Universidad Abierta Interamericana

Campus Virtual UAIOnline

LENGUAJES DE ÚLTIMA GENERACIÓN

ACCESO A DATOS

XML

UNIDAD 2/CLASE ACTUAL:
ACCESO A DATOS- XML /CLASE

Autor de contenidos:
Mauricio Prinzo

PRESENTACIÓN

En la actualidad, persistir datos es una necesidad indispensable de los sistemas de información. Existen clasificaciones de software en tecnología de la información que agrupa los programas para la administración de recursos de información. Nosotros distinguiremos dos tipos de programas:

1) Las bases de datos relacionales

2) Las bases de datos orientadas a objetos

La utilización de XML podría simplificar notablemente la persistencia y recuperación de datos que provienen de instancias de clases en memoria. Esta unidad brindará información básica acerca del manejo de XML.

Para evaluar los nuevos conocimientos, buscaremos simular la persistencia de información usando un lenguaje de marcas como XML

Esperamos que al terminar la clase y que usted alcance la comprensión de los contenidos para poder:

Trabajar con estructuras planas de datos que faciliten el intercambio de datos.

1. CONCEPTOS GENERALES

Desde la aparición de los servicios en Internet y la diversificación de plataformas de desarrollo, nació la necesidad de encontrar un estándar que facilite la transferencia de datos entre aplicaciones.

XML es un lenguaje de marcas que cumple con las necesidades que demanda el estándar de modo tal que, sin importar en qué plataforma nos encontremos trabajando, todos reconocen XML. Por lo tanto, si todos reconocen XML podremos usarlo para compartir datos entre plataformas.

2. ¿QUÉ ES XML?

Inicialmente IBM desarrollo un lenguaje de marcas que permitía el formateo de palabras dentro de un archivo de texto. Este lenguaje de marcas se conoce como SGML y de sus especificaciones nacieron otros lenguajes de marcas, entre ellos HTML que dio fuerza al crecimiento de la gran red de redes.

Los **lenguajes de marcas** se clasifican en:

- Lenguajes de formateo
- Lenguajes descriptivos

Precisamente XML es un lenguaje descriptivo, pues se utiliza para describir datos.

Los lenguajes de marcas son archivos de texto, pero son mucho más efectivos para compartir datos que los archivos de texto tradicionales.

Veamos un ejemplo. Supongamos que enviamos este contenido en un archivo de texto:

```
123pepe1
```

Tal vez, si fuese un ser humano quien tenga la obligación de interpretar esto podría facilmente descubrir un nombre. Pero el 123, ¿qué es?

Podríamos suponer que es el número de registro, o son códigos de categorías con lo cual no sería solo el número 123, podrían ser el número 1, el número 2 y el número tres. Por lo tanto, como verán, para interpretar con claridad esa información estoy obligado a diseñar un estándar de lectura.

Ese estándar podría consistir en separar los datos por un símbolo, por ejemplo #. Partiendo de esa idea, el nuevo resultado sería:

```
#123#pepe#1#
```

Ahora está un poco más claro, por lo menos sabemos dónde comienza y termina cada dato. Pero no es suficiente, pues, no sabemos que representa cada uno de esos datos. Esto nos obliga a establecer un contrato entre el emisor y el lector en donde se establezca cuál es la secuencia de datos.

Entonces, decidimos en nuestro contrato que el primer dato será el id del alumno; el segundo, el nombre; el tercero, el apellido; el cuarto, la categoría, luego, el número de registro y por último, la localización.

Con todo esto, vemos que enviar información en un archivo de texto es algo complejo.

¿Cuál es la ventaja de usar un archivo XML?

Que ese contrato está implícito en el mismo formato del archivo, el mismo ejemplo con XML sería:

```
<Facultad>
<Alumnos>
        <Alumno>
                <ID>123</ID>
                <Nombre>Mauricio</Nombre>
                <Apellido>Prinzo</Apellido>
                <Categoria>1</Categoria>
                <Registro>33509</Registro>
                <Localizacion>091</Localizacion>
        </Alumno>
</Alumnos>
</Facultad>
```

Como se puede observar es más efectivo usar un archivo XML pues facilita mucho la lectura y la comprensión con mínimo esfuerzo.

Un archivo XML es una representación jerárquica que utiliza marcas para indicar cada uno de los campos. Lo veremos a continuación.

2.1 Archivo Bien Formado

El consorcio de nombre W3C, define el estándar para un archivo XML (**http://www.w3.org/TR/#tr_XML**).

Estas reglas permiten evitar los acuerdos previos entre las partes. Cuando un archivo XML cumple con estas reglas se llama **archivo XML bien formado.**

Algunas de las reglas definen que:

Todo archivo XML comienza con la directiva de procesamiento

```
<?xml versión=”1.0”?>
```

Todos los nodos se forman por tres partes: apertura + valor+ cierre

```
<Nombre>Pepe</Nombre>
```

Si el nodo no tiene valor se representa como un nodo vacío

```
<Nombre />
```

Todos los atributos deben estar en la marca de apertura y entre comillas

```
<Persona Ndni=”20555888”> …..
```

Siempre debe tener un nodo Raíz

Todos los nodos menos el raíz deben estar dentro de otro nodo

Tomando lo considerado anteriormente estamos en condiciones de mostrar un ejemplo de XML

```
<?xml version="1.0" encoding="utf-8" ?>
<materia nombre="LUG">
    <Alumno Legajo="1">
            <Nombre>Martin</Nombre>
            <Apellido>Belgrano</Apellido>
    </Alumno>
    <Alumno Legajo="2">
            <Nombre>Angela</Nombre>
            <Apellido>Sarmiento</Apellido>
    </Alumno>
</materia>
```

2.2 DTD Y ESQUEMAS

Frecuentemente cuando se comparte información entre soluciones diferentes lograr un acuerdo entre las partes se vuelve una necesidad. Esta necesidad mejora los procesos de carga de las aplicaciones debido a que se puede asegurar el contenido del archivo previamente.

Esto es posible usando son dos tecnologías de XML, los **DTD** y los esquemas. Ambos se usan para validar la estructura de los archivos XML, indicando por ejemplo si los nodos pueden repetirse, si son obligatorios, los nombres que deben usarse, entre otras cosas.

Comencemos con los DTD. Estos describen las reglas para validar un archivo XML. En pocas palabras, permiten confrontar la estructura del contenido de un archivo XML con el acuerdo entre las partes.

Una característica muy particular es que su sintaxis es diferente al modelo usado por XML y a la vista resulta un poco más complejo. Otro aspecto que podemos señalar como una limitación es que no permite manejar tipos de datos no obstante, pueden aplicarse embebidos en el XML o fuera de él como un archivo externo.

Los **nodos** se describen con una palabra reservada llamada ELEMENT (note que está en mayúsculas).

Los elementos están formados por un nombre y una **regla**, respetando la sintaxis ELEMENT nombre (regla). Las reglas pueden ser los nodos que contiene o el valor.

Existen tres tipos de reglas cuando no son nodos:

#PCDATA

ANY

EMPTY

Un ejemplo de un conjunto de definiciones de reglas en un archivo DTD es el que le mostramos a continuación:

```
<! ELEMENT pelicula (titulo, reparto?)>

<! ELEMENT titulo(#PCDATA)>
<! ELEMENT repartoi(interprete, interprete)>
<! ELEMENT interprete (#PCDATA)>
```

Por otro lado, se encuentran los **esquemas**. Estos son archivos con extensión XSD que como los DTD también pueden definir las reglas para validar un archivo XML. Antes de continuar, haremos una valoración importante: los esquemas no reemplazan a los DTD y tampoco se cumple a la inversa.

Entonces, los **esquemas** tienen una particularidad destacable: son archivos XML bien formados (si no comprende la importancia de esto le recomendamos volver al ítem 2.1). Esta virtud, los hace mucho más fáciles de mantener y de leer, puesto que mantienen una lógica similar a la experiencia previa adquirida con XML.

Además, permiten manejar tipos de datos como *string*, enteros entre otros habilitando un nivel mayor de detalle en la descripción de las reglas.

Por otro lado, se encuentran los **esquemas**. Estos son archivos con extensión XSD que como los DTD también pueden definir las reglas para validar un archivo XML. Antes de continuar, haremos una valoración importante: los esquemas no reemplazan a los DTD y tampoco se cumple a la inversa.

Entonces, los **esquemas** tienen una particularidad destacable: son archivos XML bien formados (si no comprende la importancia de esto le recomendamos volver al ítem 1.2). Esta virtud, los hace mucho más fáciles de mantener y de leer, puesto que mantienen una lógica similar a la experiencia previa adquirida con XML.

Además, permiten manejar tipos de datos como *string*, enteros entre otros habilitando un nivel mayor de detalle en la descripción de las reglas.

Para terminar esta descripción, es importante enunciar que cada esquema comienza con una directiva de procesamiento http://www.w3.org/2001/XMLSchema.

Los **elementos** se describen con la palabra reservada elemento, ya habrá notado que en este caso es todo en minúsculas. Recuerde que cuando usamos DTD se escribe en mayúsculas.

La sintaxis para describir un **nodo** es similar a la siguiente:

```
<element name="pelicula" type="string">
```

Permite armar grupos que describen las relaciones de jerarquías entre nodos.

```
<element name="pelicula">

<complexType>
  <all>
    <element name="titulo" type="string"/>

    <element name="minutos" type="integer"/>
  </all>
</complexType>
</element>
```

Los **atributos**, tienen una sintaxis similar usando la palabra reservada *attribute*.

Veamos algunas formalidades para destacar es el uso de tres palabras reservadas:

1) *all*= Todos los elementos

2) *Sequence*= Todos los elementos respetando el orden

3) *Choice*= Es necesario incluir uno de los elementos

Conociendo estas obligaciones en la declaración de las reglas de validación, podremos entender el siguiente archivo XSD

```
<?xml version="1.0"?>

<schema xmlns=http://www.w3.org/2001/XMLSchema> <element name="pelicula">
<complexType>

<sequence>
<element name="titulo" Type="string"
</sequence>
</complexType>

</element>
</schema>
```

Como conclusión podemos decir que todo archivo XML bien formado que cumplen con las definiciones de un DTD o un esquema es un **archivo XML válido.**

En el siguiente cuadro, le presentamos las diferencias sobresalientes entre ambos.

DTD	XML Schema
Basadas en una sintaxis especializada.	Basados en XML
Compactas	No compactas
Hay muchas herramientas disponibles para procesar documentos XML con las DTD	Hay muy pocas herramientas para los esquemas XML
Tratan todos los datos como cadenas o cadenas enumeradas	Soportan una serie de tipos de datos (int, flota, bolean, date, etc.)
Emplean un modelo de datos cerrados que no permite prácticamente la ampliación.	Presentan un modelo de datos abierto, lo cual permite ampliar los vocabularios y establecer relaciones de herencia entre los elementos sin invalidar a los documentos.
Sólo permiten una asociación entre un documento y su DTD a través de la declaración de tipos de documentos.	Soportan la integración de los espacios de nombres, lo que permite asociar nodos individuales en un documento con las declaraciones de tipos de un esquema.
Soportan una forma primitiva de agrupación a través de entidades de parámetros, lo que supone una gran limitación, ya que el procesador XML no sabe nada acerca del agrupamiento.	Soportan los grupos de atributos, que le permiten combinar atributos lógicamente.

3. SERVICIOS OFRECIDOS POR EL FRAMEWORK

A continuación, describiremos algunos servicios que brinda el Framework.NET. A todas las funciones de XML las encontrará en System.XML.

La línea de código que deberá agregar en sus programas es:

```
using System.XML;
```

3.1 CLASE XMLTEXTREADER

Permite la lectura de un archivo XML, tiene un funcionamiento similar al *dataset* y es menos potente que un analizador DOM.

La creación del objeto es similar a cualquier objeto, siguiendo esta sintaxis:

```
XmlTextReader ArchivoXML = new XmlTextReader();
```

Algunas sobrecargas del objeto permiten seleccionar varias opciones: se puede indicar la dirección url del archivo XML o cargar el texto directamente.

El resultado se obtiene en la variable que abre el archivo, y que devuelve una lista de nodos. Estos se acceden por medio de *XmlNodeType*. Los tipos se enumeran en la siguiente lista:

1. *Document*
2. *Element*
3. *EndElement*
4. *Text*
5. *CData*

Un ejemplo de código podría ser el siguiente:

```
XmlTextReader ArchivoXML = new XmlTextReader(Archivo);
Try {
        While (ArchivoXML.Read())
           if (ArchivoXML.NodeType == XmlNodeType.Element)
             {  if (ArchivoXML.HasAttributes)
                 {While (ArchivoXML.MoveToNextAttribute())
                    This.Arbol.Nodes.Add(ArchivoXML.Value)
                 }
              }
           }

           if (ArchivoXML.NodeType == XmlNodeType.Text)
             { this.Arbol.Nodes.Add(ArchivoXML.Value);}

        }
Catch (System.Xml.XmlException  ex)
        MessageBox.Show(ex.Message);
}
```

3.2 CLASE XMLTEXTWRITER

Escribe un archivo XML y se asegura que sea un archivo bien formado.

La sintaxis del objeto es la siguiente:

```
XmlTextWriter archivoxml = new XmlTextWriter ("ejemplo2.xml",
System.Text.Encoding.UTF8) ;
```

UTF8 es la clasificación del set de caracteres que será válido para el contenido.

```
XmlTextWriter archivoxml = new XmlTextWriter("ejemplo2.xml",
System.Text.Encoding.UTF8);
archivoxml.Formatting = Formatting.Indented;
archivoxml.Indentation = 2;
archivoxml.WriteStartDocument(true);
archivoxml.WriteStartElement("Alumno");
archivoxml.WriteAttributeString("DNI", 1);
archivoxml.WriteElementString("Apellido", "Perez");
archivoxml.WriteEndElement();
archivoxml.WriteEndDocument();
archivoxml.Close();
```

3.3 DOMXML

Otra forma de leer un archivo XML es usando **DOMXML**. Este es un parseador incluido en el framework. Permite un manejo más profundo de los archivos XML. Además, facilita las acciones de agregar y borrar nodos.

Una característica es que cada nodo se identifica con un objeto y utiliza la clase *XmlNode* y *XmlDocument.*

```
XmlDocument doc = new XmlDocument();
doc.Load("Ejemplo.xml");
```

Para recorrer los nodos se utiliza la estructura de repetición *For each*, la variable es de tipo *XmlNode*. Como es un objeto, su interface brinda mucha información de sí mismo, es el caso de la propiedad *HasChilNode* que informa la cantidad de nodos hijos que posee.

Si se necesita leer el atributo de un nodo, manejaremos la propiedad *attribute* que devuelve una lista de atributos. Junto con esto tenemos *innerText* e *innerXML* que devuelven el contenido del nodo en sus respectivos formatos.

```
XmlDocument doc = new XmlDocument();
doc.Load("Ejemplo.xml");

foreach(XmlNode f In doc.DocumentElement)

        { foreach (XmlNode a In f.Attributes)
           {this.Arbol.Nodes.Add(a.InnerText); }

          foreach( XmlNode ff In f.ChildNodes)
           {this.Arbol.Nodes.Add(ff.InnerText);}

 }
```

Para borrar los nodos se podrán utilizar las siguientes características

RemoveAll : Borra todos los nodos

RemoveChild: Borra un nodo

Attributes.Remove : Borra un atributo con un nombre especifico

AppendChild: Agrega un elemento de tipo *XmlElement*

3.4 LINQTOXML

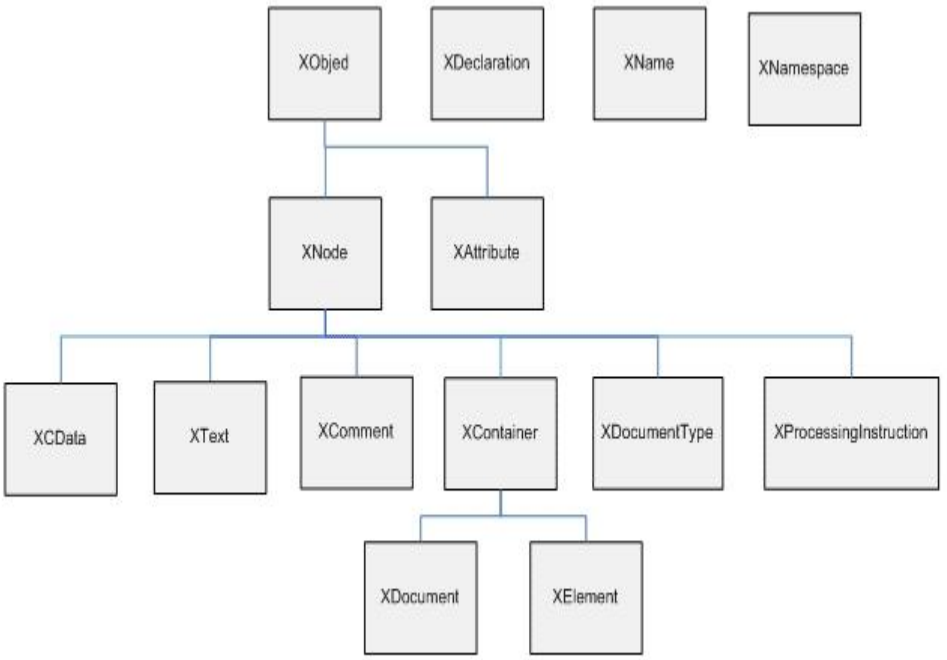
"LINQ a XML" no es otra cosa que proporcionar toda la potencialidad del "Lenguaje Integrado de Consultas" aplicado a la información que se encuentra contenida en XML

Permite leer y modificar estos archivos de forma parecida a como lo hacíamos a través de XMLReader y XMLWritter.

Notable mejora en el consumo de memoria y la cantidad de código que debemos escribir.

Utiliza la librería System.XML.Linq

Las jerarquías en las que se maneja LINQ



Ventajas

- ✓ Elementos principales. Los elementos básicos para trabajar con formatos XML a nuestro criterio son XElement y XAttribute.
- ✓ Visión simplificada de los archivos XML: se puede tratar tanto al documento como a los nodos que lo conforman como un Xelement
- ✓ Proporciona una identificación a los nodos de los archivos XML. Podemos dar un nombre a cada elemento de nuestro árbol XML a través de la propiedad XName, esta utilidad será de gran ayuda al momento de realizar copias, reemplazos o al renombrar los Nodos.
- ✓ Creación de estructuras XML: Utilizando XElement o Xattributte
- ✓ Memoria: Mejor uso de los recursos de memoria, lo que ayudará a mejorar el rendimiento de nuestras aplicaciones.

Dado el siguiente XML

```
<?xml version="1.0" encoding="utf-8"?>
<Jugadores>
 <Jugador>
  <Nombre>C.Ronaldo</Nombre>
  <Equipo>Real madrid</Equipo>
  <Posicion>Delantero</Posicion>
 </Jugador>
 <Jugador>
  <Nombre>L.Messi</Nombre>
  <Equipo>Barcelona</Equipo>
  <Posicion>Delantero</Posicion>
 </Jugador>
 <Jugador>
  <Nombre>p.Goltz</Nombre>
  <Equipo>Boca Juniors</Equipo>
  <Posicion>Defensor</Posicion>
 </Jugador>
 <Jugador>
  <Nombre>G.Lux</Nombre>
  <Equipo>River Plate</Equipo>
  <Posicion>Arquero</Posicion>
 </Jugador>
 <Jugador>
  <Nombre>Aguero</Nombre>
  <Equipo>Manchester City</Equipo>
  <Posicion>Mediocampista</Posicion>
 </Jugador>
</Jugadores>
```

Realizaremos la lectura y escritura del mismo con LINQ

```
private void LeerXML()
{  //Cargamos el documento de Jugadores
XDocument xmlDoc = XDocument.Load("Jugadores.xml");
//realizamos la consulta en LINQ
var Consulta = from Jugador in xmlDoc.Descendants("Jugador")
          select new
          {
          Nombre = Jugador.Element("Nombre").Value,
          Equipo = Jugador.Element("Equipo").Value,
          Posicion = Jugador.Element("Posicion").Value,
          };

textBox1.Text = null;
//De la consulta muestro la informacion tomada
          foreach (var a in Consulta)
          {
          textBox1.Text = textBox1.Text + "Nombre: " + a.Nombre + " \n";
          textBox1.Text = textBox1.Text + " Team: " + a.Equipo + " \n";
          textBox1.Text = textBox1.Text + " Posicion: " + a.Posicion + " \r\n";
          }

          if (textBox1.Text == null)
          textBox1.Text = "No existen resultados.";

}
```

```
 private void AgregarXml()
     {
        XDocument xmlDoc = XDocument.Load("Jugadores.xml");

        xmlDoc.Element("Jugadores").Add(new XElement("Jugador",
                          new XElement("Nombre", this.txtNombre.Text),
                          new XElement("Equipo",txtEquipo.Text),
                          new XElement("Posicion", CboPosicion.SelectedItem.ToString())));

        xmlDoc.Save("Jugadores.xml");
        LeerXML();
     }
```

4. XML AVANZADO

Un aspecto destacable que no podemos pasar por alto es la facilidad de transformar un archivo XML en un archivo con otro formato. Por ejemplo, en un archivo PDF.

A modo de ejemplo, explicaremos la receta para transformar un archivo XML en un archivo HTML - algo muy utilizado por los generadores de contenido en periódicos online.

Para completar la transformación se usan dos tecnologías **XSLT** y **xPAth**.

XSLT o transformación de XSL es un estándar que presenta la manera de convertir un documento XML en otro, incluso en formatos que no son XML.

Las plantillas diagramadas con esta tecnología definen las pautas de transformación del archivo XML. Las transformaciones del documento fuente se constituyen por una o varias reglas de **XSLT** que conviven con estructuras de texto propias del formato al que será transformado.

Esto quiere decir que si se desea transformar XML en otro lenguaje de marcas, junto con las reglas de XSLT se deberán procesar etiquetas de marcas del lenguaje destino.

Actualmente XSLT se utiliza en plataformas WEB, la armonía de XML y XSLT permite separar contenido de presentación, mejorando notablemente el mantenimiento del sitio.

Pero esta tecnología necesita apoyo para completar el proceso y es aquí donde aparece una tecnología complementaria llamada **xPath** (XML Path Language). La utilidad de xPath es encontrar y navegar los nodos del archivo XML para procesar sus datos junto a las reglas XSLT.

El camino, a partir de ahora Path, puede definirse de dos maneras diferentes: Path absoluto y Path relativo.

- El **path absoluto** es aquel donde se especifica el camino en forma completa arrancando desde el nodo Raiz,

- El **path relativo**, toma la posición donde se encuentra y completa el Path para llegar al nodo. Concluyendo, permite construir expresiones que recorren y procesan el archivo XML.

Existen diversas expresiones para encontrar un nodo en el árbol de jerarquías de XML. De ellas, solo comentaremos algunas pues el resto escapa al alcance planteado para esta asignatura.

Todos los archivos de transformación necesitan de un primer posicionamiento, en este caso, será el nodo raíz. Notará también, que todas las expresiones de xPath comienzan con un espacio de nombre (*namespace*) **xsl**.

```
<xsl:tempalte match="/"></xsl:template>
```

Tal como explicamos en párrafos anteriores, con esta tecnología podemos leer el valor de un nodo en particular.

```
<xsl:value-of select="[Path al nodo]" />
```

Otras de los servicios que brinda esta tecnología, se asemeja bastante a los lenguaje de programación. Por ejemplo, podemos recorrer una lista de nodos por medio de una expresión repetitiva.

```
<xsl:for-each select="[Path al nodo]"></xsl:for-each>
```

Para resumir lo explicado, le mostramos un ejemplo de *archivo de transformación.*

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

    xmlns:msxsl="urn:schemas-microsoft-com:xslt" exclude-result-
prefixes="msxsl">

    <xsl:template match="/">

      <xsl:value-of
      select="materia/Alumno/Nombre"/>

      <xsl:for-each select ="materia/Alumno">

        <p><xsl:value-of select="Nombre"/>,<xsl:value-of
select="Apellido"/>

      </p>

      </xsl:for-each>

    </xsl:template>

</xsl:stylesheet>
```

La **transformación** posee un proceso que no se resuelve solo con las tecnologías planteadas pues, como es de esperarse, es necesario manipular archivos e interactuar con puertos de entrada y salida.

El Framework.Net brinda dos espacios de nombre XML y Xml.XSL que junto a las librerías IO pueden crear y gestionar los archivos físicamente.

```
using System.Xml;
using System.Xml.Xsl;
using System.IO;
```

Un código de ejemplo, es el que le acercamos a continuación:

```
using System.Xml;
using System.Xml.Xsl;
using System.IO;
using System.Net;

public class Form1 {

   private void Button1_Click(object sender, EventArgs e)
     {
       XslTransform transform = new XslTransform();
       transform.Load("PersonnelHTML.xsl");
       transform.Transform("Personnel.xml", "Personnel.html");

     }

   private void Button2_Click (object sender, EventArgs e)
       {XslTransform transform = new XslTransform();
       transform.Load("xmlTransform\t.xsl");
       transform.Transform("materia.xml", " materia.html");
       }

     }
}
```

5. CONCLUSIÓN

XML es un estándar que facilita un modelo descriptivo intuitivo utilizado en la actualidad por todos los programadores. Puede ser utilizado como estructura de datos, como base para crear páginas web XHTML o, como verán en otra etapa de su aprendizaje, como uno de los pilares para la creación y utilización de servicios Web

Para ampliar los contenidos estudiados en la clase, le solicitamos la lectura del siguiente texto:

Deitel, Harvey M.; Deitel, Paul J.; Vidal Romero Elizondo, Alfonso(Traductor); y otros. **Cómo programar en C#.** 2a.ed.-- México, DF