**LINQ A XML**

**INTRODUCCION**

“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 este formato, permitiéndonos leer y modificar estos archivos de forma parecida a como lo hacíamos a través de XMLReader y XMLWritter pero con una notable mejora en el consumo de memoria y la cantidad de código que debemos escribir para obtener los mismos resultados, adicionando la ventaja que con esta nueva tecnología podemos conectarnos y trabajar con otras fuentes de datos.

**Estructura Jerárquica de “LINQ a XML”**



**LINQ a XML”:**

*Visión simplificada de los archivos XML.-* Ya no tenemos que acceder a los nodos de los arboles XML a través de un documento, En LINQ 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.-* El Ejercicio 1 expone la gran facilidad con la que se puede crear estructuras XML a través del “Lenguaje Integrado de Consultas”.

*Memoria.-* Mejor uso de los recursos de memoria, lo que ayudará a mejorar el rendimiento de nuestras aplicaciones.

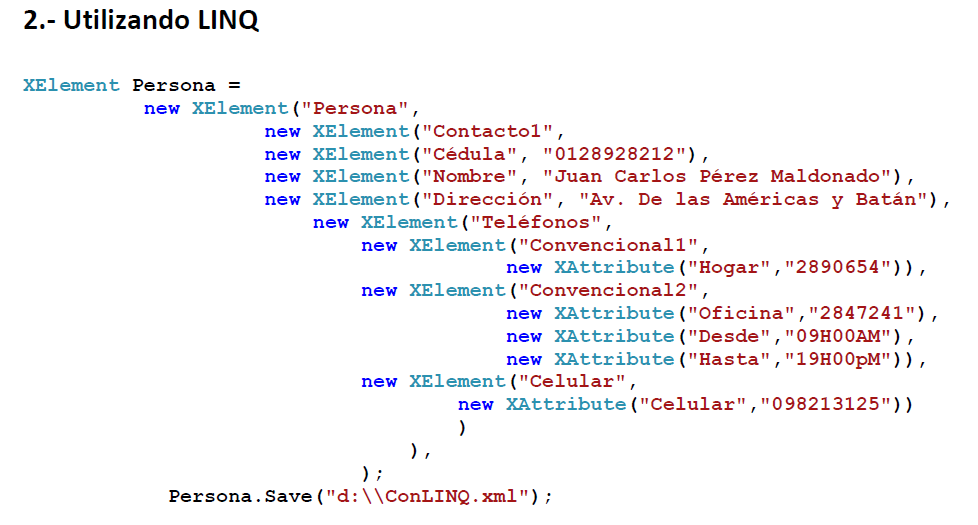
*Acceso a datos.- Permite leer y modificar los nodos de los archivos XML con sentencias muy similares a SQL y tomando en cuenta que LINQ también implementa conectividad con bases de datos, se pueden consultar las dos fuentes con una misma sintaxis.*

*Tratamiento de Tipos de Variable.- “LINQ a XML” permite tratar el contenido de los elementos y atributos respetando sus tipos con tan solo hacer un cast.*

*Elementos principales.- Los elementos básicos para trabajar con formatos XML son XElement y XAttribute.*

**Ejemplo 1**

Crear un archivo XML, que contenga la información básica de una persona. Para poder probar el ejercicio deberá crear un formulario con los eventos necesarios

***Código:*** *Este ejercicio tiene como finalidad introducir a la funcionalidad que implementa LINQ a XML, para esto hemos creado dos veces el mismo archivo, la diferencia en la cantidad y la complejidad del código resalta a la vista. Esto se profundizará cuando estudiemos la clase XElement y XAttribute que son dos de los pilares de la funcionalidad XML.*

*No es indispensable un documento XML.-* En la programación común es indispensable primero crear o disponer de un documento Xml para poder continuar con la costrucción de árboles XML, en cambio con LinXml en ningún momento tenemos que crear o cargar un documento contenedor (Documento XML).

*Compatibilidad con espacios en blanco.-* Los espacios en blanco siempre han significado un problema al momento de desarrollar código para leer, cargar o editar archivos XML, con “LINQ a XML” podemos tratar los espacios a través de nodos texto conocidos como xText.

*Control simplificado de nombres y espacios de nombres.-* Evita tener que manejar espacios de nombres o prefijos de espacios de nombres, es decir si el desarrollador desea los podrá hacer uso, caso contrario al momento de la serialización se definirá los prefijos.

Así mismo para cambiar el nombre de un nodo XML, se debía crear un nodo con diferente nombre y copiar todo el contenido del nodo original. En cambio LINQ soluciona este problema apoyándose en la propiedad xName de los nodos.

*Diferencias entre “LINQ a XML” y XML Reader.-* XML Reader nos permite realizar lecturas de documentos XML, pero será el programador quien deba analizar y decidir que va a utilizar.

A lo que queremos hacer referencia es que en muchas ocasiones no es suficiente solo leer los orígenes de datos, es más, tal vez ni siquiera haya un único origen de datos entonces.

los que nos toca es tomar los datos de las diferentes fuentes luego darles el tratamiento necesario y generar un resultado. En este caso será de mucha utilidad el Lenguaje Integrado de Consultas, o podría también ser una combinación entre XMLReader y las bondades que nos ofrece LINQ.

**Clases XElement, XAttribute y XDocument**

La clase XElement es la clase base con la trabaja el Lenguaje Integrado de Consultas. Mediante esta clase se puede realizar varias operaciones con los elementos o nodos de los archivos XML, así también se dispone de la clase XAttribute que tiene la misma función que la clase XElement pero en este caso con los atributos.

Los elementos difieren de los atributos en el aspecto que los últimos no son nodos, más bien podríamos decir que estos son parte de los nodos y que están compuestos por dos valores, uno que representa el nombre del atributo y el segundo que indica el contenido del atributo.

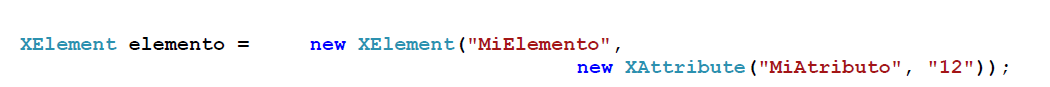
Por otra parte la clase XDocument si bien forma parte de LINQ, lo mejor es saber analizarlo de manera correcta, ya que en muchas ocasiones resulta más facil manejar archivos unicamente con elementos y atributos.

Cuando utilizamos la clase XDocument debemos tomar en cuenta que un documento solo puede tener un elemento secundario, siendo este el nodo ráiz. No cambia mucho o quizá nada, ya que todo documento XML requiere que el nodo principal sea único.

Así mismo hay otros elementos que puede tener un documento XML, como son XDeclaration, XProcessingInstruction, XComment, XDocumentType, etc. La clase XDocument se deriva de Container por lo que su funcion es similar a la de un contenedor.

El aprendizaje de estas tres clases es muy sencillo ya que la mayoría de métodos que implementan cada una de ellas son muy similares entre sí.

Ejemplo:



***Código:*** *Este es uno de los paso fundamentales para poder crear estructuras XML, en este ejemplo hemos creado un nodo XElement y le asignado un solo atributo, al leer el código podemos darnos cuenta que instanciamos un objeto XElement, el primer parámetro es el nombre y el segundo es vector que puede tener uno o varios atributos o incluso subnodos. Lo único que debemos tomar en cuenta es anteponer la palabra new a cada elemento o atributo que se crea.*

En el ejemplo podemos ver que tanto la clase XElement como XAttribute son similares, XElement recibe de parámetros el nombre del elemento en este caso MiElemento y como segundo parámetro los atributos de cada elemento. En cambio XAttribute recibe el nombre del atributo y en segundo lugar el valor o contenido del atributo.

Para el caso de los Documentos XDocument funciona de manera muy similar a las dos clases anteriores, pero esta recibe parametros como: XComent, XProcessingInstruction, XDeclarative, XElement, etc. Según la instancia que se utilice.

Para el caso de los Documentos XDocument funciona de manera muy similar a las dos clases anteriores, pero esta recibe parametros como: XComent, XProcessingInstruction, XDeclarative, XElement, etc. Según la instancia que se utilice.

**Cargar un archivo XML**

Podemos cargar archivos XML a través de su path, con el método XElement.Load(), este método tiene instancias adicionales que nos permite dar opciones de carga como es mantener los espacios en blanco. La sintaxis de este método es la siguiente:



El método Load() permite cargar archivos XML en un XElement o en un XDocument, pero la diferencia radica; Cuando lo hacemos con un XElement el nodo primario del documento pasa a ser el documento, mientras que cuando lo hacemos a través de XDocument se crea un nodo para el documento y el nodo raíz queda como nodo secundario. Sin dejar este de ser el nodo principal.

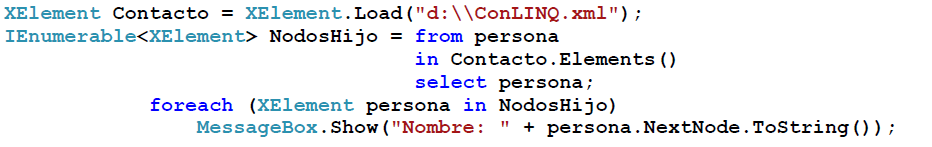
Cuando deseamos realizar la carga de un documento XML con algunas restricciones como por ejemplo preservar los espacios en blanco, podemos enviar el parámetro LoadOptions indicando como se debe realizar la carga. En el ejemplo siguiente vamos a cargar el documento Con LINQ.xml preservando los espacios en blanco. Igual que el ejemplo anterior es la misma sintaxis tanto para XElement como para XDocument.



**Código:** El segundo parámetro del metodo Load() indica que debe preservar los espacios en blanco

**EJERCICIO 2**

En este ejercicio vamos a cargar el archivo XML ConLINQ.xml que creamos al principio de este tema y después vamos a presentar el nombre del contacto.



***Código:*** *La instrucción de consulta carga todos los subnodos del nodo Contactos (NodoRaiz) y a través del bucle foreach los va presentando de uno en uno*

Si queremos obtener el valor de un elemento XML, entonces solo debemos cambiar la presentacion del resultado por:



**ACCESO A LOS ARCHIVOS QUE CONTIENEN LA INFORMACION**

**CLAÚSULA SELECT CON LINQ**

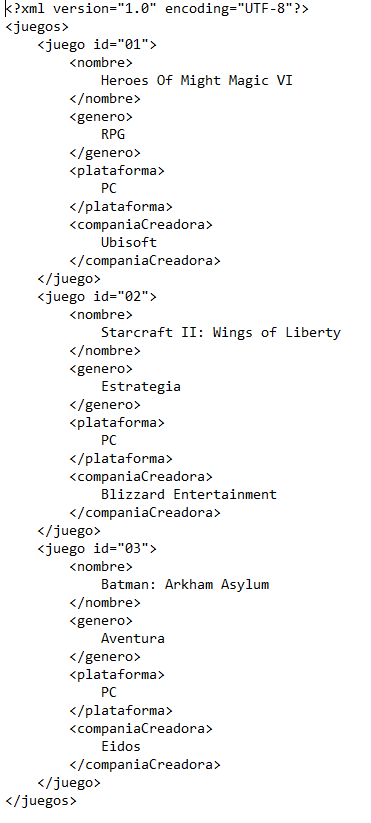
**SENTENCIA SELECT CON ORIGENES DE DATOS XML**

**Recuperar el Valor Superificial de un Elemento**

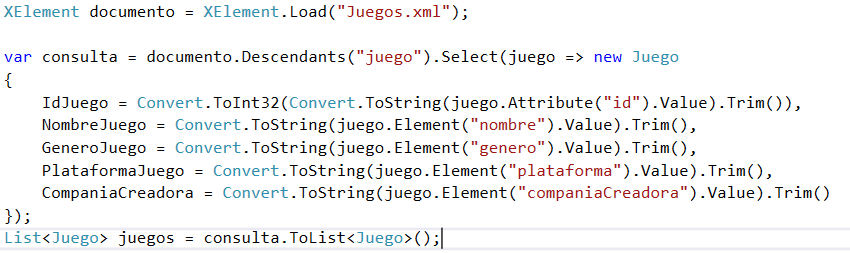
El valor superficial hace referencia al valor que tiene un nodo específico independiente del contenido que tengan sus nodos hijos.

**Ejemplo 3**

Este creará el archivo xml en el path de la aplicación, el código es el siguiente:

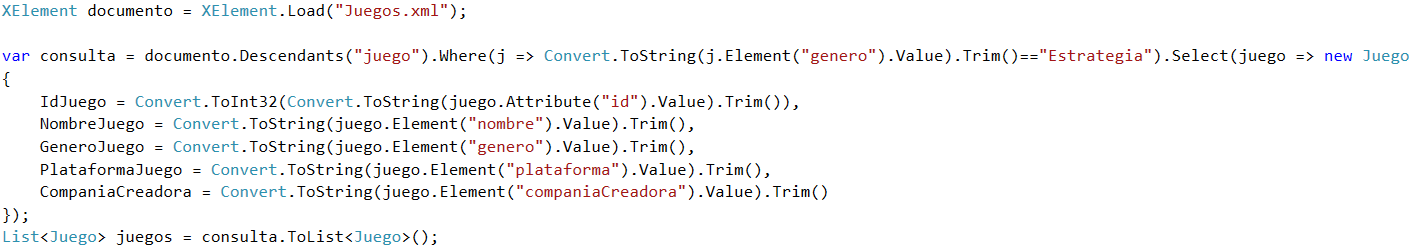


***Código:*** *Creamos una sentencia LINQ en donde cargue todos los nodos descendientes del documento.*



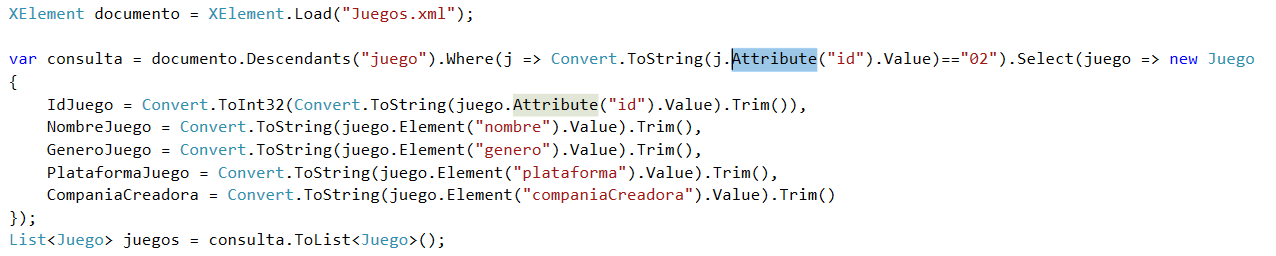
**Buscar Descendientes con un dato Específico**

El siguiente ejercicio nos muestra una forma sencilla de buscar descendientes con un id específico.



**Como Buscar un Elemento a Partir de un Atributo**

El siguiente ejemplo muestra como realizar la búsqueda de un elemento dentro de un árbol a partir de un atributo. El árbol XML que debemos tener para la busqueda es el siguiente:

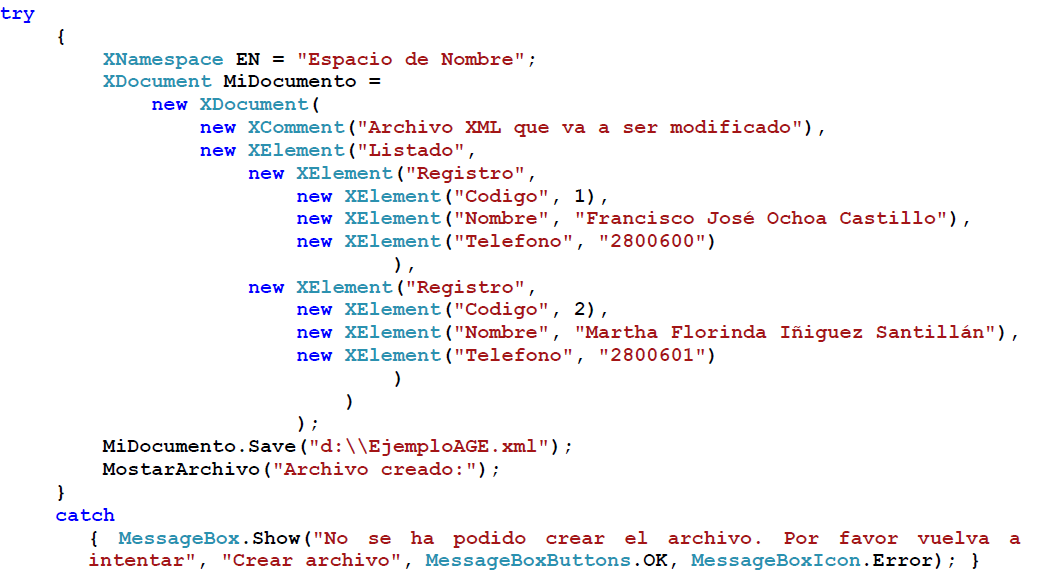


**Insertar, Modificar y Eliminar nodos de un archivo XML.**

Consecuentemente nos faltaría hacer el ejercicio que haga un Agenda (Actualizar, Guardar y Eliminar) a los datos de estos orígenes XML utilizando el potencial de “LINQ a XML ” para esto crearemos un pequeño formulario como se muestra en la siguiente figura:

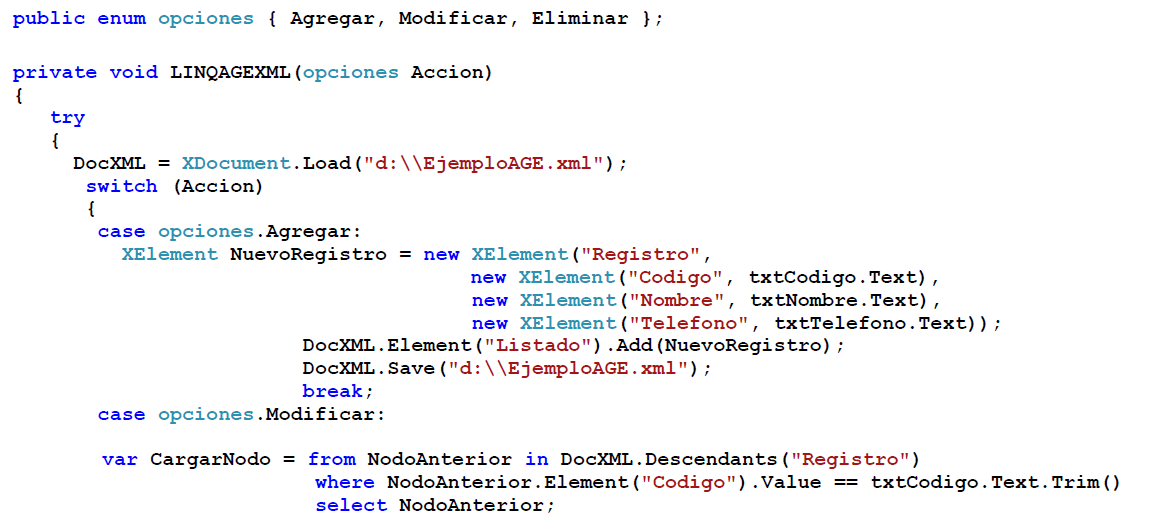
Creamos nuestra estructua XML con la que desarrollaremos todo nuestro ejemplo:

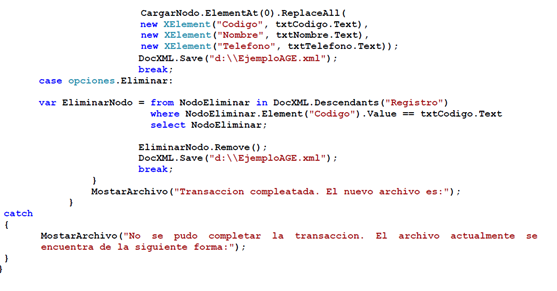




***Código:*** *Creamos el archivo XML para poder realizar las operaciones de agregar, modificar y eliminar. Podemos observar que en la primera linea hemos utilizado la clase XNameSpace, esta clase permite asignar un espacio de nombre al archivo XML que estamos creando. También podemos notar que después de hacer la instancia del documento utilizamos la clase XComment que agrega un comentario al documento XML.*

El siguiente paso consiste en crear una enumeración y luego dos métodos el primero que será el encargado de agregar, modificar, y eliminar nuestro documento xml:





***Código:*** *Instanciamos y cargamos nuestro archivo con el método Load().*

**- *Agregar.-*** *Creamos la estructura que se va a adicionar en nuestro caso a través de la clase XElement instanciada con el nombre de Nuevo Registro. Terminada de crear la nueva estructura la implementamos a través del método Add() y finalmente guardamos el documento con el método Save().*

**- *Modificar.-*** *A través de una consulta LINQ, colocamos en la variable CargarNodo el elemento XElement que va a ser modificado, después con el método ReplaceAll(), reemplazamos los nuevos valores del nodo y guardamos el documento con el método Save().*

**- *Eliminar.-*** *Así mismo con una consulta LINQ cargamos el nodo que va a ser eliminado. En nuestro caso se llama EliminarNodo. El proceso de eliminar se realiza a través del método Remove(), y de igual forma que en guardar y actualizar guardamos el documento con el método Save().*

Para formatear el documento XML realizar un Parse al documento

XDocument.Parse(DocXMl.ToString()).Save("d:\\ejemploAGE.xml");