

# XML în Perl

## Utilizarea procesorului Expat și a modelului DOM pentru manipularea documentelor XML în Perl

– Sabin Corneliu Buraga

În cadrul acestui articol vom urmări să ilustrăm câteva dintre facilitățile puse la dispoziție de modulele Perl pentru prelucrarea facilă a documentelor XML, folosind *SAX* (*Simple API for XML*) și *DOM* (*Document Object Model*).

### Modulele Perl

Înainte de a discuta despre cum prelucrăm documentele XML, este necesar să prezentăm pe scurt conceptele de pachet și de modul Perl.

Un *pachet* Perl poate fi considerat drept implementarea unei clase pe care o putem instanția în cadrul unui script. Subrutinele incluse într-un pachet pot juca, de asemenea, rolul de metode, existând posibilitatea definirii de constructori și destructori. Mai mult, se oferă suport pentru derivarea unei metode aparținând unui pachet, astfel încât pachetele Perl pot fi ierarhizate. Vom referi variabilele din alte pachete prefixând identificatorul variabilei respective cu numele pachetului urmat de „:”, după cum se poate observa din următorul exemplu:

```
$intrare = $main::STDIN;
```

La fel, pentru metode:

```
$imagine = new GD::Image(174, 333);
```

Dacă nu este specificat numele pachetului, se consideră implicit pachetul *main*. Astfel, construcția `$$var` este echivalentă cu `$main::var`. Dacă dorim să accesăm metode sau date-membru definite într-un pachet derivat din altul vom specifica numele ambelor pachete:

```
$Pachet::Subpachet::variabila
```

Un *modul* reprezintă un pachet public definit într-un fișier `.pm` cu scopul de a fi reutilizat ulterior. Modulele Perl vor fi incluse în program, spre a fi folosite, prin construcția:

```
use Modul;
```

Sunt puse la dispoziție mai multe module standard (disponibile în orice distribuție Perl actuală), dintre care se pot menționa:

- *Carp* (pentru controlul erorilor și avertismentelor);
- *Config* (pentru acces la opțiunile de configurare);
- *CGI* (pentru generarea facilă de scripturi CGI);
- *Env* (pentru accesarea variabilelor de mediu);
- *ExtUtils::Embed* (pentru includerea de cod Perl în programele C);
- *File::Find* (pentru traversarea recursivă a unui arbore de directoare);
- *File::Handle* (pentru manipularea fișierelor folosind descriptori de fișier);
- *File::Path* (pentru operații cu directoare);
- *Math::Complex* (pentru prelucrarea numerelor complexe);
- *POSIX* (pentru asigurarea interfeței cu standardul POSIX IEEE 1003.1);

- *Search::Dict* (pentru căutarea unei chei într-un fișier dicționar);
- *Socket* (pentru realizarea de operațiuni cu *socket*-uri);
- *Time::Local* (pentru acces la timpul local).

Pentru a găsi toate modulele instalate în sistem (inclusiv cele care nu au documentații sau au fost instalate în afara distribuției standard) putem folosi următoarea linie de comenzi:

```
find `perl -e 'print "@INC"'` -name "*.pm" -print
```

În mod normal, fiecare modul posedă propria lui documentație, accesibilă prin intermediul comenzii `man` din UNIX. Se poate utiliza și comanda `perldoc`. De reținut faptul că anumite module pot fi scrise în alte limbaje, în speță C (cazul modulelor *Socket* sau *POSIX*).

**CPAN** Succesul limbajului Perl rezidă, în principal, din posibilitatea de a extinde limbajul cu noi funcționalități oferite de module. În afara modulelor din distribuțiile Perl standard, există o colecție globală a tuturor materialelor publice referitoare la Perl, colecție referită sub denumirea *CPAN* (*Comprehensive Perl Archive Network*). CPAN oferă un număr impresionant de module grupate pe următoarele categorii:

- extensii de limbaj și unelte de documentare;
- suport pentru dezvoltare de programe/module;
- interfețe (la nivel scăzut sau ridicat) cu sistemul de operare;
- comunicarea între procese, în rețea și controlul dispozitivelor (*e.g.* modemuri);
- tipuri de date și conversii;
- interfețe cu bazele de date;
- interfețe cu utilizatorul;
- interfețe cu alte limbaje de programare;
- procesarea fișierelor și sistemelor de fișiere;
- procesarea caracterelor;
- procesarea fișierelor de configurație și a parametrilor în linia de comandă;
- suport pentru diverse limbi și alfabetice (internaționalizare);
- autentificare, securitate și criptare;
- suport pentru poșta electronică și grupurile de știri;
- suport pentru Web (HTML, HTTP, CGI, XML etc.);
- utilitare pentru daemoni;
- suport pentru arhivarea și compresia datelor;
- procesarea informațiilor grafice;
- controlul fluxului (excepții, erori etc.);
- procesarea fluxurilor de date și a fișierelor;
- altele.

Pentru un *listing* al tuturor locațiilor Internet referitoare la CPAN, consultați <http://www.perl.com/perl/>.

**Instalarea unui modul** În unele cazuri va trebui să luăm un modul de la CPAN pentru a-l instala și folosi ulterior în cadrul scripturilor noastre. Pentru a fi instalat pe un sistem UNIX/Linux, un modul Perl se regăsește fie ca fișier *tar* arhivat cu *gzip* (deci are extensia `.tar.gz` sau `.tgz`), fie ca fișier `.pm` (deja dezarhivat). Orice modul Perl necesită pentru instalare existența interpretorului Perl în sistem. După dezarhivare (cu `tar -xzf numearhiva.tgz`), în directorul în care a fost stocat

modulul dorit a fi instalat se dau următoarele comenzi (pentru a se putea executa ultima linie, utilizatorul trebuie să aibă drepturi de *root*):

```
perl Makefile.PL
make
make test
make install
```

În mod uzual, fiecare modul este acompaniat și de documentația necesară exploatării lui. Pentru a avea acces la ea, se folosește utilitarul `perldoc`:

```
(infoiasi)$ perldoc XML::Parser
```

Pentru convertirea documentației în format text sau HTML se pot utiliza comenzile `pod2text` și, respectiv, `pod2html`, ca în exemplul următor:

```
(infoiasi)$ pod2text Parser.pm >Parser.txt
(infoiasi)$ pod2html Parser.pm >Parser.html
```

### Prelucrarea documentelor XML

În continuare vom urmări să prelucrăm documentele XML via scrip-turile Perl, în vederea transformării lor în pagini Web.

**Utilizarea analizorului Expat** Una dintre cele mai facile modalități de a prelucra documentele XML este cea a utilizării analizorului *Expat* dezvoltat de James Clark, a cărui funcționalitate este încapsulată de modulul `XML::Parser`. Acest modul va pune la dispoziție obiectele `XML::Parser` și `XML::Parser::Expat`.

Analiza XML este bazată pe evenimente, fiecare tip de nod al arborelui asociat documentului XML declanșându-se un anumit eveniment care va trebui tratat de o rutină Perl specificată de programator. Astfel, după inițializarea analizorului, va trebui să folosim metoda `setHandlers` pentru a stabili ce funcții vor fi apelate pentru fiecare tip de eveniment.

Cele mai importante evenimentele generate de procesorul XML sunt:

- `Start` - indică apariția *tag*-ului de început al unui element;
- `End` - desemnează apariția *tag*-ului de sfârșit al unui element;
- `Char` - indică apariția conținutului text al unui element (caracterele de text neprocesat dintre *tag*-ul de început și cel de sfârșit);
- `Comment` - indică apariția unui comentariu.

Într-un prim exemplu de utilizare a modulului `XML::Parser` vom asocia pentru evenimentele `Start`, `End` și `Char` câte o subrutină care va fi apelată la fiecare apariție a evenimentului în cauză.

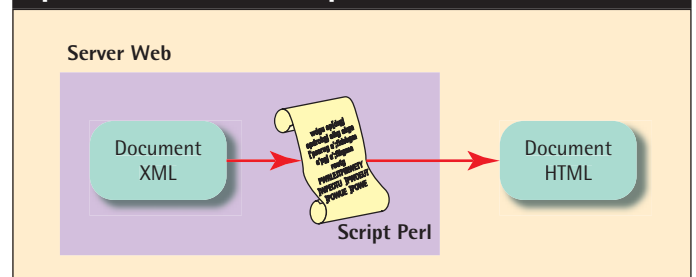
Documentul XML `biblio.xml` care urmează stochează informații despre împrumuturile dintr-o bibliotecă:

```
<?xml version="1.0" ?>
<imprumuturi>
  <imprumut>
    <carte autor="H. Hesse" an="1999">
      Lupul de steпа
    </carte>
    <client adresa="mituc@ac.tuiasi.ro">
      Victor Tarhon-0nu
    </client>
  </imprumut>
  <imprumut>
    <carte autor="H. Hesse" an="1998">
      Jocul cu margelele de sticla
    </carte>
    <client adresa="mihaela@infoiasi.ro">
      Mihaela Brut
    </client>
  </imprumut>
</imprumuturi>
```

Dorim să generăm un tabel XHTML cu aceste informații, prin transformarea documentului XML de mai sus. Vom scrie următorul script Perl, în care vom substitui fiecare element XML cu elementele XHTML corespunzătoare (aceste substituții vor fi stocate în tablouri asociative):

```
#!/usr/bin/perl
# utilizam modulul XML
use XML::Parser;
# definim tablourile hash de inlocuire a tag-urilor
# definim substitutiile de tag-uri de inceput
%start = (
  "imprumuturi" => "<table border='1'">",
  "imprumut"    => "<tr>",
  "carte"       => "<td><b>",
  "client"      => "<td align='center'">"
);
# definim substitutiile de tag-uri de sfirsit
%sfirsit = (
  "imprumuturi" => "</table>\n",
  "imprumut"    => "</tr>",
  "carte"       => "</b></td>",
  "client"      => "</td>"
);
# instantiem analizorul XML
my $parser = new XML::Parser(ErrorContext => 2);
# setam functiile de prelucrare
# a elementelor si continutului lor
$parser->setHandlers(
  Start => \&procesare_start,
  # functia de procesare tag-uri de inceput
  End   => \&procesare_sfirsit,
  # functia de procesare tag-uri de sfirsit
  Char  => \&procesare_continut
  # functia de procesare a continutului
);
# afisam antetul HTTP
print "Content-type: text/html\n\n";
# incarcam fisierul si il analizam
$parser->parsefile("biblio.xml");
# definim subrutinele pentru prelucrarea
# elementelor XML si continutului lor
sub procesare_start
{
  my $procesor = shift;
  # primul argument este instanta procesorului XML
  my $element = shift;
```

### Transformarea unui document XML în pagină Web prin intermediul unui script Perl



```
# al doilea argument este numele elementului
# corespunzator tag-ului de inceput

# afisam codul HTML, folosind tabloul hash
print $start{$element};
}

sub procesare_sfirsit
{
    my $procesor = shift;
    # primul argument este instanta procesorului XML
    my $element = shift;
    # al doilea argument este numele elementului
    # corespunzator tag-ului de sfirsit

    # afisam codul HTML, folosind tabloul hash
    print $sfirsit{$element};
}

# rutina de afisare a continutului

sub procesare_continut
{
    # am preluat argumentele furnizate
    my ($procesor, $data) = @_;

    # afisam datele
    print $data;
}

```

Funcțiile asociate evenimentelor de procesare XML vor primi drept argumente instanța procesorului Expat și numele elementului curent (pentru evenimentele Start și End) sau conținutul dintre tag-urile de început și de sfârșit (pentru evenimentul Char).

Ieșirea scriptului prezentat mai sus este:

```
Content-type: text/html
```

```
<table border="1">
  <tr>
    <td><b>
      Lupul de stepa
    </b></td>
    <td align="center">
      Victor Tarhon-Onu
    </td>
  </tr>
  <tr>
    <td><b>
      Jocul cu margelele de sticla
    </b></td>
    <td align="center">
      Mihaela Brut
    </td>
  </tr>
</table>
```

Analizorul Expat oferă programatorului funcționalități multiple, denumite *stiluri de procesare*. Stilul implicit a fost utilizat în scriptul precedent. Mai pot fi folosite:

- stilul de depanare Debug,
- stilul de analiză cu subrutine Subs în care fiecare apariție a unui tag distinct va fi tratată de o subrutină purtând același nume cu elementul corespunzător aceluși tag, iar pentru tratarea tag-urilor de sfârșit va

fi invocată o subrutină având numele elementului urmat de caracterul „\_” (vezi mai jos),

- stilul de analiză arborescentă Tree, în care procesorul va returna arborile de analiză al documentului XML, fiecare nod al arborelui fiind de forma (nume de tag, conținut),
- stilul de analiză cu obiecte Objects care este similar cu stilul Tree, dar se vor genera câte un obiect de tip *hash* pentru fiecare element.

Stabilirea stilului de procesare se va face la inițializarea analizorului, prin intermediul opțiunii Style. Constructorul respectiv va putea avea drept argumente și alte atribute de procesare precum:

- protocolul de codificare a documentului XML: ProtocolEncoding (pot fi specificate valori, e.g. UTF-8, ISO-8859-1 sau UTF-16);
- modalitatea de raportare a erorilor: ErrorContext a cărui valoare este numărul de linii care vor fi afișate la apariția unei erori de analiză XML (de obicei, se preferă valoarea 2);
- funcțiile asociate evenimentelor generate de procesorul Expat: Handlers (este un tablou asociativ având drept chei nume de evenimente și drept valori referințe la subrutinele de tratare a evenimentului respectiv);

În continuare vom utiliza stilul de procesare Subs pentru a putea prelucra comod și valorile atributelor unui element particular. Folosind *biblio.xml* ne propunem să afișăm atât titlul cărților împrumutate, cât și autorul și data apariției (acestea se regăsesc ca atribute ale elementului <carte>). Pentru fiecare element al documentului, va trebui să scriem o rutină de tratare a apariției tag-ului de început al acestuia. La fel, va trebui să concepem o rutină de tratare a fiecărei apariții a tag-ului de sfârșit. Tot în cadrul acestui exemplu vom vedea cum putem accesa valorile atributelor unui element, prin utilizarea unui tablou asociativ.

Codul sursă al scriptului este:

```
#!/usr/bin/perl
# utilizam modulul XML
use XML::Parser;
# numarul de rinduri de tabel
$rinduri = 0;
# instantiem analizorul XML
my $parser = new XML::Parser(
    Style => 'Subs',
    # apelare de subrutine pentru fiecare tag
    ErrorContext => 2);
# setam functiile de prelucrare
# a elementelor si continutului lor
$parser->setHandlers(
    Char => \&procesare_continut
    # functia de procesare a continutului
);
# afisam antetul HTTP
print "Content-type: text/html\n\n";
# incarcam fisierul si il analizam
$parser->parsefile("biblio.xml");
# rutina de afisare a continutului
sub procesare_continut
{# am preluat argumentele furnizate
    my ($procesor, $data) = @_;
    # afisam datele
    print $data;}
# rutinele care vor fi apelate pentru
# fiecare aparitie a unui tag de inceput
sub imprumuturi
{ print "<!-- Generat de Perl -->\n";
  print "<table align=\"center\" border=\"1\">";}
sub imprumut
{ $rinduri++;
  # rindurile pare vor avea fundal diferit
  if ($rinduri % 2 == 0) {
```

```

    print "<tr bgcolor=\"#CCCCCC\">";
else {print "<tr>";}
}

sub carte
{
    my $procesor = shift;
    my $element = shift;
    # preluam atributele si le memoram
    # intr-un tablou asociativ
    while (@_) {
        my $atribut = shift;
        my $valoare = shift;
        $atribute{$atribut} = $valoare;
    }
    # preluam atributele care ne intereseaza
    my $autor = $atribute{'autor'};
    my $aparitie = $atribute{'an'};
    print "<td> $autor ($aparitie) <b>";
}

sub client
{print "<td align=\"center\">";}

# rutinele care vor fi apelate pentru
# fiecare aparitie a unui tag de sfirsit

sub imprumuturi_
{
    print "</table>\n";
    print "<!-- Final de generare -->\n";
}

sub imprumut_
{print "</tr>";}

sub carte_
{print "</b></td>";}

sub client_
{print "</td>";}

```

**Utilizarea modelului DOM** Pentru a beneficia de interfețele puse la dispoziție de DOM, vom recurge la modulul `XML::DOM` care extinde o serie din funcționalitățile procesorului Expat. Modulul oferă obiectul `XML::DOM::Parser` care este derivat din `XML::Parser`. Vom putea procesa documentele XML conform specificațiilor DOM - nivelul 1 descrise de Consorțiul Web. Un document va fi regăsit în DOM ca instanță a clasei `Document`. Un obiect de tip `Document` va fi compus din obiecte de tipul `Node`. Un obiect `Document` va putea include noduri de tip `Element`, `Text`, `Comment` și `CDATASection`, iar un `Element` va putea avea noduri de tip `Attr`, `Element`, `Text`, `Comment` sau `CDATASection`. Alte tipuri de noduri nu vor avea nici un descendent.

Un exemplu, în care vom afișa toți autorii cărților împrumutate din bibliotecă (vom folosi pentru exemplificare tot `biblio.xml`):

```

#!/usr/bin/perl
use XML::DOM;
# instantiem analizorul
my $parser = new XML::DOM::Parser;
# incarcam fisierul XML
my $doc = $parser->parsefile("biblio.xml");
# afisam toate atributele 'autor' ale elementelor <carte>
# preluam lista noduri element <carte>
my $noduri = $doc->getElementsByTagName("carte");
# numarul de noduri gasite
my $nr = $noduri->getLength;
# pentru fiecare nod gasit,
# preluam valoarea atributului
for (my $i = 0; $i < $nr; $i++)
{

```

```

    my $nod = $noduri->item($i);
    my $autor = $nod->getAttribute("autor");
    print $autor->getValue . "\n";
}

```

După cum se observă, datorită faptului că modulul este derivat din `XML::Parser`, putem utiliza metodele `parsefile()` sau `parse()` pentru a încărca un document XML în vederea procesării.

**Alte module** Comunitatea programatorilor Perl are la dispoziție o sumedenie de module utile pentru diverse procesări asupra documentelor XML. În cele ce urmează vom descrie câteva dintre aceste module:

- `XML::Simple` oferă o interfață foarte simplă pentru citirea și scrierea de documente XML (indicat a fi utilizat pentru procesarea fișierelor de configurație, a tabelelor de date de mici dimensiuni etc.);
- `XML::Twig` permite procesarea rapidă a documentelor XML de dimensiuni considerabile;
- `XML::Generator` util pentru generarea de documente XML;
- `XML::Grove` oferă acces la date marcate în SGML, XML sau HTML prin intermediul tabelelor asociative;
- `XML::XSLT` implementează funcționalitățile specificației XSLT, fiind bazat pe `XML::DOM`;
- `XML::XQL` oferă posibilitatea de a realiza interogări asupra documentelor XML;
- `XML::Checker` verifică validitatea documentelor XML sau a arborilor DOM;
- `XML::RSS` permite crearea sau modificarea fișierelor RSS (Rich Site Summary) bazate pe RDF (aceste documente sunt folosite pentru crearea de descrieri folosite de Netscape Netcenter sau Meerkat (O'Reilly), putând fi regăsite pe situri precum Slashdot ori Freshmeat);
- `XML::Writer` oferă posibilitatea de a crea documente XML, într-un mod asemănător modulului CGI;
- `DBIx::XML_RDB` exportă date dintr-o bază de date accesată via DBI și le reprezintă în XML.

*Sabin Corneliu Buraga este doctorand în Computer Science la Facultatea de Informatică, Universitatea „Al. I. Cuza” din Iași și poate fi contactat la [busaco@infoiasi.ro](mailto:busaco@infoiasi.ro) sau <http://www.infoiasi.ro/~busaco/>. ■ 10*

## Resurse

- S. Buraga, **Tehnologii Web**, Editura Matrix Rom, București, 2001: <http://www.infoiasi.ro/~busaco/books/web.html>
- S. Buraga, **XML prin SAX**, NET Report, vol.9, 01 (89), ian. 2000
- S. Buraga, **XML DOM – O cale de acces**, NET Report, vol.8, 10 (85), oct. 1999
- Ș. Trăușan-Matu et al., **Prelucrarea documentelor folosind XML și Perl**, Editura Matrix Rom, București, 2001
- L. Wall et al., **Programming Perl (Third Edition)**, O'Reilly & Associates, Cambridge, 2000
- \*\*\*, **CPAN**: <http://www.perl.com/CPAN>
- \*\*\*, **Expat**: <http://www.jc1ark.com/xml/expat.html>
- \*\*\*, **Module XML la CPAN**: <http://www.perl.com/CPAN-local/modules/by-module/XML/>
- \*\*\*, **The Perl-XML FAQ**: <http://www.perlxml.com/faq/perl-xml-faq.html>