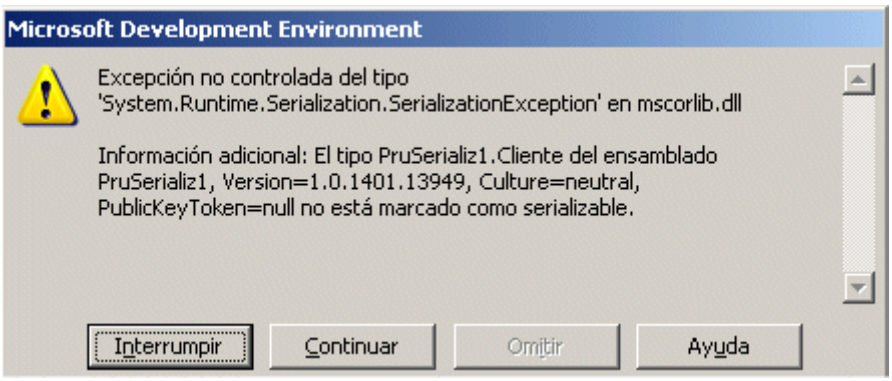
Serialización o persistencia de objetos (II)

Luis Miguel Blanco Ancos

En la primera parte de este artículo realizamos una introducción a la persistencia de objetos en .NET Framework, presentando al lector los diversos tipos de serialización disponibles en la plataforma. En esta segunda entrega abordaremos algunos aspectos avanzados, tales como la persistencia aplicada a las clases propias del programador, y a las clases interrelacionadas, de manera que podamos aplicar estas interesante características a nuestros propios objetos.

Serialización de clases propias

Por defecto, .NET Framework no proporciona directamente capacidades de serialización a las clases que pueda crear el programador, por lo que si intentamos serializar directamente un objeto de una clase propia, se producirá una excepción de tipo SerializationException, que nos informará de que el objeto no está marcado como serializable, como vemos en la siguiente imagen.



Excepción al intentar serializar un objeto de una clase propia.

Para poder hacer persistente uno de nuestros propios objetos, debemos aplicar a su clase el atributo Serializable, como vemos en el siguiente fuente.

Marcar clase como serializable

```
<Serializable()> _
Public Class Cliente
   ' campos públicos
   Public Codigo As Integer
   Public Nombre As String
```

```
  Public FechaNacim As Date
  Public Ciudad As String
  ' variables de propiedad
  Private mSaldo As Integer
  Public Property Saldo() As Integer
    Get
      Return mSaldo
    End Get
    Set(ByVal Value As Integer)
      mSaldo = Value
    End Set
  End Property
End Class
```

Una vez declarada la clase Cliente con este atributo, ya es posible hacer persistente sus objetos tanto de forma binaria como XML, al igual que con objetos nativos de .NET como Hashtable.

Control de los miembros serializables de una clase

El atributo Serializable permite, como hemos explicado en el apartado anterior, la posibilidad de hacer persistente una clase propia. Sin embargo, puede que en determinadas circunstancias no nos interese que todos los miembros de la clase sean serializados, existiendo algunos que deseemos mantener en el anonimato.

Para que un determinado miembro de una clase no sea serializado, debemos aplicarle el atributo NonSerialized, con lo cual, la información de dicho miembro no será hecha persistente al serializar el objeto al que pertenece.

Supongamos que en la clase Cliente del ejemplo anterior, no queremos que el campo Ciudad sea serializado. Para ello marcaremos este campo con el atributo NonSerialized como vemos en el siguiente código fuente.

Marcar miembro de clase como no serializable

```
<Serializable()> _
Public Class Cliente
  ' campos públicos
  '....
  <NonSerialized()> Public Ciudad As String
  '....
End Class
```

Tras efectuar estos ajustes sobre la clase, podemos crear un formulario con dos botones, uno se encargará de crear un objeto Cliente y hacerlo persistente, y el otro de recuperar dicho objeto. A continuación se muestra el código de cada uno de estos botones.

Botones de formulario para serializar/deserializar un objeto propio

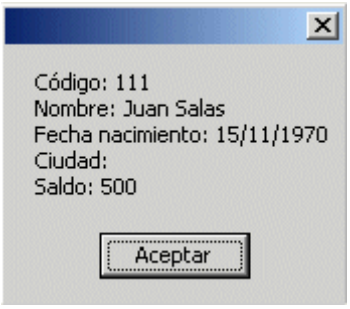
```
Imports System.IO
Imports System.Runtime.Serialization.Formatters.Soap
'....
Private Sub btnSerializar_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnSerializar.Click
   ' instanciar objeto a serializar
   Dim oCliente As New Cliente()
   oCliente.Codigo = 111
   oCliente.Nombre = "Juan Salas"
   oCliente.FechaNacim = "15/11/1970"
   oCliente.Ciudad = "Sevilla"
   oCliente.Saldo = 500
   ' crear archivo para grabar el objeto
   Dim oStream As Stream
   oStream = New FileStream("C:\Pruebas\MiObjPersonaliz.xml", _
      FileMode.Create)
   ' crear formateador soap-xml y hacer
   ' persistente el objeto
   Dim oFormatXML As New SoapFormatter()
   oFormatXML.Serialize(oStream, oCliente)
   oStream.Close()
   MessageBox.Show("objeto serializado")
End Sub
Private Sub btnDeserializar_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnDeserializar.Click
   ' abrir el archivo
   Dim oFS As FileStream = File.OpenRead("C:\Pruebas\MiObjPersonaliz.xml")
   ' crear formateador
   Dim oFormatXML As New SoapFormatter()
   ' recuperar el objeto que está en el archivo
   Dim oCliente As Cliente
   oCliente = CType(oFormatXML.Deserialize(oFS), Cliente)
   oFS.Close()
   MessageBox.Show("Código: " & oCliente.Codigo & ControlChars.CrLf & _
      "Nombre: " & oCliente.Nombre & ControlChars.CrLf & _
      "Fecha nacimiento: " & oCliente.FechaNacim & ControlChars.CrLf & _
      "Ciudad: " & oCliente.Ciudad & ControlChars.CrLf & _
      "Saldo: " & oCliente.Saldo)
End Sub
```

Al ejecutar este ejemplo, cuando pulsemos el botón que serializa el objeto en un archivo xml, el contenido de dicho archivo carecerá de la información referente al campo Ciudad del objeto Cliente, como vemos en la siguiente imagen.

```
- <SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
    ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  - <SOAP-ENV:Body>
    - <a1:Cliente id="ref-1"
        xmlns:a1="http://schemas.microsoft.com/clr/nsassem/SerializPersonalizada
        2C%20Version%3D1.0.1450.28763%2C%20Culture%3Dneutral%2C%
        20PublicKeyToken%3Dnull">
        <Codigo>111</Codigo>
        <Nombre id="ref-3">Juan Salas</Nombre>
        <FechaNacim>1970-11-15T00:00:00.0000000+01:00</FechaNacim>
        <mSaldo>500</mSaldo>
      </a1:Cliente>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Archivo xml con la información del objeto Cliente, pero sin el campo Ciudad.

Al recuperar el objeto hecho persistente, comprobaremos que la información referente al campo Ciudad no existe, como consecuencia de haber aplicado el atributo NonSerialized. Veamos el resultado en la siguiente imagen.



El campo Ciudad del objeto Cliente no ha sido serializado.

Gracias a estos atributos, se facilita enormemente la gestión y administración de la persistencia para las clases que el programador diseñe con finalidades de serialización.

ISerializable. Manipulación avanzada de la serialización

Supongamos que al hacer persistente un objeto, queremos grabar, además de los campos o propiedades que el objeto pudiera tener, información adicional generada en tiempo de ejecución en base a los datos existentes en el objeto. En definitiva, se trataría de poder controlar el proceso de serialización.

Pues bien, esto es posible a través de la interfaz ISerializable, que se encuentra en el espacio de nombres System.Runtime.Serialization. Si al crear una clase, implementamos en ella dicho interfaz, el motor de ejecución crea internamente para la clase un objeto de tipo SerializationInfo, que a modo de bolsa de propiedades, utilizaremos para guardar la información del objeto.

Para guardar los valores de nuestro objeto (tanto de campos públicos y propiedades, como calculados en tiempo de ejecución) en el tipo SerializationInfo, necesitaremos escribir un método con el nombre GetObjectData; este método será llamado al ejecutar el método Serialize del formateador.

El método GetObjectData es la clave de este mecanismo, ya que en él es donde controlamos qué elementos de nuestro objeto hacemos persistentes y cómo.

En cuanto al modo de recuperación de la información serializada, debemos escribir en nuestra clase un constructor con ámbito privado, el cual también use el tipo SerializationInfo, para devolver los valores de nuestro objeto que anteriormente hicimos persistentes. Este constructor será llamado automáticamente por el formateador al ejecutar el método Deserialize.

Obviamente, la clase que escribamos con todas estas características, deberá estar marcada con el atributo Serializable, debiendo importar los espacios de nombre correspondientes a las operaciones de serialización.

Siguiendo con la clase Cliente que hemos usado en nuestros ejemplos sobre serialización, supongamos que al hacer persistente un objeto de esta clase, queremos grabar también el nombre del mes de la fecha de nacimiento, el nombre del cliente en mayúsculas, y un descuento basado en el saldo. Estos valores, no obstante, no van a ser propiedades de nuestro objeto, sino que los vamos a generar en tiempo de ejecución partiendo de la información base que ya tiene el objeto en sus campos y propiedades.

En primer lugar vamos a crear un nuevo proyecto, en el que escribiremos el código de la clase Cliente que vemos en el siguiente fuente. Como podemos observar, se implementa la interfaz ISerializable, y codificamos los métodos que hemos explicado anteriormente.

Clase Cliente que implementa la interfaz ISerializable

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

<Serializable()> _
Public Class Cliente
  Implements ISerializable

  ' campos públicos
  Public Codigo As Integer
  Public Nombre As String
  Public FechaNacim As Date

  ' campos privados para valores calculados
  Private mMesAlta As String
  Private mMayusculas As String
```

```
Private mDescuento As Single

' variables de propiedad
Private mSaldo As Integer

Public Property Saldo() As Integer
   Get
      Return mSaldo
   End Get
   Set(ByVal Value As Integer)
      mSaldo = Value
   End Set
End Property

' cuando llamemos al método Serialize del formateador
' para hacer persistente este objeto, se llamará a este
' método, en el cual guardaremos en el parámetro info
' los valores del objeto que vamos a hacer persistentes;
' debemos observar que se guardan la información típica
' de la clase: campos públicos y propiedades, y también
' valores generados en tiempo de ejecución en este método
Public Sub GetObjectData(ByVal info _
   As System.Runtime.Serialization.SerializationInfo, _
   ByVal context As System.Runtime.Serialization.StreamingContext) _
   Implements System.Runtime.Serialization.ISerializable.GetObjectData

   info.AddValue("Codigo", Codigo)
   info.AddValue("Nombre", Nombre)
   info.AddValue("Saldo", Me.Saldo)
   info.AddValue("FechaNacim", FechaNacim)
   info.AddValue("MesAlta", FechaNacim.ToString("MMMM"))
   info.AddValue("Mayusculas", Nombre.ToUpper())
   info.AddValue("Descuento", ((Me.Saldo * 2) / 100))
End Sub

' al implementar en la clase un constructor privado,
' es necesario crear al menos un constructor público,
' aunque no tenga código
Public Sub New()

End Sub

' cuando el método Deserialize del formateador
' instancie este objeto, tomará de la bolsa de
' propiedades del parámetro info, todos los valores
' que hicimos persistentes, tanto propiedades normales
' como valores calculados
```

```
  Private Sub New(ByVal info _
    As System.Runtime.Serialization.SerializationInfo, _
    ByVal context As System.Runtime.Serialization.StreamingContext)

    Codigo = info.GetInt32("Codigo")
    Nombre = info.GetString("Nombre")
    Saldo = info.GetInt32("Saldo")
    FechaNacim = info.GetDateTime("FechaNacim")
    mMesAlta = info.GetString("MesAlta")
    mMayusculas = info.GetString("Mayusculas")
    mDescuento = info.GetSingle("Descuento")
  End Sub

  ' este método devuelve valores del objeto
  ' creados en el momento de serializar el objeto
  Public Function ValoresCalculados() As String
    Dim sDatosCalc As String

    sDatosCalc = "MesAlta: " & mMesAlta & ControlChars.CrLf
    sDatosCalc &= "Mayúsculas: " & mMayusculas & ControlChars.CrLf
    sDatosCalc &= "Descuento: " & mDescuento

    Return sDatosCalc

  End Function

End Class
```

Comentemos brevemente los elementos principales del fuente anterior:

- En el método GetObjectData, mediante el parámetro *info*, de tipo SerializationInfo, añadimos los valores usando el método AddValue, indicando un nombre para el valor, y el propio valor a guardar.

- En el constructor privado (el que contiene la lista de parámetros), usando también el parámetro *info*, de tipo SerializationInfo, recuperamos la información utilizando los mismos nombres que empleamos en GetObjectData.

- El método ValoresCalculados será explicado posteriormente. Baste por el momento con saber que lo utilizaremos para recuperar información del objeto.

Seguidamente, al igual que en otros ejemplos, añadiremos al formulario del proyecto dos botones, uno para crear un objeto, y hacerlo persistente en un archivo xml con un formateador Soap, y otro para recuperar el objeto del archivo xml.

A continuación se muestra el código fuente correspondiente al botón usado para serializar el objeto.

Botón del formulario para serializar un objeto de la clase Cliente

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

Private Sub btnSerializar_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnSerializar.Click
   ' crear objeto
   Dim oCliente As New Cliente()
   oCliente.Codigo = 951
   oCliente.Nombre = "Elena Peral"
   oCliente.FechaNacim = "20/03/2002"
   oCliente.Saldo = 1275

   ' crear stream
   Dim oStream As Stream
   oStream = New FileStream("C:\Pruebas\DatosObj.xml", _
      FileMode.Create)

   ' crear formateador
   Dim oFormatXml As IFormatter
   oFormatXml = New SoapFormatter()

   ' serializar; aquí se llama al método GetObjectData
   ' del objeto oCliente
   oFormatXml.Serialize(oStream, oCliente)
   oStream.Close()

   MessageBox.Show("Objeto serializado")
End Sub
```

En la siguiente imagen podemos ver el fragmento del archivo xml con la información serializada del objeto Cliente. Observemos cómo además de los campos y propiedades, los valores calculados adicionales también se han añadido.

```
- <a1:Cliente id="ref-1"
    xmlns:a1="http://schemas.microsoft.com/clr/nsassem
    2C%20Version%3D1.0.1450.28748%2C%20Culture%3
    20PublicKeyToken%3Dnull">
    <Codigo>951</Codigo>
    <Nombre id="ref-3">Elena Peral</Nombre>
    <Saldo>1275</Saldo>
    <FechaNacim xsi:type="xsd:dateTime">2002-03-
      20T00:00:00.0000000+01:00</FechaNacim>
    <MesAlta id="ref-4">marzo</MesAlta>
    <Mayusculas id="ref-5">ELENA PERAL</Mayusculas>
    <Descuento>25.5</Descuento>
  </a1:Cliente>
```

Datos serializados del objeto Cliente.

Seguidamente se muestra el código del botón encargado de deserializar el objeto del archivo, y recuperarlo en el programa.

Botón del formulario para deserializar un objeto de la clase Cliente

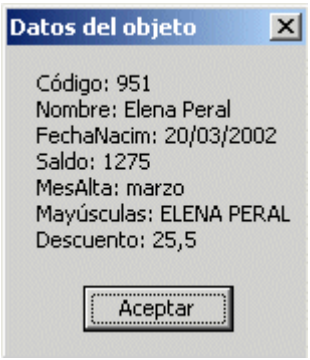
```
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
'....

Private Sub btnDeserializar_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnDeserializar.Click
   ' abrir el stream
   Dim oFS As FileStream = File.OpenRead("C:\Pruebas\DatosObj.xml")

   ' crear formateador xml
   Dim oFormatXML As IFormatter = New SoapFormatter()

   Dim oCliente As Cliente
   ' deserializar; aquí se instancia un objeto
   ' de la clase Cliente, y el formateador
   ' llama al constructor privado del objeto Cliente
   oCliente = CType(oFormatXML.Deserialize(oFS), Cliente)
   oFS.Close()

   ' mostrar el contenido del objeto que estaba en el archivo
   Dim sDatosObj As String
   sDatosObj = "Código: " & oCliente.Codigo & ControlChars.CrLf
   sDatosObj &= "Nombre: " & oCliente.Nombre & ControlChars.CrLf
   sDatosObj &= "FechaNacim: " & oCliente.FechaNacim & ControlChars.CrLf
   sDatosObj &= "Saldo: " & oCliente.Saldo & ControlChars.CrLf
   sDatosObj &= oCliente.ValoresCalculados()

   MessageBox.Show(sDatosObj, "Datos del objeto")
```

```
End Sub
```

Vamos a explicar ahora el propósito del método ValoresCalculados del objeto Cliente, que dejamos pendiente con anterioridad.

Cuando deserializamos el objeto Cliente, tenemos acceso directo desde el exterior del objeto a los miembros públicos del mismo. Sin embargo, los valores que calculamos en tiempo de ejecución al hacer persistente el objeto en el método GetObjectData, correspondientes a los campos MesAlta, Mayúsculas y Descuento, al tener estos ámbito privado, no son accesibles desde el código externo al objeto. Por este motivo, escribimos el método ValoresCalculados, que se encarga de tomar estos campos y devolverlos concatenados.

El resultado, al deserializar el objeto, y *devolverlo a la vida*, se muestra en la siguiente imagen.



Objeto deserializado de la clase Cliente, que implementa la interfaz ISerializable.

Gráfica de objetos. Tratamiento de la persistencia en objetos interdependientes

Probablemente el lector se haya preguntado cómo debe actuar cuando está trabajando con un objeto que a su vez tiene una referencia hacia otro objeto, y necesita serializar ambos, ¿debe serializarlos manualmente por separado?. Afortunadamente esto no es necesario, ya que la plataforma .NET Framework incorpora en su mecanismo de persistencia la capacidad de detectar cuándo un objeto depende de otro, para serializar todos automáticamente sin necesidad de que el programador tenga que escribir código extra para controlar los detalles de este proceso.

Propongamos la siguiente situación: creamos un proyecto en el que escribimos las clases Empresa, Empleado y Salario. Uno de los miembros de Empresa es un objeto Empleado, y a su vez, uno de los miembros de Empleado es un objeto Salario, con lo que establecemos una relación de interdependencia entre los objetos, como se muestra en el siguiente código fuente.

Clases con miembros interdependientes

```
<Serializable()> _
```

```
Public Class Empresa
   Private msCIF As String
   Private msNombre As String
   Private moEmpleado As Empleado

   Public Property CIF() As String
      Get
         Return msCIF
      End Get
      Set(ByVal Value As String)
         msCIF = Value
      End Set
   End Property

   Public Property Nombre() As String
      Get
         Return msNombre
      End Get
      Set(ByVal Value As String)
         msNombre = Value
      End Set
   End Property

   Public Sub New()
      msCIF = "A58111222"
      msNombre = "Gestoría Montes"
      moEmpleado = New Empleado()
   End Sub
End Class

'---------------------------------
<Serializable()> _
Public Class Empleado
   Private msNomApe As String
   Private mdtFechaAlta As DateTime
   Private moSalario As Salario

   Public Property NombreApellidos() As String
      Get
         Return msNomApe
      End Get
      Set(ByVal Value As String)
         msNomApe = Value
      End Set
   End Property

   Public Property FechaAlta() As DateTime
```

```
      Get
         Return mdtFechaAlta
      End Get
      Set(ByVal Value As DateTime)
         mdtFechaAlta = Value
      End Set
   End Property

   Public Sub New()
      msNomApe = "Juan Naranjo"
      mdtFechaAlta = New DateTime(2004, 1, 18)
      moSalario = New Salario()
   End Sub
End Class

'---------------------------------
<Serializable()> _
Public Class Salario
   Private mnCantidad As Integer

   Public Property Cantidad() As Integer
      Get
         Return mnCantidad
      End Get
      Set(ByVal Value As Integer)
         mnCantidad = Value
      End Set
   End Property

   Public Sub New()
      mnCantidad = 587
   End Sub

End Class
```

Cuando hagamos persistente a disco el objeto Empresa con un código similar al siguiente, el mecanismo de serialización detectará los miembros de dicho objeto que a su vez son objetos encadenados y los serializará igualmente.

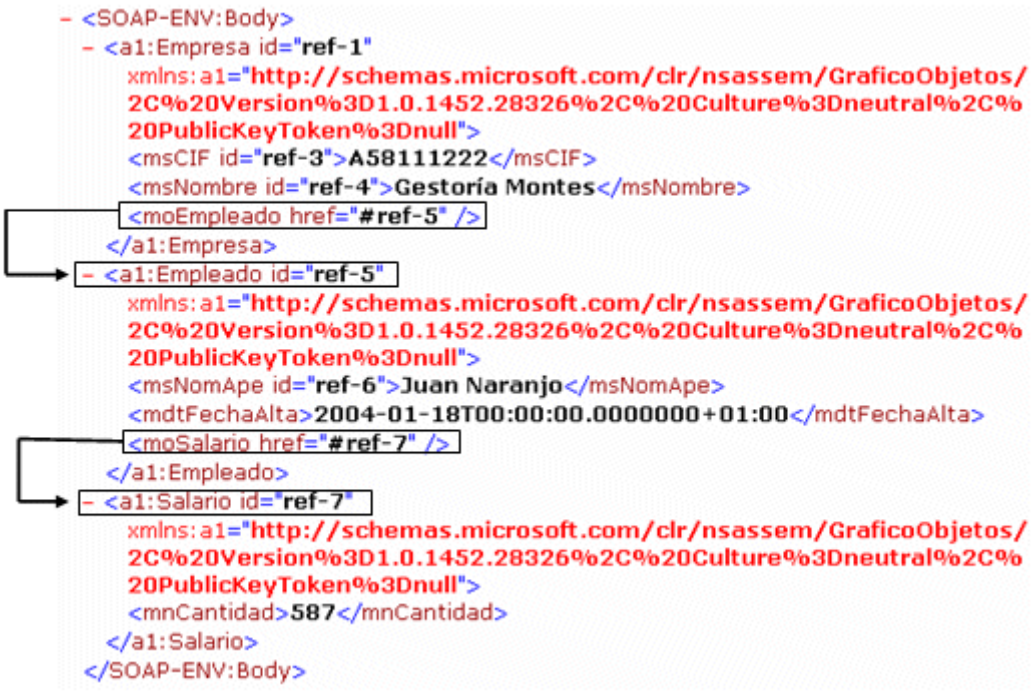
Clases con miembros interdependientes

```
Imports System.IO
Imports System.Runtime.Serialization
Imports System.Runtime.Serialization.Formatters.Soap
'....
'....
Dim oEmpresa As New Empresa()
```

```
' pasar el objeto Empresa y los dependientes a un archivo:
' crear un Stream, un formateadorXML y hacer persistente
' el objeto

' objeto stream
Dim oFS As New FileStream("C:\Pruebas\MisObjetos.xml", _
   FileMode.Create)

' objeto formateador
Dim oFormateadorXml As IFormatter
oFormateadorXml = New SoapFormatter()

' serializar
oFormateadorXml.Serialize(oFS, oEmpresa)

' cerrar el stream
oFS.Close()
```

Al hacer persistente un objeto, el sistema de serialización de .NET Framework asigna a cada objeto un identificador único que le permite ser reconocido de forma unívoca. Si serializamos los objetos como en el ejemplo anterior, utilizando un formateador SOAP, al observar en el archivo xml resultante, las etiquetas correspondientes a cada uno de estos objetos, veremos que tienen un identificador asignado.

Visualizando el archivo xml al completo, podemos comprobar como cada miembro de un objeto que a su vez es otro objeto, está referenciado a través del atributo href mediante el valor correspondiente al identificador único asignado al objeto. En la siguiente imagen se pueden observar estas relaciones remarcadas.



Dentro del contexto de la serialización, a este modo de establecer las relaciones entre objetos se le denomina Gráfica de objetos (Object Graph).

Concluyendo

Y con esto terminamos. En el presente artículo hemos abordado las técnicas necesarias para aplicar la serialización a nuestras clases, y el modo en cómo .NET Framework organiza los objetos relacionados al hacerlos persistentes. Espero que sea de utilidad para todos los lectores. Un saludo.