### Modulul 15. XML și C#.

#### Ce este XML?

XML sau Extensible Markup Language este un limbaj simplu creat pentru stocarea, transmiterea și prezentarea datelor.

În prezent XML este unul din cele mai populare limbaje pentru transmiterea informației dintre:

* Program către program
* Persoană către persoană
* Persoană către program

Din motiv că XML este ușor de citit și în același timp sigur pentru programe(XML include sistem de detectare a erorilor).

XML este un limbaj destinat pentru stocarea, transmiterea și uneori afișarea datelor, acesta nu execută comenzi.

**Sintaxa XML:**

|  |
| --- |
| <notita>  <data>2020-06-23</data>  <hour>12:00</hour>  <to>XXX</to>  <from>YYY</from>  <body>Exemplu!</body>  </notita> |

Ca și HTML, XML este un limbaj markup, dar spre deosebire de HTML, XML-ul nu a fost construit pentru prezentarea datelor, mai mult pentru stocare și transmiterea lor.

O altă diferentă este că în XML nu există tag-uri predefinite ca și în HTML, adică tag-urile precum <p> <h1> etc. care deja există în HTML și au o funcție prestabilită, în XML nu există.

În XML utilizatorul crează singur tag-uri și modul în care acestea vor fi prezentate.

#### Istoria și motivele apariției.

XML a apărut din necesitatea simplă de a stoca și transmite date între utilizatori, programe independent de sistemul de operare, limbajul de programare sau limba acestor date. Pe scurt XML permite transmiterea datelor și prelucrarea datelor într-un mod universal.

XML a fost creat în anul 1996 de către un grup de 11 programatori și procesul de dezvoltare a limbajului a continuat activ până în anul 2000. După aceea au fost făcute unele modificări și îmbunătățiri dar esența limbajului a rămas neschimbată.

După apariția și popularizarea XML-ului o mare parte din aplicații desktop, web indiferent de limbajul de programare pe care îl utilizau sau platforma pe care activau au început să implementeze limbajul pentru că acesta oferă oportunitatea de a primi sau transmite date către oricare altă aplicație care folosește XML.

**În esență, XML-ul reprezintă un traducător universal care permite aplicațiilor să comunice între ele.**

#### Structura și sintaxa unui document XML.

Structura unui document XML este una ierarhică, ceea ce semnifică că acesta începe de la un elemente părinte de bază, care poate include mai multe elemente copil. Aceste elemente, la rândul lor, pot avea alte elemente copil, formând astfel o structură ierarhică.

**Structura unui document XML:**

|  |
| --- |
|  |
| <root>  <copil1>  <copil11>.....</copil11>  </copil1>  <copil2>  </child2>  <copil3>  </child3>  </root> |

Observăm câteva lucruri în segmentul de cod de mai sus:

1. Fiecare copil este inclus în interiorul altui tag părinte, și toate sunt incluse în interiorul tag-ului root(rădăcină). **Denumirea acestor tag-uri nu este prestabilită o puteți seta pe oricare.**
2. Fiecare tag deschis cu sintaxa <copil> va fi închis cu sintaxa </copil>. Conținutul tag-ului este plasat între secvența de deschidere și închidere.

Să considerăm un document care conține informația despre câteva cărți.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <bookstore>  <book category="cooking">  <title lang="en">Everyday Italian</title>  <author>Giada De Laurentiis</author>  <year>2005</year>  <price>30.00</price>  </book>  <book category="children">  <title lang="en">Harry Potter</title>  <author>J K. Rowling</author>  <year>2005</year>  <price>29.99</price>  </book>  <book category="web">  <title lang="en">Learning XML</title>  <author>Erik T. Ray</author>  <year>2003</year>  <price>39.95</price>  </book>  </bookstore> |

După cum vedeți fiecare document XML începe cu linia **<?xml version="1.0" encoding="UTF-8"?>** aceasta include versiunea XML care se folosește și modul de codare a datelor(UTF-8 sau UTF-16).

În continuare avem tag-ul de bază al documentului care este **bookstore**, în interiorul acestuia vom plasa tot restul conținutului din document.

În interiorul la bookstore vom avea mai multe cărți cu tagul book și o anumită categorie category, care va ajuta la găsirea cărților dintr-o anumită sau mai multe categorii. În interior este plasată informația despre autor, an și prețul cărții.

#### Utilizarea XML în C#.

Platforma .NET oferă o multitudine de clase care permit citirea, înscrierea în documente XML și multe alte operații. Toate aceste operații se află în librăria **System.Xml** pe care trebuie necesar să o includem când efectuăm operații cu XML.

1. **Crearea și înscrierea unui document XML**

Cea mai simplă operație pe care o putem efectua este crearea unui document XML, dar anume aceasta face posibilă stocarea datelor dintr-un program și transmiterea acestora către alte programe.

Pentru a crea și înscrie într-un document XML vom folosi clasa XmlTextWriter și un constructor care ne permite să creăm un obiect nou din această clasă.

**Exemplu:**

|  |
| --- |
| namespace XML  {  class CreareXML  {  static void Main()  {  XmlTextWriter writer = new XmlTextWriter("exemplu.xml", System.Text.Encoding.UTF8);  writer.WriteStartDocument(true);  writer.Formatting = Formatting.Indented;  writer.WriteStartElement("Exemplu"); //<Exemplu>  writer.WriteStartElement("Numar"); //<Numar>  writer.WriteString("1"); //continut  writer.WriteEndElement(); //</Numar>  writer.WriteEndElement(); //</Exemplu>  writer.WriteEndDocument();  writer.Close();  Console.Write("Fisier XML creat cu succes");  }  }  } |

După ce am creat un XmlTextWriter vom folosi metodele WriteStartElement și WriteEndElement pentru a deschide și închide taguri(<Exemplu></Exemplu>). Pentru a înscrie o valoarea în interiorul tagului folosim funcția WriteString.

La sfârșitul documentului folosim metodele WriteEndDocument și Close pentru a indica că nu mai înscriem nimic și că vrem să închidem Writerul creat.

În final vom indica la consolă că am creat cu succes documentul XML.

Probleme pentru rezolvare.

### **Program 1:**

1. Pe baza exemplului de mai sus creați un alt fișier XML care va conține cel puțin 2 elemente cu 2 caracteristici.

**Exemplu**:

<UtilizatoriDatabase>

<Utilizator>

<Utilizator\_ID>123wertqy</Utilizator\_ID>

<Utilizator\_password>aaa</Utilizator\_password>

</Utilizator>

<Utilizator>

<Utilizator\_ID>2563wertqy</Utilizator\_ID>

<Utilizator\_password>bbb</Utilizator\_password>

</Utilizator>

</UtilizatoriDatabase>

Deși programul de mai sus ne permite să realizăm obiectivele propuse acesta nu este eficient și util pentru cantități mari de informație, din acest motiv putem extrage operați de înscriere în document și să o includem într-o metodă aparte.

**Exemplu:**

|  |
| --- |
| namespace XML2  {  class CreareXMLEficient  {  static void Main()  {  //Inceput document  XmlTextWriter writer = new XmlTextWriter("exemplu2.xml", System.Text.Encoding.UTF8);  writer.WriteStartDocument(true);  writer.Formatting = Formatting.Indented;  writer.WriteStartElement("Exemplu2"); //<Exemplu>  //Continut  adaugaProdus("1", "Produs 1", "400", writer);  adaugaProdus("2", "Produs 2", "600", writer);  adaugaProdus("3", "Produs 3", "800", writer);  //Sfirsit document  writer.WriteEndDocument();  writer.Close();  Console.Write("Fisier XML creat cu succes");  }  private static void AdaugaProdus(string pID, string pName, string pPrice, XmlTextWriter writer)  {  writer.WriteStartElement("Produs"); //<Produs>  writer.WriteStartElement("Produs\_ID"); //<Produs\_ID>  writer.WriteString(pID);  writer.WriteEndElement(); //</Produs\_ID>  writer.WriteStartElement("Produs\_nume"); //<Produs\_nume>  writer.WriteString(pName);  writer.WriteEndElement(); //</Produs\_nume>  writer.WriteStartElement("Produs\_pret"); //<Produs\_pret>  writer.WriteString(pPrice);  writer.WriteEndElement(); //</Produs\_pret>  writer.WriteEndElement(); //</Produs>  }  }  } |

Precum am menționat, dezavantajul programului precedent era faptul că acesta era ineficient, pentru a rezolva acest neajuns am extras toate metodele responsabile de introducerea unui produs și le-am inclus în metoda AdaugaProdus, acum putem introduce un produs scriind doar un singur rind, sau putem apela metoda dintr-o clasă externă pentru a introduce un produs.

Probleme pentru rezolvare.

### **Program 1:**

1. Îmbunătățiți exemplul de mai sus.
2. Într-o clasă exterioară veți avea mai multe masive care vor conține caracteristicile produsului Produs\_ID[], Produs\_nume[] etc.
3. Crearea fișierului XML fi inclusă într-o metodă care va primi toate masivele precedente ca parametri. (nu în Main ca acuma).
4. Folosiți un ciclu pentru a introduce fiecare caracteristică în locul potrivit cu ajutorul metodei AdaugaProdus.
5. **Deschiderea și citirea documentelor XML**

Din motive evidente platforma .NET ne oferă și oportunitatea de a deschide documentele XML și de a prelucra datele din interior.

Atunci când procesăm un document XML în C#, în esență îl tratăm ca pe o listă de tag-uri. Folosing documentul putem identifica toate tag-urile cu un anumit nume cu ajutorul **XmlNodeList** și a metodei GetElementsByTagName din interiorul acestei clase.

**Exemplu:**

|  |
| --- |
| namespace CitireaXML  {  class CitireXML  {  static void Main()  {  XmlDataDocument documentXML = new XmlDataDocument();  XmlNodeList xmlnode;  FileStream fs = new FileStream("exemplu2.xml", FileMode.Open, FileAccess.Read);  documentXML.Load(fs);  xmlnode = documentXML.GetElementsByTagName("Produs");  string str = null;  for (int i = 0; i < xmlnode.Count; i++)  {  for (int j = 0; j < xmlnode[i].ChildNodes.Count; j++)  Console.Write(xmlnode[i].ChildNodes.Item(j).InnerText+" ");  Console.WriteLine();  }  }  }  } |

În exemplul de mai sus am extras toate elementele cu tag-ul produs din documentul creat anterior și am folosit 2 cicluri, primul pentru a procesa toate produsele și al doilea pentru a trece prin toate elementele componente ale fiecărui produs.

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus ca să afișeze doar numele și prețul produselor.
2. **XML și DataSet**

Una din clasele pe larg utilizate pentru a crea și prezenta documentele XML în C# este clasa DataSet. În esență un DataSet este un set de date în formă tabelară. Aceasta conține un set larg de funcționalitate care poate fi folosită pentru a extrage parametri dintr-un anumit rând sau coloană, pentru a sorta datele după un anumit parametru, de a găsi un element după valoare etc.

În cazul dat vom folosi clasa DataSet pentru a sorta datele din documentul nostru XML care l-am creat anterior.

**Exemplu:**

|  |
| --- |
| namespace SortareXML  {  class SorteazaXML  {  static void Main()  {  XmlReader xmlFile;  xmlFile = XmlReader.Create("exemplu2.xml", new XmlReaderSettings());  DataSet ds = new DataSet();  DataView dv;  ds.ReadXml(xmlFile);  dv = new DataView(ds.Tables[0]);  dv.Sort = "Produs\_pret";  for (int i = 0; i < dv.Count; i++)  {  Console.WriteLine(dv[i][0] + " " + dv[i][1] + " " + dv[i][2]);  }  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați numele produselor din documentul XML și folosiți exemplul de mai sus pentru a le sorta în ordine alfabetică după nume.
2. **Căutarea într-un document XML**

Unul din dezavantajele principale ale XML este verbozitatea acestora, ceea ce semnifică că acestea conțin un volum foarte mare de cuvinte, multe dintre care sunt irelevante pentru unii utilizatori. Denumirile tag-urilor, parametrii care urmează după acestea fac volumul documentelor XML mult mai mare decât dimensiunea reală a datelor care sunt stocate înăuntrul acestora.

Din acest motiv implementarea unui mecanism de căutare este vital pentru oricare programator care are nevoie de a găsi un elemente dintr-un document XML, sau de a crea funcționalitatea pentru utilizatori de a găsi ceva în documentul dat.

**Exemplu:**

|  |
| --- |
| namespace CautareXML  {  class CautareXML  {  static void Main()  {  XmlReader xmlFile;  xmlFile = XmlReader.Create("exemplu2.xml", new XmlReaderSettings());  DataSet ds = new DataSet();  ds.ReadXml(xmlFile);  DataView dv = new DataView();  dv = new DataView(ds.Tables[0]);  dv.Sort = "Produs\_pret";  int index = dv.Find("400");  if (index == -1)  Console.WriteLine("Produsul nu a fost găsit");  else  Console.WriteLine("Produs gasit!\n" + dv[index]["Produs\_nume"] + " are pretul:" + dv[index]["Produs\_pret"]);  }  }  } |

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus pentru a găsi un produs după numele acestuia.
2. **Filtrarea rezultatelor după criterii**

O altă funcționalitate utilă atunci când aveți un volum mare de date este posibilitatea de a afișa doar rezultatele care coincid cu o anumită condiție, ceea ce în esență numim filtrare. Pentru a filtra rezultatele dintr-un document XML vom folosi clasa **DataView** care permite setarea unor anumiți parametri după care se va face filtrarea.

**Exemplu:**

|  |
| --- |
| namespace FiltrareXML  {  class FiltrareaXML  {  static void Main()  {  XmlReader xmlFile;  xmlFile = XmlReader.Create("exemplu2.xml", new XmlReaderSettings());  DataSet ds = new DataSet();  ds.ReadXml(xmlFile);  DataView dv = new DataView();  dv = new DataView(ds.Tables[0], "Produs\_pret > = 500", "Produs\_pret", DataViewRowState.CurrentRows);  dv.ToTable().WriteXml("Rezultat.xml");  Console.WriteLine("XML creat");  }  }  } |

În exemplul de mai sus am folosit filtrarea pentru a extrage toate produsele ce au un preț mai mare de 500, le-am sortat în ordine crescătoare după preț și le-am plasat într-un fișier XML nou nu numele Rezultat.xml.

Probleme pentru rezolvare.

### **Program 1:**

1. Modificați exemplul de mai sus pentru a sorta produsele după numele lor.
2. Afișați toate produsele din fișierul Rezultat.xml la consolă.

Problemă finală.

### **Program:**

1. Creați o clasă care va conține 4 masive:

* string[] IDTranzactie,
* DateTime[] dataTranzactiei,
* decimal[] sumaTranzactiei,
* bool[] statutulTranzactiei.

1. În această clasă adăugați metoda Main, și generați 50 de valori aleatorii pentru fiecare masiv.
2. Creați o altă clasă XMLController.
3. În această clasă creați o metodă care va primi 4 masive din clasa 1 și va stoca într-un fișier XML datele despre aceste 50 de tranzacții.
4. Deschideți fișierul XML și afișați la consolă datele despre tranzacții.
5. Sortați tranzacțiile după ID-ul acestora.
6. Filtrați și afișați tranzacțiile cu o valoare mai mare de 1000 lei.