Platforma .NET – Wykład 9
Obsługa standardu XML

Osoba prowadząca wykład, laboratorium i projekt:
dr hab. inż. Marek Sawerwain, prof. UZ

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

e-mail : M.Sawerwain@issi.uz.zgora.pl
tel. (praca) : 68 328 2321,
pok. 328a A-2,
ul. Prof. Z.Szafrana 2,
65-246 Zielona Góra

Ostatnia kompilacja pliku: Monday 5th June, 2023, t: 23:34

Spis treści

Plan wykładu – spotkania tydzień po tygodniu

(1) Informacje o wykładzie, pojęcie platformy, podstawowe informacje o platformie .NET

(2) Składowe platformy .NET: CLR, CTS, języki programowania, biblioteki klas, pojęcie podzespołu (ang. assembly)

(3) Programowanie w C# – środowisko VS, MonoDevelop, syntaktyka C#, wyjątki, współpraca z DLL

(4) Programowanie w C# – model obiektowy, typy uogólnione, lambda wyrażenia

(5) Programowanie w C# – aplikacje „okienkowe", programowanie wielowątkowe

(6) Programowanie w F# – podstawy, przetwarzanie danych tekstowych,

(*) "Klasówka I", czyli egzamin cześć pierwsza

(7) Dostęp do baz danych

Plan wykładu – tydzień po tygodniu

(8) Język zapytań LINQ, Entity Framework
(9) Obsługa standardu XML
(10) Technologia ASP.NET 1/2
(11) Technologia ASP.NET 2/2
(12) Model widok i kontroler – Model View Controller
(13) Tworzenie usług sieciowych SOAP i WCF (komunikacja sieciowa)
(14) Wykład monograficzny .NET 1
(15) Wykład monograficzny .NET 2
(*) "Klasówka II", czyli egzamin cześć druga

Plan wykładu

1. czym jest format XML
   1. nieco historii i SGML
   2. format XML
   3. zastosowania standardu XML
2. model DOM oraz uproszczone modele dostępu do danych
   1. czym jest model DOM
   2. odczyta dokumentu,
   3. uproszczone modele dostępu do danych XML,
   4. transformacja XSLT.

Znacznie i źródła standardu XML

XML to odmiana standardu SGML (ISO 8879). Elastyczność formatu SGML do opisu cyfrowych dokumentów zostało zauważona już w latach 80-tych. Jeden z pierwszych edytorów przeznaczonych do SGML, to LEXX opracowany przez firmę IBM około roku 1985, autorem był Mike Cowlishaw.

XML

XML (ang. Extensible Markup Language – rozszerzalny język znaczników) to niezależny od platformy sprzętowo-programowej standard przeznaczony do opisu zawartości dokumentów wymienianych pomiędzy różnymi platformami. Obsługą standardu XML zajmuje się organizacja W3C (`www.w3.org`). Obecnie dostępne są dwie wersje podstawowa 1.0 oraz 1.1 która odzwierciedla tylko zmiany w standardzie Unicode. Polecana i obowiązująca wersja to 1.0.

Format XML

Dokument XML może rozpoczynać się od informacji o wersji standardu XML oraz o sposobie kodowania znaków. W razie braku znacznika **?xml** wartości domyślne to wersja 1.0 oraz standard kodowania UTF-8.

```
<?xml version="1.0" encoding="UTF-8"?>
<książka-telefoniczna kategoria="bohaterowie książek">
 <!-- to jest komentarz -->
  <osoba charakter="dobry">
    <imię>Papa</imię>
    <nazwisko>Smerf</nazwisko>
    <telefon/>
  </osoba>
  <osoba charakter="dobry">
    <imię>Ambroży</imię>
    <nazwisko>Kleks</nazwisko>
    <telefon>123-456-789</telefon>
  </osoba>
  <osoba charakter="zły">
    <imię>Alojzy</imię>
    <nazwisko>Bąbel</nazwisko>
    <telefon/>
  </osoba>
</książka-telefoniczna>
```

Kluczowe pojęcia związane z plikami XML

Kluczowe pojęcia plików XML są następujące:

- wszystkie elementy pliku XML mogą być kodowanie za pomocą znaków ze standardu Unicode,
- postać znaczników np.: `<abc> abc </abc>` i zawartość pomiędzy znacznikami,
- elementy logiczne np. `<book> <title> <title/> </book>`,
- atrybuty np.:
  `<cstep number="three">Connect A to B.</cstep>`,
- nagłówek plików XML:
  `<?xml version="1.0" encoding="UTF-8" ?>`,
- parser typu DOM albo SAX.

Istotne elementy związane z poprawnością plików to opis zawartości pliku XML, tzw. DTD (ang. Document Type Definition).

Poprawność plików XML – „uformowanie”

Specyfikacja dokumentu XML określa dokumenty XML jako dokument poprawnie „uformowany” jeśli spełnione są odpowiednie reguły, kilka kluczowych reguł podano poniżej:

- plik XML zawiera wyłącznie znaki kodowane w standardzie Unicode,
- znaki "<" oraz "&" występuje tylko w kontekście znaczników,
- początek znacznika oraz jego koniec są poprawnie zagnieżdżone, nie brakuje znaczników oraz znaczniki się nie nakładają,
- elementy znaczników są identyczne łącznie z wielkością liter, bowiem wielkość liter jest brana pod uwagę,
- istnieje tylko jeden element główny „root” który zawiera pozostałe znaczniki.

Definicja zawartości pliku XML

Najstarszym opisem zawartości pliku XML jest Document Type Definition (DTD), pochodzący z SGML. Opis DTD posiada następujące zalety:

- wsparcie DTD jest dostępne we większości rozwiązaniach dla technologi XML,
- opis DTD jest bardzo zwięzły w przeciwieństwie do innych rozwiązań stosowanych w XML,
- DTD może wykorzystywać standardowe publiczne zbiory encji,
- DTD określa także typ dokumentu, pozwala na grupowanie wszystkich elementów w pojedynczym dokumencie.

Definicja zawartości pliku XML

Do głównych wad DTD należą min.:

- brak wsparcia dla nowszych rozwiązań XML, przede wszystkim do przestrzeni nazw,
- brakuje także lepszej ekspresywności, opis DTD dla XML to uproszczona wersja DTD dla SGML i pewne struktury nie mogą być wyrażona za pomocą gramatyki regularnej,
- dość często opisu DTD szybko stają się nieczytelne ze względu na parametryzowanie encji,
- opis DTD operacja się na syntaktyce wyrażeń regularnych odziedziczonej po SGML, większość tzw. szybkich parserów XML, nie oferuje wsparcia dla wyrażenia struktury dokumentu zapisanego w takiej semantyce, co utrudnia dostęp do dokumentu, gdyż zazwyczaj dostęp jest wyłącznie jednokierunkowy.

Ważnym elementem DTD jest możliwość jego włączenia do opisywanego pliki XML oraz możliwość definiowania encji. Ze względu na powszechność DTD, nadal ten typ opisu stanowi istotny element w procesie weryfikowania poprawności dokumentu.

Przykłady DTD – program telewizyjny

Opis DTD programu telewizyjnego:

```
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
```

Przykłady DTD – program telewizyjny plik XML

Plik XML zgodny z przykładem DTD opisu programu telewizyjnego:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE TVSCHEDULE SYSTEM "example1.dtd">
<TVSCHEDULE NAME="Tydzień 1">
   <CHANNEL CHAN="Ch.1">
      <BANNER>text</BANNER>
      <DAY>
         <DATE>01.01.2011</DATE>
         <HOLIDAY>YES</HOLIDAY>
      </DAY>
   </CHANNEL>
</TVSCHEDULE>
```

Przykłady DTD – katalog z częściami

Opis zawartości XML zawierający katalog części:

```
<!DOCTYPE CATALOG [

<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">

<!ELEMENT CATALOG (PRODUCT+)>

<!ELEMENT PRODUCT (SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>
<!ATTLIST PRODUCT
NAME CDATA #IMPLIED
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"
PARTNUM CDATA #IMPLIED
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"
INVENTORY (InStock|Backordered|Discontinued) "InStock">

<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST SPECIFICATIONS
WEIGHT CDATA #IMPLIED
POWER CDATA #IMPLIED>
```

Przykłady DTD

dokończenie pliku DTD:

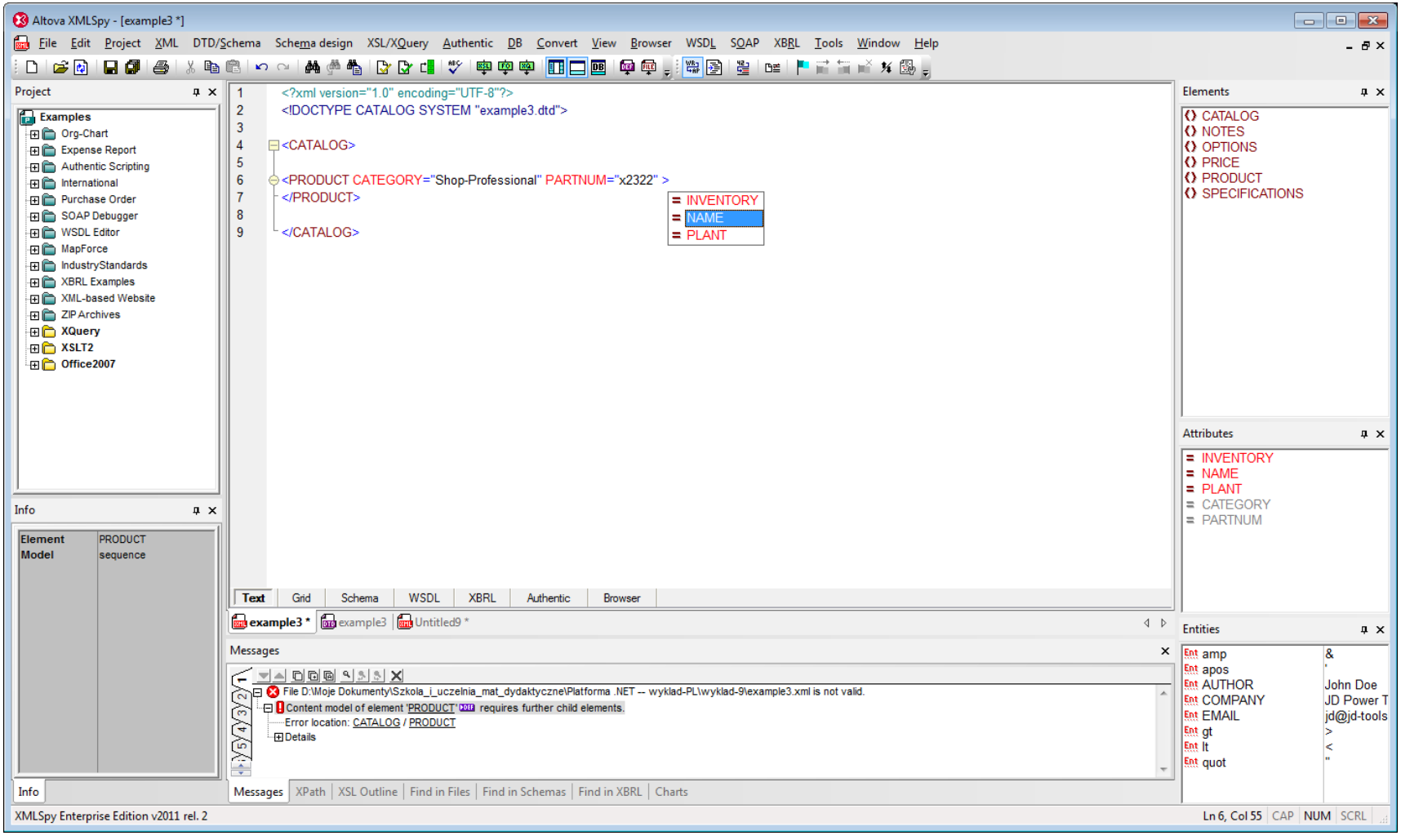
```
<!ELEMENT OPTIONS (#PCDATA)>
<!ATTLIST OPTIONS
FINISH (Metal|Polished|Matte) "Matte"
ADAPTER (Included|Optional|NotApplicable) "Included"
CASE (HardShell|Soft|NotApplicable) "HardShell">

<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST PRICE
MSRP CDATA #IMPLIED
WHOLESALE CDATA #IMPLIED
STREET CDATA #IMPLIED
SHIPPING CDATA #IMPLIED>

<!ELEMENT NOTES (#PCDATA)>

]>
```

Edycja z pliku XML w edytorze XMLSpy



XML Schema Definition Language – XSD

XSD to obecnie najbardziej rozposzechniona metoda opisu zawartości i struktury dokumentów XML. Specyfikacja XSD jest rekomendowana przez konsorcjum W3C. Kluczową zaletą XSD jest fakt iż możliwie jest opisywania typów poszczególnych atrybutów oraz znaczników. Postać opisu XSD to zestaw znaczników, więc zawartość dokumentu XML jest opisywana za pomocą innego języka znaczników. Zaleca się, aby nowotworzone dokumenty XML posiadały opis XSD zamiast opisu DTD, czy innych standardów opisu zawartości dokumentów XML.

Sposoby tworzenia XSD

- samodzielne, przy użyciu odpowiedniego edytora np.: XMLSpy, Visual Studio XML Designer,
- tworzenie XSD na podstawie istniejącej klasy, tabeli w bazie danych, na podstawie istniejącego dokumentu XML,
- wykorzystanie narzędzia xsd.exe do operacji związanych z XML Schema min. do tworzenia XML Schema z istniejących klas zapisanych w podzespołach.

Elementy XSD

Główne elementy plików opisu XSD:

Nazwa	Opis
Element	reprezentacja pojedynczego elementy
Attribute	reprezentacja atrybutu pojedynczego elementu
Attribute group	grupa atrybutów która może być użyta w typie złożonym
Simple type	typ prosty zawiera tylko dane tekstowe nie posiada podelementów
Complex type	typ złożony może zawiera typy proste

Główne typy danych dostępne w XSD:

XSD Type	.NET Type	XSD Type	.NET Type	XSD Type	.NET Type
Boolean	System.Boolean	decimal	System.Decimal	Int	System.Int32
Byte	System.SByte	Double	System.Double	Long	System.Int64
dateTime	System.DateTime	Float	System.Single	String	System.String

XML Schema – 1/3

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
   <xs:element name="TVSCHEDULE">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="CHANNEL" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="NAME" type="xs:anySimpleType" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="CHANNEL">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="BANNER"/>
            <xs:element ref="DAY" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="CHAN" type="xs:anySimpleType" use="required"/>
      </xs:complexType>
   </xs:element>
```

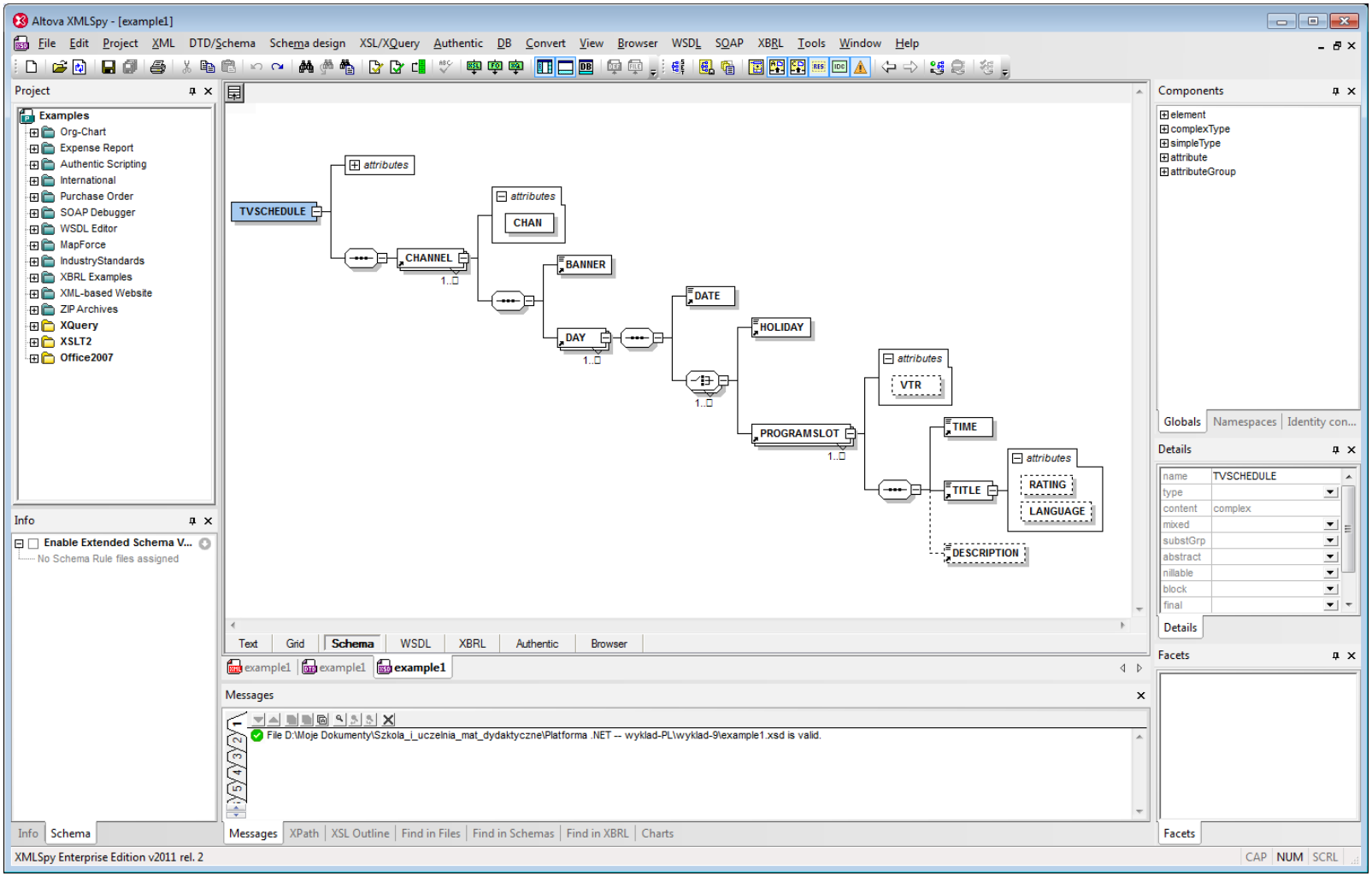
XML Schema – 2/3

```
<xs:element name="BANNER">
   <xs:complexType mixed="true"/>
</xs:element>
<xs:element name="DAY">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="DATE"/>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="HOLIDAY"/>
            <xs:element ref="PROGRAMSLOT" maxOccurs="unbounded"/>
         </xs:choice>
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="HOLIDAY">
   <xs:complexType mixed="true"/>
</xs:element>
<xs:element name="DATE">
   <xs:complexType mixed="true"/>
</xs:element>
```

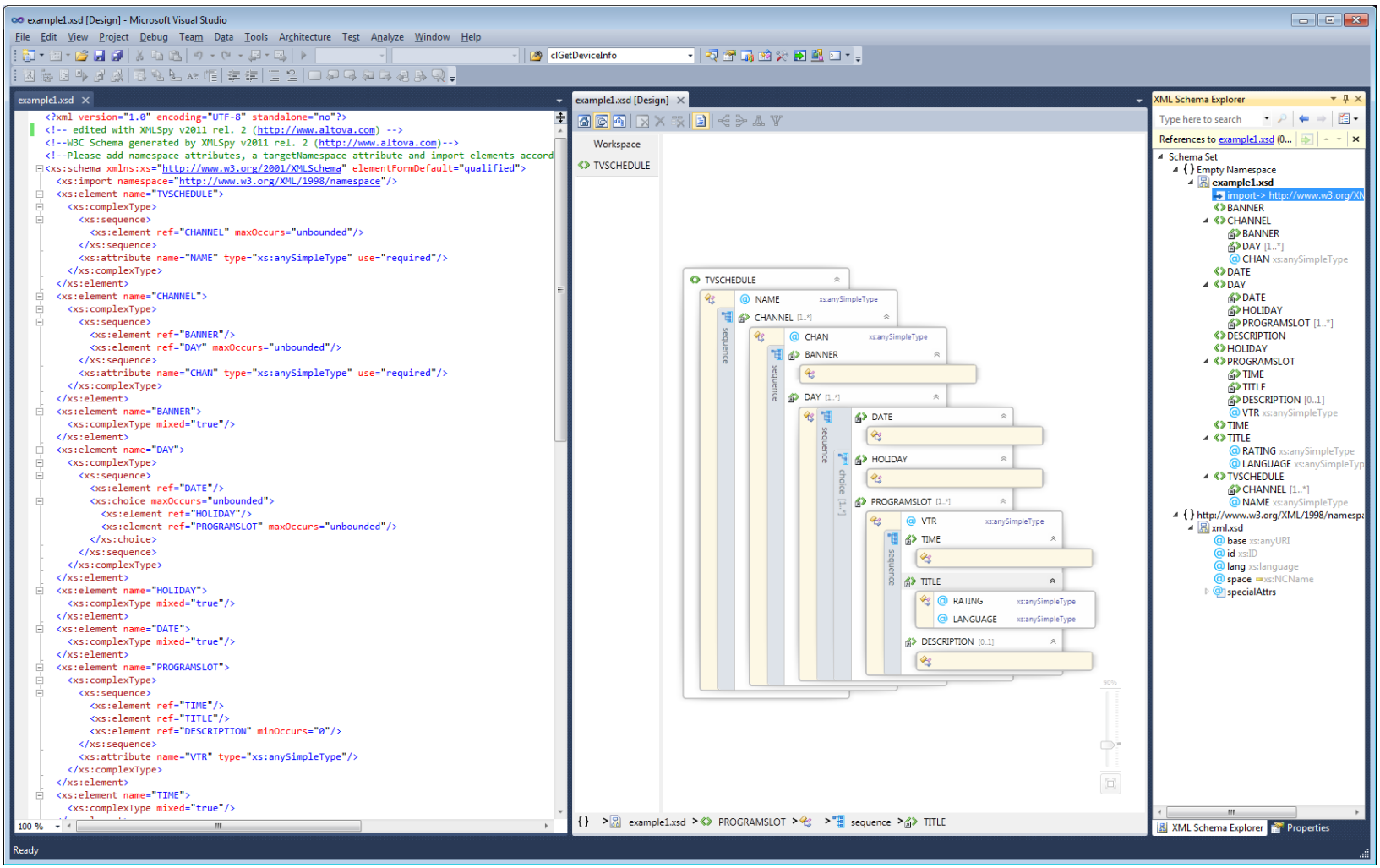
XML Schema – 3/3

```
   <xs:element name="PROGRAMSLOT">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="TIME"/>
            <xs:element ref="TITLE"/>
            <xs:element ref="DESCRIPTION" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="VTR" type="xs:anySimpleType"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="TIME">
      <xs:complexType mixed="true"/>
   </xs:element>
   <xs:element name="TITLE">
      <xs:complexType mixed="true">
         <xs:attribute name="RATING" type="xs:anySimpleType"/>
         <xs:attribute name="LANGUAGE" type="xs:anySimpleType"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="DESCRIPTION">
      <xs:complexType mixed="true"/>
   </xs:element>
</xs:schema>
```

Reprezentacja graficzna – XMLEdit



Edycja XSD w Visual Studio



Standard RELAX NG

Mniej popularny format, ale o mniej skomplikowanych regułach. Dla następującego DTD:

```
<!DOCTYPE addressBook [
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
]>
```

Opis w standardzie RELAX NG:

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

Formaty i języki oparte o XML

Przykład formatów i języków opartych o standard XML:

- RDF – Resource Description Framework, opis zawartości stron www,
- OpenDocument – OASIS Open Document Format for Office Applications, dokumenty biurowe,
- SMIL – Synchronized Multimedia Integration Language, opis prezentacji multimedialnych,
- SOAP – Simple Object Access Protocol, protokół dla usług sieciowych,
- SVG – Scalable Vector Graphics, grafika wektorowa,
- DAE - Collada format for 3D graphics, grafika 3D,
- MathML – Mathematical Markup Language, opis formuł matematycznych,
- XAML – Extensible Application Markup Language, GUI w .NET począwszy od wersji 3.0,
- XHTML – Extensible HyperText Markup Language, strony WWW,
- XSL – Extensible Stylesheet Language, przekształcanie XML-a,
- XUL – XML-based User-interface Language, GUI,
- WML – Wireless Markup Language, strony WAP.

Obecność XML w platformie .NET

Wsparcie dla technologi XML w BCL obejmuje następujące obszary platformy .NET:

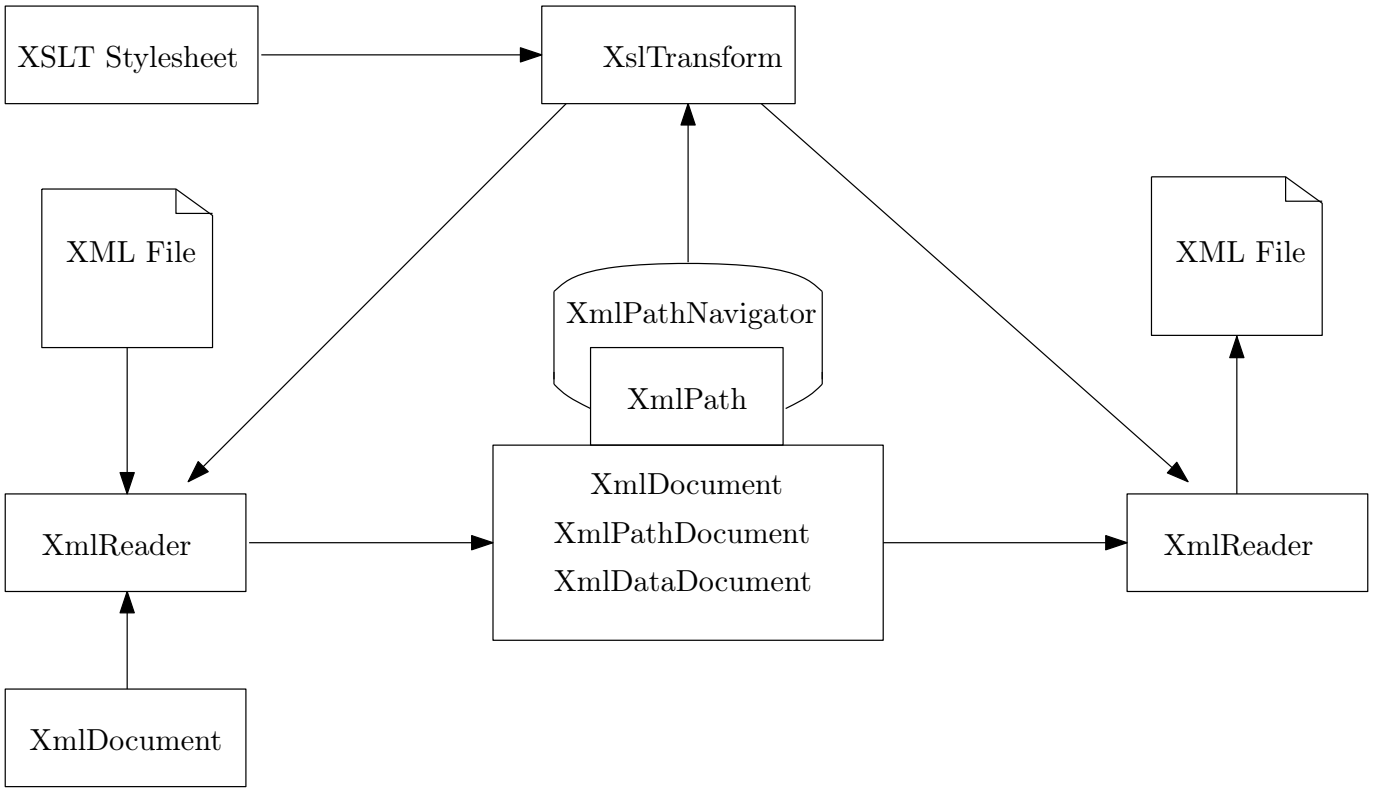
- .NET configuration files
- ADO.NET
- ASP.NET server controls
- XML serialization
- Remoting
- Web services
- XML documentation
- SQL Server XML features
- XML parsing
- XML transformation

Przestrzeń nazw System.Xml

Główna przestrzeń dot. standardu XML to System.Xml, zawiera ona następujące kolejne przestrzenie zajmujące się wybranymi obszarami problemów dotyczących przetwarzania danych opisanych przez języków znaczników XML:

- System.Xml.Schema
- System.Xml.XPath
- System.Xml.Xsl
- System.Xml.Serialization
- System.Xml.Linq

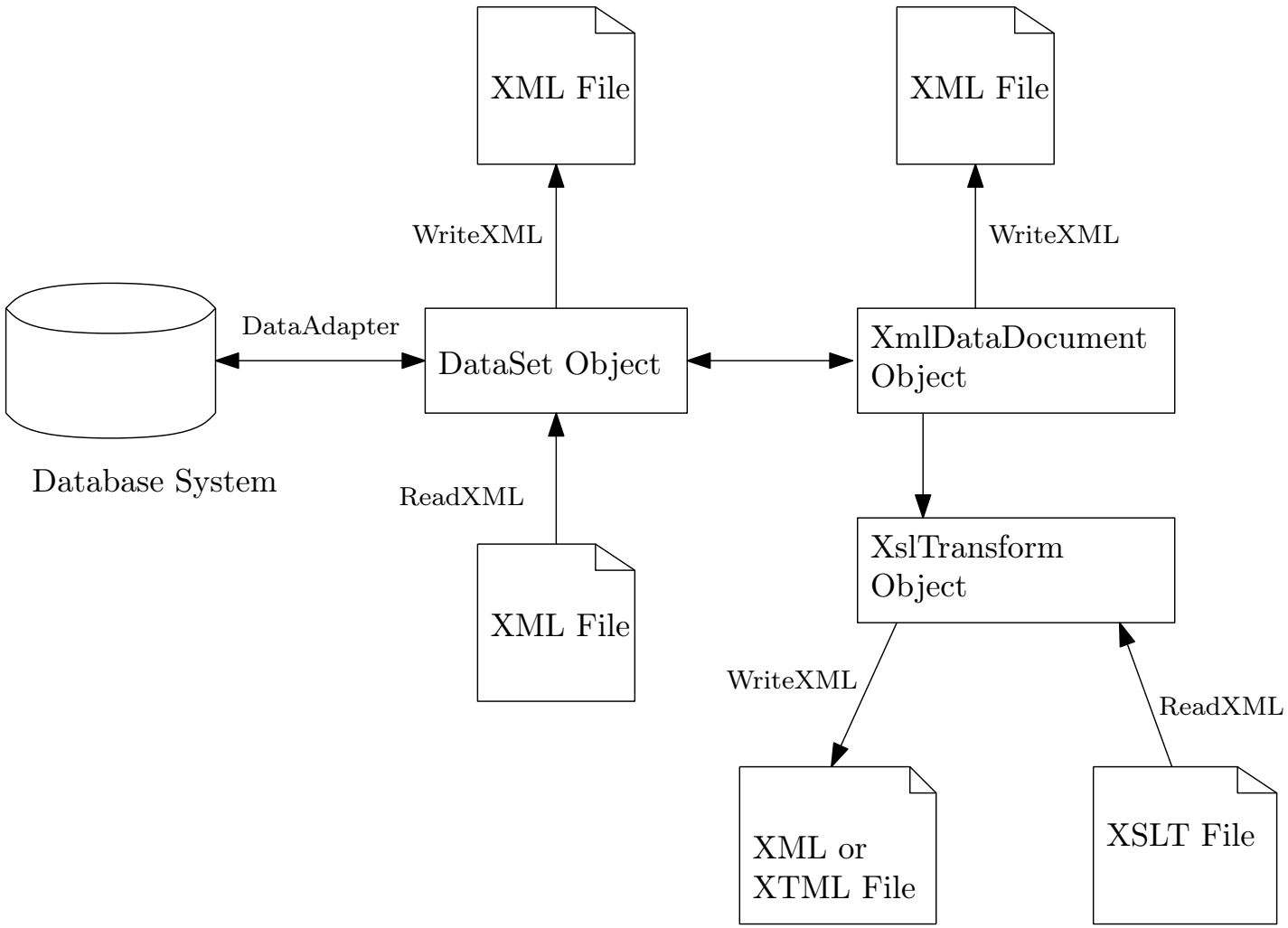
Przetwarzanie dokumentów XML



Opis poszczególnych obiektów:

- XmlReader: odczyt danych XML,
- XmlDocument, XmlNode: obiekty dostępu do danych w modelu XML DOM,
- XmlWriter: obiekt odpowiedzialny za zapis danych XML,
- XPathNavigator: ścieżki dostępu w standardzie XPath,
- XslTransform: transformacja dokumentów XML.

XML i bazy danych



Przykład pliku XML

Plik XML z tagami opisujących klientów:

```
<?xml version="1.0"?>
<customers>
    <customer CustomerID="C001">
        <name>ACME Inc.</name>
        <phone>12345</phone>
        <comments>Regular customer since 1895</comments>
    </customer>
    <customer CustomerID="C002">
        <name>Star Wars Corp.</name>
        <phone>23456</phone>
        <comments>A small but healthy company.</comments>
    </customer>
</customers>
```

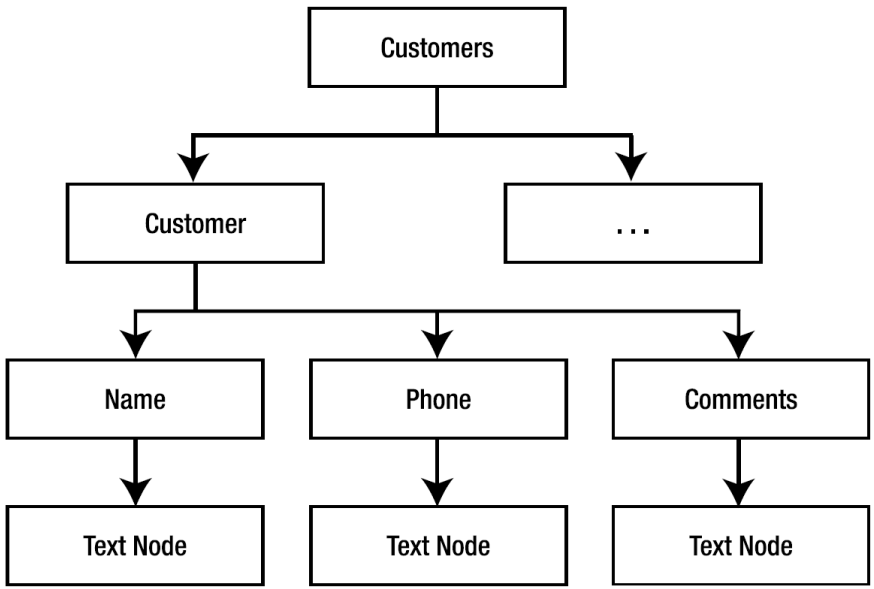
Przykład pliku XML

Plik XML opisujących klientów:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- This is list of employees -->
<employees>
  <employee employeeid="1">
    <firstname>Nancy</firstname>
    <lastname>Davolio</lastname>
    <homephone>(206) 555-9857</homephone>
    <notes>
      <![CDATA[includes a BA in psycho...onal.]]>
  </notes>
  </employee>
  <employee employeeid="2">
   ...
  </employee>
  <employee employeeid="3">
   ...
  </employee>
</employees>
```

Stosowanie Document Object Model – DOM

Zależności pomiędzy znacznikami przedstawione w postaci drzewa:



Główne elementy i obiekty w modelu DOM

- DocumentElement: reprezentuje wierzchołek drzewa dokumentu,
- Node: każdy z węzłów (w tym elementy),
- Element a węzeł: element to jeden z występujących w dokumencie DOM, rodzajów węzłów (m.in. węzły tekstowe, atrybutowe),
  - Węzły elementowe: zwykle zawierają węzły potomne: elementowe, tekstowe lub obu rodzajów.
  - Węzły atrybutowe: informacja podległa konkretnemu węzłowi elementowemu. Nie są określane jako potomkowie.
  - Węzeł dokumentu: rodzic dla wszystkich pozostałych węzłów.
  - CData: odpowiada sekcji znakowej, zawierającej zawartość nie przeznaczoną do przetwarzania;
  - Comment: (komentarz); ProcessingInstruction (instr. przetwarzania);
  - DocumentFragment: stosowany jako węzeł roboczy przy budowaniu lub rekonstruowaniu dokumentu XML.
  - Ponadto: Entity, EntityReference, Notation.

Document element – XmlElement, Processing instructions – XmlProcessingInstruction, Element – XmlElement, Attribute – XmlAttribute, Text values – XmlText, Nodes – XmlNode.

Kiedy warto stosować podejście DOM?

Odpowiadając na pytanie kiedy warto stosować model DOM, warto zadać następujące pytania:

- *zapis oraz odczyt:* model DOM oferuje zapis i odczyt dokumentu XML, jednak trzeba się zapytać, czy dana aplikacja istotnie potrzebuje obu operacji w każdym momencie edycji danych dostępu do danych XML,
- *pamięć:* dokumenty w modelu XML, w całości rezydują w pamięci operacyjnej, w przypadku dużych dokumentów, wymagania na pełną reprezentację mogą być bardzo wysokie.
- *typ dostępu:* ze względu na dostęp do całości dokumentu, możliwy jest losowy dostęp do każdego węzła, należy zadać pytanie, czy w danej aplikacji wystarczy dostęp sekwencyjny.

Wobec tych uwag najlepiej stosować podejście DOM w następujących przypadkach:

- wymagana jest pełen dostęp do pliku XML, dostęp tylko do odczytu bądź zapisu nie jest wystarczający,
- istotny jest dostęp do każdego węzła, a odczyt sekwencyjny bez możliwości powrotu jest nie wystarczający,
- przetwarzane dokumentu nie są duże,
- ograniczenia pamięciowe nie są istotne.

Wczytanie dokumentu – model DOM

Trzy sposoby odczytanie pliku XML z adresu URL, z pliku reprezentowanego jako strumień oraz z ciągu znaków reprezentującego ciągi XML:

```
try {
    XmlDocument d1 = new XmlDocument();
    XmlDocument d2 = new XmlDocument();
    XmlDocument d3 = new XmlDocument();

    d1.Load(textBox1.Text); // adres URL

    FileStream stream = new FileStream(PathToFile, FileMode.Open);
    d2.Load(stream);
    stream.Close();

    d3.LoadXml( stringXML ); // XML jako ciąg znaków w zmiennej
                             // typu string
}
catch(Exception ex) {
    ...
}
```

Odczytanie danych

Odczytanie tytułu piosenki z bazy płyt CD:

```
XmlDocument doc = new XmlDocument();
doc.Load("test.xml");
XmlElement firstCD = (XmlElement) doc.DocumentElement.FirstChild;
XmlElement artist  = (XmlElement) firstCD.GetElementsByTagName("artist")[0];
XmlElement title   = (XmlElement) firstCD.GetElementsByTagName("title")[0];
Console.WriteLine("Artysta={0}, Tytuł={1}", artist.InnerText, title.InnerText);
```

Odczytanie imion pracowników:

```
XmlDocument doc = new XmlDocument();
doc.Load("employees.xml");
list = doc.GetElementsByTagName( "firstname" );
listBox1.Items.Clear();
foreach (XmlNode node in list) {
    listBox.Items.Add(node.Name);
}
```

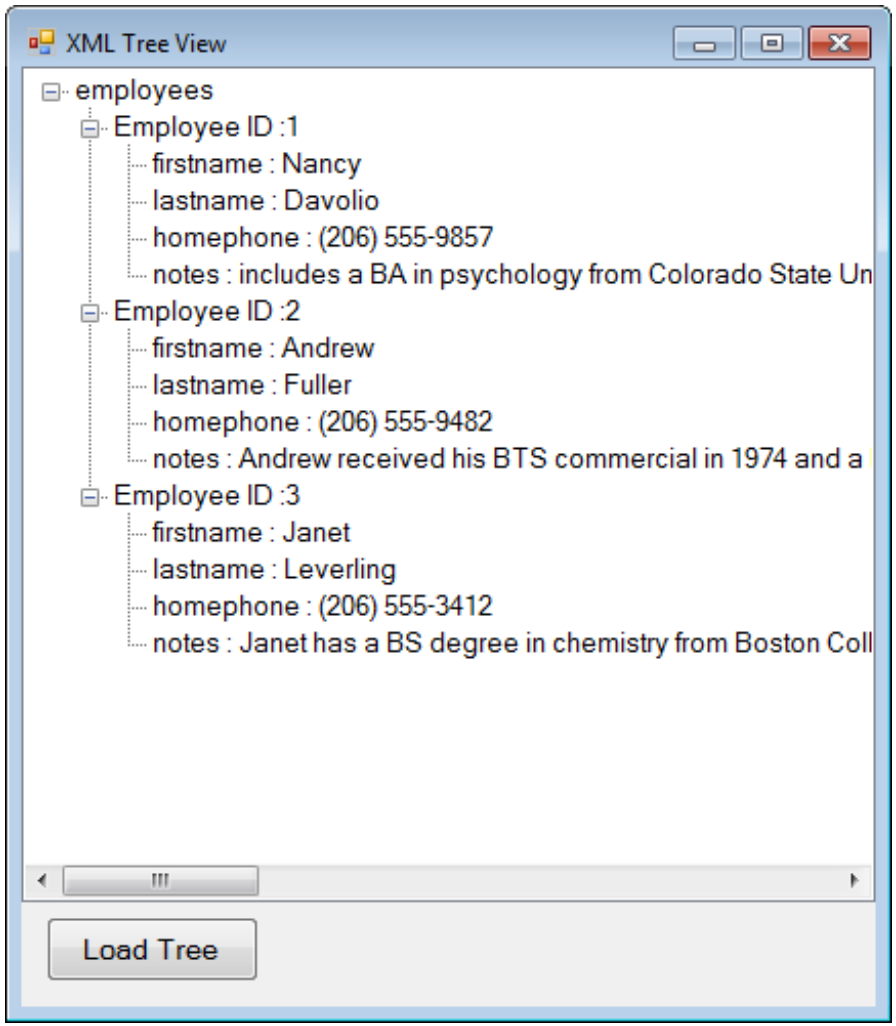
Odczyt danych XML

Odczytanie danych z pliku XML i przepisanie danych do kontrolki TreeView:

```
XmlDocument doc = new XmlDocument();
doc.Load(Application.StartupPath + "/datacust.xml");
TreeNode root = new TreeNode(doc.DocumentElement.Name);
treeView1.Nodes.Add(root);
foreach (XmlNode node in doc.DocumentElement.ChildNodes) {
    TreeNode employee = new TreeNode("Employee ID :" +
    node.Attributes["employeeid"].Value);
    root.Nodes.Add(employee);
    if (node.HasChildNodes) {
        foreach (XmlNode childnode in node.ChildNodes) {
            TreeNode n2 = new TreeNode(childnode.Name + " : "+ childnode.InnerText);
            employee.Nodes.Add(n2);
        }
    }
}
```

Aplikacja XML i TreeView

Aplikacja do wczytywania danych XML:



Modyfikacja danych w modelu DOM

Modyfikacja danych we wskazanym węźle:

```
XmlNode node=doc.SelectSingleNode(
     "//employee[@employeeid='" + +comboBoxIDS.SelectedItem + "']");
if (node != null) {
    node.ChildNodes[0].InnerText = textBox1.Text;
    node.ChildNodes[1].InnerText = textBox2.Text;
    node.ChildNodes[2].InnerText = textBox3.Text;
    XmlCDataSection notes = doc.CreateCDataSection(textBox4.Text);
    node.ChildNodes[3].ReplaceChild(notes, node.ChildNodes[3].ChildNodes[0]);
}
doc.Save(Application.StartupPath + "/employees.xml");
```

Usunięcie wskazanego węzła z dokumentu XML w modelu DOM:

```
XmlNode node = doc.SelectSingleNode(
     "//employee[@employeeid='" +comboBoxIDS.SelectedItem + "']");
if (node != null) {
    doc.DocumentElement.RemoveChild(node);
}
doc.Save("employees.xml");
```

Dodanie nowych danych

Dodanie nowego pracownika wymaga utworzenia nowych węzłów, elementów oraz nadanie wartości atrybutom:

- elementy: `<employee>`, `<firstname>`, `<lastname>`, `<homephone>`, `<notes>`,
- wartość dla atrybutu employeeid w elemencie `<employee>`,
- wartości w węzłach tekstowych `<firstname>`, `<lastname>`, `<homephone>`,
- wartość CDATA dla `<notes>`

Pierwszy krok to utworzenie węzłów:

```
XmlElement employee  = doc.CreateElement("employee");
XmlElement firstname = doc.CreateElement("firstname");
XmlElement lastname  = doc.CreateElement("lastname");
XmlElement homephone = doc.CreateElement("homephone");
XmlElement notes     = doc.CreateElement("notes");
```

Dodanie nowych danych

Reszta kodu dodająca nowy węzeł do listy pracowników w pliku XML:

```
XmlAttribute employeeid = doc.CreateAttribute("employeeid");
employeeid.Value = comboBox1.Text;

XmlText firstnametext = doc.CreateTextNode(textBox1.Text);
XmlText lastnametext = doc.CreateTextNode(textBox2.Text);
XmlText homephonetext = doc.CreateTextNode(textBox3.Text);
XmlCDataSection notestext = doc.CreateCDataSection(textBox4.Text);

employee.Attributes.Append(employeeid);
employee.AppendChild(firstname);
employee.AppendChild(lastname);
employee.AppendChild(homephone);
employee.AppendChild(notes);

firstname.AppendChild(firstnametext);
lastname.AppendChild(lastnametext);
homephone.AppendChild(homephonetext);
notes.AppendChild(notestext);

doc.DocumentElement.AppendChild(employee);
doc.Save(Application.StartupPath + "/employees.xml");
```

Klasy odczytu i zapisu – podejście typ SAX

Dostęp do XML w uproszczonym trybie jest realizowany przez następujące klasy (podstawowe stanowią abstrakcyjne klasy XmlReader i XmlWriter):

- XmlTextReader — odczyt danych XML z pliku, sprawdzając czy dokument jest dobrze uformowany, bez wykorzystania W3C Schema, możliwy jest tylko odczyt, nie ma możliwości cofnięcia się,
- XmlNodeReader — zapewnia dokładnie tą samą funkcjonalność co XmlTextReader, lecz odczytuje dane tylko z podanego węzła,
- XmlValidationReader -– podobnie jak XmlTextReader lecz z możliwością sprawdzenia poprawności z podanym plikiem opisu w standardzie W3C Schema,
- XmlTextWriter — zapewnia zapisywanie danych do pliku XML, także bez możliwości cofnięcia się do wcześniej zapisanych danych.

Kiedy warto stosować uproszczone podejście

Model DOM, niestety nie sprawdza się w przypadku bardzo obszernych dokumentów XML, dlatego jeśli:

- potrzebny jest tylko odczyt dokumentu,
- dokument jest obszerny,
- ilość dostępnej pamięci jest niewielka,
- istnieje konieczność współpracy z wieloma dokumentami,
- nie ma potrzeby losowego dostępy,

lepiej stosować podejście oparte o klasę XmlReader. Analogicznie w przypadku zapisu, podejście oparte o klasę XmlWriter jest lepsze, jeśli:

- dane są tylko zapisywane,
- ilość dostępnej pamięci jest niewielka,
- zapisywane są bardzo duże ilości danych tworzące znaczącej wielkości pliki XML.

Odczyt dokumentu

Odczyt danych w modelu uproszczonym wymaga sprawdzania jaki aktualnie znacznik jest wczytywany, wymaga to wcześniejszej znajomości struktury plików XML:

```
XmlTextReader reader = new XmlTextReader( "employees.xml");
reader.WhitespaceHandling = WhitespaceHandling.None;
TreeNode employeenode=null; TreeNode rootnode = null;
while (reader.Read()) {
  if (reader.NodeType == XmlNodeType.Element) {
      if (reader.Name == "employees") {
        rootnode = treeView1.Nodes.Add("Employees");
      }
      if (reader.Name == "employee") {
        string employeeid = reader.GetAttribute("employeeid");
        employeenode = new TreeNode("Employee ID :" + employeeid);
        rootnode.Nodes.Add(employeenode);
      }
      if (reader.Name == "firstname") {
        string firstname = reader.ReadElementString();
        TreeNode node = new TreeNode(firstname);
        employeenode.Nodes.Add(node);
      }
      if (reader.Name == "lastname") { ... }
      if (reader.Name == "homephone") { ... }
    if (reader.Name == "notes") { ... }
  }
}
reader.Close();
```

Zapis dokumentu

Zapis dokumentu XML można zrealizować bezpośrednio bez stosowania dodatkowych klas. Stosowanie klas opartych o XmlWriter jest podobne do klas odczytujących dane:

```
XmlTextWriter writer = new XmlTextWriter(Console.Out);
writer.Formatting = Formatting.Indented;
WriteQuote(writer, "MSFT", 74.125, 3.89, 69020000);
WriteQuote(writer, "MSFT", 24.422, 5.23, 69020001);
WriteQuote(writer, "MSFT", 34.257, 1.26, 69020002);
writer.Close();
```

Pomocnicza funkcja przedstawia się następująco:

```
static void WriteQuote(XmlWriter writer, string symbol,
                double price, double change, long volume)
{
   writer.WriteStartElement("Stock");
   writer.WriteAttributeString("Symbol", symbol);
   writer.WriteElementString("Price", XmlConvert.ToString(price));
   writer.WriteElementString("Change", XmlConvert.ToString(change));
   writer.WriteElementString("Volume", XmlConvert.ToString(volume));
   writer.WriteEndElement();
}
```

Podstawy XPath

XPath to oparty o XML język zapytań do wybierania danych (węzłów) z dokumentów XML (adresowania danych w plikach XML). Możliwości XPath obejmują także możliwość obliczenia wartości dla następujących typów: ciąg znaków, liczby, wartości logiczne. Standardem XPath opiekuje się World Wide Web Consortium (W3C). Ogólnie ścieżka ma następującą postać:

```
Axis::node-test[predicate]
```

Węzeł pracownik o identyfikatorze jeden:

```
//employee[@employeeid='1']
```

Główne pojęcia XPath

Location Path, Axis, Node tests, Predicates.

Osie XPath

Oś	Opis
ancestor	przodkowie węzła kontekstowego
ancestor-or-self	węzeł kontekstowy i jego przodkowie
attribute	atrybut węzła kontekstowego
child	dzieci węzła kontekstowego
descendant	potomkowie węzła kontekstowego
descendant-or-self	węzeł kontekstowy i jego potomkowie
following	wszystkie węzły znajdujące się po węźle kontekstowym
following-sibling	wszystkie węzły znajdujące się na tym samym poziomie co węzeł kontekstu
parent	rodzica węzła kontekstowego
preceding	wszystkie węzły z dokumentu, do którego należy węzeł kontekstu oraz węzły znajdujące się przed nim
preceding-sibling	wszystkie węzły znajdujące się na tym samym poziomie co węzeł kontekstu oraz przed tym węzłem
self	aktualny węzeł kontekstowy

Dostęp do danych XPath

Odczytanie ciągu znaków reprezentujących tytuł bądź reprezentujących artystę:

```
XPathDocument doc = new XPathDocument( "exdata.xml" );
XPathNavigator nav = doc.CreateNavigator();
XPathNodeIterator iterator = nav.Select("/items/compact-disc[1]/artist
                                        | /items/compact-disc[1]/title");

iterator.MoveNext();
Console.WriteLine("A={0}", iterator.Current);

iterator.MoveNext();
Console.WriteLine("T={0}", iterator.Current);
```

Odczyt danych z określonego węzła

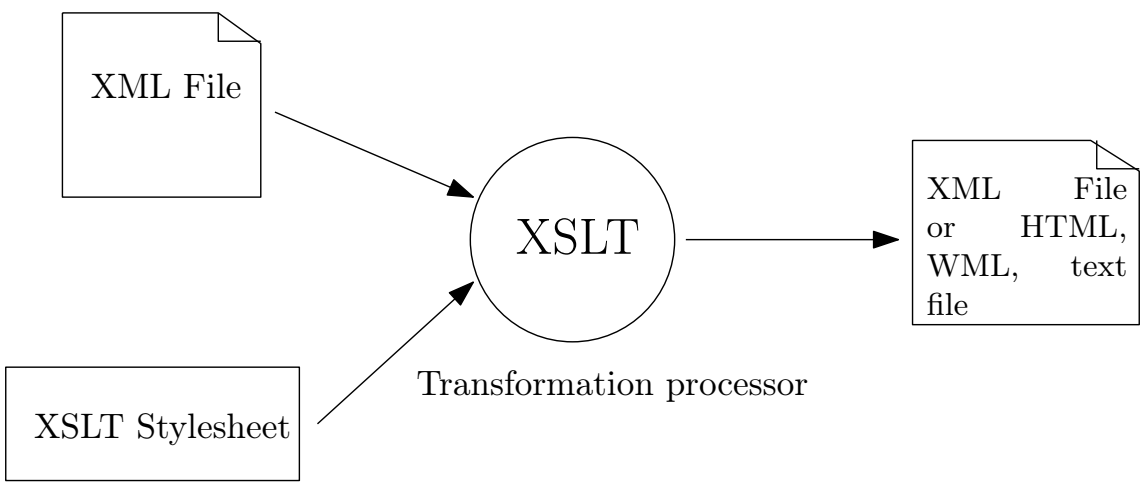
Wybieranie danych dot. pracowników po wskazanym identyfikatorze ID zapisanym w kontrolce **textEmpID**:

```
XPathDocument doc = new XPathDocument(
                Application.StartupPath + @"\employees.xml");
XPathNavigator navigator = doc.CreateNavigator();
XPathNavigator result = navigator.SelectSingleNode(
        @"employees/employee[@employeeid=" + textEmpID.Text + "]");
result.MoveToFirstChild();
do {
  switch (result.Name) {
    case "firstname":
      FNLBL.Text=result.Value;
    break;
    case "lastname":
      LNLBL.Text=result.Value;
    break;
    case "homephone":
      HPLBL.Text=result.Value;
    break;
    case "notes":
      NLBL.Text=result.Value;
    break;
  }
} while (result.MoveToNext());
```

Język XSL – eXtensible Stylesheet Language

Ponieważ XML opisuje tylko strukturę i semantykę, toteż nie opisuje sposobów prezentacji informacji. Dlatego, wprowadzona została dodatkowa warstwa która oddziela informację od prezentacji. A uzyskuje się to poprzez wykorzystanie arkuszy stylów.

- CSS – Cascading Style Sheets (Level 1 oraz Level 2),
- XSL – eXtensible Stylesheet Language - opracowany dla dokumentów XML, obejmuje:
  - XSLT (XSL Transformations) - język przekształceń o charakterze deklaratywnym, oparty na regułach przekształceń (rule-based) struktury XML,
  - XSL FO (XSL Formating Objects) – język opisu formatu.



Transformacja XML za pomocą XSLT

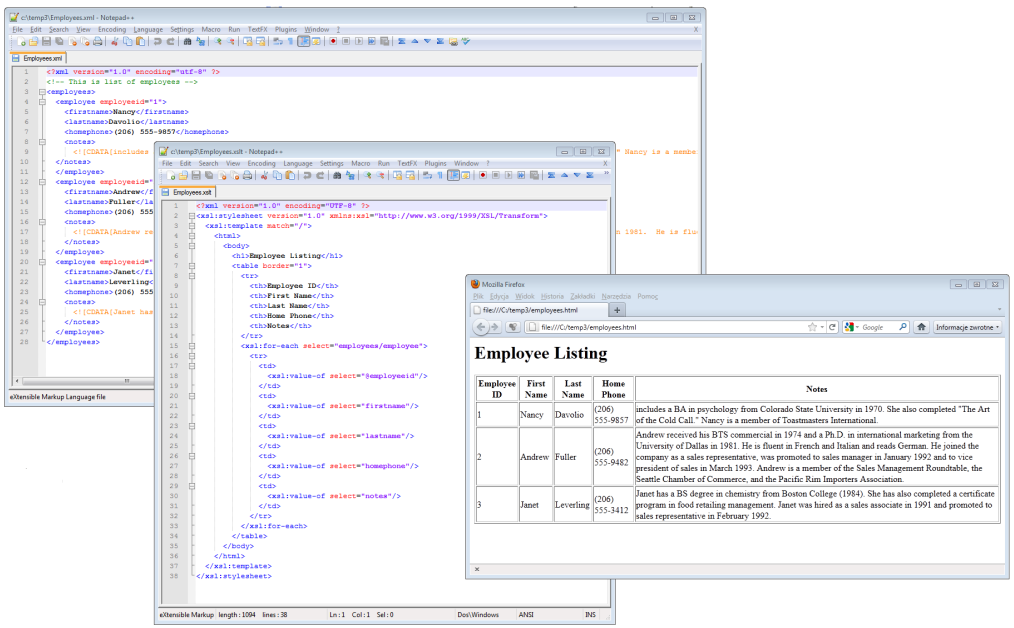
Postać pliku XSLT odpowiedzialnego za transformację danych:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h1>Employee Listing</h1>
        <table border="1">
          <tr>
            <th>Employee ID</th> <th>First Name</th> <th>Last Name</th>
            <th>Home Phone</th> <th>Notes</th>
          </tr>
          <xsl:for-each select="employees/employee">
            <tr>
              <td> <xsl:value-of select="@employeeid"/> </td>
              <td> <xsl:value-of select="firstname"/> </td>
              <td> <xsl:value-of select="lastname"/> </td>
              <td> <xsl:value-of select="homephone"/> </td>
              <td> <xsl:value-of select="notes"/> </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Realizacja przekształcenia

Kod realizujący przekształcenie w najbardziej podstawowym przypadku jest bardzo krótki:

```
XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load( "file.xslt");
xslt.Transform("file.xml", "file.html");
```



Możliwe sposoby weryfikacji poprawności dokumentu XML: None, Auto, DTD, Schema, XDR.

Podstawowy kod sprawdzający poprawność jest następujący:

```
using System.Xml;
using System.Xml.Schema;

XmlReaderSettings settings = new XmlReaderSettings();
settings.ValidationType=ValidationType.Schema;
settings.Schemas.Add("", "file.xsd");

settings.ValidationEventHandler += new ValidationEventHandler(OnValidationError);
XmlReader reader=XmlReader.Create(textBox1.Text, settings);
while (reader.Read()) {
 // odczyt danych
}
reader.Close();
```

Funkcja obsługującą błędy ma następującą postać:

```
void OnValidationError(object sender, ValidationEventArgs e) {
    if (e.Severity == XmlSeverityType.Warning) {
        // ostrzeżenie
    } else {
        // tekst błędu: e.Message
    }
}
```

Weryfikacja dla XmlDocument

Wczytanie dokumentu i weryfikacja jego poprawności:

```
XmlReaderSettings settings = new XmlReaderSettings();

settings.ValidationType = ValidationType.Schema;
settings.Schemas.Add("", "file.xsd");
settings.ValidationEventHandler += new ValidationEventHandler(OnValidationError);

XmlReader reader = XmlReader.Create( "file.xml", settings);

doc.Load(reader);
reader.Close();
```

Weryfikacja odczytanego węzła z dokumentu:

```
XmlNode node = doc.SelectSingleNode("//employee[@employeeid='" +
                                                comboBox1.SelectedItem + "']");
if (node != null) {
    // odczyt danych
}

doc.Validate(OnValidationError,node);
if (!isError) {
    // obsługa błędu
}
```

Czym jest serializacja obiektów

Serializacja jest procesem zamiany obiektów lub kolekcji obiektów na strumień danych binarnych. Wykorzystywana do zapisywania stanu systemu/obiektu w celu późniejszego odtworzenia lub przy przesyłaniu obiektów przez sieć (dość często w tym kontekście stosuje się określenie marshaling). Hasło deserializacja reprezentuje proces odwrotny.
Trzy główne klasy dostępne w platformie .NET odpowiedzialne za serializację to min.:

- serializacja binarna – klasa **BinaryFormatter** do serializowania struktur danych do strumienia binarnego. Tworzona jest dokładna/pełna reprezentacja obiektu, zawarte są również informacje o typach, wysoka wydajność ale niska przenośność pomiędzy różnymi platformami,
- serializacja standardem SOAP – klasa SoapFormatter oferuje serializację danych do postaci strumienia XML zgodnego ze standardami protokołu SOAP,
- serializacja do formatu XML – klasa XMLSerializer do serializowania danych do postaci dokumentu w formacie XML, wysoka przenośność pomiędzy platformami .

Uwaga

Wyróżnia się także dwa rodzaje serializacji: głęboką obejmującą wszelkie pola (publiczne i prywatne, a także typy zagnieżdżone) w obiekcie oraz płytką, gdzie serializacji podawane są tylko pola publiczne.

Serializacja binarna

Przygotowanie danych:

```
Serialiacja klas obecnych w BCL jest łatwa do realizacji:

Hashtable addresses = new Hashtable();

addresses.Add("Jeff", "123 Main Street, Redmond, WA 98052");
addresses.Add("Fred", "987 Pine Road, Phila., PA 19116");
addresses.Add("Mary", "PO Box 112233, Palo Alto, CA 94301");
```

Właściwa serializacja:

```
FileStream fs = new FileStream("DataFile.dat", FileMode.Create);
BinaryFormatter formatter = new BinaryFormatter();

try {
    formatter.Serialize(fs, addresses);
}
catch (SerializationException e) {
    Console.WriteLine("Błąd: " + e.Message);
    throw;
}
finally {
    fs.Close();
}
```

Serializacja binarna

Deserializacja jest realizowana w równie prosty sposób:

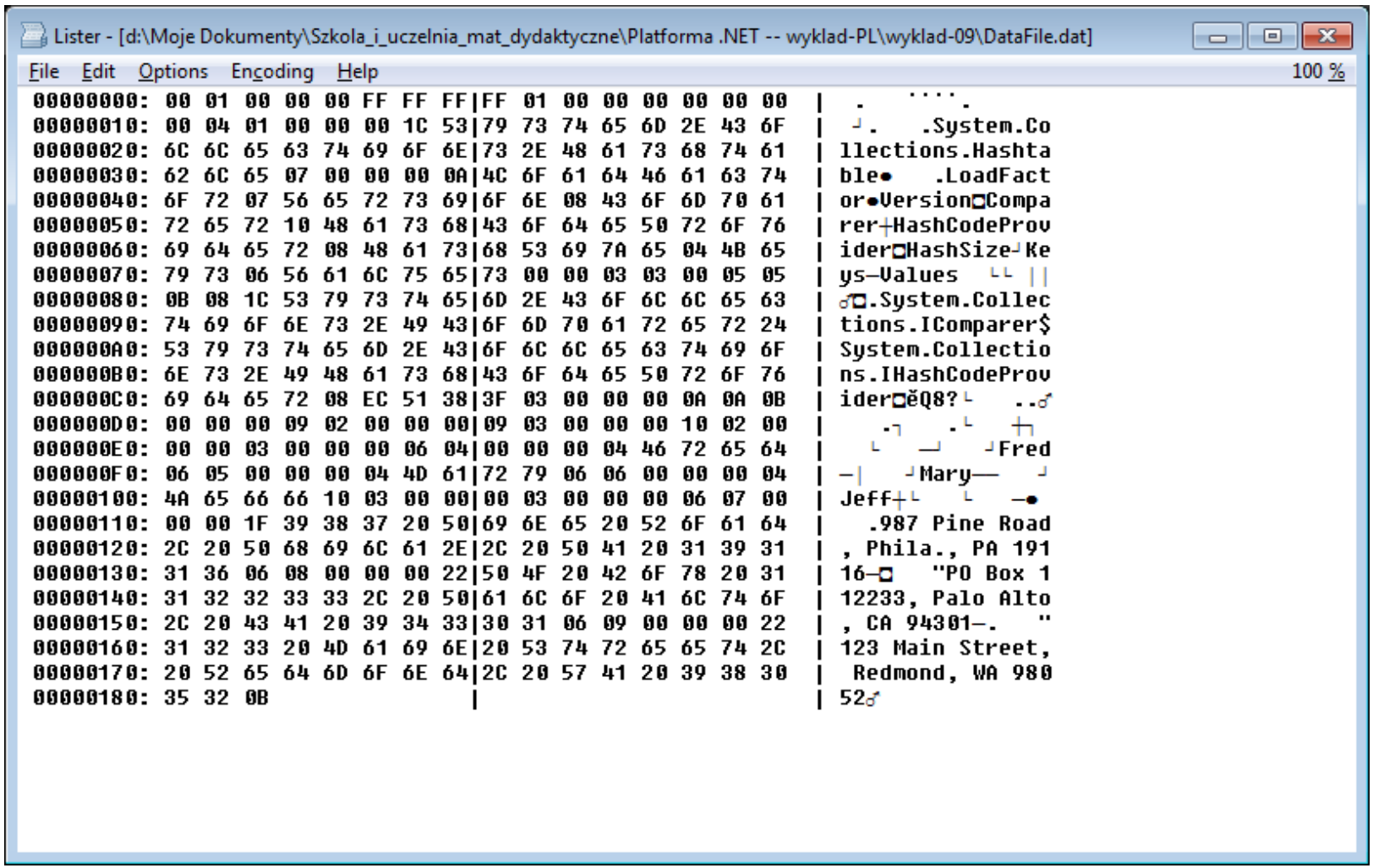
```
Hashtable addresses  = null;
FileStream fs = new FileStream("DataFile.dat", FileMode.Open);

try  {
    BinaryFormatter formatter = new BinaryFormatter();
    addresses = (Hashtable) formatter.Deserialize(fs);
}
catch (SerializationException e) {
    Console.WriteLine("Błąd: " + e.Message);
    throw;
}
finally {
    fs.Close();
}
```

Serialiacja własnej klasy wymaga dodania atrybutu: **[Serializable]**

```
[Serializable]
public class Osoba {
    public string Imię;
    public string Nazwisko;
    public string[] Adres;
[NonSerialized]
    public string osobaID = "ID-A34";
}
```

Postać binarna pliku z obiektem HashTable



Serializacja z pomocą standardu XML

Jeśli dana klasa posiada tylko pola publiczne, i struktura powstałego pliku nie jest istotna to serializacja sprowadza się do poniższego kodu:

```
XmlSerializer xmlFormat = new XmlSerializer(typeof(Osoba));

using(Stream fStream = new FileStream(fileName, FileMode.Create,
                                      FileAccess.Write, FileShare.None)) {
    xmlFormat.Serialize(fStream, objGraph);
}
```

Odczytanie osoby również jest podobne do przypadku binarnego:

```
Osoba o;

FileStream stream = new FileStream("file.xml", FileMode.Open);
XmlSerializer serializer = new XmlSerializer(typeof(Osoba));

p=(Osoba)serializer.Deserialize(stream);

stream.Close();
```

Dodatkowe atrybuty do kontroli serializacji w XML

W typowych zastosowaniach, domyślny schemat serializacji dostępny w platformie .NET jest wystarczających do poprawnego zapisu stanu obiektu. Dodanie atrybutu [Serializable] do typów wystarczy do poprawnej serializacji stanu obiektu, przy czym postać serializowanego obiektu zależy naturalnie od klasy „formatera". W pewnych przypadkach warto wpłynąć na ostateczną postać drzewa dokumentu XML poprzez dodatkowy zestaw atrybutów. Pozwala to dopasowanie postaci dokumentu XML do szczególnego wymaganego formatu lub umieszczenie dodatkowych informacji.

Atrybut	Znacznie
`[XmlAttribute]`	pole lub własność zostanie zapisanie w pliku XML w postaci atrybutu
`[XmlElement]`	pole lub własność zostanie zapisane jako element
`[XmlEnum]`	nazwa elementu w typie wyliczeniowym
`[XmlRoot]`	określenie elementu ROOT jego nazwy i przestrzeni
`[XmlText]`	własność lub pole zostanie zapisane w postaci tekstu
`[XmlType]`	określenie nazwę i przestrzeń typu XML

Klasa z atrybutami serializacji

```
[XmlRoot(ElementName="MyEmployee")]
 public class Employee {
     private int intID;
     private string strFName;
     private string strLName;
     private string strHPhone;
     private string strNotes;
     private string[] strEmails;
     private EmployeeType enumType;
     private Address objAddress=new Address();

     [XmlAttribute(AttributeName="EmployeeCode")]
     public int EmployeeID {
         get { return intID;  }
         set { intID = value; }
     }

     [XmlElement(ElementName="FName")]
     public string FirstName {
         get { return strFName;  }
         set { strFName = value; }
     }

     [XmlElement(ElementName = "LName")]
     public string LastName  { ... }
```

Klasa z atrybutami serializacji

```
    [XmlIgnore]
    public string HomePhone {
        get { return strHPhone;  }
        set { strHPhone = value; }
    }

    [XmlElement(ElementName="Remarks")]
    public string Notes { ... }

    [XmlElement(ElementName="EmployeeType")]
    public EmployeeType Type { ... }

    [XmlArray(ElementName="EmailAddresses")]
    [XmlArrayItem(ElementName="Email")]
    public string[] Emails {
        get { return strEmails;  }
        set { strEmails = value; }
    }

    [XmlElement(IsNullable=true)]
    public Address Address  { ... }
}
```

Postać pliku po serializacji z atrybutami

```
<?xml version="1.0"?>
<MyEmployee xmlns:xsi="---" xmlns:xsd="---" EmployeeCode="1">
    <FName>frist name</FName>
    <LName>last name</LName>
    <Remarks>some notes</Remarks>
    <EmployeeType>Permanent Employee</EmployeeType>
    <EmailAddresses>
        <Email>asb@t00g.pl</Email>
    </EmailAddresses>
    <Address>
        <Street>a</Street>
        <City>aa</City>
        <State>a</State>
        <Country>a</Country>
        <PostalCode>a</PostalCode>
    </Address>
</MyEmployee>
```

Zapytania LINQ to XML

Zalety zapytań zintegrowanych w kontekście XML:

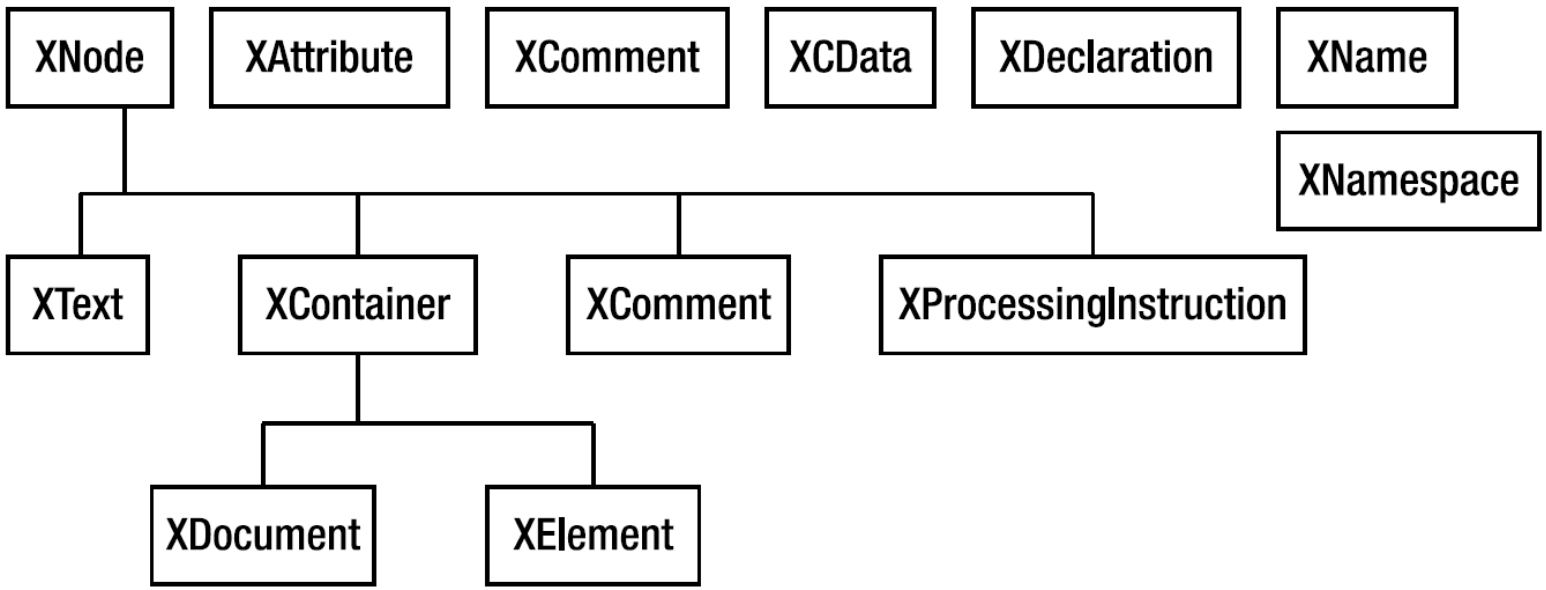
- pozwalają na dostęp do danych w XML bez konieczności stosowania dodatkowych technologi jak XPath oraz XSLT,
- wykorzystuje się standardowe polecenia LINQ,
- możliwie jest tworzenie zapytań do dokumentów XML umieszczonych w pamięci, aby otrzymywać kolekcje elementów i atrybutów XML,
- łączenie danych z różnych źródeł,
- otrzymywane rezultaty to typu XElement i XAttribtue, co pozwala na łatwe i elastyczne tworzenie nowych drzew XML.

Dwie uwagi podsumowujące LINQ:

- bogatsze i łatwiejsze w stosowaniu niż istniejące rozwiązania API oparte o klasy XmlReader/XmlWriter,
- bardziej efektywne rozwiązanie niż DOM API (w tym także mniejsze wymagania pamięciowe).

Zapytania LINQ to XML

Obsługa zapytań „LINQ to XML" została umieszczone w przestrzeni System.Xml.Linq. Hierarchię obecnych tam klas można przedstawić w następujący sposób:



Abstrakcyjna klasa XNode reprezentuje węzeł w drzewie XML. Klasy XContainer, XText, XComment, oraz XProcessingInstruction dziedziczą z klasy XNode. Klasa XContainer jest klasą bazową dla klas XDocument i XElement. Reprezentują one odpowiednio węzeł tekstowy, komentarz, instrukcję przetwarzającą, dokument XML oraz element. Klasy takie jak XAttribute, XComment, XCData oraz XDeclaration są niezależne od hierarchii klasy XNode i reprezentują atrybut, komentarz, sekcję CDATA oraz deklarację XML. Klasa XName reprezentuje nazwę elementu (XElement) lub atrybutu (XAttribute).

Podstawowe klasy do obsługi LINQ

LINQ to XML oferuje dwie klasy do wczytywania danych XML: XDocument oraz XElement. W większości przypadków, XElement jest wystarczający do wczytywania danych XML, bowiem obejmuje ładowanie plików XML, strumieni a także fragmentów XML reprezentowanych jako łańcuchy znaków.
Stosowanie XDocument jest zalecany w następujących przypadkach:

- istnieje potrzeba dodawania komentarzy na najwyższym poziomie drzewa XML,
- istnieje konieczność dołączenia instrukcji przetwarzających na najwyższym poziomie drzewa XML,
- opis DTD będzie używany do weryfikacji określonego dokumentu XML.

Sposób wczytania danych XML aby możliwie było zadawanie zapytań LINQ:

```
XElement root = null;
root = XElement.Load("file.xml");

StreamReader reader = File.OpenText("file.xml");
root = XElement.Load(reader);

XmlReader reader = XmlReader.Create("file.xml");
root = XElement.Load(reader);

root = XElement.Parse("ciąg wyrażeń XML");
```

Zapytanie do dokumentu XML

Odszukanie autora w dokumencie XML:

```
XElement books = XElement.Parse( @"<books>
    <book>
        <title>tytuł pierwszy</title>
        <author>Imie Nazwisko</author>
    </book>
    <book>
        <title>tytuł drugi</title>
        <author>Imie Nazwisko</author>
    </book>
    <book>
        <title>tytuł trzeci</title>
        <author>Imie Nazwisko</author>
    </book>
</books>");

...

var titles =
    from book in books.Elements("book")
    where (string) book.Element("author") == "Imie Nazwisko"
    select book.Element("title");
```

Zapytanie do dokumentu XML

Dokument XML zawierający tagi *p* znajdujące się na różnym poziomie zagnieżdżenia:

```
XElement doc = XElement.Parse(@"<Root>
        <p id=""1""/>
        <ul>abc</ul>
        <Child>
            <p id=""2""/> <notul/>
            <p id=""3""/> <ul>def</ul>
            <p id=""4""/>
        </Child>
        <Child>
            <p id=""5""/>  <notul/>
            <p id=""6""/>  <ul>abc</ul>
            <p id=""7""/>
        </Child>
    </Root>");
```

Zapytanie o elementy **p** posiadające znacznik **ul**

```
IEnumerable<XElement> items =
    from e in doc.Descendants("p")
    let z = e.ElementsAfterSelf().FirstOrDefault()
    where z != null && z.Name.LocalName == "ul"
    select e;
```

Dodawanie elementów do innego dokumentu XML

Budowa dwóch drzew XML:

```
XElement srcTree = new XElement("Root",
                new XElement("El1", 1), new XElement("El2", 2),
                new XElement("El3", 3), new XElement("El4", 4),
                new XElement("El5", 5)
            );
XElement xmlTree = new XElement("Root",
                new XElement("Ch1", 1), new XElement("Ch2", 2),
                new XElement("Ch3", 3), new XElement("Ch4", 4),
                new XElement("Ch5", 5)
            );
```

Dodanie nowych elementów z drzewa **srcTree**:

```
xmlTree.Add(new XElement("NewChild", "new content"));
xmlTree.Add(    from el in srcTree.Elements()
                where (int)el > 3
                select el );
xmlTree.Add(srcTree.Element("Child9"));
Console.WriteLine(xmlTree);
```

W następnym tygodniu między innymi

1. aplikacje ASP.NET,
2. struktura aplikacji WEB,
3. pojęcie formularza,
4. hierarchia klas WebControls,
5. zarządzanie sesją (ciasteczka – cookies).

Proponowane tematy prac pisemnych:

1. znaczenie i rola formatu XML,
2. weryfikacja dokumentu XML za pomocą DTD, Schema, RELAX NG w języku C#,
3. porównanie API do obsługi formatu XML w językach C# i Java.

Dziękuje za uwagę!!!