Artículo **Revista de Sistemas Computacionales y TIC's**
Junio 2016 Vol.2 No.4 58-67

# La serialización de datos utilizando un Framework de desarrollo integrado

OCHOA, Raquel*†

*Instituto Tecnológico de Ciudad Guzmán, Tecnológico Nacional de México, México*

Recibido Abril 4, 2016; Aceptado Junio 7, 2016

**Resumen**

Esta investigación analiza el concepto de serialización de datos y los tipos de formatos permitidos en el Framework de Visual Studio. La serialización proporciona un soporte para manipular información contenida en un objeto y transferirla a otras aplicaciones. Una de las mayores ventajas de la serialización es que proporciona almacenamiento e intercambio de datos, manteniendo además información importante entre aplicaciones. Si la información se transmite en binario o XML se facilita la comunicación entre datos, permitiendo la migración de los mismos.

**Framework, Serialización, Datos, Aplicaciones, Formatos**

**Abstract**

This research analyzes the concept of serialization of data types and formats allowed in the Framework of Visual Studio. Serialization provides support for manipulating information in an object and transfer it to other applications. One of the biggest advantages of serialization is that it provides storage and data exchange also keeping important information between applications. If the information is transmitted in binary or XML data communication between is provided, allowing the migration thereof.

**Framework, serialization, Data, Applications, Formats**

**Citación:** OCHOA, Raquel. La serialización de datos utilizando un Framework de desarrollo integrado. Revista de Sistemas Computacionales y TIC'S 2016, 2-4: 58-67

* Correspondencia al Autor (Correo Electrónico: raqueoo@itcg.edu.mx)
† Investigador contribuyendo como primer autor.

 

## Introducción

La serialización se pude definir como un proceso que guarda el estado de un objeto en un medio de almacenamiento. Durante este proceso los atributos públicos y privados del objeto, así como el nombre de la clase, se convierten en un formato ya sea binario, XML o SOAP. Al proceso inverso se denomina deserialización, donde se crea un objeto exacto al original.

La serialización o *marshalling* es un proceso de codificación de un objeto a un medio de almacenamiento, para transmitirlo por red como un conjunto de bytes o en formatos XML, JSON o SOAP. De esta manera, los datos transportados pueden ser utilizados para crear un nuevo objeto idéntico al original. La serialización resulta ser un mecanismo para transportar objetos a través de la red, con el fin de hacer persistente un objeto en un archivo y ser distribuido en aplicaciones. Los datos que son serializados son fáciles de transportar, representar y almacenar.

## Marco Teórico

El concepto de serialización se define como el proceso de convertir un objeto o estructura de objetos en un formato leíble y transportable a otras aplicaciones. La deserialización permite convertir de un formato leíble a su formato original (Díaz, 2012). La serialización consiste en transformar una variable, ya sea tabla o un objeto, en una cadena de caracteres (Vigouroux,, 2015).

La serialización sirve para convertir un objeto a una secuencia de bytes y trasmitirlo a un archivo, base de datos o memoria. El objeto serializado contiene además información sobre el tipo de objeto. El estado del objeto es guardado con la finalidad de crearlo de nuevo (Torres, 2013).

## El sistema de archivos

### La clase *FileStream*

En Visual Studio se usa para leer y escribir en archivos binarios, donde se debe especificar el modo de apertura, el tipo de acceso y el tipo de bloqueo. Permite además operaciones de lectura y escritura síncrona y asíncrona en un archivo. Admite el acceso aleatorio mediante el método Seek para grabar o leer en cualquier posición dentro del archivo (Hugon, 2014). En la Tabla 1 se muestra la descripción de los métodos principales de la clase *FileStream*:

| Métodos | Descripción |
|---|---|
| FileStream(String, FileMode, FileAccess, FileShare) | Inicializa un objeto de la clase *FileStream* con ruta de acceso, modo de creación. |
| Close | Cierra la secuencia actual y libera todos los recursos asociados. |
| Read | Lee un bloque de bytes de la secuencia y escribe datos en un búfer determinado. |
| Seek | Establece la posición actual de esta secuencia actual en el valor dado. |
| Write | Escribe un bloque de bytes en esta secuencia mediante el uso de datos de un búfer |

**Tabla 1** Métodos principales de la clase FileStream

Para la creación de un archivo que devuelve un objeto de tipo *FileStream* se especifican ciertos argumentos presentados en la Tabla 2 (Hugon, 2014):

ISSN-2444-5002

| Parámetro | Descripción | Métodos |
|---|---|---|
| FileMode | Especifica la manera de abrir el archivo. | Append<br>Create<br>CreateNew<br>Open<br>OpenOrCreate<br>Truncate |
| FileAccess | Especifica los tipos de operación. | Read<br>ReadWrite<br>Write |
| FileShared | Define uno o varios tipo de acceso. | None<br>Read<br>Write<br>ReadWrite<br>Delete<br>Inheritable |
| FileInfo | Proporciona propiedades y métodos de instancia y contribuye a la creación de objetos. | Equals<br>Reference - Equals |
| FileOptions | Representa opciones avanzadas para crear un objeto. | Asynchronous<br>DeleteOnClose<br>Encrypted<br>None<br>RandomAccess<br>SequentialScan<br>WriteThrough |
| FileSecurity | Determina el control de acceso y auditoría del archivo. | |

**Tabla 2** Parámetros del objeto FileStream

## TextReader y TextWriter

La clase *StringReader* accede al contenido de una variable de tipo *String*. *StringWriter* requiere ser del tipo *StringBuilder*. *StringBuilder* es un búfer con datos. Con las clases *StringReader/StringWriter* las cadenas de texto pueden ser modificadas, accedidas línea a línea o en su totalidad.

Para la transferencia de datos se utilizan los objetos *Stream*, como una fuente externa y la aplicación para la lectura de datos o al contrario para la escritura de datos. *FileStream* es una clase que se utiliza para escribir y leer datos. Las clases *StreamReader* y *StreamWriter* están disponibles para leer y escribir archivos de texto (Hugon, 2014).

En la Tabla 3 se muestra una descripción de las clases derivadas de *TextReader* y *TextWrite*:

| Clase base | Clase derivada | Utilidad |
|---|---|---|
| TextReader | StreamReader | Lee caracteres o líneas de un archivo |
| | StringReader | Lee caracteres o líneas de una cadena de texto de tipo String |
| TextWrite | StreamWriter | Escribe caracteres o líneas a un archivo de texto |
| | StringWriter | Escribe caracteres a una variable de tipo StringBuilder. |

**Tabla 3** Clases derivadas de TextReader o TextWriter

Clases derivadas de *Stream*:

- System.IO.Buffered.Stream

  Adiciona un búfer de lectura para ser utilizado con otro *Stream*.

- System.IO.FileStream

  Accesa a un archivo tanto de forma asíncrona como síncrona.

- System.Net.Sockets.Network.Stream

  Permite gestionar información desde y hacia una red.

- System.Security.Cryptography.
  CryptoStream

  Aplica un cifrado a un conjunto de datos.

En la Tabla 4 se presenta una lista de las excepciones en el acceso a archivos, (Microsoft, 2016f):

| Nombre | Descripción |
|---|---|
| DirectoryNotFoundException | La carpeta no existe. |
| EndOfStreamException | Se ha sobrepasado el final del contenido. |
| FileNotFoundException | El archivo no existe. |
| PathTooLongException | La longitud de la ruta o archivo excede el largo máximo definido por el sistema operativo. |

**Tabla 4** Excepciones StreamReader/StreamWriter

**Elementos que intervienen en la serialización**

Para efectuar la serialización o deserialización son necesarios: un objeto a ser serializado, un stream conteniendo el objeto serializado y un formateador para serializar la stream. Un formateador permite determinar el formato de la serialización (León, 2016).

Los espacios de nombres en el Framework de Visual Studio 2015 utilizados para los diferentes formatos en la serialización, son los siguientes: (Microsoft, 2016g):

- Binario
  System.Runtime.Serialization.Formatters. Binary.BinaryFormatter

- SOAP
  System.Runtime.Serialization.Formatters. Soap
- XML
  System.Xml.Serialization.XmlSerializer
  System.Runtime.Serialization.DataContractSerializer
- JSON
  System.Web.Script.Serialization.JavascriptSerializer
  System.Runtime.Serialization.DataContractJsonSerializer

La serialización permite el registro de una instancia de clase en un archivo para asegurar la persistencia de los datos y facilitar el intercambio de información entre aplicaciones. Para que una clase sea utilizable por el mecanismo de serialización es importante marcar los miembros de la instancia con el atributo <Serializable()> al inicio de la clase. El atributo Serializable indica al compilador que todo el contenido de la clase se puede guardar en un archivo. Si algunos miembros de la instancia no deben guardarse en el archivo, se utiliza el atributo <NonSerializedAttribute()> (Groussard, 2013).

**Tipos de serialización**

**Serialización binaria**

Su función es utilizar la codificación binaria y generar una serialización compacta para almacenamiento a nivel de redes y socket. (Torres, 2013). Si existe una propiedad que contiene una referencia a otra clase, serializará también esta última. Logrando además un tamaño pequeño de archivo fácil de gestionar en diferentes medios.

La serialización binaria almacena la instancia de una clase como un flujo de bytes, este flujo se puede guardar en un archivo de memoria o transmitirse por la red. Contiene el estado del objeto, incluyendo los campos privados y públicos de la instancia. Los datos no son legibles como texto pero tampoco están encriptados. En el acceso binario es posible escribir campos de longitud variable. Para guardar información de tipos diferentes a String se debe hacer uso de las clases *BinaryReader* y *BinaryWrite*. Se utiliza para guardar valores de variables que se recuperan más tarde (Microsoft, 2016a).

En la Tabla 5 se presentan los métodos de las clases mencionadas:

| Clase | Método | Descripción |
|---|---|---|
| BinaryReader | ReadBytes | Lee un carácter de tipo Byte y avanza a la próxima posición. |
| | ReadChar | Lee un caracter de tipo Char y avanza a la próxima posición. |
| | ReadDouble | Lee un carácter de tipo Double y avanza a la próxima posición. |
| | ReadSingle | Lee un carácter de tipo Single y avanza a la próxima posición. |
| BinaryWriter | Flush | Vacía el búfer intermedio y lo envía al destino. |
| | Seek | Establece la nueva posición de escritura. |
| | Write | Escribe un valor teniendo en cuenta su tipo |

**Tabla 5** Descripción de métodos BinaryReader y BinaryWriter

## Serialización XML

Serializa como elementos a los atributos públicos de un determinado objeto, así como los valores de los métodos, en un esquema de sintaxis XML (Torres, 2013). Si hay una propiedad que contiene una referencia a otra clase dará error al serializar. Se genera texto que puede ser interpretado por cualquier sistema operativo para ser transmitido o guardado sobre cualquier medio. Solamente se serializan las propiedades públicas de los objetos. La serialización XML no convierte los métodos, indizadores, campos privados ni colecciones de sólo lectura. Si se desea serializar campos públicos y privados de un objeto, se deberá utilizar la serialización binaria (Evans, Kamanna y Mueller, 2002). La serialización XML guarda propiedades públicas del objeto, sin embargo, la serialización binaria guarda cualquier propiedad sin importar su tipo (Herrarte, 2016).

En la serialización XML se debe definir una variable de tipo *XmlSerializer* del espacio System.Xml.Serialization. La clase *XMlSerializer* serializa y deserializa objetos y documentos XML. Para transferir datos entre objetos y XML es necesario asignar las construcciones del lenguaje de programación (Microsoft, 2016h). La serialización XML convierte los campos públicos de un objeto o los parámetros y valores devueltos de los métodos en una secuencia XML para su almacenamiento y transporte. Al ser XML un estándar abierto, cualquier aplicación podrá procesar esta secuencia a través de la red. Este tipo de serialización también se puede ajustar a la especificación SOAP, ya que es un protocolo basado en XML, diseñada para transportar llamadas a procedimientos con XML (Microsoft, 2016c).

En la Tabla 6 se muestra la descripción de los métodos utilizados para serializar y deserializar archivos XML:

| Métodos | Descripción |
|---|---|
| XmlSerializer | Método constructor, inicializa una nueva instancia de la clase XmlSerializer. |
| Deserialize(Stream) | Deserializa el documento XML contenido en un Stream especificado |
| Serialize(Stream, Object) | Serializa object especificado y escribe el documento XML en un archivo con Stream especificado. |

**Tabla 6** Métodos de la clase XmlSerializer

## SOAP

SOAP (*Simple Object Access Protocol*) es un protocolo diseñado en base a XML y que sirve para transportar llamadas a procedimientos utilizando XML (Torres, 2012). Un mensaje SOAP se crea mediante *XmlSerializer* seleccionando clases y generando mensajes SOAP codificados. Si hay una propiedad que contiene una referencia a otra clase, serializará también esta última.

Este tipo de serialización es recomendable para el uso en tecnologías de invocación remota RPC (Remote procedure call) (Microsoft, 2016e). Es necesario añadir una referencia en el proyecto hacia la librería .NET, ya que no es incluida por defecto el espacio de nombres (Hugon, 2014):

1. En el **Explorador de soluciones**, seleccionar el proyecto.

2. Hacer clic en la opción **Agregar referencia** del menú **Proyecto**.

3. En el cuadro de diálogo **Agregar referencia**, hacer clic en la ficha **.NET** y seleccionar el componente:

   System.Runtime.Serialization.Formatters. Soap

4. Hacer clic en **Aceptar** para cerrar el cuadro de diálogo.

5. En el **Editor de código**, agregar la siguiente instrucción al principio del módulo Form1:

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

*XmlSerializer* se puede usar para serializar clases y generar mensajes SOAP codificados. Al crear un servicio Web XML que se comunica mediante mensajes SOAP, es posible personalizar la secuencia XML. Para las aplicaciones Web o los servicios Web, es conveniente guardar el objeto en un archivo XML con un formato SOAP, lo que facilita el uso compartido del objeto (Microsoft, 2016d):

La clase *SoapFormatter* serializa o deserializa un objeto en formato SOAP. Las clases *SoapFormatter* y *BinaryFormatter* implementan la interfaz *IRemotingFormatter* para permitir llamadas a procedimientos remotos. La interfaz *IFormatter* admite la serialización de objetos. *SoapFormatter* no permite compatibilidad de serialización en diferentes versiones de .NET Framework, por lo que se pueden producir errores (Microsoft, 2016e).

**JSON**

JSON es acrónimo de JavaScript Object Notation, es un formato que se basa en la sintaxis de JavaScript y que resulta útil para la transportación de datos, ya que genera un formato más liviano que XML. La clase DataContractJsonSerializer que se encuentra en el espacio de nombres: System.ServiceModel.Web.dll. Del lado del servidor se localiza en el ensamblado System.Runtime.Serialization.dll (Díaz, 2012).

Para realizar la serialización JSON se utiliza la clase *DataContractJsonSerializer*, además la clase definida del objeto a serializar debe tener definido el atributo [DataContract], así también, cada uno de los elementos de la clase se debe definir con el atributo [DataMember] (Corrales, 2010).

**Resultados, análisis e interpretación de los modelos aplicados**

Para diferenciar cada uno de los formatos de serialización se presenta a continuación una aplicación desarrollada en Visual Studio 2015 que integra la serialización binaria, XML y SOAP, ver Figura 1:


**Figura 1** Opciones de la aplicación

A continuación se muestra el código de la clase Casa que contiene tres atributos para realizar las pruebas de la serialización:

```
<Serializable()>
Public Class Casa
    Public Color As String
    Public Habitaciones As Integer
    Public Cochera As Boolean
End Class
```

Posteriormente se importan las clases requeridas para cada uno de los formatos de serialización binaria, XML y SOAP:

```
Imports System.IO
Imports
System.Runtime.Serialization.Formatters.Bi
nary
Imports System.Xml.Serialization
Imports
System.Runtime.Serialization.Formatters.So
ap
```

A nivel formulario en un botón de comando se muestra la codificación de la serialización binaria por medio de la creación de un objeto de la clase Casa, así como la creación de un archivo binario para almacenar la información del objeto en formato *BinaryFormatter*:

```
Public Class Form1
    Private Sub Form1_Load(sender As
Object, e As EventArgs) Handles
MyBase.Load

    End Sub

    Private Sub
cmdSerializaBin_Click(sender As Object, e
As EventArgs) Handles
cmdSerializaBin.Click
        Dim MiCasa As New Casa
        MiCasa.Color = "Azul"
        MiCasa.Habitaciones = 4
        MiCasa.Cochera = True
        'Serialización
        Dim FS As New
FileStream("MiCasa.bin", FileMode.Create)
        Dim Formato As New BinaryFormatter
        Formato.Serialize(FS, MiCasa)
        FS.Close()
    End Sub
```

La deserialización prepara un objeto vacío de la clase Casa para posteriormente deserializar la información contenida en el archivo binario ejecutando el método *Deserialize* y asignar la información al objeto declarado inicialmente.

```
      Private Sub
DesearializaBin_Click(sender As Object, e
As EventArgs) Handles
DesearializaBin.Click
        'Declarar casa vacía
        Dim MiCasa As Casa
        'Deserializar al objeto
        Dim FS As New
FileStream("MiCasa.bin", FileMode.Open)
        Dim Formato As New BinaryFormatter
        MiCasa = Formato.Deserialize(FS)
        'Muestra los valores
        MsgBox(MiCasa.Color & " " &
MiCasa.Habitaciones & " " &
MiCasa.Cochera)
        FS.Close()
    End Sub
```

Para generar la serialización en formato XML se crea un archivo de datos por medio de la clase *FileStream* asignando posteriormente atributos a un objeto de la clase Casa, para finalmente serializarlos en el formato *XmlSerializer*.

```
    Private Sub
cmdSerializaXML_Click(sender As Object, e
As EventArgs) Handles
cmdSerializaXML.Click
        Dim FS As New
FileStream("MiCasa.xml", FileMode.Create)

        Dim MiCasa As New Casa
        MiCasa.Color = "Amarilla"
```

```
        MiCasa.Habitaciones = 5
        MiCasa.Cochera = False
        'Serialización

        Dim Formato As New
XmlSerializer(MiCasa.GetType)
        Formato.Serialize(FS, MiCasa)
        FS.Close()

    End Sub
```

Una vez que se ha serializado el objeto de la clase Casa, se abre el archivo generado para ejecutar el método de deserialización y asignar a un objeto vacío de la clase Casa la información correspondiente.

```
    Private Sub
cmdDeserializaXML_Click(sender As Object,
e As EventArgs) Handles
cmdDeserializaXML.Click
        Dim FS As New
FileStream("MiCasa.xml", FileMode.Open)

        'Declarar casa vacía
        Dim MiCasa As New Casa
        'Deserializar al objeto

        Dim Formato As New
XmlSerializer(MiCasa.GetType())

        MiCasa = Formato.Deserialize(FS)

        'Muestra los valores
        MsgBox(MiCasa.Color & " " &
MiCasa.Habitaciones & " " &
MiCasa.Cochera)
        FS.Close()
    End Sub
```

Para el caso de serializar en formato SOAP se crea un archivo con extensión xml y se almacena en el mismo la información que contiene un objeto de la clase Casa en formato *SoapFormatter*.

```
    Private Sub
cmdSerializarSOAP_Click(sender As Object,
e As EventArgs) Handles
cmdSerializarSOAP.Click
        Dim FS As New
FileStream("MiCasaSOAP.xml",
FileMode.Create)

        Dim MiCasa As New Casa
        MiCasa.Color = "Blanca"
        MiCasa.Habitaciones = 2
        MiCasa.Cochera = False
        'Serialización
        Dim Formato As New SoapFormatter()
        Formato.Serialize(FS, MiCasa)
        FS.Close()
    End Sub
```

Para deserializar el archivo creado anteriormente se abre el archivo y se declara un objeto vacío de la clase Casa el cual recibirá la deserialización en formato *SoapFormatter* y mostrará los datos del objeto original.

```
    Private Sub
cmdDeserializarSOAP_Click(sender As
Object, e As EventArgs) Handles
cmdDeserializarSOAP.Click
        Dim FS As New
FileStream("MiCasaSOAP.xml",
FileMode.Open)

        'Declarar casa vacía
        Dim MiCasa As New Casa
        'Deserializar al objeto
        Dim Formato As New SoapFormatter()
        MiCasa = Formato.Deserialize(FS)
        'Muestra los valores
        MsgBox(MiCasa.Color & " " &
MiCasa.Habitaciones & " " &
MiCasa.Cochera)
        FS.Close()
    End Sub
End Class
```

En la Figura 2 se muestra el contenido del archivo binario MiCasa.bin generado a través de la serialización:

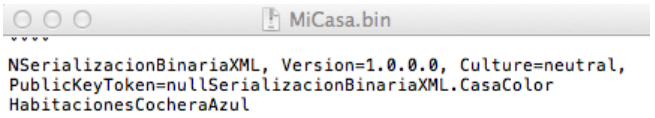


**Figura 2** Visualización del archivo binario

En la Figura 3 se presenta el contenido del archivo MiCasa.XML que fue generado por medio de la serialización:

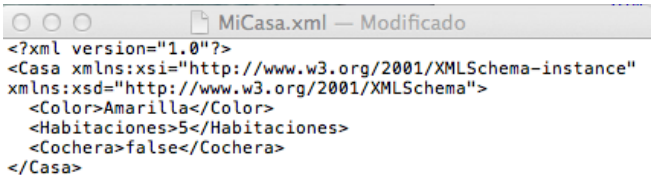


**Figura 3** Visualización del archivo XML



ISSN-2444-5002

En la Figura 4 se despliega el contenido del archivo MiCasaSOAP.xml:


**Figura 4** Visualización del archivo XML-SOAP

## Conclusiones

La serialización XML no guarda métodos, campos privados, propiedades de solo lectura o indexadores, en caso de requerirlo se deberá utilizar la serialización binaria. Por otra parte, si se está trabajando con la serialización XML y la clase tiene un constructor con parámetros, es importante crear también de forma explícita un constructor sin parámetros.

En la serialización binaria y en SOAP, todos los miembros de la clase son serializados ya sean incluso de ámbito privado o protegido. Por otra parte, en la serialización *XML* sólo se serializan los miembros públicos.

Al implementar un mecanismo de la serialización en un entorno orientado a objetos, es necesario realizar intercambios y valoraciones entre la facilidad de uso y la flexibilidad. El presente artículo examina el mecanismo de la serialización robusta proporcionado con .NET Framework y resalta varias características importantes de cada uno de los formatos permitidos en la serialización.

Las ventajas que proporciona la serialización se enfocan principalmente en la interoperabilidad entre aplicaciones, capas o servidores. Además de que se requieren protocolos simples para su envío o almacenamiento.

## Referencias

Díaz, C. (2012). *Aplicaciones de negocio con Microsoft Silverlight 5*. España: RC Libros. p. 388-393

Groussard, T. (2013). *Recursos informáticos Visual Basic 2012 (VB.NET) Los fundamentos del lenguaje*. Barcelona, España: Ediciones ENI. p. 171

Hugon, J. (2014). *C# 5 Desarrolle aplicaciones Windows con Visual Studio 2013*. Barcelona, España: Ediciones ENI. p. 255

Torres, M. (2012). *Programación orientada a objetos con Visual Basic 2012, ADO NET 4.5 FrameWork 4.5*. Lima, Perú: MACRO. p. 217-218

Vigouroux, C. (2015). *Aprender a desarrollar con JavaScript*. Barcelona, España. p. 303.

Corrales, D. (2010). *C#, Serializar JSON (DataContract)*. Recuperado el 22 de abril de 2016 de http://www.esasp.net/2010/06/c-serializar-json-datacontract.html

Evans, K., Kamana, A. Mueller, J. (2002). *XML Serialization*. Recuperado el 22 de abril de 2016 de http://www.informit.com/articles/article.aspx?p=29457&seqNum=5

Herrarte, P. (2016). *Serialización:XmlSerializer y BinaryFormater*. Recuperado el 21 de abril de 2016 de http://www.devjoker.com/contenidos/catss/433/Serializaci%C3%B3n-XmlSerializer-y-BinaryFormater.aspx

León, S. (2010). *Programación desordenada*. Recuperado el 10 de abril de 2016 de http://panicoenlaxbox.blogspot.mx/2010/11/serializacion-en-net.html

Microsoft. (2016a). *Binaryreader (Clase)*. Recuperado el 10 de abril de 2016 de https://msdn.microsoft.com/es-es/library/system.io.binaryreader(v=vs.110).aspx

Microsoft. (2016b). Serialización XML y SOAP. Recuperado el 22 de abril de 2016 de https://msdn.microsoft.com/es-es/library/90c86ass(v=VS.80).aspx

Microsoft. (2016c). Serialización de SOAP a XML. Recuperado el 5 de abril de 3016 de https://msdn.microsoft.com/es-es/library/90c86ass(v=vs.120).aspx

Microsoft. (2016 d). *Cómo serializar un objeto como secuencia XML con codificación SOAP*. Recuperado el 19 de abril de 2016 de https://msdn.microsoft.com/es-es/library/d5wt2he6(v=VS.80).aspx

Microsoft. (2016e). *SoapFormatter (Clase)*. Recuperado el 19 de Abril de 2016 de https://msdn.microsoft.com/es-es/library/system.runtime.serialization.formatters.soap.soapformatter(VS.80).aspx

Microsoft. (2016f). *DirectoryNotFoundException (Clase)*. Recuperado el 20 de abril de 2016 de https://msdn.microsoft.com/es-es/library/system.io.directorynotfoundexception(v=vs.110).aspx

Microsoft. (2016g). *Tutorial: Conservar un objeto (C# y Vuisual Basic)*. Recuperado el 23 de abril de 2016 de https://msdn.microsoft.com/es-es/library/et91as27.aspx