Unidad 4 - Laboratorio 2

1. Lectura y Escritura de Archivos TXT y XML con ADO.Net.

Objetivos

Consultar y modificar un archivo .txt utilizando ADO.Net
Consultar y modificar un archivo .xml utilizando ADO.Net

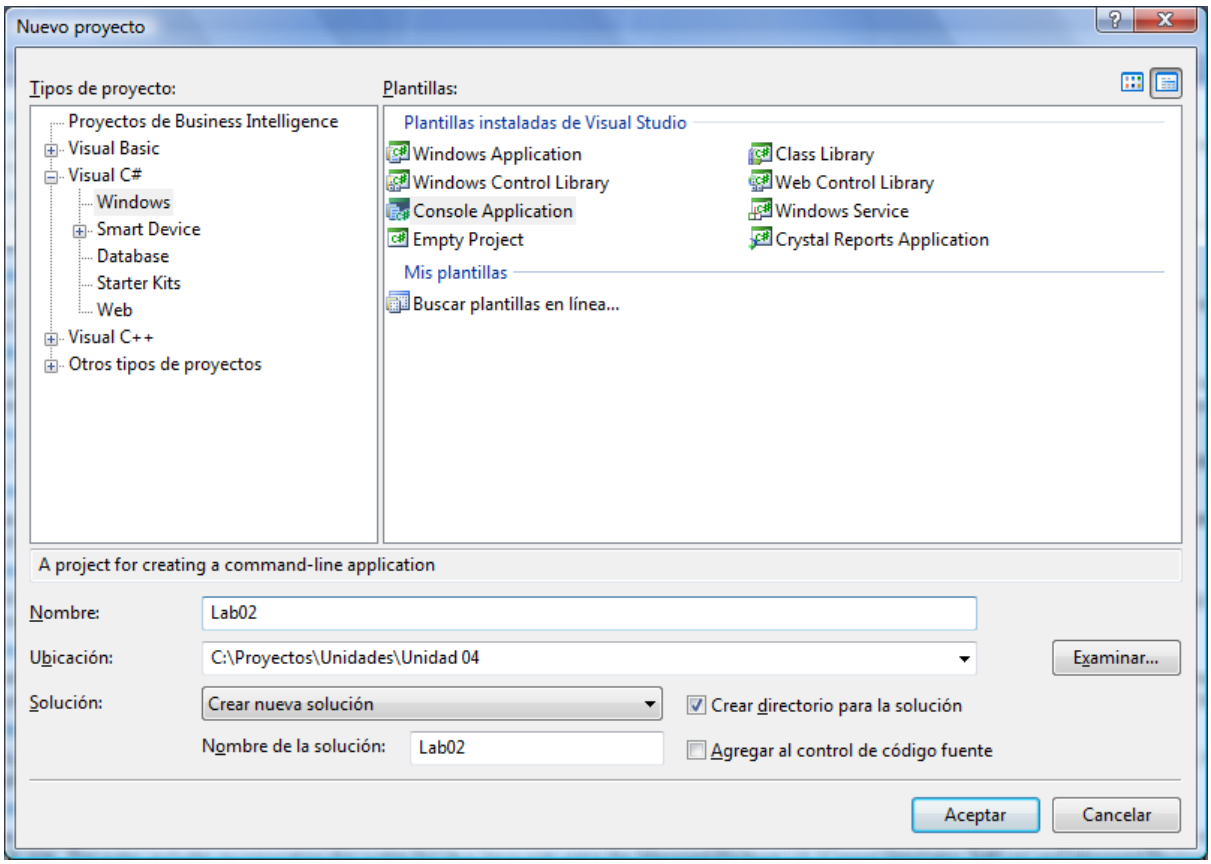
Duración Aproximada
20 minutos

Pasos

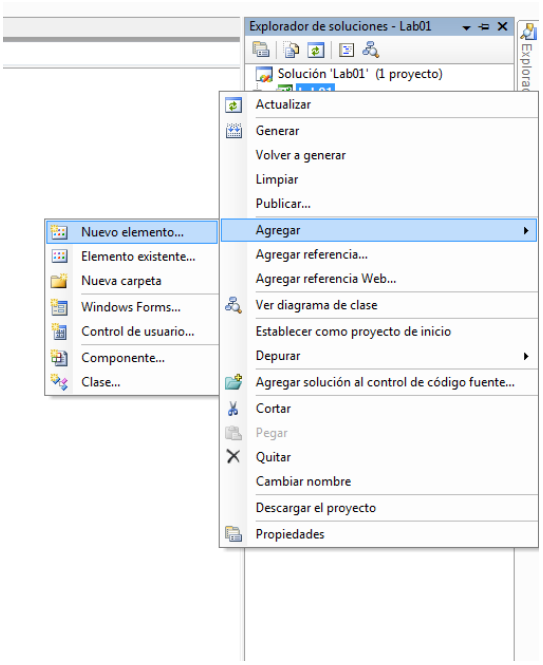
1) Abra el Visual Studio 2005 y haga clic en Archivo -> Nuevo -> Proyecto
2) Elija un proyecto de la categoría C# -> Windows -> Console Aplication.

   A) Completar los siguientes datos:
   - Nombre del Proyecto (**Name**): *Lab02*
   - Ubicación de la solución **(Location)**:

C:\Proyectos\Unidades\Unidad 04



3) Presione Aceptar.
4) Hacemos clic con el botón derecho sobre el proyecto Lab02. Allí elegimos Agregar -> Nuevo Elemento

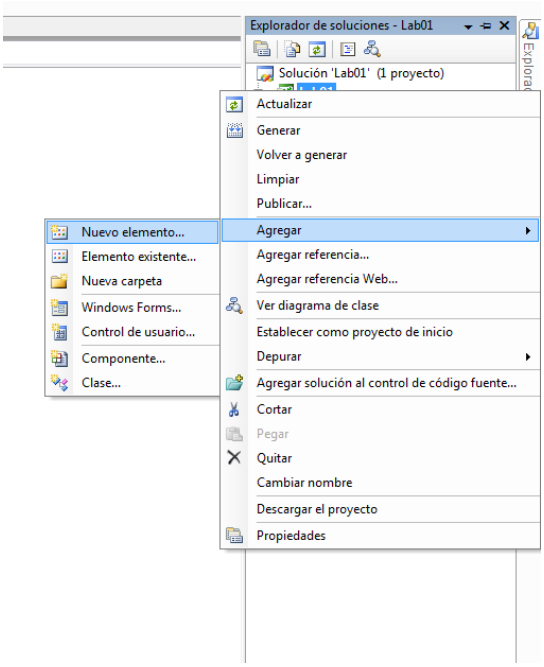


5) Elegimos Text File y lo llamamos agenda.txt

6) En agenda.txt pegamos:

```
ID,Nombre,Apellido,E-mail,Telefono
1,Juana,De Blanco,jdblanco@gmail.com,411-1111
2,Jose,Gonzales,johny_smithy@gmail.com,422-2222
3,Rodrigo,Rodriguez,rodrirodri@gmail.com,433-3333
```

7) Hacemos clic con el botón derecho sobre el proyecto Lab02. Allí elegimos Agregar -> Nuevo Elemento

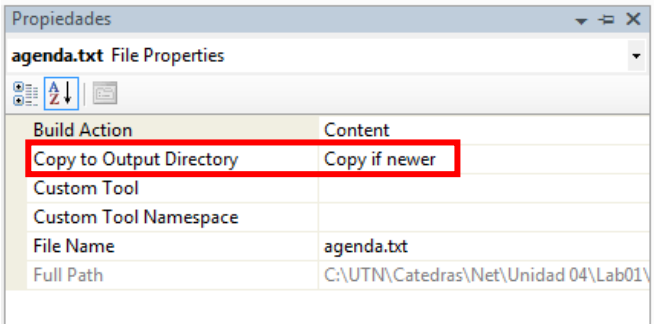


8) Elegimos XML File y lo llamamos agenda.xml

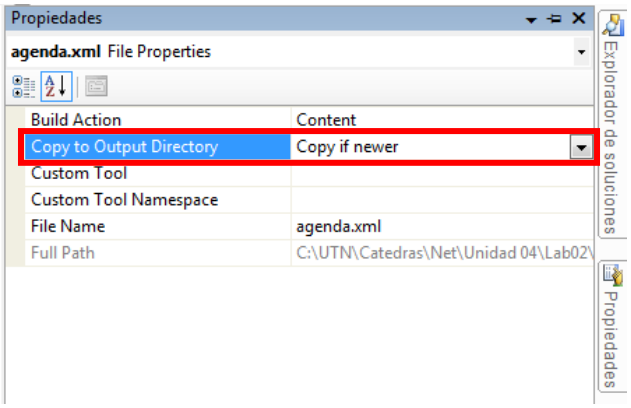
9) En agenda.xml pegamos:

```
<?xml version="1.0" standalone="yes"?>
<agenda>
	<contactos>
		<ID>1</ID>
		<Nombre>Juana</Nombre>
		<Apellido>De Blanco</Apellido>
		<E-mail>jdblanco@gmail.com</E-mail>
		<Telefono>411-1111</Telefono>
	</contactos>
	<contactos>
		<ID>2</ID>
		<Nombre>Jose</Nombre>
		<Apellido>Gonzales</Apellido>
		<E-mail>johny_smithy@gmail.com</E-mail>
		<Telefono>422-2222</Telefono>
	</contactos>
	<contactos>
		<ID>3</ID>
		<Nombre>Rodrigo</Nombre>
		<Apellido>Rodriguez</Apellido>
		<E-mail>rodrirodri@gmail.com</E-mail>
		<Telefono>433-3333</Telefono>
	</contactos>
</agenda>
```

10) Hacemos clic sobre agenda.txt y vamos a la pestaña de propiedades. Cambiamos la propiedad Copy to Output Directory a Copy if newer. Para que al compilar si el archivo agenda.txt no existe el mismo se genere.



11) Hacemos lo mismo con agenda.xml



Gracias a ADO.Net las operaciones para manipulan los datos en memoria son iguales tanto para los datos obtenidos del archivo TXT como del XML. Por este motivo crearemos una clase llamada ManejadorArchivo que implementará los métodos con todas las operaciones de manipulación de datos en memoria:

- listar
- nuevaFila
- editarFila
- eliminarFila

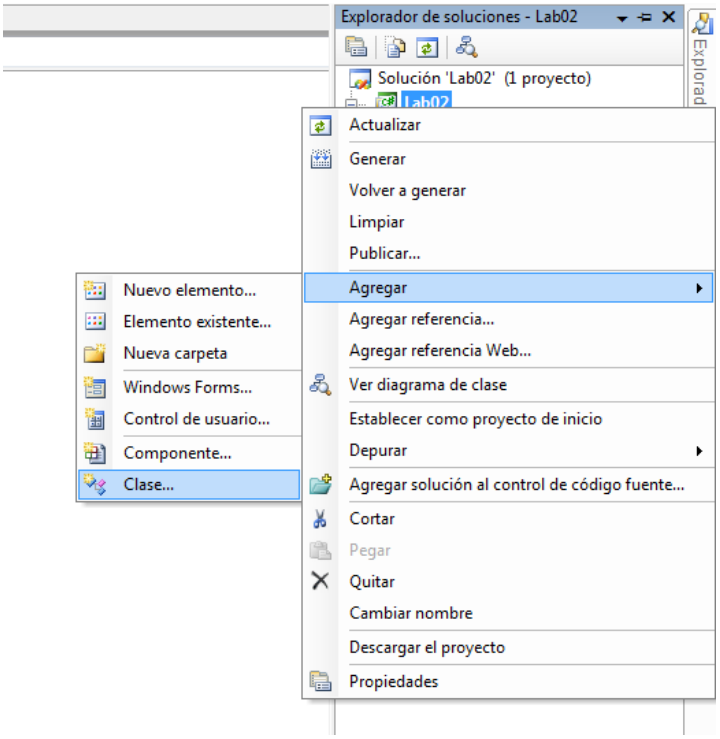
Además definiremos en la misma dos métodos:

- getTabla. Se encarga de acceder a los datos del archivo y darles forma de tabla.
- aplicaCambios. Se encarga de traducir los cambios hechos en la tabla en cambios para los archivos

Obviamente tanto la forma de acceder a los datos como la forma de realizar modificaciones en los archivos son diferentes para cada tipo de archivo, por ello la implementación específica de getTabla y aplicarCambios en archivos TXT y XML la realizarán las clases ManejadorArchivoTxt y ManejadorArchivoXml respectivamente.

Estas dos clases heredan de ManejadorArchivo.

12) Hacemos clic con el botón derecho sobre el proyecto Lab02. Allí elegimos Agregar -> Clase



La llamamos ManejadorArchivo.cs

13) En el archivo ManejadorArchivo.cs agregamos la directiva

using System.Data; y creamos la variable de instancia misContactos, la misma es del tipo DataTable.

La clase entonces queda así:

```
namespace Lab02
{
    public class ManejadorArchivo
    {

        protected DataTable misContactos;

        public ManejadorArchivo()
        {
            this.misContactos = this.getTabla();
        }

        public virtual DataTable getTabla()
        {

            return new DataTable();
        }

        public virtual void aplicaCambios()
        {

        }


        public void listar()
        {
            foreach (DataRow fila in this.misContactos.Rows)
            {
                if (fila.RowState != DataRowState.Deleted)
                {
                    foreach (DataColumn col in this.misContactos.Columns)
                    {
                        Console.WriteLine("{0}: {1}", col.ColumnName, fila[col]);
                    }
                    Console.WriteLine();
                }
            }
        }

        public void nuevaFila()
        {
            DataRow fila = this.misContactos.NewRow();
            foreach (DataColumn col in this.misContactos.Columns)
            {
                Console.Write("Ingrese {0}:",col.ColumnName);
                fila[col] = Console.ReadLine();
            }
            this.misContactos.Rows.Add(fila);
        }


        public void editarFila()
        {
            Console.WriteLine("Ingrese el número de fila a editar");
            int nroFila = int.Parse(Console.ReadLine());
            DataRow fila = this.misContactos.Rows[nroFila-1];
            for (int nroCol = 1; nroCol < this.misContactos.Columns.Count; nroCol++)
                //el 0 se omite por ser la ID
            {
                DataColumn col = this.misContactos.Columns[nroCol];
                Console.Write("Ingrese {0}:", col.ColumnName);
                fila[col] = Console.ReadLine();
            }

        }


        public void eliminarFila()
        {
            Console.WriteLine("Ingrese el número de fila a eliminar");
            int fila = int.Parse(Console.ReadLine());
            this.misContactos.Rows[fila-1].Delete();
        }
    }
}
```

Los métodos getTabla y aplicaCambios tienen el modificador virtual para permitir ser sobrescritos en las clases hijas.

Listar Contactos

En el método listar los contactos recorremos la colección Rows de la DataTable.
Para listar el nombre de la columna recorremos la colección de columnas del DataTable y utilizamos la propiedad ColumnName y para acceder a cada una de las celdas de la DataTable recorremos la colección de filas (DataRows) con el foreach y luego para cada fila accedemos a los valores de las celdas como si fuese un array pero no sólo podemos acceder con el índice de la columna sino también con el nombre de la columna o como en este caso con el objeto columna.

Nuevos Contactos

Para crear un nuevo contacto creamos una nueva fila en la DataTable con: this.misContactos.NewRow();

Es muy importante destacar que la única forma de crear una nueva fila es a partir de un DataTable. Esto se debe a que la fila debe tener una estructura de celdas y la misma es proporcionada por el DataTable a partir del cual se genera. Cabe aclarar que aunque una fila se genere a partir de una DataTable no significa que la misma esté agregada dentro de la colección de filas de la DataTable

Luego asignamos los valores a cada una de las celdas de la fila con:

```
foreach (DataColumn col in this.misContactos.Columns)
{
     Console.Write("Ingrese {0}:",col.ColumnName);
     fila[col] = Console.ReadLine();
}
```

Finalmente agregamos la fila a la tabla con:
`this.misContactos.Rows.Add(fila);`

Editar Contacto

Primero identificamos la fila de la tabla que queremos modificar aquí lo hacemos con el índice: `DataRow fila = this.misContactos.Rows[nroFila-1];`

También podría hacerse con un método de la DataTable llamado Select que permite filtrar un grupo de filas cuyos datos cumplen con determinados criterios.

Luego modificamos los datos de la fila con:

```
for (int nroCol = 1; nroCol < this.misContactos.Columns.Count; nroCol++)
//el 0 se omite por ser la ID
{
    DataColumn col = this.misContactos.Columns[nroCol];
    Console.Write("Ingrese {0}:", col.ColumnName);
    fila[col] = Console.ReadLine();
}
```

Aquí no es necesario agregarla a la DataTable como en el caso anterior porque la fila ya existía y se encontraba en la DataTable.

Eliminar Contacto

Para eliminar una fila simplemente localizamos la DataRow que deseamos eliminar. Luego simplemente eliminamos la fila con el método Delete de la DataRow.

14) Luego desde el explotador de soluciones agregamos las otras dos nuevas clases ManejadorArchivoTxt y ManejadorArchivoXml

15) En la clase ManejadorArchivoTxt utilizaremos un conector Ole Db para realizar las operaciones sobre un archivo plano delimitado por comas. Estas clases se encuentran en el namespace System.Data.OleDb por lo que incluiremos la directiva using System.Data.OleDb;

16) Luego creamos la propiedad de sólo lectura connectionString la misma devolverá el siguiente valor:

```
@"Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\Proyectos\Unidades\Unidad 04\Lab02\Lab02\bin\Debug;" +
"Extended Properties='text;HDR=Yes;FMT=Delimited'"
```

Un Connection String es un string que tiene los datos necesarios para conectarse a un origen de datos a través de una conexión. Para más información visitar http://www.connectionstrings.com/

Cada origen de datos requiere datos específicos, en el caso de archivos planos se requiere:

- Provider: indica cual driver de conexión utilizaremos.
- DataSource: en el caso de los archivos planos es la carpeta donde se encuentran los archivos.

Además de los datos obligatorios los connection string tienen datos opcionales, en este caso las Extended Properties que indica que el archivo es de tipo texto, tienen Header (encabezado con el nombre de las columnas) y el formato del archivo (en este caso delimitado por comas).

17) Sobrescribimos entonces el método getTabla. Dentro del mismo creamos una conexión y la abrimos.

18) Luego en el creamos un comando OleDb para obtener con la sentencia SQL para obtener los datos. Para ejecutar el comando utilizamos el método ExecuteReader que crea un objeto DataReader que nos permite acceder leer los datos de los archivos y cargarlos en el DataTable (a través del método Load.

19) A continuación cerramos la conexión.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.OleDb;

namespace Lab02
{
    public class ManejadorArchivoTxt:ManejadorArchivo
    {
        protected string connectionString
        {
            // en este connection string:
            //      HDR=Yes       : indica que el primer registro contiene los encabezados (nombres) de las columnas, no datos.
            //      FMT=CSVDelimited : indica que el los campos están delimitados por un caracter ,
            //      la arroba es para evitar tener que usar \\ para escribir la barra invertida en el string
            get
            {
                return @"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Proyectos\Unidades\Unidad 04\Lab02\Lab02\bin\Debug;" +
                       "Extended Properties='text;HDR=Yes;FMT=Delimited'";
            }
        }

        public override DataTable getTabla()
        {
            using (OleDbConnection Conn = new OleDbConnection(connectionString))
            {
                OleDbCommand cmdSelect = new OleDbCommand("select * from agenda.txt", Conn);
                Conn.Open();
                OleDbDataReader reader = cmdSelect.ExecuteReader();
                DataTable contactos = new DataTable();
                if (reader != null)
                {
                    contactos.Load(reader);
                }
                Conn.Close();
                return contactos;
            }
        }
```

20) Finalmente devolvemos la DataTable para utilizarla

21) A continuación sobrescribimos el método aplicaCambios

22) En el mismo creamos 3 comandos uno para agregar filas otro para modificarlas y finalmente uno para eliminarlas.

Para A cada uno de ellos escribimos la sentencia SQL que permite realizar dichas acciones y finalmente creamos los parámetros necesarios para que dada una fila se utilicen esos datos para insertar, modificar o eliminar dichos datos

23) En el mismo creamos 3 comandos uno para agregar filas otro para modificarlas y finalmente uno para eliminarlas.

24) Luego abrimos la conexión

25) Sobre la DataTable misContactos formamos buscamos cuales filas fueron agregadas con:

```
DataTable filasNuevas = this.misContactos.GetChanges(DataRowState.Added);
```

Sobre ellas ejecutamos el comando cmdInsert que realiza las inserciones.

26) Acto seguido buscamos las filas eliminadas con:

```
DataTable filasBorradas = this.misContactos.GetChanges(DataRowState.Deleted);
```

Sobre ellas ejecutaremos el comando cmdDelete que sirve para eliminar filas.

27) Luego buscamos las filas que tiene modificaciones con:
DataTable filasModificadas = this.misContactos.GetChanges(DataRowState.Modified);
Para ejecutar sobre ellas el comando cmdUpdate que es el encargado de modificar los datos.

```
public override void aplicaCambios()
{
    using (OleDbConnection Conn = new OleDbConnection(connectionString))
    {
        OleDbCommand cmdInsert = new OleDbCommand("insert into agenda.txt values(@id,@nombre,@apellido,@email,@telefono)", Conn);
        cmdInsert.Parameters.Add("@id", OleDbType.Integer);
        cmdInsert.Parameters.Add("@nombre", OleDbType.VarChar);
        cmdInsert.Parameters.Add("@apellido", OleDbType.VarChar);
        cmdInsert.Parameters.Add("@email", OleDbType.VarChar);
        cmdInsert.Parameters.Add("@telefono", OleDbType.VarChar);

        OleDbCommand cmdUpdate = new OleDbCommand(
              "update agenda.txt set nombre=@nombre, apellido=@apellido, e-mail=@email,telefono=@telefono where id=@id", Conn);
        cmdUpdate.Parameters.Add("@id", OleDbType.Integer);
        cmdUpdate.Parameters.Add("@nombre", OleDbType.VarChar);
        cmdUpdate.Parameters.Add("@apellido", OleDbType.VarChar);
        cmdUpdate.Parameters.Add("@email", OleDbType.VarChar);
        cmdUpdate.Parameters.Add("@telefono", OleDbType.VarChar);

        OleDbCommand cmdDelete = new OleDbCommand("delete from agenda.txt where id=@id", Conn);
        cmdDelete.Parameters.Add("@id", OleDbType.Integer);

        DataTable filasNuevas = this.misContactos.GetChanges(DataRowState.Added);
        DataTable filasBorradas = this.misContactos.GetChanges(DataRowState.Deleted);
        DataTable filasModificadas = this.misContactos.GetChanges(DataRowState.Modified);

        Conn.Open();

        if (filasNuevas != null)
        {
            foreach (DataRow fila in filasNuevas.Rows)
            {
                cmdInsert.Parameters["@id"].Value = fila["id"];
                cmdInsert.Parameters["@nombre"].Value = fila["nombre"];
                cmdInsert.Parameters["@apellido"].Value = fila["apellido"];
                cmdInsert.Parameters["@email"].Value = fila["e-mail"];
                cmdInsert.Parameters["@telefono"].Value = fila["telefono"];
                cmdInsert.ExecuteNonQuery();
            }
        }

        if (filasBorradas != null)
        {
            foreach (DataRow fila in filasBorradas.Rows)
            {
                cmdDelete.Parameters["@id"].Value = fila["id", DataRowVersion.Original];
                cmdDelete.ExecuteNonQuery();
            }
        }

        if (filasModificadas != null)
        {
            foreach (DataRow fila in filasModificadas.Rows)
            {
                cmdUpdate.Parameters["@id"].Value = fila["id"];
                cmdUpdate.Parameters["@nombre"].Value = fila["nombre"];
                cmdUpdate.Parameters["@apellido"].Value = fila["apellido"];
                cmdUpdate.Parameters["@email"].Value = fila["e-mail"];
                cmdUpdate.Parameters["@telefono"].Value = fila["telefono"];
                cmdUpdate.ExecuteNonQuery();
            }
        }
```

28) Para ejecutar cada uno de los comandos utilizamos el método ExecuteNonQuery.

29) Luego creamos en la clase ManejadroArchivoXml donde sobrescribimos los métodos de getTable y aplicaCambios

Los métodos son substancialmente más simples debido a que todo ADO.Net está construido sobre la idea de transferir datos y definir su estructura a través de XML, por este motivo la mayoría de los objetos de almacenamiento de datos tienen incorporados métodos para leer desde archivos XML y almacenar los datos en archivos XML.

En este caso utilizaremos el objeto DataSet, el cual puede contener varias DataTable.

El mismo permite guardar todas las DataTable con o sin su estructura a través del método WriteXml.

Adicionalmente, como en este caso, puede crear los DataTable y cargarlos con datos a partir de un XML a través del método ReadXml.

Los DataTables también tienen estas capacidades pero la razón por la cual elegimos el objeto DataSet es que tiene la capacidad de extrapolar la estructura de los DataTable a partir de un XML bien formado y valido.

La razón por la cual escribimos el XML dos veces es porque la primera vez lo hacemos sólo con los datos y en el segundo caso también almacenamos la estructura del mismo.

Luego de ejecutar la aplicación podrán ver las diferencias en los archivos XML en la carpeta C:\Proyectos\Unidades\Unidad 04\Lab02\Lab02\bin\Debug\

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace Lab02
{
    public class ManejadorArchivoXml: ManejadorArchivo
    {
        protected DataSet ds;

        public override System.Data.DataTable getTabla()
        {
            this.ds = new DataSet();
            this.ds.ReadXml("agenda.xml");
            return this.ds.Tables["contactos"];
        }

        public override void aplicaCambios()
        {
            this.ds.WriteXml("agenda.xml");
            this.ds.WriteXml("agendaconschema.xml", XmlWriteMode.WriteSchema);
        }
    }
}
```

30) Modificamos el programa principal de la siguiente forma

```
using System;
using System.Collections.Generic;
using System.Text;


namespace Lab02
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            ManejadorArchivo manejadorArch;
            Console.WriteLine("Elija el modo:");
            Console.WriteLine("1 - TXT");
            Console.WriteLine("2 - XML");
            if (Console.ReadLine() == "2")
            {
                manejadorArch=new ManejadorArchivoXml();
            }
            else
            {
                manejadorArch = new ManejadorArchivoTxt();
            }
            manejadorArch.listar();
            menu(manejadorArch);
        }

        static void menu(ManejadorArchivo manejadorArch)
        {
            string rta="";
            do
            {
                Console.WriteLine("1 - Listar");
                Console.WriteLine("2 - Agregar");
                Console.WriteLine("3 - Modificar");
                Console.WriteLine("4 - Eliminar");
                Console.WriteLine("5 - Guardar Cambios");
                Console.WriteLine("6 - Salir");
                rta = Console.ReadLine();
                switch (rta)
                {
                    case "1":
                        manejadorArch.listar();
                        break;
                    case "2":
                        manejadorArch.nuevaFila();
                        break;
                    case "3":
                        manejadorArch.editarFila();
                        break;
                    case "4":
                        manejadorArch.eliminarFila();
                        break;
                    case "5":
                        manejadorArch.aplicaCambios();
                        break;
                    default:
                        break;
                }
            } while (rta != "6");
        }
    }
}
```

Aclaraciones:

El manejador de archivos XML no realiza modificaciones en el archivo original. Simplemente lo borra y lo crea nuevamente. Hay muchas formas de manipular archivos XML, aquí se utilizó una pero también es muy

popular utilizar el objeto XMLDocument o XMLDataDocument pero la manipulación del mismo no se realiza a través de una tabla sino de un árbol con nodos. El beneficio de no manipular el archivo XML como una tabla es que permite la flexibilidad de manipular un XML que no puede representarse con una estructura fija relacional (filas y columnas).

El manejador de archivos de texto sólo permitirá listar y aplicar los cambios de las filas agregadas. Desafortunadamente el conector OleDB permite realizar selecciones y agregar filas al final pero no permite realizar modificaciones ni eliminaciones porque para eso el conector OleDB necesita utilizar un índice, algo que los archivos planos no tienen.

Sin embargo aquí lo se agregó a este laboratorio ya que esa es una forma de aplicar las modificaciones y eliminaciones cuando se realizan los cambios en una tabla de la base de datos.

31) Presionamos F5 para ejecutar.

Se recomienda poner un punto de interrupción dentro del método getTabla del ManejadorArchivoTxt. Avanzar paso a paso (F10) hasta luego de ejecuta la sentencia contactos.Load(reader);
Posicionar el mouse sobre la variable contactos y aparecerá un menú flotante con una pequeña lupa. Haga Clic sobre ella para visualizar la estructura y datos del DataTable.