Cuprins

Metoda `appendChild` poate insera mai multe noduri simultan dacă primeşte ca parametru un obiect de tipul `XMLList` (evident, nodurile vor fi elementele listei).

Copilul nou adăugat obiectului `xmlObj` va fi aşezat pe ultima poziţie.

Pentru a adăuga copii pe primele poziţii, în locul comenzii `appendChild` utilizaţi metoda `prependChild`.

Acestea nu sunt singurele posibilităţi. Pentru a insera noul nod după un anumit nod deja existent, se poate utiliza metoda `insertChildAfter` care are ca parametru două obiecte de tip XML: primul este nodul după care trebuie făcută inserarea, al doilea este nodul care trebuie inserat. De exemplu, pentru a insera nodul reprezentând noul prieten între cele două noduri deja existente în XML ar trebui ca, în locul metodei `appendChild`, să se utilizeze:

```
xmlObj.insertChildAfter( xmlObj.children()[0],prietenNou);
```

Metoda `insertChildBefore` are tot doi parametri şi inserează obiectul reprezentat de cel de-al doilea parametru înaintea nodului identificat prin primul parametru.

Metoda cea mai simplă pentru inserarea unui nou nod este, totuşi, prin adăugarea utilizând operatorul punct. Pentru aceasta, scrieţi numele instanţei obiectului, urmată de punct şi de numele marcatorului ce va fi adăugat. Folosiţi, apoi, operatorul egal pentru a atribui noului nod un obiect XML:

```
xmlObj.prieten = prietenNou;
```

Pentru a afla poziţia pe care se află un anumit nod (de exemplu, pentru a o utiliza împreună cu comanda `children`), se poate apela metoda `childIndex`. Pentru exemplul de mai sus, se va afişa pe ce poziţie se află nodul „`data_nastere`" în cadrul primului prieten existent în XML:

```
trace( xmlObj.children()[0].data_nastere.childIndex() );
```

Copierea unui obiect `XML` se va realiza cu metoda `copy`. Metoda `copy` nu necesită niciun argument, va returna un obiect XML identic cu cel care a apelat metoda, aflat în schimb la altă locaţie de memorie. Atunci când selectaţi un anumit nod şi operaţi o modificare asupra nodului, modificarea se reflectă supra XML-ului original (încercaţi să selectaţi un prieten din lista de prieteni şi să-i adăugaţi încă un prenume şi veţi vedea că modificarea se reflectă şi în XML-ul original). Dacă utilizaţi metoda `copy`, pentru a copia în prealabil nodul într-un nou obiect, `XML`-ul rezultat va fi unul total nou şi orice modificare făcută asupra sa nu va influenţa `XML`-ul din care a fost extras:

```
// vom clona primul nod într-un nou obiect XML:
var prieten : XML = xmlObj.children()[0].copy();

// clonei îi vom adăuga încă un element:
prieten.insertChildAfter( prieten.child("nume"),
                          <prenume>Cristi</prenume> );

// putem observa că originalul a rămas neschimbat:
trace( prieten.toXMLString() );
trace( "--------" );

// chiar dacă în clonă a apărut noul nume:
trace( xmlObj.toXMLString() );
```

Se poate observa că în cel de-al doilea element, primul nod de tip prieten din listă rămâne neschimbat.

Clasa `XML` nu furnizează o anumită metodă pentru eliminarea unui nod din arborele XML. Totuşi, în AS3 există o metodă generică „`delete`" care elimină orice legătură dintre numele variabilei şi locul din memorie în care este stocat obiectul. Obiectul în sine nu va mai putea referit şi, din acest motiv, poate fi considerat şters. În momentul în care un obiect nu mai poate fi referit, el este trimis automat către *garbage-collector* (colectorul de gunoaie) care are rolul de a-l elimina din memorie (pentru a lăsa memoria liberă altor viitoare obiecte). Pentru a elimina primul nod din `xmlObj` putem să utilizăm o construcţie de tipul:

```
delete xmlObj.children()[0];
trace( xmlObj );
```

Metoda `replace` are rolul de a înlocui un nod cu unul nou. Fie două obiecte: `xmlObj` şi `prietenNou`. Metoda `replace` va fi apelată pentru obiectul în care se va face înlocuirea având ca parametri poziţia în XML a nodului ce trebuie înlocuit şi noul nod. De exemplu, dacă se doreşte înlocuirea primului nod-prieten (identificat ca fiind pe poziţia 0) cu `prietenNou`, se va scrie:

```
xmlObj.replace( 0 , prietenNou );
```

Atunci când obiectul XML conţine mai multe noduri de tip text consecutive, ele pot fi concatenate într-un singur nod. De asemenea, un element care nu conţine niciun fel de informaţie poate să nu fie scris ca `<element></element>` şi direct `<element/>`.

Atunci când apare unul din cele două cazuri de mai sus, este bine ca XML-ul să fie normalizat (va ocupa mai puţin spaţiu în memorie şi interpretorul îl va putea parcurge mai uşor). Normalizarea unui obiect de tip XML se realizează cu ajutorul metodei `normalize`. Această metodă nu

are niciun parametru de intrare, nu returnează nimic, dar modifică în memorie obiectul XML apelant astfel încât să poată fi parcurs mai rapid. De exemplu:

```
var xml:XML =
   <test>
       <nod_vid></nod_vid>
   </test>

xml.appendChild("text1");
xml.appendChild("text2");
xml.appendChild("text3");
trace( xml );
trace("Inainte de normalizare:",xml.children().length());

xml.normalize();
trace( xml );
trace( "Dupa normalizare: ", xml.children().length() );
```

Datorită adăugării nodurilor „text1", „text2", „text3", înainte de normalizare, obiectul XML are patru copii (deşi ultimii trei sunt doar noduri textuale, fără marcatori şi nu ar trebui să se facă diferenţa între ei). După normalizare, cele trei noduri textuale sunt unite într-unul singur şi elementul vid va fi afişat ca `<nod_vid />` (de către `trace( xml );`).

Un conţinut simplu este fie un nod textual, fie unul de tip element vid (chiar cu atribute). Un nod are conţinut complex în toate celelalte cazuri. În cazul agendei, primul prieten (luat în ansamblu) formează un nod de tip complex, nodul care conţine doar numele său este un nod de tip simplu. Pentru a testa dacă un nod are conţinut simplu se va folosi `hasSimpleContent`, în mod identic se va apela `hasComplexContent` pentru a afla dacă un nod este complex. Cele două metode nu au niciun parametru şi returnează o valoare logică în funcţie de conţinutul nodului apelant:

```
// pentru primul nod (prieten):
trace( xmlObj.children()[0].hasSimpleContent() ); //false
trace( xmlObj.children()[0].hasComplexContent() ); //true

// pentru nodul nume din primul nod prieten:
trace(xmlObj.children()[0].children()[0].
                              hasSimpleContent()); // true
```

Atribute

Un atribut este o construcţie de tipul `nume="valoare"` şi poate fi adăugat în marcatorul de început al unui nod-element. Rolul unui atribut este de a

descrie proprietăţile elementelor conţinute de marcatorul în cauză. În cazul agendei, am adăugat, de fiecare dată, elementelor de tip prieten câte un atribut indicând sexul persoanei respective.

Pentru a prelua valorile tuturor atributelor existente într-un obiect de tip XML se va apela metoda `attributes` pentru respectivul obiect XML. Astfel, putem afişa valorile atributelor din primul nod prieten printr-o construcţie:

```
trace( xmlObj.children()[0].attributes() );
```

Metoda `attributes` returnează un obiect de tip `XMLList`. Comanda `trace` apelează metoda `toString` pentru a putea afişa conţinutul.

Pentru a prelua un anumit atribut (presupunând că sunt mai multe), se va utiliza metoda `atribute` care primeşte, ca argument, numele atributului şi returnează valoarea acestuia. De exemplu:

```
trace( xmlObj.children()[0].attribute("sex") ); // m
```

O variantă şi mai simplă de obţinere a unui atribut este utilizarea operatorului `@`:

```
trace( xmlObj.children()[0].@sex ); // m
```

Pentru a adăuga un atribut unui marcator se va construi o referinţă către acel marcator plecând din rădăcina XML-ului, după care se va utiliza operatorul punct (utilizat şi la adăugarea copiilor) urmat de caracterul `@` specific atributelor, numele atributului, operatorul de atribuire şi valoarea atributului. În continuare, se va adăuga atributul „`ani`" primului prieten din agendă:

```
xmlObj.children()[0].@ani = 30;
trace( xmlObj.children()[0].attribute("ani") ); // 30
```

Utilizarea spaţiilor de nume

Din exemplele pe care le-am dat până în acest moment, se poate observa că o aplicaţie „se mulează" pe structura XML-ului pe care trebuie să-l utilizeze. Dacă dorim să construim un program care să utilizeze agenda, acest program trebuie să folosească corect semnificaţia marcatorului `prieten` sau a datei de naştere a acestuia.

Unele documente XML sunt standardizate, tocmai pentru ca fiecare aplicaţie ce ar încerca să le acceseze să cunoască un format clar, în care va găsi datele. Mai mult, pentru a indica că aceste tipuri de documente XML sunt particularizate, pentru a deservi anumitor scopuri, au fost redenumite (extensiile lor nu sunt XML, ci SVG, RSS etc.) şi, în acelaşi timp, tipul lor

MIME[1] a fost schimbat (de exemplu: `application/xml+rss`). În acest mod, oricine va discuta despre SVG, de exemplu, va şti că acest format este special pentru reprezentarea elementelor grafice vectoriale (iar în cazul în care este un cunoscător al acestui format, ar putea enumera marcatorii ce pot fi utilizaţi în construcţia unui SVG[2]). RSS este un alt format derivat din XML care permite sumarizarea conţinutului unui sit (RSS = *Really Simple Sindication*). Acestea nu sunt singurele formate derivate din XML. Chiar şi XML-ul din exemplul nostru (cel cu agenda) ar putea deveni un standard şi toată lumea să ştie în ce format trebuie să adauge informaţiile în el pentru a fi compatibil cu aplicaţia noastră. De fapt, există un format de specificare a prietenilor utilizând documente XML cunoscut sub numele FOAF (*Friend Of A Friend*).

Nu este de ajuns să denumim altfel un fişier XML pentru a-l transforma într-un SVG sau RSS. O aplicaţie ar putea interpreta un anumit marcator dintr-un fişier SVG într-un anumit fel, în timp ce altă aplicaţie ar putea interpreta acelaşi marcator în alt fel. De exemplu, marcatorul „`circle`" din SVG ar putea fi interpretat de o aplicaţie realizată în Anglia ca un cerc, iar de alta realizată în Japonia ca fiind un grup de oameni. Pentru a fi siguri că toate aplicaţiile „înţeleg" acelaşi marcaj în acelaşi fel şi pentru a evita ambiguităţile, se utilizează un aşa-zis spaţiu de nume (specific fiecărui tip particular de document XML). Pentru un fişier în format SVG, acest spaţiu de nume este specificat de adresa **http://www.w3.org/2000/svg**. La o astfel de adresă nu este neapărat să găsiţi informaţii despre formatul respectiv (aşa cum se întâmplă în cazul SVG-ului). Deoarece toată lumea foloseşte acelaşi spaţiu de nume (care este unic pentru SVG), SVG-ul este recunoscut de toată lumea în acelaşi fel. Un obiect de tip XML poate fi inserat într-un document SVG. Pentru a nu se confunda tag-urile din HTML cu cele din SVG, se poate defini spaţiul de nume caracteristic SVG şi preciza această definiţie de fiecare dată când este utilizat un marcator specific SVG. De exemplu (salvaţi într-un fişier cu extensia *svg*):

```
<html:html xmlns:html="http://www.w3.org/1999/xhtml"
           xmlns:svg="http://www.w3.org/2000/svg">
<html:body>
<html:p>Acesta este un paragraf HML</html:p>
  <svg:svg width="300" height="100" version="1.1">
```

1. Tipul MIME (*Multipurpose Internet Mail Extensions*) a fost utilizat pentru a identifica corect formatele de fişiere din cadrul unui e-mail. În prezent a fost extins şi la alte protocoale (HTTP, RTP). Detalii la adresa: http://en.wikipedia.org/wiki/MIME_type.
2. Mai multe despre SVG la: http://www.w3.org/Graphics/SVG/.

```
    <svg:circle cx="100" cy="50" r="40" stroke="black"
                stroke-width="2" fill="red" />
  </svg:svg>
</html:body>
</html:html>
```

În acest exemplu, putem identifica câteva marcaje specifice HTML: `html`, `body`, `p` şi câteva marcaje specifice SVG-ului: `svg`, `circle`. Pentru a nu se confunda între ele (pentru că nu fac parte din acelaşi limbaj), s-au definit spaţiile de nume pentru HTML şi pentru SVG (*xmlns* provine de la *XML Name Space*) şi de fiecare dată când se utilizează un marcator (specific SVG-ului spre exemplu), acesta va fi prefixat de spaţiul de nume corespunzător (*svg*) şi de caracterul două puncte. Pentru a putea vizualiza[1] elementele grafice definite în exemplul de mai sus, veţi avea nevoie să instalaţi un *player* de SVG: **http://www.adobe.com/svg/viewer/install/**.

Pentru a concluziona, numele de spaţii identifică tipul datelor care sunt conţinute în documentul XML. Spaţiul de nume poate fi definit ca un atribut din spaţiul de nume *xmlns* care are o anumită valoare – un URI unde este definit spaţiul de nume.

AS3 permite includerea spaţiilor de nume prin intermediul unui obiect special numit `Namespace` (care se află, de asemenea, în *„Top Level"*, deci nu trebuie importată nici o clasă specială). Singurele chestiuni importante dintr-un obiect de tip `Namespace` sunt cele două proprietăţi ale sale: `prefix` şi `uri`, ambele de tip `String`. Proprietatea `prefix` este cea care indică cu ce se va identifica spaţiul de nume (de exemplu, „svg" – cel definit în *xmlns* în exemplul de mai sus), proprietatea `uri` va conţine adresa la care se află definiţia spaţiului de nume.

Exemplu de declarare şi iniţializare a unui obiect de tip `Namespace`:

```
var xmlns:Namespace = new Namespace("soap",
                "http://www.w3.org/2001/12/soap-envelope");
```

Cu ajutorul comenzii `setNamespace` se poate asocia un spaţiu de nume unui anumit obiect XML:

```
var xmlns:Namespace = new Namespace("soap",
              "http://www.w3.org/2001/12/soap-envelope");

var xml:XML =
    <Envelope>
        <Body/>
    </Envelope>;
```

1. Unele browsere (ca, de exemplu, Firefox) au preinstalat un *player* (interpretor) de SVG.