C.R.U.D. Ops
Each tutorial is C# - if you get bored – when you check them out – note what is in the program.cs file – are there files for a client front end – each New Project type in VS can have similar page types but placed in different folders – More vocab EF Core – ASP.NET Core – fancy for C# with more or less files.

migrations – scaffolding – what VS can do to create dbase – CRUD code – Client forms etc. from C#.

Most if not all have a GitHub repo for a completed project to run.

c# - Beginner CRUD Console Application - Code Review Stack Exchange –

I have not run this one – but it is still pretty old school C# console – the code is a good read because of the operations the author is attempting.

- There is a discussion in the middle of using a loop device instead of recursion Recursion in C# - GeeksforGeeks – a function calling itself.
- Another reason maybe to loop besides iterating thru an array.

This was written in 2024 - web app that manages a database of movies – forms & pages CRUD to dbase – VS Studio does heavy lifting.

- Very quick to create New project and Run the template right away – uses bootstrap last time I checked.

Tutorial: Get started with Razor Pages in ASP.NET Core | Microsoft Learn

1. C# classes are added for managing movies in a database. The app's model classes use Entity Framework Core (EF Core) to work with the database. EF Core is an object-relational mapper (O/RM) that simplifies data access.
2. You write the model classes first, and EF Core creates the database.
3. The model classes are known as POCO classes (from "Plain-Old CLR Objects") because they don't have a dependency on EF Core.
4. AspNetCore.Docs/aspnetcore/tutorials/razor-pages/razor-pages-start/sample/RazorPagesMovie80 at main · dotnet/AspNetCore.Docs · GitHub

Remember API has no "Client" but is designed to do CRUD ops – URLs with commands attached might be part.

Tutorial: Create a minimal API with ASP.NET Core | Microsoft Learn

Advanced – more pages and parts – creates & updates dbase from C# using migrations - scaffolding

Razor Pages with Entity Framework Core in ASP.NET Core - Tutorial 1 of 8 | Microsoft Learn

Reference

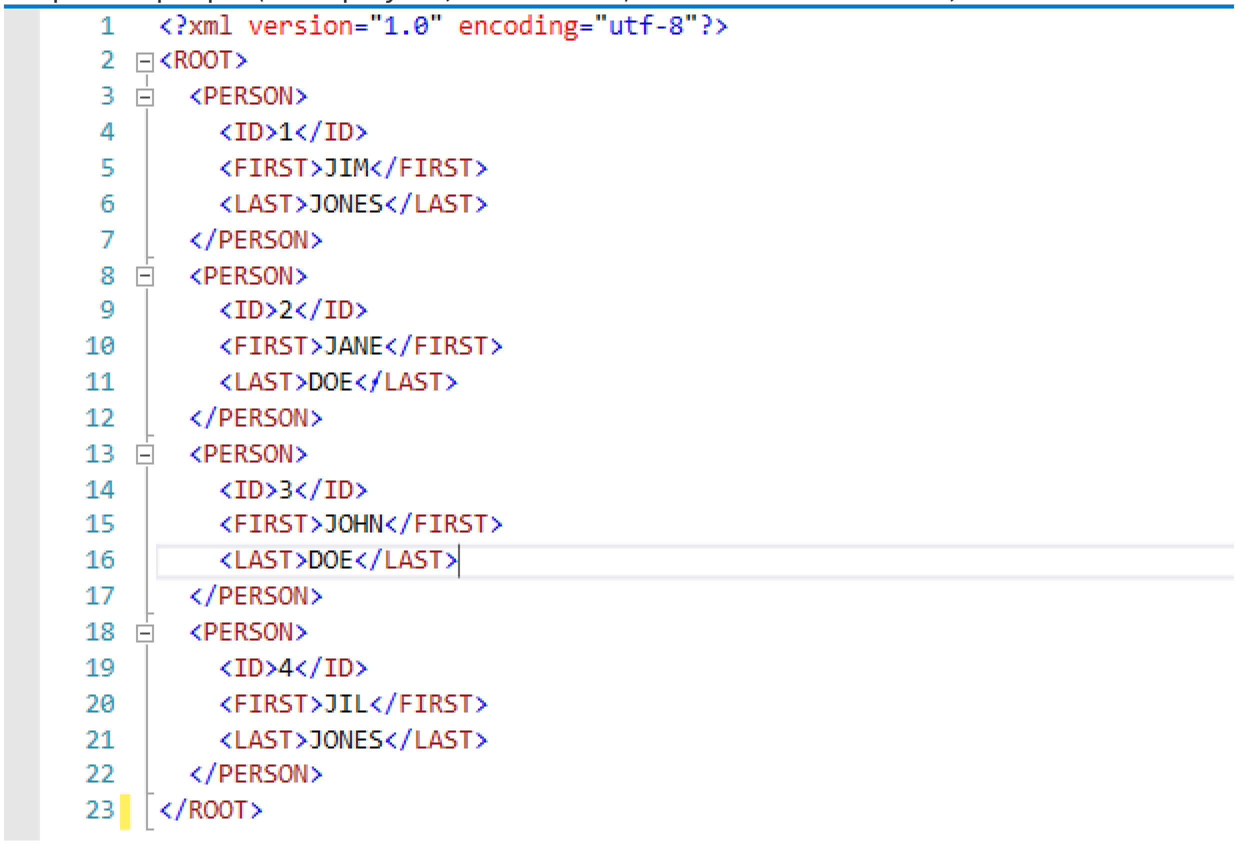
ConnectionStrings.com - Forgot that connection string? Get it here!

Other examples
C# CRUD Operations – CRUDzone (wordpress.com) – xml file

C# CRUD Operations Using XML file as data storage

There are two ways that you can perform xml CRUD operations. Using XPath or LinqToXml. We will see both here in action..XPath is simpler to learn. However you may need to know a thing or two while using Linq such as clauses or lambdas. I personally use LinqToXml more often due to less coding..

I've named my xml file as crud.xml but you can name it anything you prefer. Just make sure to use the correct path-filename. Below how it looks like. Basically a simple xml file where we keep list of people (or employees, students etc) with three elements: ID,FIRST and LAST.

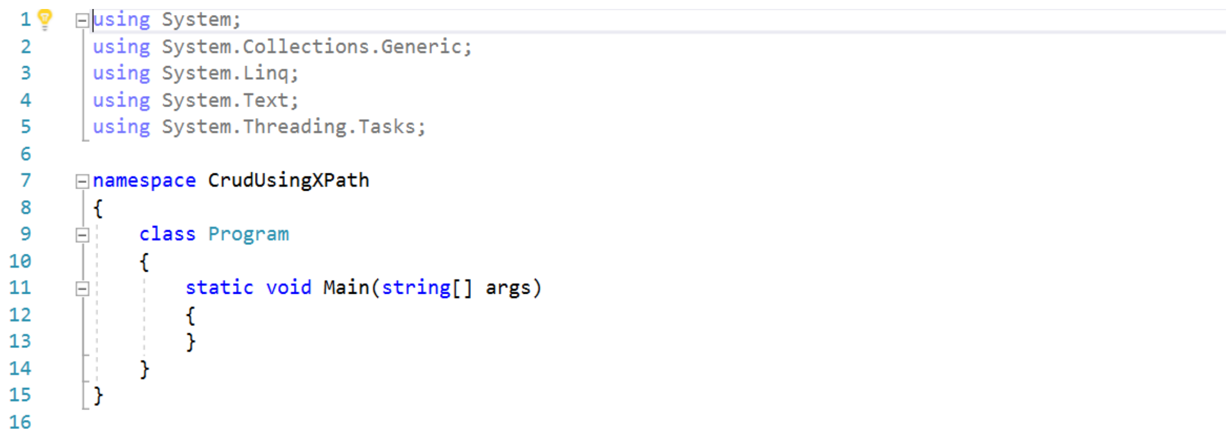


```xml
<?xml version="1.0" encoding="utf-8"?>
<ROOT>
  <PERSON>
    <ID>1</ID>
    <FIRST>JIM</FIRST>
    <LAST>JONES</LAST>
  </PERSON>
  <PERSON>
    <ID>2</ID>
    <FIRST>JANE</FIRST>
    <LAST>DOE</LAST>
  </PERSON>
  <PERSON>
    <ID>3</ID>
    <FIRST>JOHN</FIRST>
    <LAST>DOE</LAST>
  </PERSON>
  <PERSON>
    <ID>4</ID>
    <FIRST>JIL</FIRST>
    <LAST>JONES</LAST>
  </PERSON>
</ROOT>
```

First off, lets take a look at CRUD operations using XPath. Please note i'm going to try to keep this as simple as possible so no error handling etc will be discussed..

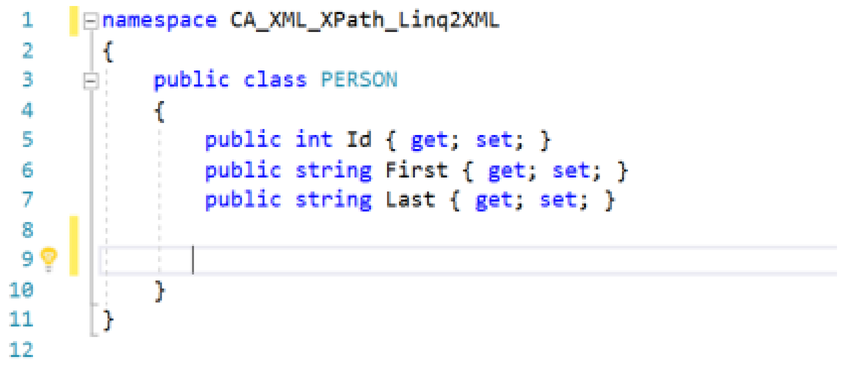
1. XML CRUD Operation using XPath

Lets create a new Console Application…



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CrudUsingXPath
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

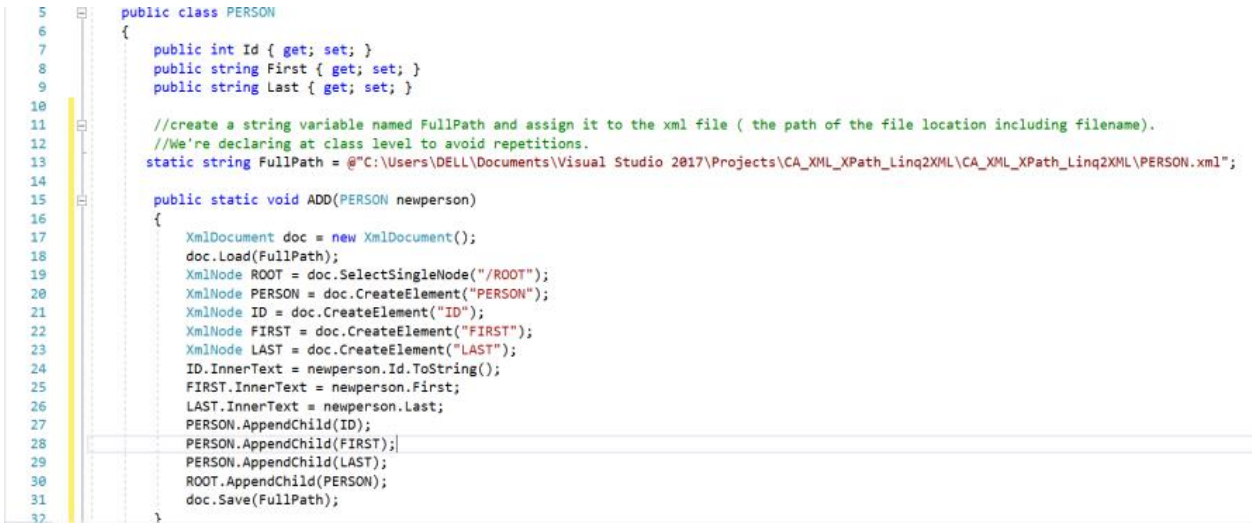
Next, lets add a Person class to our solution with 3 properties: Id, First, Last.



```
namespace CA_XML_XPath_Linq2XML
{
    public class PERSON
    {
        public int Id { get; set; }
        public string First { get; set; }
        public string Last { get; set; }


    }
}
```

Now that we have added our class.. Next add system.xml namespace.. lets move on to creating our Methods. I'm going to create inside the Person class that we've just added as static methods to perform CRUD operations on our xml file.
The beauty of XPath when you look at the code you can see what you're selecting or doing clearly in a tree like structure . Easier to understand.

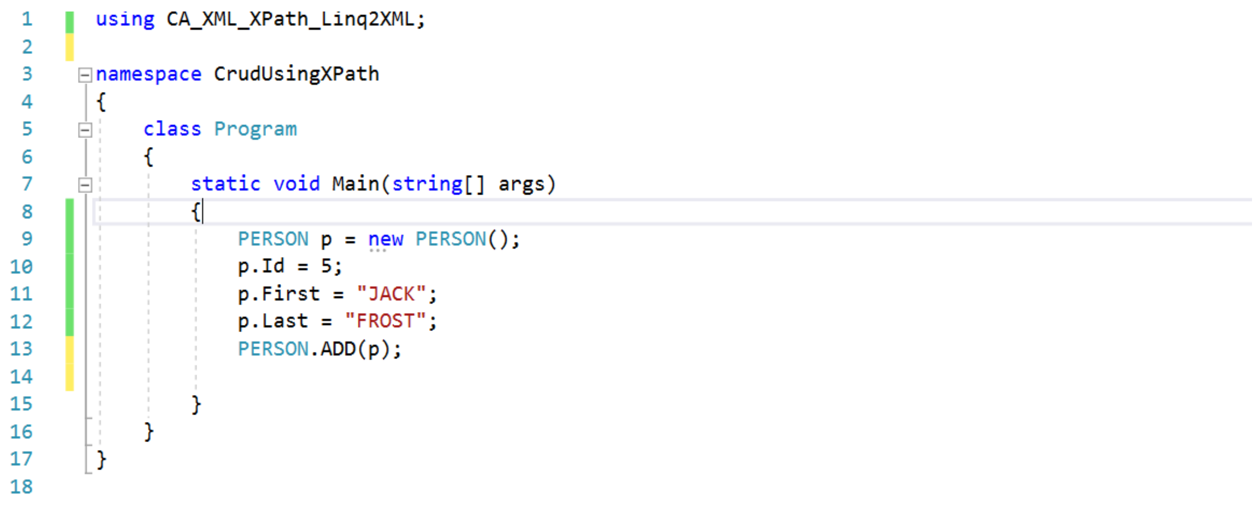
1.ADD or CREATE



```
    public class PERSON
    {
        public int Id { get; set; }
        public string First { get; set; }
        public string Last { get; set; }

        //create a string variable named FullPath and assign it to the xml file ( the path of the file location including filename).
        //We're declaring at class level to avoid repetitions.
       static string FullPath = @"C:\Users\DELL\Documents\Visual Studio 2017\Projects\CA_XML_XPath_Linq2XML\CA_XML_XPath_Linq2XML\PERSON.xml";

        public static void ADD(PERSON newperson)
        {
            XmlDocument doc = new XmlDocument();
            doc.Load(FullPath);
            XmlNode ROOT = doc.SelectSingleNode("/ROOT");
            XmlNode PERSON = doc.CreateElement("PERSON");
            XmlNode ID = doc.CreateElement("ID");
            XmlNode FIRST = doc.CreateElement("FIRST");
            XmlNode LAST = doc.CreateElement("LAST");
            ID.InnerText = newperson.Id.ToString();
            FIRST.InnerText = newperson.First;
            LAST.InnerText = newperson.Last;
            PERSON.AppendChild(ID);
            PERSON.AppendChild(FIRST);
            PERSON.AppendChild(LAST);
            ROOT.AppendChild(PERSON);
            doc.Save(FullPath);
        }
```

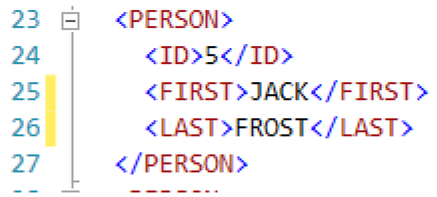
Lets take a look at the code for a minute. First, we creating a XmlDocument object and assigning to doc variable. Then loading our xml file. Next, we have a ROOT variable of type XmlNode with that we're selecting the "/ROOT" node which is the parent node. Next, we're telling our object to create a "PERSON" node that's child a child node (adds these tags to the file: <PERSON></PERSON>). Next, we're saying create the nodes id, first and last . And then assigning values to innertext of these nodes using our newperson. Next, we're telling the PERSON node to add childnodes of ID, FIRST AND LAST and telling ROOT to add PERSON child node and finally save the file.

Now we can call our brand new ADD method from the Main. As seen below we first create an object of our person class and then call the Add method inside the Person class and passing in the object just created.



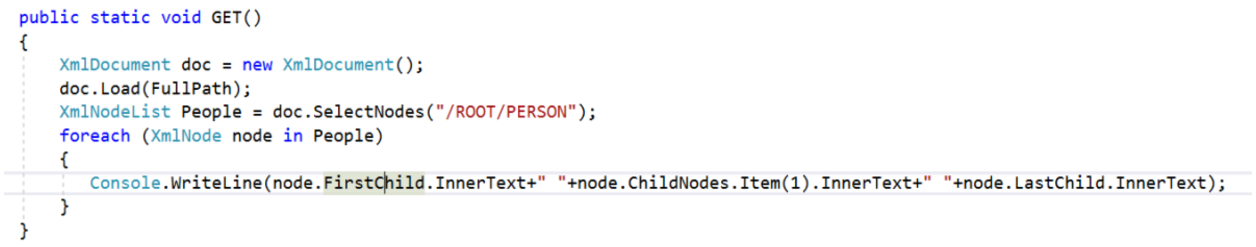
```
using CA_XML_XPath_Linq2XML;

namespace CrudUsingXPath
{
    class Program
    {
        static void Main(string[] args)
        {
            PERSON p = new PERSON();
            p.Id = 5;
            p.First = "JACK";
            p.Last = "FROST";
            PERSON.ADD(p);

        }
    }
}
```

When we run the program we'll see that a new person with id of 5, (JACK FROST) is added to the xml file.



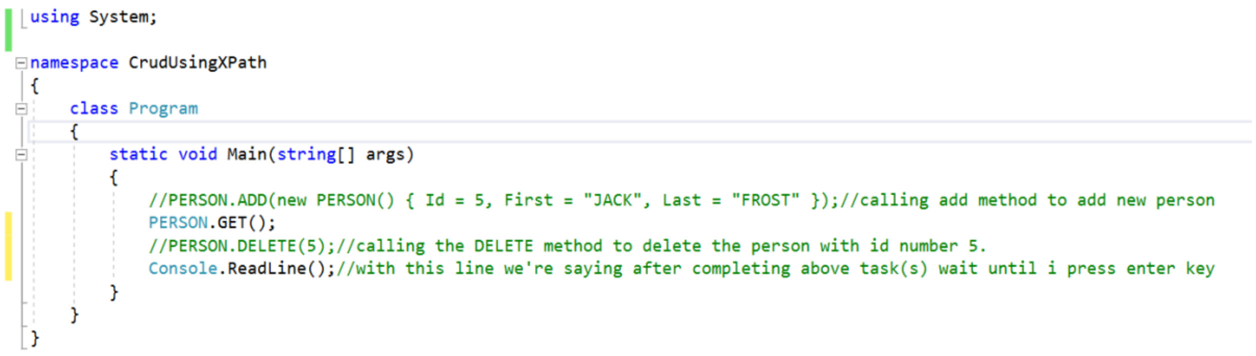
```xml
<PERSON>
  <ID>5</ID>
  <FIRST>JACK</FIRST>
  <LAST>FROST</LAST>
</PERSON>
```

2. GET or READ



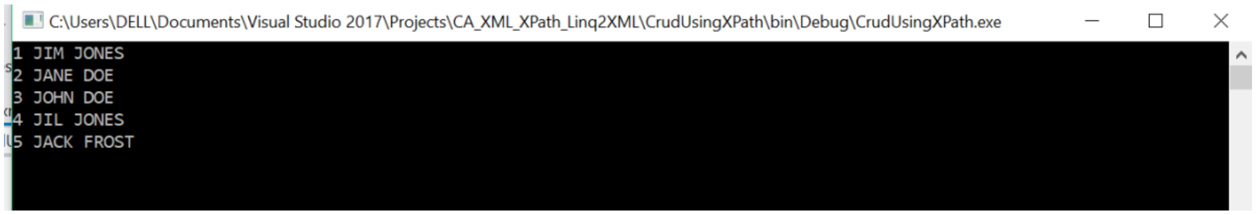
```csharp
public static void GET()
{
    XmlDocument doc = new XmlDocument();
    doc.Load(FullPath);
    XmlNodeList People = doc.SelectNodes("/ROOT/PERSON");
    foreach (XmlNode node in People)
    {
        Console.WriteLine(node.FirstChild.InnerText+" "+node.ChildNodes.Item(1).InnerText+" "+node.LastChild.InnerText);
    }
}
```

Here, after loading the file we're creating a new variable type of XmlNodeList (notice we've used SelectNodeList instead of SelectSingleNode) and assigning all PERSON nodes. And finally, looping thru each node and out putting the InnerText to screen.



```csharp
using System;

namespace CrudUsingXPath
{
    class Program
    {
        static void Main(string[] args)
        {
            //PERSON.ADD(new PERSON() { Id = 5, First = "JACK", Last = "FROST" });//calling add method to add new person
            PERSON.GET();
            //PERSON.DELETE(5);//calling the DELETE method to delete the person with id number 5.
            Console.ReadLine();//with this line we're saying after completing above task(s) wait until i press enter key
        }
    }
}
```

When we call and run it we get a list of people displayed in console window



```
C:\Users\DELL\Documents\Visual Studio 2017\Projects\CA_XML_XPath_Linq2XML\CrudUsingXPath\bin\Debug\CrudUsingXPath.exe
1 JIM JONES
2 JANE DOE
3 JOHN DOE
4 JIL JONES
5 JACK FROST
```

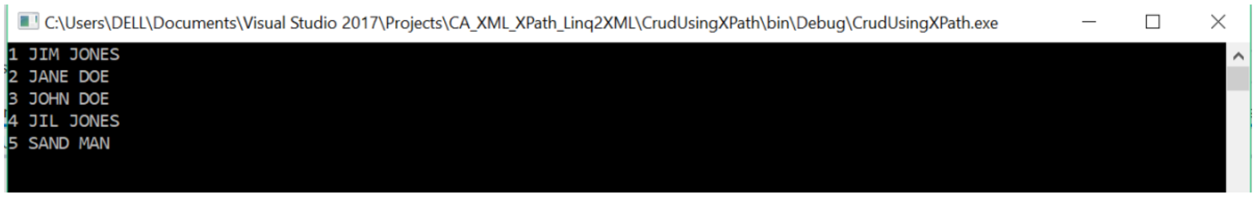
3.EDIT or UPDATE : Lets now see how we can edit or update a person's information based on the id criteria.

```
public static void UPDATE(int id, PERSON newperson)
{
    XmlDocument doc = new XmlDocument();
    doc.Load(FullPath);
    XmlNode ROOT = doc.SelectSingleNode("ROOT");
    XmlNode OldPerson = doc.SelectSingleNode("ROOT/PERSON[ID= " + id + "]");
    OldPerson.ChildNodes.Item(1).InnerText = newperson.First;
    OldPerson.LastChild.InnerText = newperson.Last;
    doc.Save(FullPath);
}
```

As we can see we're saying go get a single node that meets our criteria in a single statement by using the complete path including criteria. Well, what happens if our criteria is a variable of string type not int. In that case, we'll need to use single quotes on each side of the variable or plain text if not using variable, like: "ROOT/PERSON[FIRST='FROST']". Lets call our method and supply the new values:

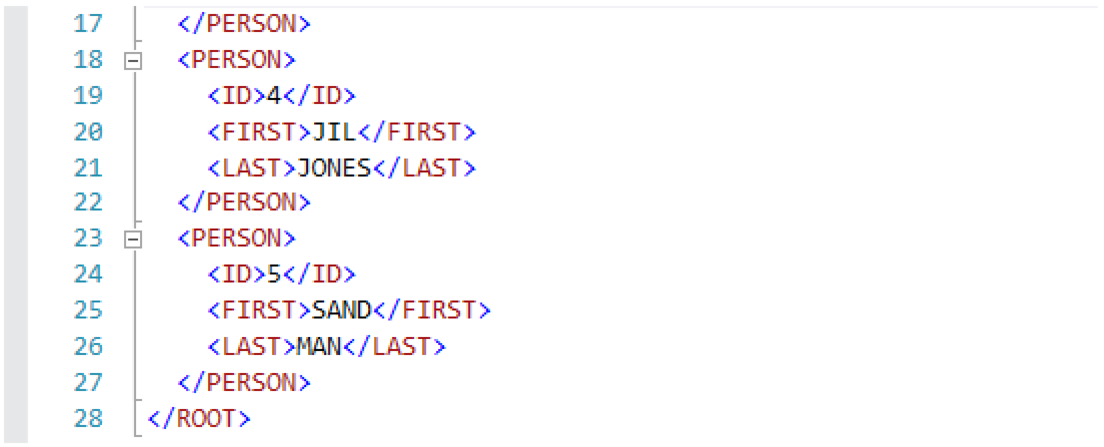
```
static void Main(string[] args)
{
    //PERSON.ADD(new PERSON() { Id = 5, First = "JACK", Last = "FROST" });//calling add method to add new person

    //PERSON.GET();//get (read) all

    PERSON.UPDATE(5, new PERSON() { First = "SAND", Last = "MAN" });//update the person with ID of 5 with the new person values

    //PERSON.DELETE(5);//calling the DELETE method to delete the person with id number 5.

    Console.ReadLine();//with this line we're saying after completing above task(s) wait until i press enter key
}
```

Now lets comment out the line where we call Update method and uncomment GET method:



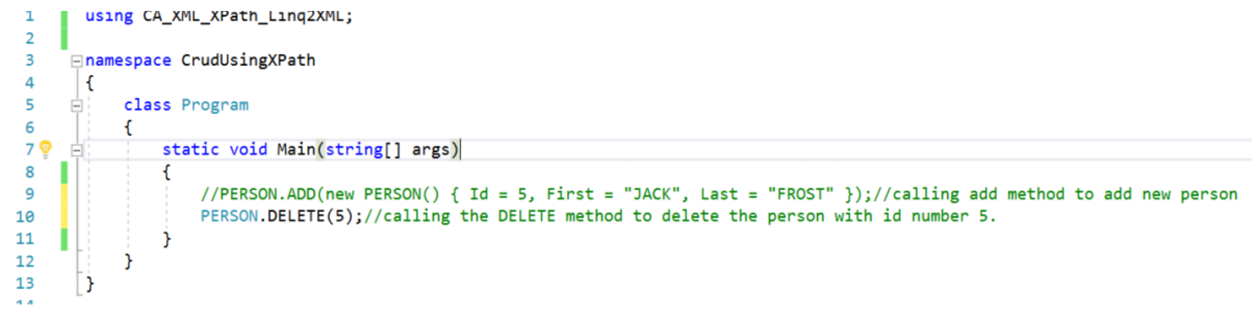
```
C:\Users\DELL\Documents\Visual Studio 2017\Projects\CA_XML_XPath_Linq2XML\CrudUsingXPath\bin\Debug\CrudUsingXPath.exe
1 JIM JONES
2 JANE DOE
3 JOHN DOE
4 JIL JONES
5 SAND MAN
```

And when we check our xml file we'll see its updated with new values.



```
  </PERSON>
  <PERSON>
    <ID>4</ID>
    <FIRST>JIL</FIRST>
    <LAST>JONES</LAST>
  </PERSON>
  <PERSON>
    <ID>5</ID>
    <FIRST>SAND</FIRST>
    <LAST>MAN</LAST>
  </PERSON>
</ROOT>
```

4.DELETE: Lets take a look at deleting now.. Here, we want to delete a person based on the ID number that was passed in from the Main.



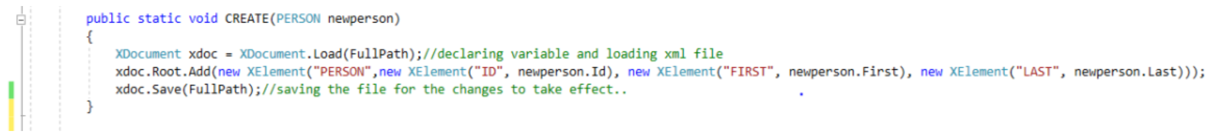
```
using CA_XML_XPath_Linq2XML;

namespace CrudUsingXPath
{
    class Program
    {
        static void Main(string[] args)
        {
            //PERSON.ADD(new PERSON() { Id = 5, First = "JACK", Last = "FROST" });//calling add method to add new person
            PERSON.DELETE(5);//calling the DELETE method to delete the person with id number 5.
        }
    }
}
```

2. XML CRUD Operations using Linq To XML

Basically we'll be doing the same operations as above but using slightly a different technique. We can do the same thing with less coding.

Lets dive in. First off, we will need to add system.linq and system.xml.linq namespaces to our Person class as using statements.

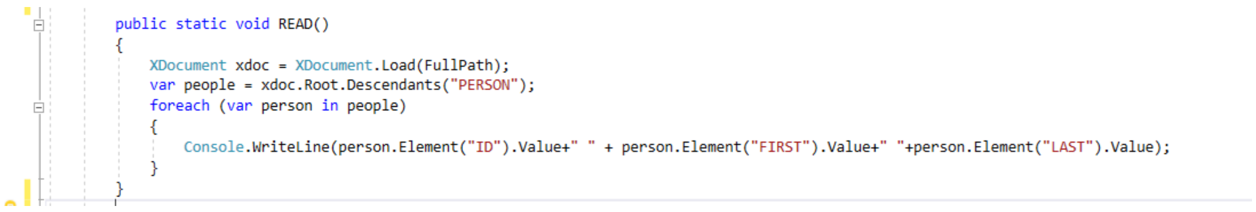
- CREATE or ADD



```
public static void CREATE(PERSON newperson)
{
    XDocument xdoc = XDocument.Load(FullPath);//declaring variable and loading xml file
    xdoc.Root.Add(new XElement("PERSON",new XElement("ID", newperson.Id), new XElement("FIRST", newperson.First), new XElement("LAST", newperson.Last)));
    xdoc.Save(FullPath);//saving the file for the changes to take effect..
}
```

As we can see we're creating the instance of XDocumnent and loading the file in a single line and assigning to "xdoc" variable. Then, we telling xdoc to grab the root and add new person child element "PERSON" . Notice before closing the bracket right after comma we're adding another xelement named "ID"which becomes child of "PERSON" and notice with "ID" we're supplying the value as well in the brackets ("ID", nerperson).. Briefly, it will add <PERSON></PERSON> tags first. Next,between these tags it will add <ID></ID> ,<FIRST></FIRST>,<LAST></LAST> tags with supplied vales.

- READ or GET

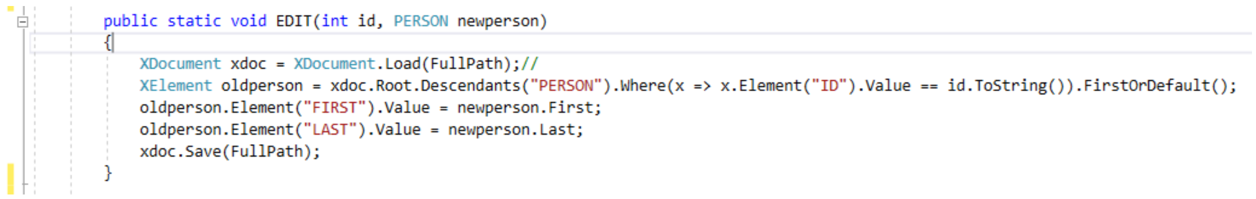


```
public static void READ()
{
    XDocument xdoc = XDocument.Load(FullPath);
    var people = xdoc.Root.Descendants("PERSON");
    foreach (var person in people)
    {
        Console.WriteLine(person.Element("ID").Value+" " + person.Element("FIRST").Value+" "+person.Element("LAST").Value);
    }
}
```

Here, after loading our xml file into our xdoc variable we're creating a variable named "people" and populating it with all child elements of "PERSON". And then looping thru each person in our foreach loop and displaying on the console screen.

- UPDATE or EDIT

From time to time we'll need to change the values of a particular record in our database and that's where EDIT/UPDATE comes in..

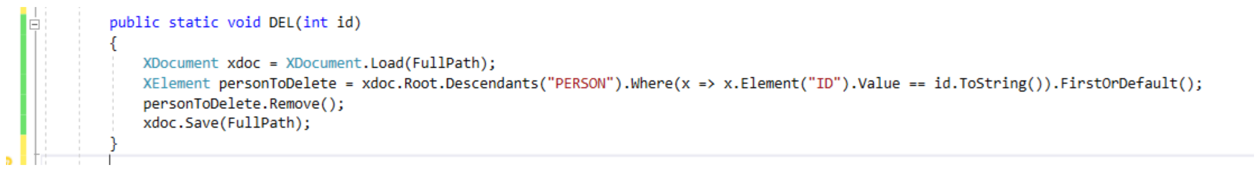


```
public static void EDIT(int id, PERSON newperson)
{
    XDocument xdoc = XDocument.Load(FullPath);//
    XElement oldperson = xdoc.Root.Descendants("PERSON").Where(x => x.Element("ID").Value == id.ToString()).FirstOrDefault();
    oldperson.Element("FIRST").Value = newperson.First;
    oldperson.Element("LAST").Value = newperson.Last;
    xdoc.Save(FullPath);
}
```

Again, first we load our xml document into our xdoc variable (we could name this anything). And then we're trying to locate the record by creating a new variable called "oldperson" and filtering our document by adding a where clause. Our criteria coming in when this method is called which is id. So we're saying find me the person where the person id equals to the id i'm supplying. And once that line is executed and the oldperson variable will be populated with the person that has the id we've supplied. And once we got hold of that person we want to change the "FIRST" and "LAST" name and assigning to the newperson variable which we're passing in when we call the method. And finally, saving our document.

- DELETE or REMOVE

This is very similar to UPDATE/EDIT.



```
public static void DEL(int id)
{
    XDocument xdoc = XDocument.Load(FullPath);
    XElement personToDelete = xdoc.Root.Descendants("PERSON").Where(x => x.Element("ID").Value == id.ToString()).FirstOrDefault();
    personToDelete.Remove();
    xdoc.Save(FullPath);
}
```

Here, we're naming our variable personToDelete and locating the record we wanna delete by filtering by id (using where clause and lambda expression) that we've supplied. And calling the Remove() method to delete the record. And finally saving the file for changes to take effect.

CRUD Operation In C# Application (c-sharpcorner.com) – SQL dabase

using SQL database to insert, update and delete operation. Before starting, you should add DLL and afterwards, you should add namespace under it.

Step 1

using System.Data.SqlClient;

You should use namespace given above to connect with SQL database.

Step2

You have to declare connection string outside the class.

```
SqlConnection con= new SqlConnection("Data Source=.;Initial Catalog=Sample;Integrated Security=true;");
SqlCommand cmd;
SqlDataAdapter adapt;
//ID variable used in Updating and Deleting Record
int ID = 0;
```

Step 3

Insert data in the database, as sgiven below.

```
if (txt_Name.Text != "" && txt_State.Text != "") {
    cmd = new SqlCommand("insert into tbl_Record(Name,State) values(@name,@state)", con);
    con.Open();
    cmd.Parameters.AddWithValue("@name", txt_Name.Text);
    cmd.Parameters.AddWithValue("@state", txt_State.Text);
    cmd.ExecuteNonQuery();
    con.Close();
    MessageBox.Show("Record Inserted Successfully");
    DisplayData();
      ClearData();
} else {
      MessageBox.Show("Please Provide Details!");
}
```

Step 4

Updating record is given below.

```
if (txt_Name.Text != "" && txt_State.Text != "") {
    cmd = new SqlCommand("update tbl_Record set Name=@name,State=@state where ID=@id", con);
    con.Open();
    cmd.Parameters.AddWithValue("@id", ID);
    cmd.Parameters.AddWithValue("@name", txt_Name.Text);
    cmd.Parameters.AddWithValue("@state", txt_State.Text);
    cmd.ExecuteNonQuery();
    MessageBox.Show("Record Updated Successfully");
    con.Close();
      DisplayData();
      ClearData();
} else {
      MessageBox.Show("Please Select Record to Update");
}
```

Step 5

Display record is shown below.

```
con.Open();
DataTable dt = new DataTable();
adapt = new SqlDataAdapter("select * from tbl_Record", con);

adapt.Fill(dt);
```

```
dataGridView1.DataSource = dt;
con.Close();
```

Step 6

Proceed, as shown below to delete the record.

```
if (ID != 0) {
    cmd = new SqlCommand("delete tbl_Record where ID=@id", con);
    con.Open();
    cmd.Parameters.AddWithValue("@id", ID);
    cmd.ExecuteNonQuery();
    con.Close();
    MessageBox.Show("Record Deleted Successfully!");
    DisplayData();
    ClearData();
} else {
     MessageBox.Show("Please Select Record to Delete");
}
}
```

At last, I have called clear method to clear all the textboxes.

```
txt_Name.Text = "";
txt_State.Text = "";
ID = 0;
```