SERIALIZACIÓN DE DATOS

La serialización de datos no es otra cosa que transformar los datos de tal manera que pueda ser transferida por un canal (Internet, archivo plano, memoria, etc) a otro sistema. En otras palabras, si queremos compartir información de nuestro sistemas con otras aplicaciones o viceversa, tendremos que utilizar serialización de datos.

De manera nativa, el Framework de .NET nos ofrece la posibilidad de serializar la información en tres formatos: **Binario, Soap, Xml.**

La serialización en formato **binario** consiste en convertir la información a bytes y se utiliza comúnmente para los escenarios donde la información es transferida por la red hacia un sistema destino, el cual recibe dicha información y realiza el proceso inverso de la serialización, Deserializacion para construir el objeto (información) que fue transferido.

La serialización en formato **Soap** consiste en convertir los datos en un "documento estándar" en el cual se incluirá además de los datos a serializar, una serie de información adicional que será utilizada por el sistemas destino para construir el objeto original. Esta serialización es la que se utiliza en escenarios con Web Services. Mejora la portabilidad con respecto al formato binario.

Finalmente la serialización en formato **Xml,** consiste en transformar la información en un documento Xml que será interpretado por el sistema destino.

Los formatos de serialización Binario y Soap están contenidos en el namespace *System.Runtime.Serialization*, mientras que el Xml esta en el namespace *System.Xml.Serialization.*

Para ejemplificar cada escenario, supóngase que en nuestra aplicación se utiliza una entidad llamada Empleado definida como se muestra a continuación:

```
public class Empleado
{
      private int ID;
      private string nombre;
      private int edad;
      private int diasTrabajados;
      //Propiedades para cada uno de los atributos
}
```

Serialización en formato Binario

Como se mencionó anteriormente, éste método se utiliza generalmente para intercambiar información con otros sistemas a través de la red. Esta serialización tiene la desventaja que solo funcionará entre aplicaciones .NET, es decir, que tanto la aplicación que serializa como la aplicación que deserializa deben ser aplicaciones desarrolladas bajo la plataforma .NET.

El primer paso para serializar un objeto en .NET, es incluir el atributo Serializable en la definición de nuestra clase así:

```
[Serializable]
public class Empleado
{…}
```

Esto le indicará al runtime de .NET que este objeto estará habilitado para ser serializado cuando se requiera, de lo contrario, se generará una excepción del tipo *System.Runtime.Serialization.SerializationException*.

Después, implementar la serialización binaria es tan sencillo como invocar el método Serialize de la clase BinaryFormatter que encontramos en el namespace System.Runtime.Serialization. Este método recibe como parámetros un Stream y el objeto que deseamos serializar

```
Empleado e = new Empleado();
e.ID=1234;
e.nombre=”Pepe Lopez”;
e.edad=35;
e.diasTrabajados=157;

FileStream fichero = new FileStream(“empleados.bin”,FileMode.Create);
BinaryFormatter formater = new BinaryFormatter();
formater.Serialize(fichero, e);
fichero.Close();
```

Como se puede apreciar en el fichero creado, la información obtenida después del proceso de serialización contiene una serie de caracteres especiales. Cuando sea recibido por una aplicación destino, ésta utilizará el método Deserialize del objeto BinaryReader para obtener el objeto Empleado que fue enviado inicialmente, así:

```
FileStream fichero = new FileStream(“empleados.bin”, FileMode.Open);
Empleado e = null;
BinaryFormatter formater = new BinaryFormatter();
e = (Empleado)formater.Deserialize(fichero);
MessageBox.Show(“Nombre:”+e.nombre+”,Edad:”+e.edad.ToString()+”,Dias
Trabajados:”+e.diasTrabajados.ToString());
```

Profundizando un poco más en el como el runtime de .NET serializa los datos, debemos saber que en tiempo de ejecución el runtime convierte en bytes cada uno de los miembros del objeto sin importar el tipo o nivel de acceso (public, private, protected, etc). Esto en algunas ocasiones puede no ser lo que deseemos hacer, sino mas bien omitir algunos campos que consideramos no son necesarios al momento de serializar. En nuestro ejemplo especifico, ¿para qué queremos serializar el campo díasTrabajados si puede ser calculado en tiempo de ejecución?. Esto implica un costo que podemos evitar ya que a mayor cantidad de miembros a serializar, mayor cantidad de información tendrá que ser transferida por red o almacenada en disco.

Para modificar el comportamiento de la serialización binaria, podemos utilizar el atributo `NonSerialized` en cada uno de los miembros que deseamos omitir, así:

```
[NonSerialized]
```

```
private int diasTrabajados;
```

Cuando deserializemos ahora, podemos observar que no se está calculando el valor del campo diasTrabajados. Esto se debe a que el proceso de deserialización lo que hace es "*recrear"* el estado del objeto serializado y como el campo diasTrabajados no está incluido en la información serializada, es omitido en el proceso de deserialización.

Serialización en formato SOAP.

La serialización en formato SOAP, consiste en utilizar un formateador especial que genera un documento SOAP con la información del objeto que deseamos serializar. Este formato de serialización es el utilizado en los servicios Web por su gran interoperabilidad entre sistemas ya que me permite tener los sistemas desarrollados bajo diferentes plataformas e incluso ejecutándose en diferentes plataformas (Linux, Windows, etc).
Sin embargo, una de las desventajas que tiene este formato de serialización es que genera una gran cantidad de información al serializar nuestro objeto (debido a que se debe cumplir con el estándar de los documentos SOAP). Por este motivo, es necesario analizar que tipo de serialización utilizar en nuestra aplicación.
Así entonces tenemos que la serialización binaria es la más eficiente pero sólo nos sirve para escenarios donde las aplicaciones están desarrolladas en .NET. Si por el contrario, necesitamos compartir nuestros datos con aplicaciones desarrolladas en otros lenguajes y además cumplir con el estándar SOAP, utilizaremos el formato SOAP y finalmente si lo que necesitamos es compartir información con aplicaciones desarrolladas en otros lenguajes pero no es necesario cumplir con ningún estándar, podemos utilizar el formato XML.

Para serializar en formato SOAP, sólo se debe utilizar el formateador SoapFormatter que se encuentra en el namespace System.Runtime.Serialization.Formatters.Soap, así:

Ejemplo: *Proyecto SerializaciónFicheros*

Serialización en formato XML

La serialización en formato XML consiste en transformar el objeto en un documento XML, donde por defecto, el nodo raíz será el tipo de dato que estemos serializando y sus miembros serán elementos de ese nodo raíz. La implementación de la serialización XML difiere un poco de las presentadas anteriormente ya que no se utiliza un formateador sino que se hace utilizando la clase XmlSerializer que se encuentra en el namespace System.Xml.Serialization. Esta clase recibe como parámetro el tipo de dato que deseamos serializar. Por ejemplo:

Ejemplo: *Proyecto SerializaciónFicherosXML*

De esta manera le estamos indicando al objeto XmlSerializer que deseamos serializar un objeto de tipo Empleado, el cual al momento de invocar el método Serialize, genera un documento XML*. (Ver fichero)*
En el namespace System.Xml.Serialization, existen varios atributos que permiten modificar el comportamiento de la serialización XML y así obtener diferentes estructuras del documento XML generado. Uno de los atributos más usado es el **XmlAttribute** el cual se usa sobre cualquier miembro público y sirve para indicarle al runtime que dicho

miembro lo genere como un atributo del nodo raíz y no como un elemento.

Ejemplos: *Proyecto SerializaciónFicherosXML* y *XMLListadeObjetos*