia en .NET Framework, dejando algunas cuestiones avanzadas para una próxima entrega.

## ¿Persistencia o serialización, qué término usar?

En la documentación disponible sobre la plataforma .NET, tanto la propia del producto como artículos, libros, etc., cuando se habla de persistencia de objetos, se utiliza el término serialización. El motivo es que en la tecnología .NET la serialización se define como el proceso de convertir un objeto en un flujo de bytes, y una vez que tenemos dicho flujo, podemos volcarlo a disco (hacerlo persistente), o enviarlo remotamente a otro proceso (técnica conocida también como Remoting).

## Deserialización, la resurrección de los objetos

Una vez que mediante la serialización hemos grabado un objeto a disco y lo hemos destruido, la deserialización sería el proceso inverso, es decir, durante la misma ejecución del programa en que habíamos serializado el objeto, o bien, en otra ejecución diferente en tiempo y/o lugar, podremos devolver a la vida el objeto del cual anteriormente habíamos grabado su estado.

## Tipos de serialización

Cuando serializamos un objeto, el proceso de generación del conjunto de bytes en que convertimos el objeto se denomina formateo del objeto. La plataforma .NET Framework proporciona al programador dos tipos de formateo para la serialización, con sus correspondientes clases.

- **Binaria**. Se utiliza para serializar un objeto en un formato compacto binario. La clase para generar este tipo de serialización es BinaryFormatter.

- **Soap-XML**. Se utiliza para serializar un objeto en un formato basado en XML. La clase para generar este tipo de serialización es SoapFormatter.

Serialización binaria

Como ejemplo de este tipo de serialización, vamos a desarrollar paso a paso un proyecto en el que haremos persistente un tipo nativo de la plataforma .NET Framework hacia un archivo, en concreto un objeto Hashtable.

En primer lugar, crearemos un proyecto de tipo Aplicación para Windows, al que daremos el nombre SerializBinaria, e insertaremos un control Button con el nombre btnSerializar, en el diseñador del formulario.

A continuación, abriremos el editor de código e importaremos los espacios de nombre mostrados en el siguiente código fuente.

**Espacios de nombre necesarios para serialización binaria**

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
```

El espacio de nombres System.Runtime.Serialization.Formatters.Binary contiene la clase BinaryFormatter, que como hemos dicho anteriormente, necesitaremos para formatear o serializar un objeto en un flujo binario.

Por otro lado, el espacio de nombres System.IO lo usaremos para crear un objeto Stream o flujo, encargado de guardar el objeto que serializaremos con el formateador binario.

El siguiente paso consistirá en escribir el manejador del evento Click del botón del formulario, en el que procederemos a realizar el proceso de serialización, que describimos en los siguientes puntos:

- Crear el objeto Hashtable a serializar, y rellenarlo de valores.
- Crear un nuevo archivo mediante un objeto FileStream.
- Instanciar un objeto BinaryFormatter, que será el que efectúe la serialización.
- Llamar al método Serialize del BinaryFormatter, pasándole como parámetros el objeto FileStream en el que se grabará la información, y el objeto Hashtable, que será el que hagamos persistente en el archivo.
- Cerrar el flujo de datos, obteniendo el archivo con el objeto serializado.

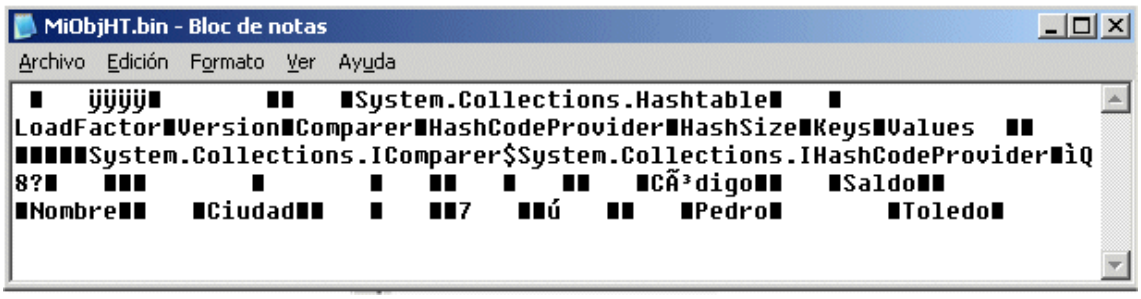
El código fuente que vemos a continuación, muestra el proceso que acabamos de explicar traducido a líneas de código.

**Proceso para serialización binaria de un objeto**

```
Private Sub btnSerializar_Click(ByVal sender As System.Object, _
```

```
            ByVal e As System.EventArgs) Handles btnSerializar.Click

  ' crear y rellenar objeto Hashtable
  Dim htPrueba As New Hashtable()
  htPrueba.Add("Código", 55)
  htPrueba.Add("Nombre", "Pedro")
  htPrueba.Add("Ciudad", "Toledo")
  htPrueba.Add("Saldo", 250)

  ' crear un stream para volcar el objeto en un archivo
  Dim oFlujo As New FileStream("C:\Pruebas\MiObjHT.bin", _
              FileMode.Create)


  ' crear un formateador binario para serializar el objeto
  Dim oFormateadorSerializ As New BinaryFormatter()

  ' serializar el objeto sobre el stream usando el formateador
  oFormateadorSerializ.Serialize(oFlujo, htPrueba)

  ' cerrar el stream
  oFlujo.Close()

  ' destruir el objeto Hashtable
  htPrueba = Nothing

  MessageBox.Show("Objeto serializado en archivo")

End Sub
```

Una vez serializado el objeto, finalizaremos la ejecución del programa, obteniendo como resultado el archivo que contiene el objeto que hemos hecho persistente. Si abrimos este archivo con el Bloc de notas, visualizaremos algo similar a la siguiente figura.




**Objeto Hashtable serializado a un archivo.**

Deserialización binaria

Una vez que hemos hecho persistente un objeto, podemos recuperarlo en cualquier momento en el mismo estado en que se encontraba cuando lo grabamos a disco.

Podríamos realizar este proceso inverso en el mismo proyecto de ejemplo que estábamos escribiendo, sin embargo, para que quede más patente esta recuperación del objeto, vamos a crear un proyecto nuevo con el nombre DeserializacionBin, que contenga en su formulario un control Button y un ListBox. El primero lo usaremos para deserializar el objeto grabado en el archivo, y el segundo para mostrar el contenido del objeto recuperado.

Al igual que en el anterior proyecto, importaremos los mismos espacios de nombre, que nos permitan manejar archivos y el formateador binario de conversión.

A continuación, escribiremos el código del botón del formulario, en el que abriremos con un tipo FileStream el archivo que contiene el objeto Hashtable, y recuperaremos este mediante el método BinaryFormatter.Deserialize.

El método Deserialize devuelve un tipo Object genérico de la plataforma .NET, y por este motivo, debemos realizar una operación de conversión de tipos (type-casting) con la función CType, sobre el valor resultante de Deserialize, para poder obtener el objeto Hashtable grabado en el archivo.

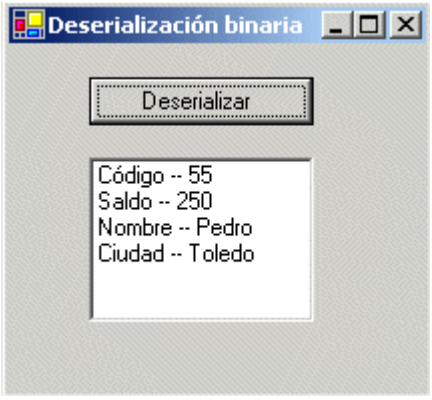
Por último, recorreremos el objeto Hashtable que hemos recuperado, y pasaremos sus valores al control ListBox. Veamos estas operaciones en el siguiente código fuente.

**Proceso para deserialización binaria de un objeto**

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Binary
Private Sub btnDeserializar_Click(ByVal sender As System.Object, _
            ByVal e As System.EventArgs) Handles btnDeserializar.Click
  ' abrir el archivo que contiene el objeto
  ' usando un stream
  Dim oFlujoLectura As New FileStream("C:\Pruebas\MiObjHT.bin", FileMode.Open)
  ' crear un formateador para deserializar el objeto
  ' que está en el archivo
  Dim oFormateadorDeserializ As New BinaryFormatter()
  ' declarar una variable del mismo tipo
  ' que el objeto que tenemos serializado
  Dim htDatos As Hashtable
  ' deserializar el objeto que está en el fichero-stream,
  ' y hacer un type casting del resultado a un objeto Hashtable
  htDatos = CType(oFormateadorDeserializ.Deserialize(oFlujoLectura), _
     Hashtable)
  ' recorrer el objeto deserializado
  ' y volcar sus elementos a un listbox del formulario
```

```
    Dim oEnumerador As IDictionaryEnumerator
    oEnumerador = htDatos.GetEnumerator()
    While oEnumerador.MoveNext()
        Me.lstDatos.Items.Add(oEnumerador.Key & " -- " & oEnumerador.Value)
    End While
End Sub
```

La siguiente figura muestra este formulario en ejecución, una vez que hemos deserializado el objeto, y volcado su contenido en el ListBox.




**Objeto Hashtable recuperado desde un archivo.**

Serialización XML

Acabamos de comprobar cómo la magia de la persistencia en .NET Framework nos permite recuperar directamente un objeto en un determinado estado, que habíamos grabado anteriormente en un soporte permanente como el disco.

Sin embargo, la serialización binaria, cuando se trata de transmitir un objeto serializado de forma remota a través de una red, no conjuga demasiado bien con los sistemas de seguridad (firewalls) de la misma, precisamente por el carácter binario de los datos enviados.

Para resolver este problema del envío de datos a través de redes, se ha adoptado el uso de XML, que al estar basado en texto plano, evita que los firewalls bloqueen la transmisión de la información que viaja en este formato.

Ya que XML es un lenguaje, y por sí solo no es suficiente para efectuar este envío de datos, se ha desarrollado el protocolo Soap (Simple Object Access Protocol), que define el conjunto de normas para el transporte de información en formato XML a través de Internet.

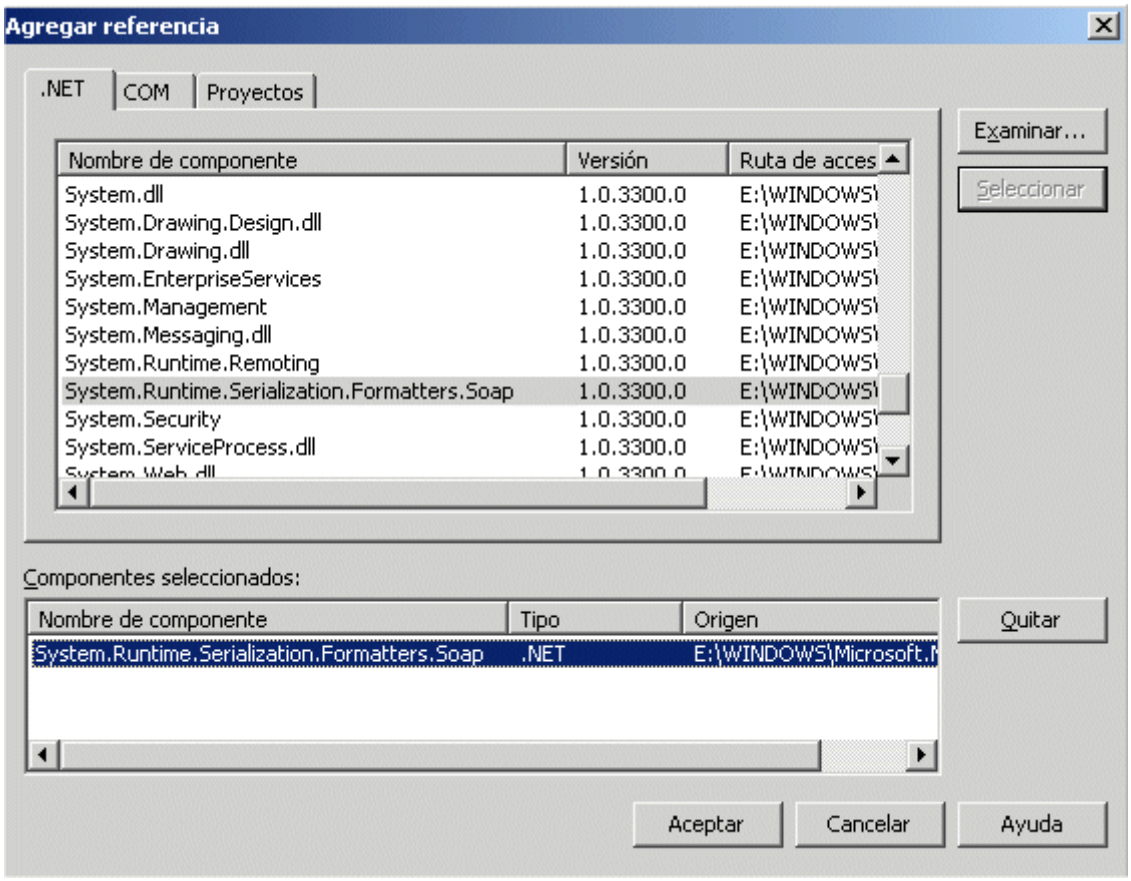
Por tal motivo, .NET Framework incluye las clases encargadas de trabajar con este protocolo en el espacio de nombres System.Runtime.Serialization.Formatters.Soap,

que deberemos importar en nuestro código cuando necesitemos hacer persistente un objeto que tengamos que enviar de forma remota por Internet o en una intranet, utilizando la serialización basada en XML, que explicaremos a continuación.

La descripción de Soap, debido a su extensión, se encuentra fuera del ámbito de este artículo, por lo que sugerimos al lector, que consulte la información que sobre este protocolo existe en la documentación de .NET Framework.

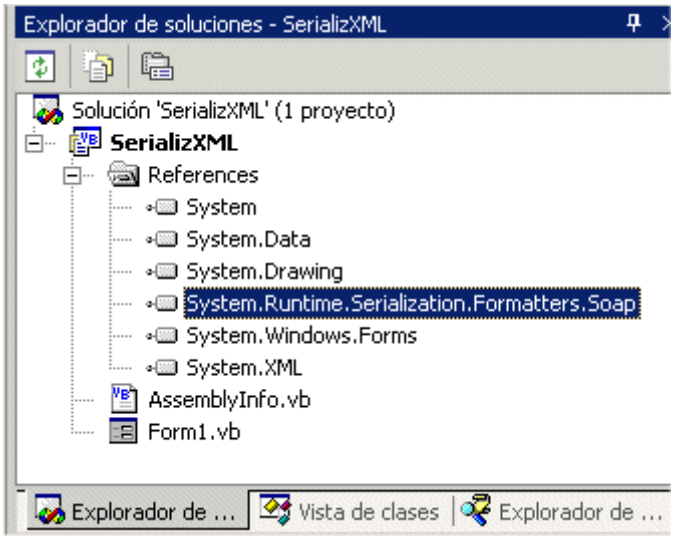
El primer paso, obviamente, consiste en tomar un objeto y hacerlo persistente hacia un archivo en disco, para lo que crearemos un proyecto con el nombre SerializXML, en el que añadiremos un botón al formulario, codificando en este control el proceso de serialización.

Es muy posible que el espacio de nombres System.Runtime.Serialization.Formatters.Soap, no esté directamente disponible en nuestro proyecto, a diferencia de lo que ocurría con el ejemplo anterior en binario. Por esta causa, deberemos situarnos en la ventana Explorador de soluciones de Visual Studio .NET, y hacer clic derecho en el nodo References, seleccionando la opción *Agregar referencia*, lo que abrirá una caja de diálogo en la que buscaremos y seleccionaremos el componente que contiene las clases Soap, como se muestra en la siguiente imagen.




**Agregar referencia de las clases Soap.**

Las referencias de nuestro proyecto, quedarán por lo tanto como se muestran en la siguiente imagen.




**Referencias del proyecto incluyendo el espacio de nombres de Soap.**

A continuación, escribiremos el código del botón del formulario con el proceso de serialización en XML, que como vemos en el siguiente código fuente, sigue las mismas pautas que el ejemplo con serialización binaria desarrollado anteriormente.

- Crear el objeto Hashtable a serializar, y rellenarlo de valores.
- Crear un nuevo archivo con extensión .xml mediante un objeto FileStream.
- Instanciar un objeto SoapFormatter, que será el que efectúe la serialización.
- Llamar al método SoapFormatter.Serialize, pasándole como parámetros el objeto FileStream en el que se grabará la información, y el objeto Hashtable, que será el que hagamos persistente en el archivo.
- Cerrar el flujo de datos, obteniendo el archivo con el objeto serializado.

**Proceso para serialización XML de un objeto**

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap
'....

Private Sub btnSerializar_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnSerializar.Click
   ' crear y rellenar objeto Hashtable
   Dim htPrueba As New Hashtable()
   htPrueba.Add("Código", 55)
   htPrueba.Add("Nombre", "Pedro")
   htPrueba.Add("Ciudad", "Toledo")
   htPrueba.Add("Saldo", 250)
   ' crear un stream para volcar el objeto en un archivo
   Dim oFlujo As New FileStream("C:\Pruebas\DatosObjeto.xml", _
```

```
    FileMode.Create)

  ' crear formateador soap-xml
  Dim oFormatXML As New SoapFormatter()

  ' serializar el objeto hacia el stream
  oFormatXML.Serialize(oFlujo, htPrueba)
  oFlujo.Close()
End Sub
```

En esta ocasión el resultado será más legible que en el caso anterior. Si hacemos doble clic en el archivo XML generado, se abrirá nuestro navegador de Internet, mostrándonos el objeto serializado en este formato. La siguiente imagen muestra un fragmento del código XML generado, en el que podemos ver cómo nuestro objeto Hashtable ha sido organizado en este formato.

```
- <SOAP-ENV:Body>
  - <a1:Hashtable id="ref-1"
      xmlns:a1="http://schemas.microsoft.com/clr/ns/System.Collections">
      <LoadFactor>0.72</LoadFactor>
      <Version>4</Version>
      <Comparer xsi:null="1" />
      <HashCodeProvider xsi:null="1" />
      <HashSize>11</HashSize>
      <Keys href="#ref-2" />
      <Values href="#ref-3" />
    </a1:Hashtable>
  - <SOAP-ENC:Array id="ref-2" SOAP-ENC:arrayType="xsd:anyType[4]">
      <item id="ref-4" xsi:type="SOAP-ENC:string">Código</item>
      <item id="ref-5" xsi:type="SOAP-ENC:string">Saldo</item>
      <item id="ref-6" xsi:type="SOAP-ENC:string">Nombre</item>
      <item id="ref-7" xsi:type="SOAP-ENC:string">Ciudad</item>
    </SOAP-ENC:Array>
  - <SOAP-ENC:Array id="ref-3" SOAP-ENC:arrayType="xsd:anyType[4]">
      <item xsi:type="xsd:int">55</item>
      <item xsi:type="xsd:int">250</item>
      <item id="ref-8" xsi:type="SOAP-ENC:string">Pedro</item>
      <item id="ref-9" xsi:type="SOAP-ENC:string">Toledo</item>
    </SOAP-ENC:Array>
  </SOAP-ENV:Body>
```

**Código XML del objeto serializado.**

Deserialización XML

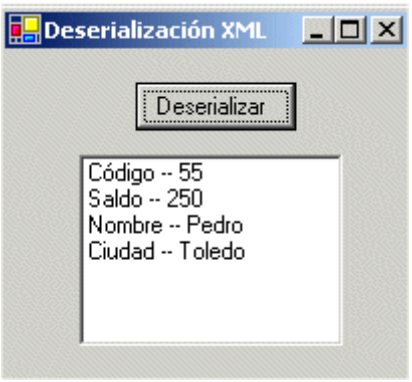
Tras haber hecho persistente el objeto en XML, supongamos que lo hemos enviado a una ubicación remota de la red, y al llegar a su destino, existe otra aplicación que se encarga de recuperar el objeto a partir del archivo XML.

Vamos por lo tanto, a crear un proyecto con el nombre DeserializXML, en el que añadiremos un control Button y un ListBox al formulario. En el botón escribiremos el siguiente código fuente, que mediante un objeto SoapFormatter, se encargará de abrir el archivo XML, y recuperar el objeto Hashtable que contiene. Finalmente recorreremos los valores del Hashtable, rellenando el ListBox.

**Proceso para deserialización XML de un objeto**

```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap
'....
Private Sub btnDeserializar_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnDeserializar.Click
  ' abrir el archivo que contiene el objeto
  ' usando un stream
  Dim oFlujoLectura As New FileStream("C:\Pruebas\DatosObjeto.xml", _
     FileMode.Open)
  ' crear un formateador soap-xml
  ' y un Hashtable para volcar el
  ' objeto serializado en el archivo
  Dim oFormatXML As New SoapFormatter()
  Dim htPrueba As Hashtable
  ' deserializar el objeto que está en el fichero-stream,
  ' y hacer un type casting del resultado a un objeto Hashtable
  htPrueba = CType(oFormatXML.Deserialize(oFlujoLectura), _
     Hashtable)
  oFlujoLectura.Close()
  ' recorrer el objeto recuperado mediante un enumerador,
  ' y visualizar en el listbox del formulario
  Dim oEnumerador As IDictionaryEnumerator = htPrueba.GetEnumerator()
  While (oEnumerador.MoveNext())
     Me.lstDatos.Items.Add(oEnumerador.Key & " -- " & oEnumerador.Value)
  End While
End Sub
```

La siguiente imagen, muestra el formulario una vez que el objeto ha sido deserializado y volcado al ListBox.




**Resultado de deserializar un objeto de un archivo XML.**

Bien, pues esto ha sido todo, a lo largo de este artículo hemos descrito en qué consiste la persistencia de objetos, y cómo .NET Framework proporciona esta técnica al programador. Igualmente hemos explicado los tipos de persistencia disponibles en la plataforma .NET, ilustrando con ejemplos prácticos la forma de hacer persistentes y recuperar nuestros propios objetos. Larga vida (persistencia) a los objetos.