

LINQ to XML

Język C# dostarcza rozwiązań do przetwarzania plików XML w "tradycyjny sposób" (parser, np. XPath, XSLT) → [Dokumenty i dane XML](#).

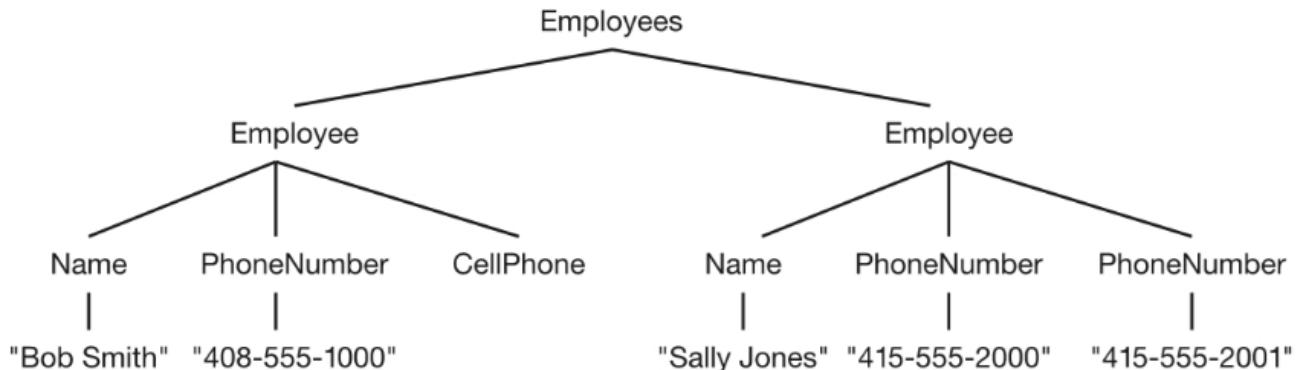
W kontekście przetwarzania danych zapisanych w plikach XML, *LINQ to XML* upraszcza te zadania
→ [Przetwarzanie danych XML przy użyciu modelu LINQ to XML](#).

Drzewo XML

Plik XML:

```
<Employees>
  <Employee>
    <Name>Bob Smith</Name>
    <PhoneNumber>408-555-1000</PhoneNumber>
    <CellPhone />
  </Employee>
  <Employee>
    <Name>Sally Jones</Name>
    <PhoneNumber>415-555-2000</PhoneNumber>
    <PhoneNumber>415-555-2001</PhoneNumber>
  </Employee>
</Employees>
```

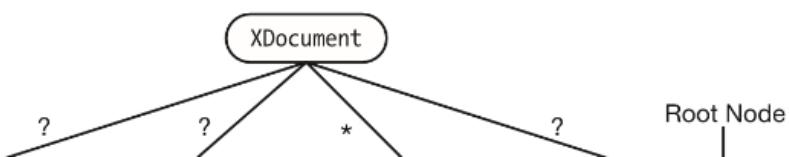
Hierarchiczna struktura pliku XML:

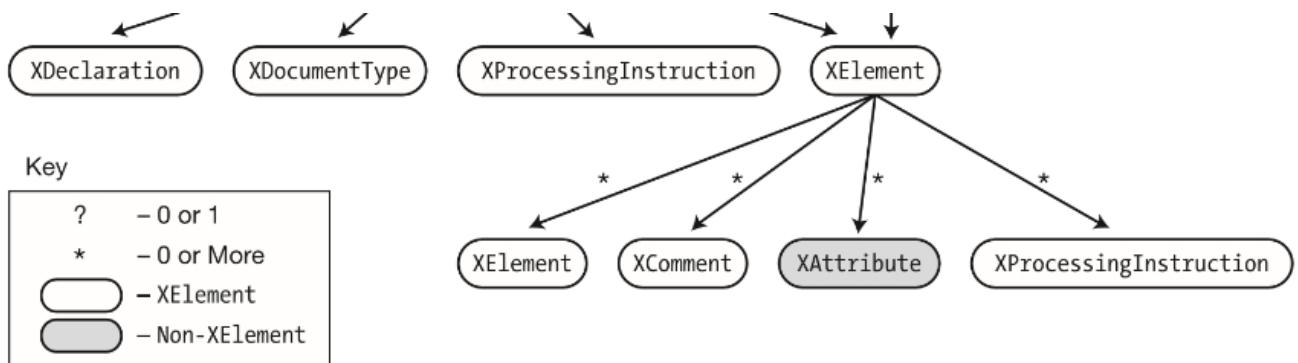


LINQ to XML API

LINQ to XML API składa się z pewnej liczby klas reprezentujących składniki drzewa XML. Najczęściej wykorzystywanymi są: `XElement`, `XAttribute` oraz `XDocument`. Klasy zlokalizowane są w przestrzeni nazw `System.Xml.Linq`.

Drzewo dokumentu XML składa się z odpowiednio powiązanych ze sobą węzłów określonych typów:

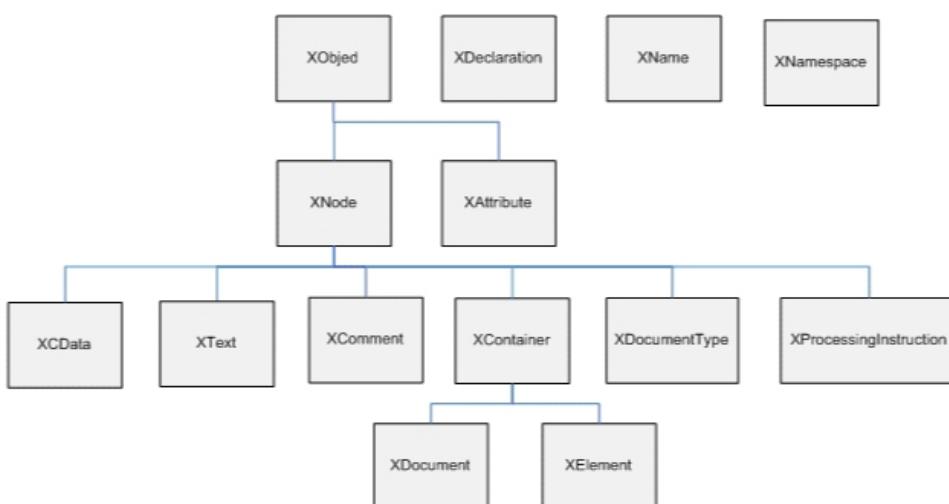




- węzłem nadrzędnym jest `XDocument`,
- jego potomkami są pojedyncze węzły typów `XDeclaration`, `XDocumentType` i `XElement` oraz * może wiele - węzłów typu `XProcessingInstruction`,
- najwyżej zlokalizowany węzeł typu `XElement` jest korzeniem (ang. *root*) dla pozostałych węzłów drzewa XML,
- węzeł typu `XElement` zawiera - być może wiele - węzłów typu `XElement` (rekurencyjnie) oraz węzły `XComment`, `XAttribute` i `XProcessingInstruction`,
- za wyjątkiem węzłów typu `XAttribute`, pozostałe dziedziczą z klasy `XNode`.

<https://docs.microsoft.com/en-us/dotnet/standard/data/xml/types-of-xml-nodes>

Architektura przestrzeni nazw `System.Xml.Linq` (uproszczona):



<https://docs.microsoft.com/pl-pl/dotnet/api/system.xml.linq>

Tworzenie i przeglądanie drzewa XML

<https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/concepts/linq/creating-xml-trees-linq-to-xml-2>

```

using System.Xml.Linq;

public static XDocument CreateXMLExample()
{
    return
        new XDocument(                               // Tworzy dokument XML
            new XElement("Employees",               // Tworzy element root
                new XElement("Employee",             // Tworzy pierwszy element Employee
                    new XElement("Name", "Bob Smith"),
                    new XElement("PhoneNumber", "408-555-1000")
                ),
                ...
            )
        );
}
  
```

```

        new XElement("Employee",
            new XElement("Name", "Sally Jones"),
            new XElement("PhoneNumber", "415-555-2000"),
            new XElement("PhoneNumber", "415-555-2001")
        )
    ) // koniec XElement Employees
); // koniec XDocument
}

```

W klasie `XDocument` przeciążona metoda `ToString()` dostarcza sformatowaną reprezentację tekstową drzewa XML:

```

XDocument employeesTree = CreateXMLExample();
Console.WriteLine( employeesTree );

```

```

<Employees>
  <Employee>
    <Name>Bob Smith</Name>
    <PhoneNumber>408-555-1000</PhoneNumber>
  </Employee>
  <Employee>
    <Name>Sally Jones</Name>
    <PhoneNumber>415-555-2000</PhoneNumber>
    <PhoneNumber>415-555-2001</PhoneNumber>
  </Employee>
</Employees>

```

Linq to XML dostarcza wygodnego wsparcia dla zapisywania i odczytywania plików XML (metody `Save`, `SaveAsync`, `Load`, `LoadAsync` oraz metody `WriteTo` i `WriteTo` zapisujące do strumienia `XmlWriter`).

```

XDocument employeesTree = CreateXMLExample();
employeesTree.Save("EmployeesFile.xml");
XDocument employeesTree1 = XDocument.Load("EmployeesFile.xml");
Console.WriteLine( employeesTree1 );

```

Jeśli dane w formacie XML mamy już załadowane do pamięci i zapamiętane w zmiennej typu `string`, do przetworzenia napisu do postaci drzewa XML używamy statycznej metody `XDocument.Parse(string text)`.

Po drzewie XML możemy się poruszać, odczytując lub modyfikując aktualne elementy.

W klasach `XDocument` oraz `XElement` zdefiniowane są metody umożliwiające odczytywanie danych podczas przemieszczania się po drzewie XML (formalnie dziedziczą z klas abstrakcyjnych `XNode` oraz `XContainer`).

Method Name	Class	Return Type	Description
Nodes	XDocument XElement	IEnumerable<object>	Returns all the children of the current node, regardless of their type
Elements	XDocument XElement	IEnumerable< XElement >	Returns all the current node's XElement child nodes or all the child nodes with a specific name
Element	XDocument XElement	XElement	Returns the current node's first XElement child node or the first child node with a specific name
Descendants	XElement	IEnumerable< XElement >	Returns all the descendant XElement nodes or all the descendant XElement nodes with a specific name, regardless of their level of

DescendantsAndSelf	XElement	IEnumerable< XElement >	nesting below the current node
Ancestors	XElement	IEnumerable< XElement >	Same as Descendants, but also includes the current node
AncestorsAndSelf	XElement	IEnumerable< XElement >	Returns all the ancestor XElement nodes or all the ancestor XElement nodes above the current node that have a specific name
Parent	XElement	XElement	Same as Ancestors, but also includes the current node
			Returns the parent node of the current node

- Metoda `Nodes()` zwraca wszystkie węzły "w dół" począwszy od węzła bieżącego w formie sekwencji `IEnumerable`. Ponieważ zwracane węzły mogą być różnych typów, np. `XElement`, `XComment`, ..., zatem sekwencja zawiera obiekty `XNode` (zwracana jest sekwencja `IEnumerable< XNode >`).
- Metoda `Elements()` daje uproszczenie zapisu metody `Node()` dla elementów typu `XElement`:
 - użyta bez argumentów zwraca wszystkie podrzędne obiekty `XElement`,
 - użyta z jednym argumentem typu `XName`, zwraca elementy podrzędne filtrowane ze względu na nazwę.

Uwaga: `XName` nie zawiera żadnych konstruktorów publicznych. Klasa zapewnia niejawną (automatyczną) konwersję z `string`, zatem w miejscach, w których oczekiwaliśmy obiektu `XName` możemy użyć obiektu typu `string`.
- Metoda `Element()` zwraca pierwszy podrzędny w stosunku do aktualnego węzła `XElement`. Podobnie jak metoda `Elements()` może być wywołana bez parametrów lub z jednym parametrem typu `XName`.
- Metody `Descendants()` oraz `Ancestors()`: działanie podobne do `Elements()`, ale zamiast zwracać bezpośrednie elementy podrzędne (lub nadrzędne), zawierają elementy poniżej (lub powyżej) bieżącego węzła, niezależnie od różnicy w poziomie zagnieżdżenia. Metody te wykorzystują odroczone wykonanie. Szczególnie często wykorzystywane w zapytaniach *Linq-to-XML*.

Przykład:

```
XDocument employeesTree = CreateXMLExample();
// XElement root = employeesTree.Element("Employees"); //pobranie XElement o nazwie "Employees"
XElement root = employeesTree.Root; //równoważne powyższemu
foreach( var emp in root.Elements() )
{
    //emp jest węzłem Employee
    XElement empNameNode = emp.Element("Name");
    Console.WriteLine( empNameNode.Value );
    IEnumerable< XElement > phones = emp.Elements("PhoneNumber");
    foreach(var phone in phones)
    {
        Console.WriteLine( $" * {phone.Value}" );
    }
}
```

```
Bob Smith
* 408-555-1000
Sally Jones
* 415-555-2000
* 415-555-2001
```

Przykład z metodą `Descendants()`:

```

XDocument employeesTree = CreateXMLExample();

var empsQuery =
    from emp in employeesTree.Descendants("Employee")
    select emp;

foreach(var emp in empsQuery)
{
    string empName = emp.Element("Name").Value;
    IEnumerable<string> empPhoneList = emp.Elements("PhoneNumber").Select(x => x.Value);

    Console.WriteLine( $"{empName} : {string.Join(", ", empPhoneList)}" );
}

```

```

Bob Smith : 408-555-1000
Sally Jones : 415-555-2000, 415-555-2001

```

Modyfikowanie drzewa XML

Do istniejącego drzewa można dynamicznie dołączyć kolejne węzły (metoda `Add()`). Można określić dodanie węzła wariantami tej metody: `AddFirst()`, `AddBeforeSelf()`, `AddAfterSelf()`.

Method Name	Call From	Description
Add	Parent	Adds new child nodes after the existing child nodes of the current node
AddFirst	Parent	Adds new child nodes before the existing child nodes of the current node
AddBeforeSelf	Node	Adds new nodes before the current node at the same level
AddAfterSelf	Node	Adds new nodes after the current node at the same level
Remove	Node	Deletes the currently selected node and its contents
RemoveNodes	Node	Removes the child nodes from an XContainer
SetElement	Parent	Sets the contents of a node

Przykład:

```

XDocument employeesTree = CreateXMLExample();
 XElement root = employeesTree.Root;
 XElement osoba = new XElement("Employee",
        new XComment("Wykładowca WSEI"),
        new XElement("Name", "Krzysztof Molenda"),
        new XElement("PhoneNumber", "+48 111-111-111")
    );
root.Add( osoba );

Console.WriteLine( employeesTree );

```

```

<Employees>
  <Employee>
    <Name>Bob Smith</Name>
    <PhoneNumber>408-555-1000</PhoneNumber>
  </Employee>
  <Employee>
    <Name>Sally Jones</Name>
    <PhoneNumber>415-555-2000</PhoneNumber>
    <PhoneNumber>415-555-2001</PhoneNumber>
  </Employee>
</Employees>

```

```
<!--Wykładowca WSEI-->
<Name>Krzysztof Molenda</Name>
<PhoneNumber>+48 111-111-111</PhoneNumber>
</Employee>
</Employees>
```

W podobny sposób można usuwać wskazane węzły:

Przykład (`Remove()` z `XNode`):

```
XDocument employeesTree = CreateXMLExample();
 XElement empToRemove = employeesTree
 .Descendants("Employee")
 .FirstOrDefault(x => x.Element("Name").Value == "Bob Smith");
empToRemove.Remove();
Console.WriteLine(employeesTree);
```

```
<Employees>
<Employee>
<Name>Sally Jones</Name>
<PhoneNumber>415-555-2000</PhoneNumber>
<PhoneNumber>415-555-2001</PhoneNumber>
</Employee>
</Employees>
```

Metoda `RemoveAll()` usuwa wszystkie elementy z drzewa XML.

W podobny sposób modyfikujemy wartości węzłów:

```
XDocument employeesTree = CreateXMLExample();
 XElement empToModify = employeesTree
 .Descendants("Employee")
 .FirstOrDefault(x => x.Element("Name").Value == "Bob Smith");
empToModify.SetElementValue("Name", "Boby Smith");
Console.WriteLine(employeesTree);
```

```
<Employees>
<Employee>
<Name>Boby Smith</Name>
<PhoneNumber>408-555-1000</PhoneNumber>
</Employee>
<Employee>
<Name>Sally Jones</Name>
<PhoneNumber>415-555-2000</PhoneNumber>
<PhoneNumber>415-555-2001</PhoneNumber>
</Employee>
</Employees>
```

Atrybuty w drzewie XML

Atrybuty dostarczają dodatkową informację o węźle `XElement` - zawarte są wewnątrz znacznika XML.

Tworząc drzewo XML z poziomu kodu, dodajemy atrybut za pomocą konstruktorów:

- dwuargumentowego - przekazującego nazwę atrybutu oraz wartość atrybutu,
- jednoargumentowego - przekazującego referencję do już istniejącego atrybutu.

Poniższy przykład tworzy drzewo XML z węzłami zawierającymi atrybuty:

```
public static XDocument CreateXMLExampleAttributes()
{
```

```

return
    new XDocument(
        new XElement("Employees",           // Tworzy dokument XML
            new XElement("Employee",         // Tworzy element root
                new XAttribute("EmpId", "101"), // Konstruktor atrybutu EmpId
                new XAttribute("Gender", "M"),   // Konstruktor atrybutu Gender
                new XElement("Name", "Bob Smith"),
                new XElement("PhoneNumber", "408-555-1000")
            ),
            new XElement("Employee",           // Tworzy drugi element Employee
                new XElement("Name", "Sally Jones"),
                new XElement("PhoneNumber", "415-555-2000"),
                new XElement("PhoneNumber", "415-555-2001")
            )
        ) // koniec XElement Employees
    ); // koniec XDocument
}

```

Wywołanie metody `CreateXMLExampleAttributes()`:

```
Console.WriteLine( CreateXMLExampleAttributes() );
```

```

<Employees>
    <Employee EmpId="101" Gender="M">
        <Name>Bob Smith</Name>
        <PhoneNumber>408-555-1000</PhoneNumber>
    </Employee>
    <Employee>
        <Name>Sally Jones</Name>
        <PhoneNumber>415-555-2000</PhoneNumber>
        <PhoneNumber>415-555-2001</PhoneNumber>
    </Employee>
</Employees>

```

Przeglądając drzewo XML możemy odczytywać atrybuty węzłów metodą `Attribute()`, podając nazwę atrybutu jako parametr:

```

XDocument employeesTree = CreateXMLExampleAttributes();
foreach(var emp in employeesTree.Descendants("Employee"))
{
    var name = emp.Element("Name")?.Value;
    var id = emp.Attribute("EmpId")?.Value;
    Console.WriteLine( $" {id}: {name}" );
}

```

```

101: Bob Smith
: Sally Jones

```

W kodzie użyto operatora `?`, aby uniknąć zgłoszenia wyjątku `NullReferenceException` w sytuacji braku przypisanego atrybutu do węzła. `Console.WriteLine()` nie drukuje `null`.

Aby usunąć atrybut lub go dodać/zmodyfikować, należy wskazać element, wskazać atrybut i wywołać na nim `Remove()` (lub `SetAttributeValue()`).

```

XDocument employeesTree = CreateXMLExampleAttributes();
 XElement empToModify = employeesTree
    .Descendants("Employee")
    .FirstOrDefault(x => x.Element("Name").Value == "Bob Smith");
empToModify?.Attribute("Gender")?.Remove();

empToModify = employeesTree
    .Descendants("Employee")

```

```
        .FirstOrDefault(x => x.Element("Name").Value == "Sally Jones");
empToModify?.SetAttributeValue("EmpId", "102");

Console.WriteLine( employeesTree );
```

```
<Employees>
  <Employee EmpId="101">
    <Name>Bob Smith</Name>
    <PhoneNumber>408-555-1000</PhoneNumber>
  </Employee>
  <Employee EmpId="102">
    <Name>Sally Jones</Name>
    <PhoneNumber>415-555-2000</PhoneNumber>
    <PhoneNumber>415-555-2001</PhoneNumber>
  </Employee>
</Employees>
```

W powyższym przykładzie usuwamy atrybut `Gender` z węzła *Bob Smitha* oraz dodajemy nowy atrybut do węzła *Sally Jones* za pomocą `SetAttributeValue()`.

Inne typy węzłów XML

XComment

Wprowadza komentarz do dokumentu XML w określonej pozycji (węźle) - już było w jednym z przykładów.

XDeclaration

Dokument XML zaczyna się od deklaracji, zawierającej opis wersji użytego standardu XML, sposobu kodowania znaków oraz czy dokument jest zależny od zewnętrznych referencji. Są to *metadane* dokumentu.

```
new XDeclaration("1.0", "utf-8", "yes")  
  
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

XProcessingInstruction

Instrukcja przetwarzania XML służy do dostarczania dodatkowych informacji o tym, w jaki sposób dokument XML powinien być używany lub interpretowany. Najczęściej instrukcje przetwarzania służą do powiązania arkusza stylów z dokumentem XML.

Można dodać instrukcję przetwarzania za pomocą konstruktora `XProcessingInstruction`, który przyjmuje dwa parametry `string` : - cel (*target*) i napis określający parametry przetwarzania.

```
static void Main()
{
    XDocument xmlDocTree = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        new XComment("This is a comment"),
        new XProcessingInstruction("xml-stylesheet",
            @"href=""style.css"" type=""text/css"""),
        new XElement("root",
            new XElement("first"),
            new XElement("second"))
    );
    Console.WriteLine( xmlDocTree );
}
```

```
<!--This is a comment-->
<?xmlstylesheet href="style.css" type="text/css"?>
<root>
  <first />
  <second />
</root>
```

Zwróćmy uwagę, że w powyższym przykładzie drugi parametr jest *verbatim string* (@), a cudzysłowy wewnątrz ciągu muszą wtedy być reprezentowane przez zestawy dwóch cudzysłówów.

Kwerendy Linq-to-XML

W przestrzeni nazw `System.Linq.Xml` znajduje się klasa statyczna `Extensions` zawierająca metody rozszerzające typy `XElement`, `XAttribute` oraz `IEnumerable< XElement >` i `IEnumerable< XAttribute >`, definiujące operatory LINQ dostosowane do przetwarzania drzewa XML.

<https://docs.microsoft.com/pl-pl/dotnet/api/system.xml.linq.extensions>

Powszechnym scenariuszem postępowania jest:

1. Załadowanie dokumentu XML
2. Zdefiniowanie kwerendy operującej na drzewie XML, wykorzystując notację *fluid* albo *query* (najczęściej zaczynamy od metody `Descendants()` wywołanej dla załadowanego drzewa XML)
3. Wywołanie kwerendy

Przykład prezentujący utworzenie drzewa XML dla danych w tabeli `Products` w bazie danych `Northwind` (komunikacja z bazą z wykorzystaniem *EF Core*):

```
static void OutputProductsAsXml()
{
    using (var db = new Northwind())
    {
        var productsForXml = db.Products.ToArray();
        var xml = new XElement("products",
            from p in productsForXml
            select new XElement("product",
                new XAttribute("id", p.ProductID),
                new XAttribute("price", p.UnitPrice),
                new XElement("name", p.ProductName)));
        Console.WriteLine(xml.ToString());
    }
}
```

```
<products>
  <product id="1" price="18">
    <name>Chai</name>
  </product>
  <product id="2" price="19">
    <name>Chang</name>
  </product>
  ...

```

Przykład prezentujący wczytywanie i przetwarzanie pliku XML:

- Dla pliku XML `settings.xml` zawierającego konfigurację pewnej aplikacji:

```
<?xml version="1.0" encoding="utf-8" ?>
<appSettings>
  <add key="color" value="red" />
  <add key="size" value="large" />
  <add key="price" value="23.99" />
</appSettings>
```

- kod wczytuje plik, wypisuje na konsolę wybrane fragmenty, dodaje nowy element na początku drzewa i zapisuje to drzewo XML na dysku do pod tą samą nazwą:

```
XDocument doc = XDocument.Load("settings.xml");
var appSettings = doc.Descendants("appSettings")
    .Descendants("add")
    .Select(node => new
    {
        Key = node.Attribute("key").Value,
        Value = node.Attribute("value").Value
    })
);

foreach (var item in appSettings.ToArray())
    Console.WriteLine($"{item.Key}: {item.Value}");

doc.Root.AddFirst(
    new XElement("add",
        new XAttribute("key", "quantity"),
        new XAttribute("value", "100"))
);
doc.Save("settings.xml");
```

☰