

# 11. POGLAVLJE

## XML

### U ovom poglavlju:

- Osnovni pojmovi vezani uz XML
- Rad s XML-om u .NET-u
- Dodavanje, traženje, mijenjanje i brisanje dijelova XML dokumenata
- Učitavanje, spremanje i pisanje XML dokumenata
- Osnove XPatha
- Pretraživanje XML dokumenata XPathom
- XML serijalizacija

**X**ML (eXtensible Markup Language) posljednjih nekoliko godina možete pronaći u reklamnim materijalima gotovo svake aplikacije – svi se hvale da mogu izvoziti svoje datoteke i spremati ih u XML formatu da interno koriste XML, kao da je to nešto što čini njihovu aplikaciju moćnijom od konkurencije. U ovom poglavlju ćemo razjasniti što je uopće XML, kako se on može uklopiti u vašu aplikaciju te gdje zapravo leži snaga XML-a.

### III. DIO: DIJELOVI .NET-A

Naravno, i Microsoft je uvidio važnost XML formata i njegova podrška je uključena u niz programa (primjerice, Microsoft Office ima mogućnost spremanja svih datoteka u XML formatu). Ipak, kako se radi o formatu koji će pretežito koristiti programeri u svojim aplikacijama i za nadogradnju postojećih rješenja, podrška za XML u Visual Studiju .NET je na svakom koraku.

## Osnovni pojmovi

Dakle, mnogo je bilo priče o XML-u kao spasitelju informatičke industrije, nečemu što će poboljšati sve aplikacije, učiniti život programera lakšim i učiniti sve kupce softvera zadovoljnim. Očito se radi o pretjeranim tvrdnjama jer je takvih tehnologija “spasitelja” kroz povijest bilo mnogo, no ipak – XML je po mnogočemu poseban i može vašem programu i načinu razmišljanja predstavljati spas.

Što je uopće XML? Radi se o običnom tekstualno strukturiranom formatu zapisa. No dobro, možete pomisliti, što je tu toliko moćno? Upravo ta jednostavnost postavlja XML na uzvišenu poziciju.



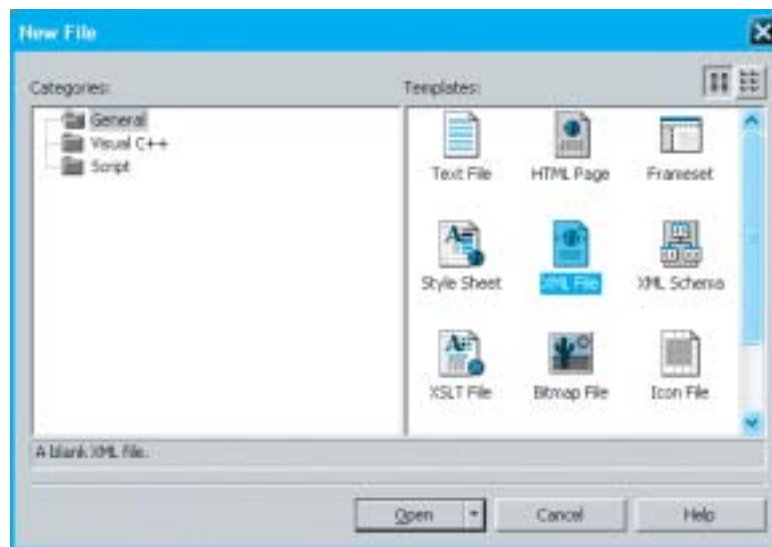
**XML format je zapravo preporuka World Wide Web Consortiuma (W3C, više informacija na <http://www.w3c.org/>), grupe koja je zadužena za donošenje različitih standarda i koja je, primjerice, već prije definirala HTML, CSS, a zadužena je i za druge tehnologije vezane uz XML koje će biti objašnjene u ovom poglavlju, poput XPatha i XSLT-a..**

XML format ćete koristiti za spremanje različitih podataka, konfiguracijskih datoteka, kao jedan od izlaznih formata za vaše datoteke, kao format za podatke koje ćete razmjenjivati s drugim aplikacijama. Kako se radi o tekstualnom tipu podatka, on je čitljiv na svim platformama. Čitati i proučavati ga mogu i korisnici, tako da ga otvori u bilo kojem uređivaču teksta.

XML se sastoji od hijerarhijskih elemenata, baš kao i HTML (npr. HTML *tag* sadržava HEAD *tag*, a on pak može sadržavati TITLE i META *tagove*). No dok HTML ima točno određene elemente koje može sadržavati, XML je mnogo slobodniji – u njemu možete definirati koju god strukturu želite. Za XML se kaže da je proširiv, što u praksi znači da može imati koliko želite različitih elemenata, na kojim god pozicijama u dokumentu odnosno da vi sami određujete format svog XML dokumenta.

Krenimo odmah i na konkretan primjer – već smo objasnili da je uloga XML-a da sadržava neke podatke pa ćemo sad izraditi jednostavnu XML datoteku koja će sadržavati informacije o kupcima u nekom dućanu.

Da biste stvorili novu XML datoteku, u Visual Studiju .NET odaberite *File – New – File* ili pritisnite CTRL+N. Pojavit će vam se prozor kao na slici 11-1, a u njemu odaberite stvaranje XML datoteke.



**Slika 11-1:**  
**Stvaranje nove XML**  
**datoteke u Visual**  
**Studiju .NET**

U njoj će već biti upisano zaglavlje dokumenta, a vi možete upisati ostatak sadržaja. Evo primjera XML dokumenta.

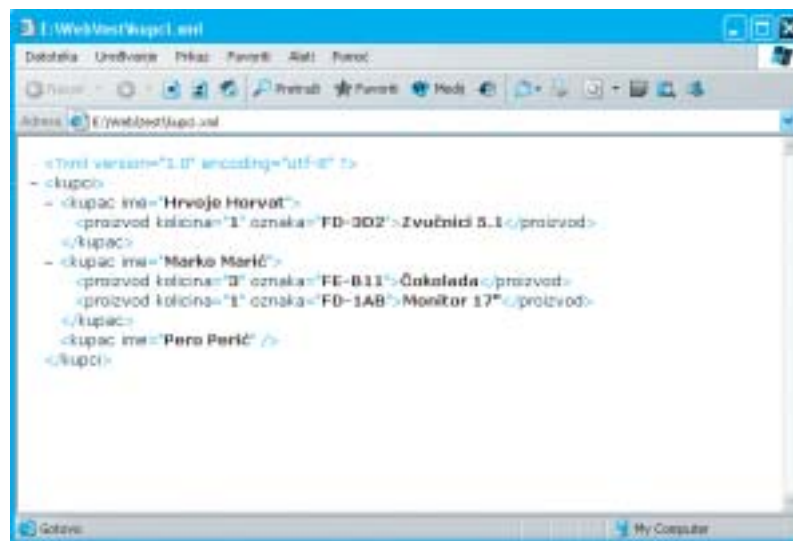
```
<?xml version="1.0" encoding="utf-8" ?>
<kupci>
  <kupac ime="Hrvoje Horvat">
    <produkt kolicina="1" oznaka="FD-3D2">Zvučnici 5.1</produkt>
  </kupac>
  <kupac ime="Marko Marić">
    <produkt kolicina="3" oznaka="FE-B11">Čokolada</produkt>
    <produkt kolicina="1" oznaka="FD-1AB">Monitor 17</produkt>
  </kupac>
  <kupac ime="Pero Perić" />
</kupci>
```

Pogledate li primjer, uočite ćete da je njegova struktura veoma slična HTML-u. Glavnu ulogu imaju elementi, odnosno *tagovi*, koji pak imaju neke atribute, a svi oni su složeni hijerarhijski.

Također, specifičnost XML formata za podatke je u tome što je on svima lako razumljiv. Gornji primjer zaista ne skriva neke tajne – sadržava informacije o kupcima koji su složeni hijerarhijski ispod glavnog *taga* nazvanog “kupci”. Za svakog kupca zapisano je i njegovo ime te proizvodi koje je kupio (logika je i dalje ista, pa je za svaki proizvod zapisana njegova količina i oznaka na skladištu te opis). Uočite da smo sami imenovali sve *tagove* i odredili koje podatke mogu sadržavati.

### III. DIO: DIJELOVI .NET-A

**Slika 11-2:**  
*Internet Explorer će  
na pregledan način  
prikazati strukturu  
XML dokumenta.*



Prethodni primjer ima i jednu posebnost. Naime, pogledate li kupca Peru Perića, vidjet ćete da za njega nema nikakvih informacija o proizvodima, vjerojatno zato što nijedan nije kupio. No posebnost se očituje u načinu zatvaranja njegovog taga "kupač". Dok se može očekivati da kao i u HTML-u postoji obavezno "</kupač>", u slučaju da tag ništa ne sadržava, može ga se zatvoriti i na kraći način – zatvaranjem taga sa znakovima "</>".

XML dokumenti su obične tekstualne datoteke, pa ih zato možete izrađivati u bilo kojem tekstualnom uređivaču, od Notepada do Visual Studija. Naravno, preporučujemo vam da ostanete uz Visual Studio jer on ima ugrađenu podršku za XML format pa vam može pomoći u pisanju – automatski će zatvarati sve *tagove* koje napišete.



Preporučujemo vam da konfiguracijske datoteke, kao uostalom i sve datoteke koje koristite u programiranju, otvarate i mijenjate pomoću Visual Studija. Naime, sve se datoteke prema inicijalnim postavkama spremaju u formatu Unicode, što mnogi editori ne podržavaju. Nakon snimanja datoteka s takvim programima, one bi mogle ispasti neispravne i neupotrebljive.

## 11. POGLAVLJE: XML

Kad budete radili s malo složenijim XML dokumentima možda ćete se i teže u njima snalaziti. XML format ne zahtijeva od vas da uvlačite sve elemente koji su niže po hijerarhiji i tako si olakšavate pregled. Što se samog formata tiče, *tagove* možete naslagati jedan na drugi, no pokušajte se tada snaći u sadržaju.

Na svu sreću, Internet Explorer u potpunosti podržava prikaz XML datoteka. Otvorite načinjenu datoteku u Internet Exploreru i vidjet ćete hijerarhijsku strukturu dokumenta. Čak možete i zatvarati podstrukture nekih elemenata klikom na znak “-” kraj njega.

Vi imate potpunu slobodu u radu s XML-om. Na vama je da osmisлите format dokumenata, napunite ga podacima i izradite programsku podršku koja će iskoristiti XML datoteke (to vas čeka u nastavku poglavlja). No ponekad baš ta sloboda može biti ograničavajući faktor. Ponekad možete podatke prezentirati i oblikovati na krivi način te oni mogu biti nerazumljivi, teški za obradu, višeznačni i slično.

Za XML je presudno da ga svi mogu razumjeti. Ljudi bi ga trebali moći čitati i shvatiti okvirno o čemu se u tom dokumentu radi, a računala i programi bi trebali moći obraditi ga i iz njega izvući jednodoznačne podatke. Primjerice, pogledajte neispravno korištenje XML-a:

```
<?xml version="1.0" encoding="utf-8" ?>
<proizvodi>
  <proizvod oznaka="FD-3D2">
    <kolicina vrijednost="1">
      <kupac ime="Hrvoje Horvat">Zvučnici 5.1</kupac>
    </kolicina>
  </proizvod>
  <proizvod oznaka="FE-B11">
    <kolicina vrijednost="3">
      <kupac ime="Marko Marić">Čokolada</kupac>
    </kolicina>
  </proizvod>
  <proizvod oznaka="FD-1AB">
    <kolicina vrijednost="1">
      <kupac ime="Marko Marić">Monitor 17</kupac>
    </kolicina>
  </proizvod>
</proizvodi>
```

Iako će to biti objašnjeno kasnije u tekstu, nije naodmet ponoviti – struktura gornjeg primjera nije korisna jer nelogično prikazuje podatke. U kasnijem tekstu ćemo se više puta referencirati na primjer XML dokumenta, a pritom ćemo misliti na njegovu prvu, logičnu verziju, a ne na ovu.



### III. DIO: DIJELOVI .NET-A

Na prvi je pogled i ovaj dokument potpuno jasan. Organiziran je po proizvodima, a za svaki je proizvod zapisana količina i koji je kupac tu količinu kupio. Na kraju cijele hijerarhije nekog proizvoda zapisan je i sam opis tog proizvoda. Naravno, možda se to moglo i bolje organizirati, no u žurbi ste, a ovaj oblik je zadovoljavajući. Ni ne slutite potencijalne probleme.

No oni su vrlo stvarni – kako ćete znati da kupac Pero Perić nije ništa kupio? Zamislite da se radi o malo većem dokumentu i pokušajte na prvi pogled saznati što je sve kupio Marko Marić. Okvirno, ovaj oblik XML dokumenta nije toliko loš (organiziran je po proizvodima), no potpuno je kriva hijerarhija u kojoj *kupac* dolazi ispod *količine*. Naravno, besmisleno je i da se opis proizvoda nalazi unutar elementa *kupac*.

## XML pravila

Ako vam se može činiti da je XML doista slobodan format, postoji nekoliko pravila kojih se morate pridržavati da bi XML dokument bio valjan:

1. Mora postojati jedan i samo jedan korijenski (glavni) element.
2. Svi *tagovi* se moraju zatvoriti.
3. Imena elemenata su osjetljiva na velika i mala slova.

U našem prvom primjeru postoji i `<?xml version="1.0" encoding="utf-8" ?>` element, no on nije obavezan i njegova je svrha opisna. Ukoliko ga izbacite, dokument će i dalje biti ispravan.

Prvo pravilo kaže da mora postojati jedan i samo jedan glavni element, a to je u našem slučaju element *kupci*. Unutar njega su sadržani svi ostali elementi. Ukoliko njega ne bi bilo, svi bi njegovi podelementi *kupac* bili glavni, a to je

nedozvoljeno, jer kao što pravilo kaže – mora postojati *jedan i samo jedan*.

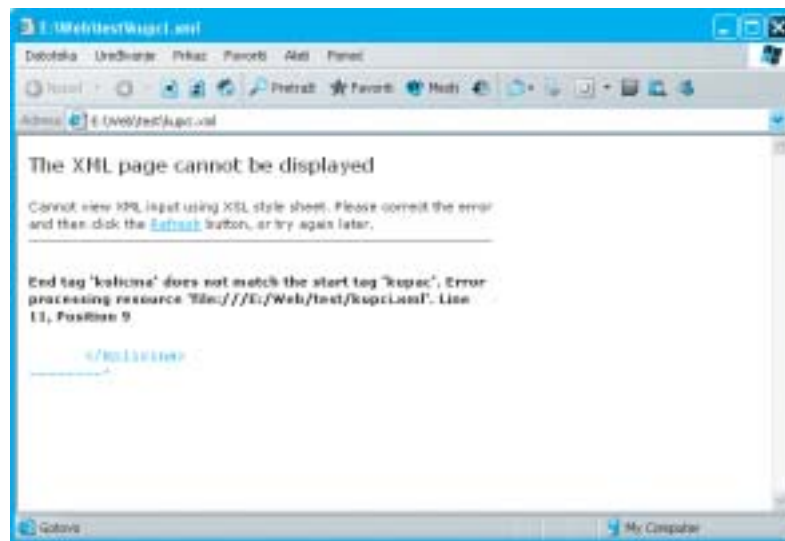
Drugo pravilo govori o zatvaranju *tagova*. To je već objašnjeno – *tagove* možete zatvoriti standardnim, HTML, načinom tako da stavite krajnji *tag* oblika `</ ... >`, no možete ih i odmah zatvoriti tako da na kraj *taga* stavite znakove `</>`.

Treće pravilo može biti problematično ukoliko ručno u Notepadu pišete XML dokument. Primjerice, greška je ako napišete `<kupacIme>Hrvoje Horvat</kupacime>`, jer *kupacIme* nije isto što i *kupacime*.

XML parseri, odnosno programi zaduženi za obradu XML dokumenata, ne znaju ništa raditi s dokumentima koji nisu ispravni. Ukoliko se vaš dokument ne podvrgava spomenutim pravilima, ne smatra ga se XML dokumentom. Pokušajte neku od opisanih grešaka napisati u XML-u, te zatim taj dokument učitati u Internet Exploreru. Kao što je i očekivano, dojavit će vam grešku.

## 11. POGLAVLJE: XML

No, da ne duljimo previše, veoma je važna ispravna organizacija XML dokumenata. Nakon što izradite svoj XML, postavite si osnovna pitanja o njegovu korištenju. Ključno je da razumijete osnovni princip – ako vi možete na jednostavan način izvući neke podatke, tad će to moći i program. Ukoliko vi morate pamtit i više stvari, puno *scrollati* po dokumentu da biste isti podatak izvukli s više mjesta, to očito nije dobro jer će istu stvar morati raditi i program, a to će biti sporo i loše.



**Slika 11-3:**  
**Internet Explorer**  
**neće otvoriti neispravan XML dokument,**  
**već će vam dojaviti grešku.**

Da biste bili sigurni u ispravnost XML dokumenata, iskoristite mogućnost njihova provjeravanja direktno iz Visual Studija .NET. Odaberite opciju XML – Validate XML Data, a u prozoru Task List će se pojaviti popis svih grešaka i upozorenja s detaljnim informacijama.



## XML DOM

Uz XML, W3C se pobrinuo i za definiranje standardiziranog API-ja nazvanog XML DOM (od Document Object Model). DOM API predstavlja XML dokument kao stablo elemenata – zahvaljujući njegovoj hijerarhijskoj strukturi, lako je posložiti sve elemente i predstaviti ih u obliku *stabla* u kojem postoje korijenski element te elementi *listovi* (koji nemaju više svoje *djece* odnosno podelemenata).

Kretanjem po stablu lako je pronaći bilo koji element u XML strukturi. Predočite li hijerarhijsku strukturu elemenata tako da svaki element može imati svoju *djecu* odnosno elemente koji su u

### III. DIO: DIJELOVI .NET-A

hijerarhiji ispod njega te *roditelja* odnosno element koji ga sadržava, lako se snalaziti razmišljajući o odnosima poput *djed* (dva elementa iznad) ili *unuk* (dva elementa ispod).



**API** je skraćenica od **Application Program Interface** i predstavlja definirane metode sučelja koje se mogu koristiti za rad s programom.

Vratimo li se na originalan ispravan primjer XML dokumenta, možete uočiti da su *unuci* elementa “kupci” zapravo elementi “proizvod” jer se nalaze dvije razine ispod njega. Isto tako, *roditelj* elementa “proizvod” je element “kupac”. Razmišljate li na taj način, vrlo lako ćete se snalaziti u XML strukturama.



Ukoliko vas zanima kompletna i detaljna specifikacija XML DOM-a, možete je pronaći na <http://www.w3.org/DOM>.

DOM API vam pruža metode za rad s XML dokumentima pa tako standardiziranim metodama možete pronaći korijenski element u XML dokumentu, popis svih elemenata određenog imena, popis svih podelemenata, tj. *djece* nekog elementa, vrijednost pojedinog atributa nekog elementa itd.

Osim dohvaćanja informacija iz XML dokumenta, korištenjem DOM API-ja možete i dodavati, mijenjati ili brisati elemente iz dokumenta, a možete i dodavati, mijenjati i brisati sve njihove atribute.

.NET u sebi ima ugrađen XML DOM API te ćemo njega koristiti u kasnijim primjerima baš za pokazivanje prethodno navedenih mogućnosti.



Uz DOM način rada s XML dokumentima postoji još jedan standard – SAX ili Simple API for XML. Razlika između DOM-a i SAX-a je u načinu njihova rada – dok DOM parseri koji obrađuju XML dokument učitavajući ga cijelog u memoriju računala (što može usporiti aplikaciju ili čak biti nemoguće za ekstremno velike XML datoteke), SAX pruža način slijednog čitanja sadržaja XML dokumenta bez potrebe da ga cijelog učitava u memoriju. Pritom se služi *callback* metodama koje se pozivaju pri određenim događajima, primjerice pri pojavi određenog tipa elementa. Iako SAX nije podržan u .NET-u, ipak postoji klasa *XmlReader* koja na sličan način obrađuje XML dokumente – slijednim čitanjem unaprijed.



## XPath

I dok DOM ima izvrsne mogućnosti dohvaćanja određenih dijelova XML dokumenta, ipak ima određena ograničenja za pretraživanje. No to nije problem – uz XML tehnologiju se veže i XPath (od XML Path), tehnologija za slobodno pretraživanje XML dokumenata.

XPath je vrlo sličan SQL upitima. Naravno, dok SQL upiti pretražuju baze podataka, XPath pretražuje XML dokumente i služi isključivo za dohvaćanje njegovih dijelova, dakle poput naredbe SELECT.

XPath upiti tako mogu sadržavati različite uvjete, koristiti logičke operatore, funkcije za rad sa znakovnim nizovima (i tako, primjerice, pronaći sve elemente koji u svom nazivu sadrže niz “abc”), aritmetičke operatore (primjerice, možete pronaći sve elemente čiji je zbroj neka dva atributa veći od 100) i slično.

Naredni XPath upit iz našeg će primjera XML dokumenta dohvatiti sve elemente imena “proizvod” čiji je atribut oznaka jednak “FE-B11”, ma gdje se oni nalazili u XML strukturi.

```
//proizvod[@oznaka='FE-B11']
```

Zanimaju li vas detalji XPath specifikacije odnosno nešto više od onoga što ćemo obraditi u ovom poglavlju, proučite dokumentaciju na web-stranici <http://www.w3.org/TR/xpath>.



## XSLT

Bavite li se izradom web-stranica, poznat vam je koncept po kojem se sadržaj web-stranica odvaja od informacija o njenom prikazu. Konkretno, za sadržaj web-stranica služe HTML datoteke, a u CSS datotekama se nalaze informacije o prikazu pojedinih dijelova sadržaja.

Na sličan način se može gledati i na odnos XML-a i XSLT-a – dok XML sadržava podatke, XSLT sadržava upute za njihovo oblikovanje i prikaz. XSLT (eXtensible Stylesheet Language Transformations) je dakle format za zapisivanje transformacija XML dokumenata. Korištenjem različitih XSL datoteka moguće je istu XML datoteku prikazati na potpuno drugačiji način.

Proučavajući XML dokumentacije često ćete naići na kraticu XSL (eXtensible Stylesheet Language). Radi se o kombinaciji triju W3C specifikacija – XPatha, XSLT-a i XSL-FO (XSL Formatting Objects).



### III. DIO: DIJELOVI .NET-A

## XML sheme

XML je izvrstan format za razmjenu podataka – vi podatke iz svoje aplikacije pretvorite u neki XML oblik i prosljedite ga autoru neke druge aplikacije na korištenje i izradu podrške za taj format. No podaci iz XML dokumenta nisu dovoljni za razumijevanje strukture XML dokumenta – tome služe XML sheme. One opisuju strukturu dokumenta i na temelju njih moguće je točno znati kakvu će XML strukturu imati neki dokument.

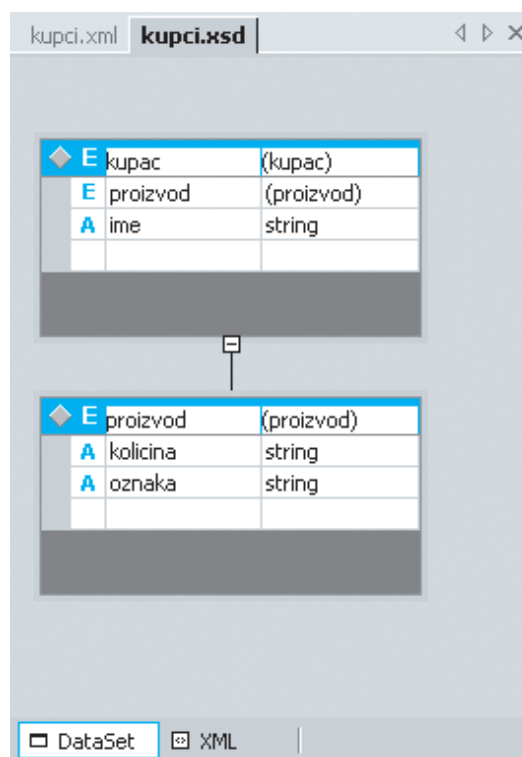


Preteča XML shema imala je ime DTD (Document Type Definition) i služila je za provjeru ispravnosti XML dokumenta, no ipak s manjim mogućnostima nego što imaju XML sheme. DTD-ovi se i danas još uvijek koriste, pa je podrška i za njih ugrađena u .NET.

Shemu odnosno opis nekog XML dokumenta možete izraditi automatski u Visual Studiju. Učitajte XML dokument i odaberite opciju *XML – Create Schema*.



**Slika 11-4:**  
**Stvorena shema za XML dokument**



Stvorite li tako shemu za naš XML primjer, dobit ćete XSD datoteku sljedećeg sadržaja, a njen grafički prikaz možete vidjeti na slici 11-4.

```
<?xml version="1.0"?>
<xs:schema id="kupci" targetNamespace="http://tempuri.org/kupci1.xsd"
xmlns:mstns="http://tempuri.org/kupci1.xsd" xmlns="http://tempuri.org/kupci1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-
com:xml-msdata" attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="kupci" msdata:IsDataSet="true" msdata:Locale="hr-HR"
    msdata:EnforceConstraints="False">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="kupac">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="proizvod" nillable="true" minOccurs="0"
                maxOccurs="unbounded">
                <xs:complexType>
                  <xs:simpleContent msdata:ColumnName="proizvod_Text"
                    msdata:Ordinal="2">
                    <xs:extension base="xs:string">
                      <xs:attribute name="kolicina" form="unqualified"
                        type="xs:string" />
                      <xs:attribute name="oznaka" form="unqualified"
                        type="xs:string" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="ime" form="unqualified" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Obratite pozornost na bitne elemente – s *xs:element* su označeni zasebni elementi u XML strukturi, a za svaki od njih su namještene i druge njegove postavke. Kad se dogovorite oko sheme XML

### III. DIO: DIJELOVI .NET-A

dokumenta, više ne može biti zabune – lako možete provjeriti odgovara li neki XML dokument zadanoj shemi i tako možete biti sigurni da će svi programi za rad s određenim tipom XML dokumenta raditi sa svim XML datotekama koje slijede zadanu shemu.



Detaljne specifikacije XML shema možete pronaći na web-adresi <http://www.w3.org/XML/Schema>.

## Podrška XML-u u .NET-u

Kao što ste vidjeli, sve dosad opisane tehnologije i formate standardizirao je W3C. Dakle, ništa od toga nije isključivo Microsoftovo – svoje znanje o XML-u, XPathu, XSLT-u ili XML shemama možete primijeniti na bilo kojoj platformi i u bilo kojem programskom jeziku.

U .NET-u postoji ugrađena podrška za rad sa svim opisanim tehnologijama. Kao i sve ostalo, ona je sadržana u nizu klasa, čiji popis možete vidjeti u tablici 11-1.

Kao što vidite, većina klasa organizirana je ispod *System.Xml namespacea*, što olakšava snalaženje.

**Tablica 11-1:**

**Popis glavnih klasa u .NET-u za rad s XML tehnologijama**

Klasa	Opis
<code>System.Xml.XmlReader</code>	apstraktna klasa za čitanje XML dokumenata
<code>System.Xml.XmlTextReader</code>	klasa koja pruža najbrži način čitanja XML dokumenta; po funkcionalnosti je slična <i>DataReaderu</i> u ADO.NET-u jer pruža mogućnost čitanja samo unaprijed (ne možete se vraćati) i pomoću nje ne možete mijenjati sadržaj XML dokumenta; ova klasa nema mogućnost validiranja XML dokumenta
<code>System.Xml.XmlValidatingReader</code>	klasa za čitanje XML dokumenata koja ima mogućnost validiranja njegove ispravnosti prema XML shemama i DTD dokumentima
<code>System.Xml.XmlNodeReader</code>	čitač XML dokumenta za DOM elemente
<code>System.Xml.XmlWriter</code>	apstraktna klasa za pisanje XML dokumenata
<code>System.Xml.XmlTextWriter</code>	implementacija klase za pisanje XML dokumenata koja ima mogućnost stvaranja XML dokumenta samo unaprijed i predstavlja najbrži način pisanja XML dokumenata

Klasa	Opis
<code>System.Xml.XmlDocument</code>	klasa koja predstavlja XML dokument
<code>System.Xml.XmlDataDocument</code>	implementacija klase za rad s XML dokumentom koja pruža mogućnost pristupa podacima relacijskim načinom ili preko DOM-a
<code>System.Xml.XPath.XPathDocument</code>	klasa za XML dokument optimizirana za zadavanje XPath upita
<code>System.Xml.XPath.XPathNavigator</code>	omogućava XPath upravljanje XML dokumentom
<code>System.Xml.Schema.XmlSchema</code>	klasa za XSD shemu XML dokumenta
<code>System.Xml.Xsl.XslTransform</code>	klasa za XSL transformacije

## Rad s XML-om

Naučiti raditi s XML-om u .NET-u najlakše je na temelju primjera. Stoga ćemo pokazati kako napisati kôd koji bi stvorio XML dokument te dodavao, mijenjao i brisao elemente. Kako XML možete koristiti u svim tipovima aplikacija, naredni kôd neće biti vezan niti uz jednu vrstu aplikacija, već ćete ga moći koristiti i u prozorskim aplikacijama, web-aplikacijama ili web-servisima.

Osim glavnih klasa prikazanih u tablici 11-1, u *namespaceu* `System.Xml` postoji još niz klasa za rad s XML dokumentima. Njih ćemo koristiti u narednim primjerima, a pobrojane su u tablici 11-2.

**Tablica 11-2:**  
**Klase za rad s dijelovima XML dokumenta**

Klasa	Opis
<code>XmlNode</code>	jedan čvor (element, komentar, CDATA dio, itd.) u XML dokumentu
<code>XmlNodeList</code>	lista objekata <code>XmlNode</code> ; kretanje po XML dokumentu zasniva se na prolazanju kroz stablo <code>XmlNodeList</code> , jer svaki XML element može sadržavati sve svoje podelemente (koji se opet daju predstaviti novim objektom <code>XmlNodeList</code> )
<code>XmlDocument</code>	XML dokument; ova klasa izvedena je iz klase <code>XmlNode</code> , jer se i na dokument može gledati kao na jedan XML tag koji sadržava svoje podelemente
<code>XmlElement</code>	element u XML dokumentu
<code>XmlAttribute</code>	atribut elementa u XML dokumentu

### III. DIO: DIJELOVI .NET-A

Napravit ćemo dakle implementaciju kôda koji će s nekim XML dokumentom raditi kao s bazom podataka. Kako XML dokumenti služe prvenstveno za spremanje nekih podataka, ponekad je lakše iskoristiti njih nego stvarati i koristiti bazu. To će biti slučaj kad ne radite s pretjerano velikom količinom podataka ili jednostavno radite aplikaciju u kojoj ne želite koristiti bazu podataka (ili nemate mogućnost njenog korištenja).



**Pretjerano korištenje baze podataka za svaku sitnicu može uvelike usporiti rad aplikacije. Radi li se o web-aplikaciji koja koristi jednu bazu podataka, velik broj istovremenih korisnika može otežati i potpuno onemogućiti rad s takvom aplikacijom koja se za sve oslanja na bazu. Stoga je preporučljivo manje podatke spremati u XML datotekama.**

Radit ćemo s XML dokumentom koji će služiti za spremanje informacija o korisnicima aplikacije. Primjerice, uz svakog korisnika možete spremiti njegove postavke i druge informacije o njegovu načinu korištenja aplikacije, što vam može biti korisno.

Za početak, u tu ćemo konfiguracijsku datoteku spremati informacije o svim korisnicima koji su se prijavili za rad u aplikaciji. Pritom nećemo imati naglasak na sigurnosti – korisnici će upisati svoje ime, osnovne podatke, i to ćemo spremiti u XML datoteku. Kad se vrate aplikaciji, ponovno će upisati svoje ime, a program će automatski učitati njihove postavke.

Ovo je osnovna ideja rada aplikacije – mi nećemo napraviti pravu aplikaciju, već ćemo pokazati kako napisati kôd koji bi radio prethodne zadatke s XML datotekom. Evo i njenog oblika – dobro ga pogledajte, jer ćemo takvu datoteku stvarati i mijenjati:

```
<?xml version="1.0" encoding="utf-8" ?>
<Korisnici>
  <Korisnik Ime="Marko">
    <PunoIme>Marko Marić</PunoIme>
    <ZadnjaPrijava>10.5.2004 15:35:40</ZadnjaPrijava>
    <Postavke>
      <MaksimiziranProzor>True</MaksimiziranProzor>
      <Tema>Crna</Tema>
    </Postavke>
  </Korisnik>
  <Korisnik Ime="Petar">
    <PunoIme>Petar Petrović</PunoIme>
    <ZadnjaPrijava>10.5.2004 12:19:59</ZadnjaPrijava>
```

```

        <Postavke>
            <MaksimiziranProzor>False</MaksimiziranProzor>
            <Tema>Plava</Tema>
        </Postavke>
    </Korisnik>
</Korisnici>

```

## Stvaranje XML dokumenta

Prvi korak u radu s XML dokumentom je njegovo stvaranje. Primjerice, ukoliko on ne postoji pri pokretanju aplikacije, stvorit će se novi XML dokument. Naredni kôd ne samo da stvara novi dokument već mu namješta XML deklaraciju (prvi `<?xml ...?>` tag) i glavni korijenski element (“Korisnici”).

Pri radu s XML dokumentima ne zaboravite uključiti *namespace System.Xml* korištenjem ključne riječi *using*.



Slijedi metoda za stvaranje novog XML dokumenta.

```

private XmlDocument NoviXmlDokument()
{
    XmlDocument xml = new XmlDocument();

    XmlDeclaration dec = xml.CreateXmlDeclaration("1.0", "utf-8", null);
    xml.PrependChild(dec);
    XmlElement korijen = xml.CreateElement("Korisnici");
    xml.AppendChild(korijen);

    return xml;
}

```

Dakle, prvo se stvara novi objekt tipa *XmlDocument*. Potom se stvara XML deklaracija naredbom *CreateXmlDeclaration*, koja prima tri parametra – prvi predstavlja verziju korištenog XML standarda, druga *encoding* (koristimo univerzalan UTF-8 koji će bez problema prikazati naše znakove na

### III. DIO: DIJELOVI .NET-A

bilo kojem računalu), a treći prikazuje je li dokument samostalan (tzv. “standalone”), što nam za ovaj slučaj nije važno, pa smo prosljedili vrijednost *null*.

Korištenjem *PrependChild* naredbe dodajemo napravljenu deklaraciju XML dokumentu. *PrependChild* umeće element prije prvog elementa. Iako njega u tom trenutku još nismo stvorili, važno je znati da će ta naredba dodati element na sam početak zadane XML strukture.

Potom stvaramo korijenski element XML dokumenta. Za stvaranje elemenata u XML-u koristimo naredbu *CreateElement* kojoj prosljeđujemo naziv elementa. No samim stvaranjem elementa nismo ga dodali u strukturu XML dokumenta, već to moramo eksplicitno napraviti naredbom *AppendChild* koja dodaje novi element.

Prethodnu metodu možemo pozivati iz bilo kojeg dijela kôda:

```
XmlDocument mojXmlDokument = NoviXmlDokument();
```

Želimo li ispisati kompletan XML kôd dokumenta (primjerice, zbog provjere), iskoristili bismo *OuterXml* svojstvo XML dokumenta koje vraća njegov kompletan XML kôd, uključujući sve podelemente.



Spomenuli smo da se na cijeli XML dokument može gledati kao na jedan XML element sa svojim podelementima. To znači i da ćemo *OuterXml* svojstvo moći iskoristiti za ispis kompletnog XML kôda pojedinog elementa u strukturi, što će ispisati i sve njegove podelemente.

Primjerice, radite li prozorske aplikacije, kôd načinjenog XML dokumenta najjednostavnije biste ispisali:

```
MessageBox.Show(mojXmlDokument.OuterXml, "Provjera");
```

Rezultat poziva gornje metode bio bi tako sljedeći XML kôd:

```
<?xml version="1.0" encoding="utf-8"?>
<Korisnici/>
```

Kao što vidite, radi se o jednostavnom XML dokumentu koji sadrži deklaraciju i prazan korijenski element imena “Korisnici”.

## Dodavanje elemenata

Sad kad imate gotov i spreman XML dokument, možete u njega dodavati različite elemente. Evo kako biste dodali informacije za pojedinog korisnika – njegovo korisničko ime, puno ime, datum



## 11. POGLAVLJE: XML

**Umjesto ručnog generiranja svakog elementa i njegova ubacivanja u XML strukturu novog dokumenta, mogli ste jednostavno učitati XML kôd korištenjem *loadXml* metode. Primjerice, sljedeći kôd rezultirao bi istim dokumentom kao i prethodni primjer:**

```
XmlDocument xml = new XmlDocument();
xml.LoadXml("<?xml version=\"1.0\"
encoding=\"utf-8\"?><Korisnici/>");
```



prijave te postavke – je li pri zadnjem korištenju aplikacije imao maksimiziran prozor i koju je temu odnosno sučelje aplikacije odabrao.

Za dodavanje novog korisnika rabi se metoda *DodajKorisnika*, a koristimo i dodatnu metodu *DodajTekstElement* koja prima nekoliko parametara – prvi određuje kojem XML dokumentu treba stvoriti novi element, drugi unutar kojeg postojećeg elementa treba dodati novi, a treći i četvrti određuju ime i vrijednost novog elementa. Primjerice, proslijedimo li za posljednja dva parametra vrijednosti “MaksimiziranProzor” i “True”, stvorit će se novi element `<MaksimiziranProzor>True</MaksimiziranProzor>`.

```
private void DodajKorisnika(XmlDocument xml, string ime, string punoime, bool max-
prozor, string tema)
{
    XmlElement glavniElement = xml.DocumentElement;

    XmlElement Korisnik = xml.CreateElement("Korisnik");
    Korisnik.SetAttribute("Ime", ime);
    glavniElement.AppendChild(Korisnik);

    DodajTekstElement(xml, Korisnik, "PunoIme", punoime);
    DodajTekstElement(xml, Korisnik, "ZadnjaPrijava", DateTime.Now.ToString());

    XmlElement Postavke = xml.CreateElement("Postavke");
    Korisnik.AppendChild(Postavke);

    DodajTekstElement(xml, Postavke, "MaksimiziranProzor", maxprozor.ToString());
    DodajTekstElement(xml, Postavke, "Tema", tema);
}
```

### III. DIO: DIJELOVI .NET-A

```
private void DodajTekstElement(XmlDocument xml, XmlElement roditelj, string ime,
string vrijednost)
{
    XmlElement noviElement = xml.CreateElement(ime);
    XmlText sadrzaj = xml.CreateTextNode(vrijednost);
    noviElement.AppendChild(sadrzaj);

    roditelj.AppendChild(noviElement);
}
```

Dakle, metoda *DodajKorisnika* prima kao prvi parametar XML dokument unutar kojeg treba dodati strukturu za pojedinog korisnika. Drugi parametar predstavlja korisničko ime, treći puno ime, četvrti logičku vrijednost je li njegov prozor maksimiziran, a peti odabranu temu.

Kako ćemo strukturu *Korisnik* dodati ispod glavnog elementa, u varijablu imena *glavniElement* učitavamo referencu na glavni element u XML dokumentu. Ispod njega ćemo dodavati nove elemente.

To odmah radimo u sljedećim naredbama – stvaramo novi element naredbom *CreateElement* imena “Korisnik” i spremamo ga u varijablu *Korisnik*. Njemu dodajemo jedan atribut imena “Ime” i u njega upisujemo korisničko ime. To obavljamo uz pomoć naredbe *SetAttribute* koju pozivamo za neki element, u ovom slučaju novi element imena “Korisnik”. Potom korištenjem već spomenute naredbe *AppendChild* dodajemo element “Korisnik” ispod glavnog elementa.

U ovom trenutku će struktura XML dokumenta imati sljedeći oblik:

```
<?xml version="1.0" encoding="utf-8"?>
<Korisnici>
    <Korisnik Ime="neko_ime"/>
</Korisnici>
```



**Primijetite i da se tekstualni sadržaj nekog elementa smatra njegovim *djetetom*. U metodi *DodajTekstElement* stvaramo tekstualni sadržaj, a potom ga naredbom *AppendChild* dodajemo već postojećem elementu.**

No sad u igru ulazi već opisana metoda *DodajTekstElement*. Njen je zadatak, dakle, da ispod nekog već postojećeg elementa u XML strukturi doda novi, tekstualni, oblika `<Ime>Vrijednost</Ime>`. Za dodavanje tekstualnog sadržaja pod neki element koristi se *XmlText* varijabla i naredba *CreateTextNode* koja dodaje sadržaj unutar nekog elementa.

Na kraju, u metodi *DodajTekstElement* dodajemo novi element s tekstualnim sadržajem ispod nekog postojećeg, koji je proslijeđen kao drugi parametar metodi.

I sad je sve jasno – u glavnoj metodi *DodajKorisnika* pozivamo metodu *DodajTekstElement* za svaki novi tekstualni element. Prva dva dodajemo ispod elementa “Korisnik”, a nose imena “PunoIme” i “ZadnjaPrijava” (čiju vrijednost računamo iz trenutnog vremena, da dokument dobije na aktualnosti), a druga dva dodajemo ispod novog elementa “Postavke”, a oni pak nose imena “MaksimiziranProzor” i “Tema”.

Opisanu metodu *DodajTekstElement* možemo pozvati na standardan način i dodati korisnika postojećem XML dokumentu, primjerice:

```
DodajKorisnika(mojXmlDokument, "Marko", "Marko Marić", true, "Crna");
```

Rezultat poziva takve metode je sljedeći XML dokument:

```
<?xml version="1.0" encoding="utf-8"?>
<Korisnici>
  <Korisnik Ime="Marko">
    <PunoIme>Marko Marić</PunoIme>
    <ZadnjaPrijava>15.5.2004 17:53:38</ZadnjaPrijava>
    <Postavke>
      <MaksimiziranProzor>True</MaksimiziranProzor>
      <Tema>Crna</Tema>
    </Postavke>
  </Korisnik>
</Korisnici>
```

Kako ćete u radu s XML dokumentima najviše vremena provesti u radu s *XmlElement* objektima, u tablici 11-3 nalazi se popis njihovih korisnih svojstava.

Naravno, za dodavanje elemenata, njihovo brisanje i sve druge operacije ključne su metode *XmlElement* objekta, a neke važnije su navedene u tablici 11-4.

III. DIO: DIJELOVI .NET-A

**Tablica 11-3:**  
**Neka korisna svojstva *XmlElement* objekata**

Svojstvo	Opis
Attributes	vraća kolekciju tipa <i>XmlAttributeCollection</i> koja sadrži sve atribute elementa
ChildNodes	vraća sve podelemente
FirstChild	vraća prvi podelement
HasAttributes	vraća logičku vrijednost, ima li element neke atribute
HasChildNodes	vraća logičku vrijednost, ima li element <i>djecu</i> (odnosno sadrži li podelemente)
InnerText	tekstualni sadržaj elementa
InnerXml XML	sadržaj elementa (dakle, samo sva njegova djeca, bez XML kôda samog elementa)
LastChild	vraća posljednji podelement Name vraća ime elementa
NextSibling	vraća sljedeći element u hijerarhiji (tzv. <i>bratski element</i> , jer se nalaze na istoj razini hijerarhije)
OuterXml	vanjski XML sadržaj elementa (uključuje XML kôd samog elementa i sve njegove djece)
ParentNode	vraća nadelement trenutnog (tzv. <i>roditelj</i> )
PreviousSibling	vraća prethodni element u hijerarhiji (tzv. <i>bratski element</i> , jer se nalaze na istoj razini hijerarhije)

Proširenja DOM-a

Microsoft je ipak proširio standardne definirane mogućnosti DOM API-ja u klasi *XmlDocument*. Tako vam na raspolaganju stoji svojstvo *InnerText*, pomoću kojeg možete definirati tekst unutar nekog elementa. Primjerice, metodu *DodajTekstElement* mogli ste napisati na sljedeći način:

```
private void
DodajTekstElement(XmlDocument xml,
XmlElement roditelj, string ime,
string vrijednost)
{
XmlElement noviElement =
```

```
xml.CreateElement(ime);
noviElement.InnerText =
vrijednost;
roditelj.AppendChild(noviElement);
}
```

Kao što vidite, radi se o pojednostavljenju i svojevrsnom unaprjeđenju mogućnosti rada s XML dokumentima. Korištenjem *InnerText* svojstva direktno smo upisali tekstualnu vrijednost nekom elementu, a nismo morali stvarati novi element tipa *XmlText* naredbom *CreateTextNode* te ga s *AppendChild* dodavati.

**Tablica 11-4:**  
**Neke korisne metode *XmlElement* objekata**

Metoda	Opis
<b>AppendChild</b>	dodaje novi element unutar postojećeg (novi element se dodaje na kraj odnosno poslije svih postojećih podelemenata)
<b>GetAttribute</b>	vraća vrijednost zadanog atributa
<b>GetElementsByTagName</b>	vraća sve podelemente zadanog imena
<b>HasAttribute</b>	vraća logičku vrijednost, ima li element zadani atribut
<b>PrependChild</b>	dodaje novi element unutar postojećeg (novi element se dodaje na, odnosno prije svih postojećih podelemenata)
<b>RemoveAttribute</b>	briše zadani atribut elementa
<b>RemoveChild</b>	briše zadani podelement
<b>ReplaceChild</b>	zamjenjuje zadani podelement novim elementom
<b>SelectNodes</b>	vraća listu elemenata dohvaćenu XPath upitom
<b>SelectSingleNode</b>	vraća prvi element dohvaćen XPath upitom
<b>SetAttribute</b>	Postavlja vrijednost atributu

## Traženje elemenata

Kao što u svom programu želite upisivati podatke o novim korisnicima, tako ih vrlo vjerojatno želite i mijenjati. Iako je naredne primjere moguće izraditi i uz pomoć XPatha (što bi bilo brže i efikasnije), to ćemo ipak ostaviti za kasnije, a sad ćemo se pozabaviti korištenjem DOM-a za rad s XML dokumentom.

Za početak će na trebati metoda koja će pronaći strukturu korisnika odnosno odgovarajući element “Korisnik”, koji se nalazi ispod elementa “Korisnici”. Kako smo na početku programa načiniili pretpostavku da korisničko ime jedinstveno određuje nekog korisnika (npr. ne mogu postojati dva korisnika s korisničkim imenom “Marko”), tražit ćemo element “Korisnik” koji ima zadano korisničko ime.

Evo kako bi izgledala ta metoda:

```
private XmlElement PronadjiKorisnika(XmlDocument xml, string ime)
{
```

### III. DIO: DIJELOVI .NET-A

```

XmlElement pronadjeniKorisnik = null;
XmlElement glavniElement = xml.DocumentElement;

XmlNodeList listaKorisnika = glavniElement.GetElementsByTagName("Korisnik");
foreach (XmlNode korisnik in listaKorisnika)
{
    if (korisnik is XmlElement)
    {
        XmlElement elementKorisnik = (XmlElement) korisnik;
        if (ime == elementKorisnik.GetAttribute("Ime"))
        {
            pronadjeniKorisnik = elementKorisnik;
            break;
        }
    }
}
return pronadjeniKorisnik;
}

```

Iako smo mogli koristiti XPath, cilj je bio pokazati korištenje *XmlNodeList* odnosno liste *XmlNode* objekata u strukturi XML dokumenta.

Dakle, metoda *PronadjiKorisnika* pretražuje XML dokument koji joj je proslijeđen kao prvi parametar u potrazi za korisnikom s korisničkim imenom koje je zadano kao drugi parametar. Ona vraća pronađeni element.

U njoj najprije određujemo varijablu *pronadjeniKorisnik* u koju će biti spremljena referenca na element u XML strukturi i postavljamo je na *null* jer ga još nismo našli. U varijablu *glavniElement* učitavamo korijenski element XML strukture.

Nad njim izvršavamo naredbu *GetElementsByTagName* koja vraća listu svih čvorova sa zadanim imenom (u našem slučaju "Korisnik"). Podsjetimo, *XmlNode* objekt može biti običan element, može biti neki atribut ili bilo koji drugi dio XML dokumenta. Zatim u *foreach*-petlji prolazi kroz sve dohvaćene zapise sa zadanim imenom. Kako lista dohvaćenih zapisa sadržava *XmlNode* objekte, tako ih sve u petlji spremamo u varijablu *korisnik* tog tipa.

No kako tražimo samo obične XML elemente (dakle, ne bi nam odgovarali, primjerice, atributi s tim imenom), u samoj petlji radimo provjeru je li objekt *korisnik* zapravo tipa *XmlElement*. Ukoliko je to XML element, eksplicitno ga pretvaramo iz *XmlNode* tipa u očekivani *XmlElement* jer ćemo tako moći pristupati njegovim atributima (sjetimo se petog poglavlja – radi se o *boxingu* i *unboxingu*).

Potom vršimo usporedbu – ukoliko parametar *ime* odgovara atributu “ime” dohvaćenog elementa, pronašli smo ono što smo tražili. U varijablu *pronadjeniKorisnik* spremamo referencu na pronađeni element te prekidamo petlju i vraćamo varijablu kao rezultat metode.

**U slučaju da ne pronađemo korisnika sa zadanim korisničkim imenom, metoda bi vratila null vrijednost jer je to početna vrijednost varijable *pronadjeniKorisnik*.**



Tu metodu možemo pozvati iz glavnog dijela programa. Evo kompletnog izvornog kôda:

```
XmlDocument mojXmlDokument = NoviXmlDokument();
DodajKorisnika(mojXmlDokument, "Marko", "Marko Marić", true, "Crna");

XmlElement k = PronadjiKorisnika(mojXmlDokument, "Marko");
if (k != null) MessageBox.Show(k.OuterXml, "Provjera");
```

U dokumentu *mojXmlDokument* tražimo korisnika s imenom “Marko”. Ako on ne bude pronađen (što ipak neće biti slučaj jer ga dodajemo jednu naredbu ranije), poruka se neće ispisati – objekt *k* u kojem je spremljen rezultat poziva bit će *null* i neće se izvršiti naredba *MessageBox.Show*. Naravno, ako se pronađe odgovarajući korisnik, ispisuje se samo XML kôd unutar pronađenog elementa imena “Korisnici”. Evo kako taj kôd izgleda:

```
<Korisnik Ime="Marko">
  <PunoIme>Marko Marić</PunoIme>
  <ZadnjaPrijava>15.5.2004 17:53:38</ZadnjaPrijava>
  <Postavke>
    <MaksimiziranProzor>True</MaksimiziranProzor>
    <Tema>Crna</Tema>
  </Postavke>
</Korisnik>
```

Uočite da smo ispisali *OuterXml* svojstvo nekog elementa i tako dobili kompletan njegov XML kôd, uključujući i podelemente.

## Mijenjanje elemenata

Nakon što dohvatite neki element prethodno opisanom metodom, vrlo lako možete mijenjati njegove podelemente ili atribute. Evo kako bi izgledala metoda koja je izmijenila puno ime korisnika i

### III. DIO: DIJELOVI .NET-A

datum njegove prije, tj. osvježila ga na ažurniju vrijednost (zbog jednostavnosti nećemo mijenjati ostale elemente, no princip je isti). I ovaj put ćemo, da pokažemo mogućnosti rada s DOM-om, koristiti što više različitih metoda i svojstava objekata za rad s XML-om.

```
private void IzmijeniKorisnika(XmlDocument xml, string ime, string novoPunoIme)
{
    XmlElement korisnik = PronadjiKorisnika(xml, ime);
    if (korisnik != null)
    {
        XmlNodeList listaElemenata = korisnik.ChildNodes;
        for (int i = 0; i < listaElemenata.Count; i++)
        {
            if (listaElemenata.Item(i) is XmlElement)
            {
                XmlElement element = (XmlElement) listaElemenata.Item(i);
                if (element.Name == "PunoIme")
                {
                    XmlText noviSadrzaj = xml.CreateTextNode(novoPunoIme);
                    element.ReplaceChild(noviSadrzaj, element.FirstChild);
                    break;
                }
            }
        }
    }
}
```

Dakle, metodi proslijeđujemo XML dokument u kojem treba načiniti izmjenu, korisničko ime korisnika te njegovo novo puno ime koje treba upisati umjesto postojećeg. Prvo koristimo načinjenu metodu *PronadjiKorisnika*, koja vraća element korisnika sa zadanim korisničkim imenom.

U varijablu *listaElemenata* učitavamo listu svih podelemenata pronađenog korisnika – lista svih podelemenata dostupna nam je preko svojstva *ChildNodes*. Potom koristimo običnu *for*-petlju za kretanje kroz sve dohvaćene podelemente (mogli smo koristiti i *foreach*-petlju, no čisto da pokažemo nešto novog kôda).

Ukoliko je neki podelement tipa *XmlElement* odnosno ako se radi o pravom elementu, eksplicitno ga pretvaramo i spremamo u varijablu *Element*. Zatim provjeravamo njeno ime pomoću svojstva *Name* – ukoliko je ono “PunoIme”, znači da smo pronašli element u kojem je spremljeno puno ime korisnika.

Tad stvaramo novi tekstualni element korištenjem *CreateTextNode* s varijablom koja sadrži novo puno ime. Korištenjem *ReplaceChild* metode zamjenjujemo jedan podelement drugim – kao prvi



parametar navodimo novi element, a kao drugi element koji želimo zamijeniti. Kako je tekstualni sadržaj jedini podelement elementa "PunoIme", koristimo *FirstChild* svojstvo i na njegovo mjesto stavljamo novi sadržaj.

I u ovom ste slučaju mogli iskoristiti *InnerText* svojstvo za zapis tekstualnog sadržaja elementa. Izmijenjeni dio kôda u tom bi slučaju imao sljedeći oblik:

```
if (element.Name == "PunoIme")
{
    element.InnerText = novoPunoIme;
    break;
}
```



## Brisanje elemenata

Važna stavka u radu s XML dokumentima je i brisanje elemenata u XML strukturi. Sve ćemo opet objasniti na primjeru brisanja korisnika. Slijedi metoda kojoj biste proslijedili korisničko ime korisnika, a ona bi pronađenu strukturu izbrisala:

```
private void BrisiKorisnika(XmlDocument xml, string ime)
{
    XmlElement korisnik = PronadjiKorisnika(xml, ime);
    if (korisnik != null)
    {
        XmlNode nadElement = korisnik.ParentNode;
        nadElement.RemoveChild(korisnik);
    }
}
```

Metoda je jednostavna – prvo pronalazi korisnika, zatim u varijablu *nadElement* učitava njegova roditelja korištenjem *ParentNode* svojstva. Potom korištenjem *RemoveChild* metode nad objektom nad-elementa, prosljeđujući joj referencu na element koji treba izbrisati, briše cijelu strukturu elementa "Korisnik", što uključuje i sve njegove podelemente.

## III. DIO: DIJELOVI .NET-A

## Učitavanje i spremanje XML dokumenata

Naravno, sve ove naredbe i način korištenja neće vam previše vrijediti ukoliko ne spremite sve promjene nad XML dokumentom u neku datoteku. Tu datoteku kasnije možete opet učitati i nastaviti gdje ste stali – tako ona dobiva funkciju svojevrzne konfiguracijske datoteke jer u njoj možete spremati različite postavke koje će biti čuvane na disku računala.

Za učitavanje i spremanje datoteka koriste se *Load* i *Save* metode XML dokumenta koje za parametar primaju naziv datoteke XML dokumenta. Primjerice, želite li na početku rada aplikacije učitati neku XML datoteku, napišite sljedeći kôd:

```
XmlDocument mojXmlDokument = NoviXmlDokument();
mojXmlDokument.Load("korisnici.xml");
```

Prethodni bi primjer tako učitao u XML dokument sadržaj datoteke "korisnici.xml" koja se nalazi u istom direktoriju kao i sama aplikacija. Naravno, možete navesti i punu putanju do XML datoteke, primjerice "C:\Programi\Aplikacija\korisnici.xml".



Osim učitavanja datoteka s lokalnog diska, imate mogućnost i učitavanja datoteka s Interneta. Primjerice, ako biste željeli u svom programu učitati XML datoteku smještenu na nekom poslužitelju na Internetu, samo biste trebali napisati njenu punu adresu:

```
mojXmlDokument.Load("http://www.server.com/aplikacija/korisnici.xml");
```

Pri dohvaćanju XML datoteka s Interneta, naravno, preporučljivo je ugraditi neki mehanizam obrade iznimki – što ako datoteka na tom serveru ne postoji ili što ako je server trenutno nedostupan? To su slučajevi za koje se ipak trebate pripremiti u svojem programu.

XML datoteke, se spremaju slično tome kako se učitavaju. Kao parametar *Save* metodi samo navedite naziv datoteke u koju želite spremiti XML strukturu. Takvu ćete akciju najčešće obavljati po završetku rada s programom, kako biste sačuvali sve izmjene.

```
mojXmlDokument.Save("korisnici.xml");
```

Korištenje XML datoteka za spremanje konfiguracijskih postavki aplikacije je dobra ideja, a to potvrđuje i ASP.NET koji u web.config datoteci sprema konfiguracijske postavke svake aplikacije.

## Pisanje XML dokumenata

Želite li stvoriti novi XML dokument i zapisati ga u neku datoteku, možete iskoristiti klasu *XmlTextWriter*. Ona služi za slijedno zapisivanje podataka i mnogo je brža nego da u memoriji stvarate novu strukturu XML dokumenata stvarajući poseban objekt za svaki element. U tablici 11-5 navedene su njene osnovne metode.

**Tablica 11-5:**  
**Osnovne metode za korištenje klase *XmlTextWriter***

Metoda	Opis
<code>WriteAttributes</code>	zapisuje sve atribute pronađene na trenutnoj poziciji u <i>XmlReaderu</i>
<code>WriteAttributeString</code>	zapisuje atribut s određenom vrijednošću
<code>WriteCData</code>	zapisuje <code>&lt;![CDATA[ ... ]]&gt;</code> blok sa zadanim tekstom
<code>WriteComment</code>	zapisuje komentar ( <code>&lt;!-- ... --&gt;</code> ) sa zadanim tekstom
<code>WriteElementString</code>	zapisuje element
<code>WriteStartAttribute</code> , <code>WriteEndAttribute</code>	otvara novi atribut odnosno zatvara atribut
<code>WriteStartDocument</code> , <code>WriteEndDocument</code>	otvara strukturu XML dokumenta odnosno zatvara strukturu dokumenta
<code>WriteStartElement</code> , <code>WriteEndElement</code>	otvara pisanje novog elementa odnosno zatvara pisanje elementa
<code>WriteRaw</code>	zapisuje čisti XML kôd napisan ručno

Da bismo pokazali način korištenja klase *XmlTextWriter*, napravit ćemo metodu koja će zapisati strukturu našeg oglednog XML dokumenta s jednim korisnikom u neku datoteku.

```
private void ZapisiXmlDokument(string datoteka)
{
    XmlTextWriter xml = new XmlTextWriter(datoteka, System.Text.Encoding.UTF8);
    xml.Formatting = Formatting.Indented;

    // Otvaramo strukturu XML dokumenta
    xml.WriteStartDocument();
}
```

### III. DIO: DIJELOVI .NET-A

```
// Zapisujemo komentar
xml.WriteComment("Automatski generirani XML dokument");

// Otvaramo element "Korisnici"
xml.WriteStartElement("Korisnici");

// Otvaramo element "Korisnik" i zapisujemo mu atribut
xml.WriteStartElement("Korisnik");
xml.WriteString("Ime", "Marko");

// Zapisujemo element "PunoIme"
xml.WriteStartElement("PunoIme");
xml.WriteString("Marko Marić");
xml.WriteEndElement();

// Zapisujemo element "ZadnjaPrijava"
xml.WriteStartElement("ZadnjaPrijava");
xml.WriteString(DateTime.Now.ToString());
xml.WriteEndElement();

// Zatvaramo element "Korisnik"
xml.WriteEndElement();

// Zatvaramo element "Korisnici"
xml.WriteEndElement();

// Zatvaramo strukturu XML dokumenta
xml.WriteEndDocument();

xml.Close();
}
```

Kôd metode je vrlo jasan – krećemo se naredbu po naredbu i upisujemo elemente u XML dokument. Ključno je otvaranje dokumenta – kao parametar konstruktoru klase *XmlTextWriter* proslijeđujemo naziv datoteke (koji metoda prima kao parametar) i *encoding* koji se nalazi u pobrojanom nizu *System.Text.Encoding* pa tako možete izabrati i ASCII *encoding* izlazne datoteke.

Postavljanjem svojstva *Formatting* određujemo kako će biti oblikovana ciljna datoteka – u pobrojanom nizu *Formatting* odabrali smo vrijednost *Indented*, što znači da će elementi u XML dokumentu biti uvučeni ovisno o njihovoj razini, što će rezultirati pregledno oblikovanim dokumentom.

Metodu iz primjera pozivamo na standardni način iz glavnog programa:

```
ZapisiXmlDokument("korisnici.xml");
```

Rezultat njenog poziva bit će nova datoteka sa zadanim imenom i upisanim XML sadržajem.



Kasnije je sve jasno iz komentara – prvo otvaramo strukturu XML dokumenta, zapisujemo komentar, zatim otvaramo element “Korisnici” i u njemu upisujemo novi element “Korisnik” s jednim atributom. Nastavljamo na isti način dalje – stvaramo elemente “PunoIme” i “ZadnjaPrijava”, a potom zatvaramo još nezatvorene strukture elementa “Korisnik”, “Korisnici” te cijele strukture XML dokumenta. Klasu *XmlTextWriter* zatvaramo s *Close()*, a nova datoteka na disku imat će sljedeći sadržaj:

```
<?xml version="1.0" encoding="utf-8"?>
<!--Automatski generirani XML dokument-->
<Korisnici>
  <Korisnik Ime="Marko">
    <PunoIme>Marko Marić</PunoIme>
    <ZadnjaPrijava>21.5.2004 11:53:11</ZadnjaPrijava>
  </Korisnik>
</Korisnici>
```

Korištenje klase *XmlTextWriter* preporučuje se kad želite nabrzinu zapisati neki sadržaj i stvoriti novu XML datoteku. Kao što je rečeno, takav pristup je mnogo brži nego stvaranjem svakog posebnog elementa, pozivanjem *AppendChild* metode i slično. Uvijek tako stvorenu datoteku možete i učitati u objekt tipa *XmlDocument* korištenjem metode *Load*.

## XPath

Zahvaljujući SQL jeziku za pristup podacima, baze podataka definitivno su otvorile svoja vrata svim znatiželjnicima. Korištenjem lako razumljivog SQL jezika, koji je zapravo industrijski standard koji sve baze više-manje doslovno implementiraju, korisnici imaju mogućnost dohvaćanja i obrade svih podataka spremljenih u bazi.

Kako se može reći da i XML format služi prvenstveno za pohranu nekih podataka, pojava standarda za pristup tim podacima bio je neupitan. Zahvaljujući W3C-u, on se 1999. godine pojavio u obliku XPatha i predstavlja ključni element u bilo kakvom radu s XML dokumentima. Kad ga već uspoređujemo sa SQL-om, budimo potpuno precizni – dok SQL omogućava i mijenjanje i dodavanje podataka, XPath služi isključivo za dohvaćanje dijelova XML dokumenata.

### III. DIO: DIJELOVI .NET-A

## Osnove XPatha

XPath za pristup dijelovima XML dokumenata koristi sintaksu vrlo sličnu onoj na koju ste navikli pri radu s datotekama ili web-stranicama, primjerice direktorij/direktorij/datoteka. Naravno, u kontekstu XML dokumenata, ne radi se o direktorijima i datotekama, već o elementima odnosno *no-deovima* u hijerarhiji.

Da bismo potpuno objasnili mogućnosti XPatha, trebat ćemo i XML dokument koji ćemo koristiti u primjerima. Radi se o dokumentu čiju ste strukturu upoznali već u prethodnim primjerima, no ponovit ćemo ga još jednom da lakše možete pratiti XPath upite:

```
<?xml version="1.0" encoding="utf-8" ?>
<Korisnici>
  <Korisnik Ime="Marko">
    <PunoIme>Marko Marić</PunoIme>
    <ZadnjaPrijava>10.5.2004 15:35:40</ZadnjaPrijava>
    <Postavke>
      <MaksimiziranProzor>True</MaksimiziranProzor>
      <Tema>Crna</Tema>
    </Postavke>
  </Korisnik>
  <Korisnik Ime="Petar">
    <PunoIme>Petar Petrović</PunoIme>
    <ZadnjaPrijava>10.5.2004 12:19:59</ZadnjaPrijava>
    <Postavke>
      <MaksimiziranProzor>False</MaksimiziranProzor>
      <Tema>Plava</Tema>
    </Postavke>
  </Korisnik>
</Korisnici>
```

## Kretanje po strukturi

Dakle, cilj XPath upita je odabrati jedan ili više elemenata ili njihovih atributa. Doista je mnogo kriterija po kojima možete vršiti selekciju, no krenimo s najjednostavnijim. Slijede tri upita:

```
/Korisnici
/Korisnici/Korisnik
/Korisnici/Korisnik/PunoIme
```

Upiti se rade tako da se napiše struktura koja se želi označiti, a XPath će pronaći sve elemente koji odgovaraju zadanoj strukturi. Prvi upit će, kao što možete i očekivati, označiti glavni korijenski element XML dokumenta, imena “Korisnici”. Drugi će pak upit označiti sve elemente imena “Korisnik” koji se nalaze ispod elementa “Korisnici”. U našem oglednom XML dokumentu postoje dva takva elementa. Slična je situacija i s trećim upitom koji označava sve elemente “PunoIme” koji se nalaze ispod jednog od elemenata “Korisnik”, koji se pak nalaze ispod elementa “Korisnici”. Tim trećim upitom dohvatili biste puna imena svih korisnika iz XML dokumenta.

Uočite da svi ti upiti počinju sa *slash* znakom (“/”) – on označava da se radi o apsolutnom putu, tj. onom koji počinje na vrhu hijerarhijske XML strukture. No stavite li na početak upita dva *slash* znaka, pretraživat ćete cijelu XML hijerarhiju za traženim elementom.

```
//Tema
```

Gornji će primjer tako za rezultat dobiti sve elemente “Tema” koji se nalaze bilo gdje u hijerarhijskoj strukturi.

Osim određivanja točnog puta za odabir elemenata, možete iskoristiti jedan ili više *wildcard* znakova (\*) za označavanje svih mogućnosti, kao što je vidljivo u narednim primjerima:

```
/Korisnici/Korisnik/*  
/Korisnici/*/PunoIme  
/*/*/PunoIme
```

U prvom primjeru odabrani su svi elementi koji se nalaze ispod hijerarhije /Korisnici/Korisnik, a to su svi elementi “PunoIme”, “ZadnjaPrijava”, “Postavke” u dokumentu. Drugi primjer pak označava sve elemente “PunoIme”, koji se nalaze u drugoj razini ispod elementa “Korisnici” (primjerice, označio bi i elemente /Korisnici/nesto\_treće/PunoIme da postoje). Treći primjer pak označava sve elemente imena “PunoIme” koji imaju dva pretka bilo kojeg imena u hijerarhijskoj strukturi.

**Napišete li “/\*”, označit ćete sve elemente u dokumentu jer, podsjetimo, dva *slash* znaka označavaju da se pretražuju svi elementi bilo gdje unutar strukture, a zvjezdica da upitu odgovaraju elementi bilo kojeg imena.**



U prethodnim smo primjerima označavali sve elemente koji bi odgovarali zadanom upitu. Ponekad ćete željeti odabrati samo neke od njih. Pogledajte sljedeće primjere:

### III. DIO: DIJELOVI .NET-A

```
/Korisnici/Korisnik[1]
/Korisnici/Korisnik[last()]
/Korisnici/Korisnik[PunoIme='Marko Marić']
```

Unutar uglatih zagrada možete napisati dodatne uvjete. Dakle, prvi upit će označiti prvi element “Korisnik” koji se nalazi ispod elementa “Korisnici”. Drugi upit će pak označiti zadnji element “Korisnik” koji se nalazi ispod elementa “Korisnici”, no primijetite da ne postoji funkcija poput *first()* koja bi vratila prvi element – želite li to postići, samo napišite [1], kao u prvom primjeru. Treći primjer pak označava element “Korisnik” koji sadrži element “PunoIme”, koji pak ima vrijednost “Marko Marić”. Rezultat tog upita bio bi prvi element “Korisnik”.

## Atributi elemenata

Osim po imenu elemenata, možete pretraživati i po njihovim atributima. Krenimo opet s primjerima:

```
//@Ime
//Korisnik[@Ime='Marko']
//Korisnik[@*]
```

U XML-u na attribute elemenata se gleda baš kao i na elemente – oni predstavljaju zasebne čvorove i također se mogu označiti i iz njih izvući podaci. Prvi primjer tako označava sve attribute “Ime” u XML dokumentu te tako dobivate sva korisnička imena u dokumentu, u konkretnom primjeru dva – “Marko” i “Petar”.

Drugi primjer označava sve elemente “Korisnik” kojima je atribut “Ime” jednak “Marko” odnosno korisnika s korisničkim imenom “Marko”. Treći pak primjer označava sve elemente “Korisnik” koji u sebi sadržavaju bilo koji atribut. Uočite da se u svim primjerima koriste dva *slash* znaka koji određuju pretraživanje svuda unutar XML hijerarhije.

## Usporedbe i odnosi

Osim dosad prikazanog operatora jednakosti (“=”), u XPathu možete koristiti i niz drugih operatora. Želite li tako provjeriti nejednakost, iskoristit ćete “!=”, tj. *not equal* operator. Naravno, dostupne su vam i druge usporedbe, primjerice *manje-od* (“<”), *manje-ili-jednako* (“<=”), *veće-od* (“>”) i *veće-ili-jednako* (“>=”).

Nad vrijednostima (primjerice, atributa) u XML dokumentu možete obavljati i aritmetičke operacije – zbrajanje (“+”), oduzimanje (“-”), množenje (“\*”), dijeljenje (“div”) i računanje ostatka dijeljenja ili modul (“mod”). Ako vam to nije dovoljno, možete koristiti i logičke operatore AND (“and”) i OR (“or”).

Osim što uz pomoć XPatha možete označivati elemente u odnosu na cijelu XML hijerarhiju (tj. apsolutno), to možete raditi i relativno, tj. u odnosu na trenutno označeni element. Pri korištenju XML



dokumenata u svom kôdu, radit ćete s različitim elementima unutar hijerarhije. Nakon što učitate XML dokument, bit ćete pozicionirani na glavni element u hijerarhiji i u odnosu na njega ćete obavljati pretraživanja. No vi vrlo lako možete označiti neki drugi element i zatim nad njim obavljati XPath upite koji će označavati sve njegove podelemente, baš kao da je on glavni element u hijerarhiji.

U tablici 11-6 možete vidjeti nekoliko ključnih riječi koje vam omogućavaju relativno označavanje elementa, tj. u ovisnosti o njihovu odnosu.

**Tablica 11-6:**

**Nekoliko ključnih riječi koje će vam koristiti kad budete označavali XML elemente u odnosu na trenutno označeni**

Metoda	Opis
<code>ancestor::</code>	svi prethodnici trenutnog elementa – njegov nadelement, zatim njegov nadelement pa tako sve do glavnog elementa u hijerarhiji
<code>ancestor-or-self::</code>	trenutni element i svi njegovi prethodnici
<code>attribute::</code>	svi atributi trenutnog elementa
<code>child::</code>	djeca odnosno podelementi trenutnog elementa
<code>descendant::</code>	suprotno od <i>ancestors</i> – svi podelementi trenutnog elementa (djeca trenutnog, zatim njihova djeca itd.)
<code>descendant-or-self::</code>	trenutni element i svi njegovi <i>nasljednici</i>
<code>following::</code>	svi sljedeći elementi u hijerarhiji (ne uključuje <i>descendant</i> elemente od trenutnog)
<code>following-sibling::</code>	svi sljedeći elementi koji se nalaze ispod istog elementa kao i trenutni ( <i>braća</i> )
<code>parent::</code>	nadelement ( <i>roditelj</i> ) trenutnog elementa <code>preceding::</code> svi prethodni elementi u hijerarhiji
<code>preceding-sibling::</code>	svi prethodni elementi koji se nalaze ispod istog elementa kao i trenutno ( <i>braća</i> )
<code>self::</code>	trenutni element

Slijedi nekoliko primjera koji će zorno prikazati njihovu uporabu:

```

/Korisnici/Korisnik[2]/preceding::*
/Korisnici/child::Korisnik[position()<3]
child::Korisnik[position()<3]

```

### III. DIO: DIJELOVI .NET-A

Prvi XPath upit kreće od drugog elementa “Korisnik” te zatim označava sve njegove prethodnike u XML hijerarhiji, a to su prvi element “Korisnik” i sva njegova djeca. Za objašnjavanje drugog i trećeg upita potrebno je pojasniti čemu služi funkcija *position()* – ona vraća poziciju nekog elementa unutar nadelementa. Tako će drugi upit vratiti sve podelemente od “Korisnici” imena “Korisnik”, i to samo prva dva, jer su njihove pozicije manje od 3 (naravno, u našem dokumentu postoje samo dva, no u slučaju da ih ima više, taj upit bi vratio samo prva dva).

Treći upit obavlja istu stvar, no u ovisnosti o trenutno odabranom elementu – ukoliko ga izvršite nad elementom “Korisnici”, dobit ćete isti rezultat kao i u prethodnom primjeru.

## Kratice

**N**e trebate uvijek koristiti ključne riječi poput *child::* ili *parent::*. Za njih postoje već objašnjene kratice. Primjerice, napišete li “*child::tekst*”, to je isto kao da ste napisali i samo “*tekst*” jer će XPath uvijek krenuti s pretraživanjem podelemenata.

Isto tako, napišete li samo “*.*”, to je isto kao da ste napisali i *self::* odnosno označili trenutni element. Primjerice, “*./tekst*” je identično “*self::no-*

*de()/child::tekst*”. Slično vrijedi i za *parent::* koji je jednak “*..*” pa su tako sljedeća dva izraza jednaka: “*../tekst*” i “*parent::node()/child::-tekst*”.

I na kraju, pretraživanje svih XML elemenata u hijerarhiji s dva *slash* znaka moguće je izvesti i s *descendant-or-self::*, primjerice “*//tekst*” je isto što i “*/descendant-or-self::node()/child::cd*”.

## Rad s XPathom

U .NET-u s XPathom možete raditi preko klase *XmlNode* nekog objekta *XmlDocument* i preko klase *XPathNavigator*. Kao što ste mogli vidjeti u tablici 11-4, na raspolaganju vam stoje dvije metode – *SelectNodes* i *SelectSingleNode* – koje koriste XPath upite za pretraživanje dokumenta.

Primjerice, želite li ispisati sve vrijednosti elementa “PunoIme” odnosno dohvatiti imena svih korisnika iz XML dokumenta, možete iskoristiti sljedeći kôd (pretpostavka je da i dalje koristite XML dokument kao u prvim primjerima te da je on učitao u varijablu *mojXmlDokument*):

```
XmlNodeList imena = mojXmlDokument.SelectNodes("//PunoIme");
foreach (XmlNode ime in imena)
{
```

```

        MessageBox.Show(ime.InnerText, "Ime");
    }

```

Prethodni kôd je već jasan – dohvaćate listu elemenata, baš kao što ste radili u prethodnim primjerima s *ChildNodes* ili *GetElementsByTagName*. Ta se lista sprema u objekt tipa *XmlNodeList*, a zatim koristi *foreach*-petlja za kretanje kroz sve dohvaćene elemente. Potom se samo ispisuje njihov sadržaj u prozoru za poruke – ta ste imena mogli učitati u neki padajući izbornik ili bilo koju drugu kontrolu.

Na sličan način radi i *SelectSingleNode* metoda koja, kao što je rečeno, vraća samo jedan element. Evo kako bi izgledao primjer u kojem dohvaćamo puno ime korisnika s korisničkim imenom “Marko”.

```

XmlNode ime =
    mojXmlDokument.SelectSingleNode("/Korisnici/Korisnik[@Ime='Marko']/PunoIme");

MessageBox.Show(ime.InnerText, "Ime");

```

**Ako XPath upit proslijeđen *SelectSingleNode* metodi vrati više elemenata, svi će se zane-mariti osim prvog elementa, koji će biti dohvaćen kao rezultat poziva.**



No kompletan API u .NET-u za rad s XPath upitima je izgrađen oko klase *XPathNavigator*. Pomoću nje i klase *XPathDocument* ćemo u narednom primjeru učitati listu svih imena korisnika.

U klasu *XPathDocument* učitat ćemo XML dokument nad kojim ćemo zadavati upite. Korištenjem klase *XPathNavigator* zadat ćemo upit koji će vratiti određeni set rezultata, a uz pomoć klase *XPathNodeIterator* ćemo se kretati kroz dohvaćene elemente. Pogledajte primjer:

```

XPathDocument xml = new XPathDocument("korisnici.xml");
XPathNavigator nav = xml.CreateNavigator();
XPathNodeIterator iter = nav.Select("/Korisnici/Korisnik/PunoIme");

while (iter.MoveNext())
{
    MessageBox.Show(iter.Current.Value, "Ime");
}

```

### III. DIO: DIJELOVI .NET-A

Korištenjem klasa za rad s XPathom ubrzavate svoje aplikacije i preporučljivo ih je koristiti jer se one ionako koriste u pozadini poziva *SelectNode* naredbe. Dakle, u primjeru stvaramo novi *XPathDocument* objekt, a iz njega pozivom metode *CreateNavigator* stvaramo objekt tipa *XPathNavigator*. On će nam služiti za zadavanje XPath upita.

Objekt tipa *XPathNodeIterator* služi nam za kretanje kroz dohvaćene elemente – u njega učitavamo listu elemenata korištenjem objekta *XPathNavigator* i metode *Select* u kojoj navodimo XPath upit.

Korištenjem *while*-petlje krećemo se kroz sve elemente, a trenutnom elementu pristupamo svojom *Current*. Na kraju ispisujemo vrijednost svih elemenata korištenjem *Value* svojstva.



**Dohvatite li XPath upitom vrijednost nekog atributa, njega također možete ispisati *Value* svojstvom – ono je predviđeno da vrati vrijednost bilo kojeg dohvaćenog elementa, bio to običan XML element ili neki atribut.**

Vrlo često ćete i XPath upite koristiti da biste dohvatili neke elemente i kasnije s njima radili – mijenjali ih ili čak brisali. No ukoliko im pristupate preko *XPathNodeIterator* kolekcije, oni su *read-only*, što znači da samo možete pročitati njihovu vrijednost, a ne možete je mijenjati.

Naredni primjer stvara *XPathNavigator* iz običnog XML dokumenta tipa *XmlDocument* i pretvara dohvaćeni element u tip *XmlNode* – time dobivate punu kontrolu nad elementom i nad njim možete raditi sve preko DOM-a. Iskoristit ćemo to za promjenu svih korisničkih imena – umjesto običnih imena, dat ćemo im brojeve koji odgovaraju rednom broju elementa “Korisnik” u XML strukturi dokumenta.

```
XmlDocument mojXmlDokument = NoviXmlDokument();
mojXmlDokument.Load("korisnici.xml");

XPathNavigator nav = mojXmlDokument.CreateNavigator();
XPathNodeIterator iter = nav.Select("/Korisnici/Korisnik");

XmlNode element;

while (iter.MoveNext())
{
    element = ((IHasXmlNode) iter.Current).GetNode();
    element.Attributes["Ime"].Value = iter.CurrentPosition.ToString();
}
```

Dakle, objekt tipa *XPathNavigator* stvaramo iz objekta *XmlDocument* tipa pozivom standardne metode *CreateNavigator*, a zatim u *XPathNodeIterator* učitavamo sve elemente koji odgovaraju zadanom XPath upitu. U gornjem slučaju to su svi elementi “Korisnik”.

Za pristup pojedinom elementu koristit ćemo varijablu *element* tipa *XmlNode*. U petlji kojom se krećemo kroz listu dohvaćenih elemenata XPath upitom, u varijablu *element* učitavamo pojedini element. To obavljamo konverzijom trenutnog elementa u tip “IHasXmlNode”. Zahvaljujući toj konverziji možemo pozvati naredbu *GetNode* koja za trenutni element vraća njegov objekt tipa *XmlNode*.

Potom s dohvaćenim elementom možemo raditi što god želimo. U gornjem primjeru odlučili smo se na promjenu atributa “Ime”. Njega postavljamo na redni broj trenutnog elementa među dohvaćenima – prvi će korisnik tako imati korisničko ime “1”, drugi “2”, itd.

## XML serijalizacija

Kako je XML postao glavni format za razmjenu informacija, u .NET je ugrađena podrška za serijalizaciju i deserijalizaciju objekata u XML format. O čemu se zapravo radi? Serijalizacija vam daje mogućnost da bilo koji objekt spremite u XML obliku i, primjerice, spremite u neku datoteku na disku. Deserijalizacija vam pak omogućava da pročitate taj XML objekt i iz njega stvorite originalni objekt.

Za serijalizaciju i deserijalizaciju ključan je *System.Xml.Serialization namespace*. U njemu se nalaze metode za ispravno upravljanje različitim objektima i njihovu (de)serijalizaciju. Sve ćemo najbolje pokazati na primjeru objekta *DataSet* s kojim smo se družili u 9. poglavlju o ADO.NET-u.

Recimo da izrađujete aplikaciju koja omogućava uređivanje podataka iz baze. Sve obavljate pomoću *DataSet* objekta u kojem je spremljena kopija podataka. Po završetku uređivanja sve promjene spremate u bazu podataka. No što ako se desi nepredviđen slučaj – baza je nedostupna, trenutno se administrira ili je nedostupan poslužitelj u mreži na kojem se baza nalazi? Zar ćete dopustiti da sve promjene budu izgubljene ili ćete natjerati korisnika da pokušava spremiti podatke sve dok veza s bazom ne bude ispravna?

Rješenje ove situacije je jednostavno i nudi vam se u obliku XML serijalizacije. Po završetku rada aplikacija će pokušati spremiti podatke u bazu – u slučaju da ne uspije, jednostavno će serijalizirati objekt *DataSet* u kojem se nalazi kopija podataka. Spremit će ga negdje na disk i po sljedećem pokretanju aplikacije će ga učitati i ponovno pokušati spremiti u bazu. Ako i tada ne uspije, nema problema – aplikacija može i dalje raditi s lokalnom kopijom podataka spremljenom u serijaliziranom XML dokumentu sve dok se ne uspostavi veza s bazom.

### III. DIO: DIJELOVI .NET-A



Pri radu s narednim primjerima ne zaboravite uključiti *System.Xml.Serialization* namespace:

```
using System.Xml.Serialization;
```

U *DataSet* objekt u varijabli *dataSet11* učitali smo podatke dohvaćene preko objekta *DataAdapter* jednostavnom SQL naredbom izvršenom nad bazom Northwind, s kojom smo se družili i u 9. poglavlju.

```
SELECT EmployeeID, LastName, FirstName, BirthDate FROM Employees
```

Radi se o jednostavnom upitu koji dohvaća osnovne podatke o zaposlenicima. Dakle, u svojoj aplikaciji ste napunili *DataSet*. Evo kako biste taj *DataSet* serijalizirali u neku datoteku na disku:

```
XmlSerializer ser = new XmlSerializer(dataSet11.GetType());
XmlTextWriter xml = new XmlTextWriter("podaci.xml", System.Text.Encoding.UTF8);

ser.Serialize(xml, dataSet11);

xml.Close();
```

Prvo smo stvorili objekt tipa *XmlSerializer* i namjestili ga da radi s *DataSet* objektima. To smo učinili prosljediivši mu kao parametar u konstruktoru tip objekta koji će trebati serijalizirati, a njega smo pak dohvatili pozivom metode *GetType* nad objektom s kojim radimo.

Potom smo stvorili objekt za zapisivanje XML dokumenata tipa *XmlTextWriter* – kao prvi parametar zadali smo mu datoteku u koju će zapisati, a kao drugi *encoding*. Potom smo pozvali metodu *Serialize* objekta *XmlSerializer* – kao prvi parametar smo joj zadali *XmlTextWriter* koji određuje gdje će se zapisati podaci, a kao drugi sam objekt koji treba serijalizirati.

Na kraju zatvaramo *XmlTextWriter* i podaci se zapisuju u datoteku. Na slici 11-5 možete vidjeti strukturu generirane XML datoteke prikazane u Internet Exploreru, a posebnu pozornost obratite na XML shemu podataka koja se nalazi na početku dokumenta.

Naravno, sama serijalizacija vam ne bi previše vrijedila bez mogućnosti deserijalizacije. Ona je, u slučaju *DataSet* objekata, mnogo jednostavnija. Primjerice, imate li u prozorskoj aplikaciji *DataGrid* u kojem se prikazuju podaci i u njemu želite prikazati podatke iz *DataSeta*, evo kako biste to učinili:

```
dataSet11.ReadXml("podaci.xml");
dataGrid1.DataSource = dataSet11;
dataGrid1.DataMember = "Employees";
```



### III. DIO: DIJELOVI .NET-A

```

public string Ime
{
    get { return this._Ime; }
    set { this._Ime = value; }
}

private string _Prezime = "";
[XmlIgnore]
public string Prezime
{
    get { return this._Prezime; }
    set { this._Prezime = value; }
}

public MojaKlasa()
{
}

public MojaKlasa(int MojBroj)
{
    Broj = MojBroj;
}
}

```

Dakle, radi se o klasi *MojaKlasa* koja ima cjelobrojnu varijablu *Broj*, znakovni niz *Ime* (realiziran preko metoda *get* i *set* i privatne varijable *\_Ime*), znakovni niz *Prezime* (također realiziran preko metoda *get* i *set* i privatne varijable *\_Prezime*). U klasi su definirana i dva konstruktora – jedan *defaultni* i jedan koji prima cjelobrojni parametar i upisuje ga u varijablu *Broj*.



Klasa koju serijalizirate obavezno mora imati definiran *defaultni* konstruktor, tj. onaj koji ne prima niti jedan parametar jer se on koristi u procesu serijalizacije odnosno deserijalizacije. Takav konstruktor je definiran i u gornjem primjeru – ne sadržava nikakav kôd i nema funkcionalnost, ali mora postojati.

Unutar klase možete vidjeti i niz direktiva koje određuju ponašanje elemenata pri serijalizaciji, a one su objašnjene u tablici 11-7.



**Tablica 11-7:****Direktive koje određuju ponašanje elemenata pri serijalizaciji**

Direktiva	Opis
<code>XmlAttribute</code>	član označen ovom direktivom bit će serijaliziran kao atribut u XML dokumentu
<code>XmlElement</code>	član označen ovom direktivom bit će serijaliziran kao element u XML dokument ( <i>defaultno</i> ponašanje)
<code>XmlIgnore</code>	član označen ovom direktivom ignorirat će se pri serijalizaciji i neće biti uključen u generirani XML dokument
<code>XmlRoot</code>	opisuje korijenski element XML dokumenta (direktiva je primjenjiva isključivo na klasu)

Sad kad znate što koja direktiva iz tablice 11-7 znači, lako je razumjeti opis klase. Tako će objekt te klase biti serijaliziran u XML objekt koji ima korijenski element imena “MojaKlasaXML”. Varijabla *Broj* će, zbog direktive *XmlAttribute*, biti stavljena kao atribut u glavni element. Isto tako, varijabla *Prezime* će se ignorirati jer je ispred nje postavljena direktiva *XmlIgnore*.

Serijalizacija objekta neke klase ne uključuje članove koji nisu označeni s *public* te metode. Tako se ne morate brinuti da će se, primjerice, serijalizirati privatna varijabla *\_Ime*, a metode se ionako ne mogu serijalizirati jer ne sadržavaju nikakve podatke.



Objekt klase *MojaKlasa* serijalizirate na sličan način kao i *DataSet* objekte – pogledajte primjer:

```
MojaKlasa obj = new MojaKlasa(15);
obj.Ime = "Marko";
obj.Prezime = "Marić";

XmlSerializer ser = new XmlSerializer(obj.GetType());
XmlTextWriter xml = new XmlTextWriter("podaci.xml", System.Text.Encoding.UTF8);

ser.Serialize(xml, obj);
xml.Close();
```

### III. DIO: DIJELOVI .NET-A

Na početku stvaramo objekt i preko konstruktora mu vrijednost varijable *Broj* postavljamo na 15. Također, postavljamo mu vrijednosti za varijable *Ime* i *Prezime*. Ostatak kôda je isti kao i kod serijalizacije *DataSeta*: stvara se novi *XmlSerializer* objekt, zatim *XmlTextWriter* koji određuje gdje će se spremiti serijalizirani objekt, te se poziva metoda *Serialize*.

Pogledamo li nakon izvršavanja prethodnog kôda u datoteku "podaci.xml", ona će imati sljedeći sadržaj:

```
<?xml version="1.0" encoding="utf-8" ?>
<MojaKlasaXML xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" Broj="15">
  <Ime>Marko</Ime>
</MojaKlasaXML>
```

Kao što smo i očekivali, objekt se serijalizirao u XML dokument s korijenskim elementom imena "MojaKlasaXML", a kao njegov atribut je postavljena i vrijednost varijable *Broj*. Unutar tijela XML dokumenta nalazi se element *Ime*, koji sadržava vrijednost varijable *Ime*, no nema vrijednosti varijable *Prezime*, jer je kod njene definicije u klasi postavljena direktiva *XmlIgnore*.

Deserijalizacija ovakvog objekta ide na malo drugačiji način nego kôd objekta *DataSet*, jer vaš objekt nema metodu *ReadXml*. No, na svu sreću, on uopće nije kompliciran – želite li u neki objekt klase *MojaKlasa* učitati podatke iz XML datoteke, evo kako biste to učinili.

```
MojaKlasa novi = new MojaKlasa();
XmlSerializer deser = new XmlSerializer(novi.GetType());
XmlTextReader x = new XmlTextReader("podaci.xml");
novi = (MojaKlasa) deser.Deserialize(x);

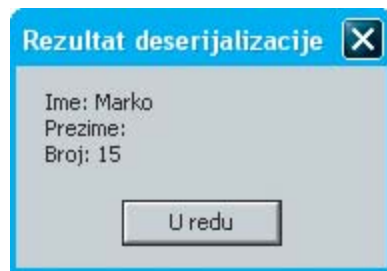
MessageBox.Show("Ime: " + novi.Ime + "\nPrezime: " + novi.Prezime + "\nBroj: " +
novi.Broj.ToString(), "Rezultat deserijalizacije");
```

Prvo stvaramo novi objekt klase *MojaKlasa* i pritom koristimo *defaultni* konstruktor, tj. onaj bez parametara, jer želimo da novi objekt bude *prazan*, bez postavljenih vrijednosti. Zatim inicijaliziramo objekt tipa *XmlSerializer*, baš kao u procesu serijalizacije.

No za razliku od serijalizacije kad smo koristili *XmlTextWriter*, jer smo zapisivali XML dokument, u procesu deserijalizacije koristimo *XmlTextReader*, jer čitamo sadržaj nekog XML dokumenta. Potom pozivamo metodu *Deserialize* koja kao parametar prima objekt iz kojeg čitamo XML dokument i dohvaćeni deserijalizirani objekt pretvaramo u objekt tipa *MojaKlasa* jednostavnim *castanjem* u taj tip.

No znakovito je ispisivanje informacija o deserijaliziranom objektu – kao što je vidljivo na slici 11-6, nije dohvaćena vrijednost za varijablu *Prezime*, što je logično, jer ona nije ni bila serijalizirana.

## 11. POGLAVLJE: XML



**Slika 11-6:**  
***Ispis informacija o deserijaliziranom objektu***

Kao što ste mogli vidjeti u ovom poglavlju, XML vam uvelike može olakšati izradu aplikacija. Zahvaljujući mogućnostima serijalizacije objekata u XML, imate otvorene mogućnosti spremanja objekata na lokalni disk za kasniju uporabu. U narednom poglavlju čekaju vas web-servisi koji se u potpunosti temelje na XML-u i SOAP-u, posebnoj gramatici XML-a.

