EXtensible Stylesheet Language - XSL

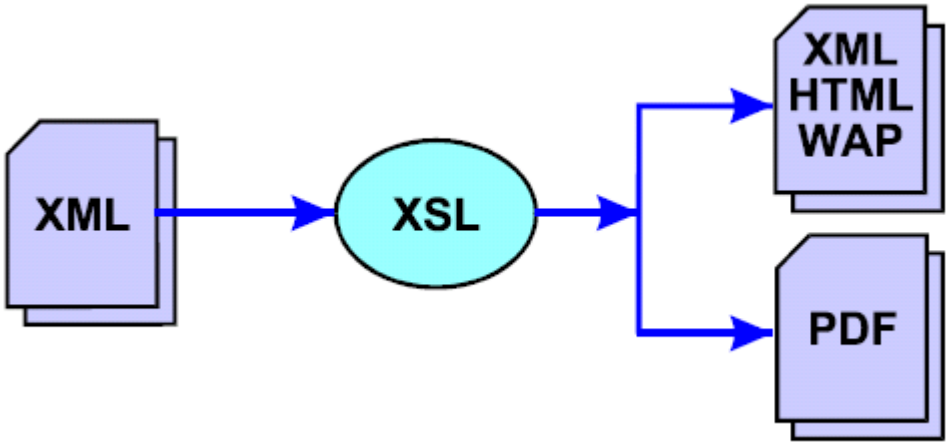
Cuprins :

1. Introducere

Tehnologia XML este foarte puternica si utila. Cu siguranta ca posibilitatea de a transforma cu usurinta un document XML într-un alt format precum HTML, WAP, text, etc. a contribuit la cresterea popularitatii XML-ului. Exista doua tehnologii care pot fi utilizate în acest sens, si anume:

CSS- Cascading Style Sheet;
XSL Extensible Stylesheet Language.

Dintre aceste doua tehnologii XSL este mai puternica si mai flexibila în comparatie
cu utilizarea CSS si, în acelasi timp, este mai apropiata de conceptele XML, fiind
practic un limbaj bazat chiar pe XML, si care utilizeaza stylesheet-uri pentru a
transforma documentului de intrare.



În practica XSL documentul de intrare este numit arbore sursa (arbore de intrare), iar documentul de iesire arbore de iesire.

În cadrul XSL sunt definite doua limbaje:

XSLT – limbaj pentru transformari;
XSL Formatting Objects – utilizat pentru transformarea documentelor XML în
documente

cu format binar precum PDF sau Microsoft Word.

Desi sunt doua limbaje al caror scop este diferit, acestea pot fi folosite împreuna în vederea transformarii documentelor XML. Trebuie mentionat totusi ca motoarele XSLT nu suporta obligatoriu si XSL Formatting Objects, XSLT fiind practic de sine statator.

Exista trei modalitati (dintre care doua mai importante) în care un document XML poate fi transformat într-un alt tip de document prin aplic area unui stylesheet XSLT:

1. Documentul XML si stylesheet-ul asociat sunt transmise aplicatiei client (browser-ului) ca ruia îi revine sarcina de a realiza efectiv transformarea în conformitate cu informatia din stylesheet-ul XSLT. În acste conditii încarcarea serverului scade, dar borwser-ul trebuie sa permita procesarea documentelor XML;

2. Aplicarea stylesheet-ului XSLT se face chiar pe server, documentul rezultat (uzual în format HTML) fiind transmis clientului. Se pot realiza astfel procesari în functie de natura clientului;

3. Cea de a treia posibilitate este extrem de putin utilizata si se refera la transformarea documentului XML cu ajutorul unei aplicatii externe si plasarea pe server a documentului rezultat (HTML), urmând ca acesta sa fie transmis clientului.
Elementul central al tehnologiei XSLT este template-ul: <xsl:template>. În cadrul acestuia se regasesc doua elemente importante:

- atributul match – specifica o cale în arborele de intrare;
- continutul – implementeaza modul în care se realizeaza transformarea.

Forma generala a unui template este:

```
<xsl:tempate match="element_XPath">
...
<xsl:tempate>
```

Asocierea unui document XML cu un stylesheet XSLT se realizeaza în cadrul documentului XML cu ajutorul instructiunii de procesare <?xml-stylesheet>:

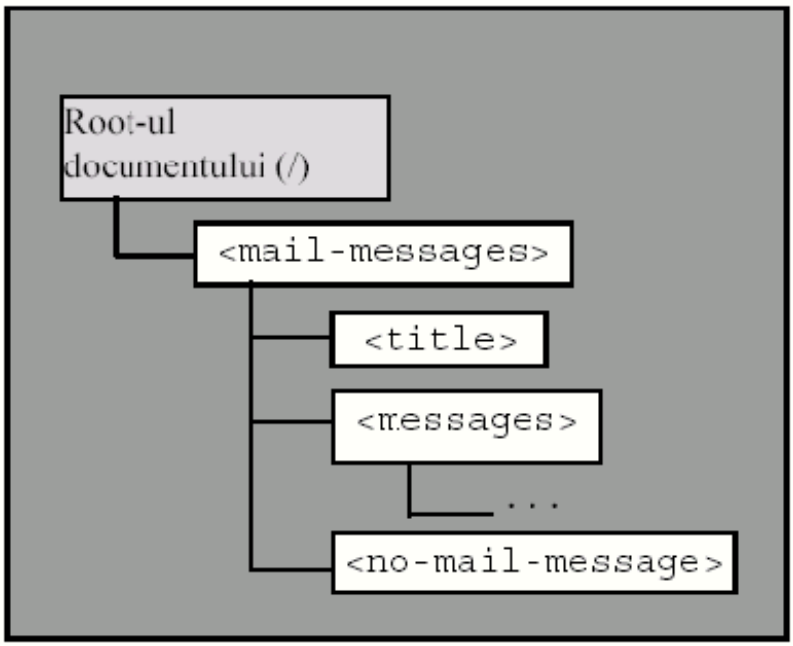
```
<?xml-stylesheet href=”stylesheet/Login” type=”text/xsl” />
```

Argumentul href specifica numele stylesheet-ului XSLT si daca este cazul si calea ca trebuie acesta.

2. XPath

Pentru a sistematiza modul în care poate fi accesat un nod

dintr-un document XML consortiul W3C a elaborat specificatia pentru limbajul XPath. Trebuie mentionat ca în cadrul XPath exista notiunea de root al documentului (document root. Astfel, se face o distinctie foarte clara între root-ul documentului si elementul root al documentului XML. Root-ul documentului, prin prisma XPath, este de fapt radacina arborelui de elemente reale - definite în cadrul documentului.



Root-ul documentului este utilizat la nivel conceptual, neavând un element corespondent în cadrul elementelor documentului XML, si este reprezentat prin caracterul “/”.

Daca se doreste procesarea nodului <mail-messages> din exemplul de mai sus, atunci trebuie utilizata o expresie XPath de tipul "/mail-messages". Expresiile XPath sunt interpretate de la stânga la dreapta, iar expres ia anterioara poate fi înteleasa ca: "pornind de la root-ul documentului selecteaza elemetul <mail-messages> care este fiu al acestuia (root-ului)".

Omiterea caracterului "/" din expresia XPath anterioara schimba radical întelesul
acesteia, caz în care "sunt selectate toate elementele <mail-messages> care sunt fii
ai nodului curent". În cazul elementelor XPath mai complexe, elementele
constituente sunt separate prin caracterul "/", care asa cum se poate observa are o
dubla semnificatie în functie de pozitia în care apare în cadrul elementului XPath.

Un element XPath precum "/mail-messages/messages/message" permite selectarea tuturor nodurilor <message> care sunt fii ai elementului <messages>, care este fiu al elementului <mail-messages>, care la rândul sau este fiu al root-ului documentului.

În cadrul elementelor XPath este permisa de asemenea si utilizarea atributelor.

Pentru a deosebi un element XML de un atribut acestea din urma sunt precedate de caracterul "@".

2.1.Elemente XPath care includ conditii

Exista situatii specifice în care un element XPath trebuie sa faca o selectie mult mai riguroasa a elementelor selectate si tratate în cadrul unui template.

Se poate presupune, spre exemplu, ca într-un anumit context se doreste selectarea doar a elementelor <input> al caror atribut "type" are o valoare diferita de"hidden". Pentru a realiza acest lucru, elementul care va fi filtrat trebuie sa fie urmat de filtrul care urmeaza sa se aplice. Acesta este alcatuit dintr-o pereche de paranteze drepte ([]) care încadreaza, în mod uzual, o conditie (poate sa fie si o conditie compusa).

3.Mod de functionare si utilizare XSL:

3.1.Functionare

XSLT foloseste XPath pentru a defini si identifica parti din documentul sursa care se potrivesc cu vreunul dintre template-urile definite in documentul XSLT.Daca e gasita o potrivire, XSLT va transforma partea din documentul sursa care corespunde potrivirii in altceva, care va face parte din documentul rezultat. Partile pentru care nu exista nici o potrivire (cu nici un template), vor aparea nemodificate in documentul rezultat.

3.2.Utilizare

Trebuie sa avem un document XML pe care dorim sa il prelucram folosind XSL.

Exemplu documentul XML, "studenti.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <grupa>
        <student>
            <nume>Gigi</nume>
            <varsta>2 ani</varsta>
        </student>
```

```
        <student>
            <nume>Gog</nume>
            <varsta>3 ani</varsta>
        </student>
    </grupa>
```

Deasemeni trebuie sa definim un document xsl.Declararea sa se face in 2 moduri:

```
    <xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/
        Transform">
```

sau

```
    <xsl:transform version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/
        Transform">
```

Documentul xsl mai contine si o serie de template-uri.
-exemplu de doc XSL, "studenti.xsl":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
      <html>
        <body>
          <h2>Studenti</h2>
          <table border="1">
            <tr>
              <th
        align="left"><big>Nume</big></th>
              <th
        align="left"><big>Varsta</big></th>
            </tr>
            <xsl:for-each
        select="grupa/student">
              <tr>
                <td><xsl:value-of
        select="nume"/></td>
                <td><xsl:value-of
        select="varsta"/></td>
              </tr>
            </xsl:for-each>
          </table>
        </body>
      </html>
    </xsl:template>
</xsl:stylesheet>
```

In acest moment aveti doua documente XML neconectate intre ele astfel, daca vizualizati documentul XML, veti vedea o reprezentare default pe care browserul o utilizeaza ptr docurile XML care nu au atasat un XSL. De ex. in IE 5.5 veti vedea o reprezentare sub forma de arbore, cu

posibilitatea de restrange arborele pana la radacina (folosind semnele + si - din dreptul fiecarui element). Pentru a specifica faptul ca dorim sa folosim un anumit document xsl ptr documentu nostru xml, trebuie sa adaugam in documentul xml:

<?xml:stylesheet type="text/xsl"
href="studenti.xsl"?>,

imediat dupa prima linie care specifica tipul docului si inainte de declararea celorlalte elemenete:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml:stylesheet type="text/xsl"
href="studenti.xsl"?>
<grupa>
    <student>
        <nume>Gigi</nume>
        <varsta>2 ani</varsta>
    </student>
    <student>
        <nume>Gog</nume>
        <varsta>3 ani</varsta>
    </student>
</grupa>
```

Dupa ce realizam acest lucru, browserul va utiliza docu. XSL si va afisa o lista a studentilor intr-un tabel, dupa cum am specificat in documentul XSL.

4.Prezentare a elementelor cele mai importante dintr-un document XSL:

• Pe primul nivel, dupa cum se vede si in exemplu, nu pot exista decat unul din urmatoarele 2 elemente:

<xsl:stylesheet>,

sau

<xsl:transform>

Dintre atribute sale, cel mai important este "version".Se observa ca se defineste si un spatiu de nume "xsl" prin *xmlns:xsl=http://www.w3.org/1999/XSL/Transform ,iar* daca se utilizeaza acest spatiu de nume, se foloseste obligatoriu si version="1.0".Spatiul de nume asigura unicitatea elementelor folosite in diverse documente XML, el putand asigura si o anumita semnificatie, bine definita, cum e cazul cu elementele din XSL.Pe de alta parte se mareste numarul de identificatori pentru elemente folosit in XML, deoarece daca se realiza asignarea unei semnificatii anume elementului <transform> atunci altcineva nu ar mai fi putut folosi acele elemente decat cu acea semnificatie. (similar cu elementele HTML). Astfel, <spatiunume1:transform> si <spatiunume2:transform> sunt elemente diferite.

"xmlns" reprezinta un spatiu de nume XML
."xmlns:spatiu_nume='un URI'" reprezinta modalitatea de a defini un spatiu de nume XML .In cazul documentelor XSL care respecta specificatiile date de W3C in nov 1999, in documentul de la adresa "http://www.w3.org/1999/XSL/Transform", se utilizeaza tocmai acest URI ptr spatiul de nume "xsl". Obligatoriu tre sa se utilizeze si "version='1.0'".
Semnificatia asociata elementelor XSL este cunoscuta de browser, deoarece el implementeaza complet (IE6) sau incomplet(IE5 si NS6) specificatia W3C.Daca browserul nu cunoaste semnificatia unor elemente, el le trateaza ca pe niste elemente oarecare.

•Pe al doilea nivel, ca elemente care apar in <xsl:stylesheet> si <xsl:transform> cele mai importante sunt:

<xsl:template>

Are ca atribut important ‘match’ (identifica portiunea din documentul XML care se se potriveste cu valoarea indicata in acest atribut, specificata conform XPath).Elementul <xsl:template> contine o serie de reguli (actiuni) care vor fi realizate cand se gaseste un element care se potriveste cu ce e specificat in match.Acest atributul match realizeaza asocierea intre template si un element XML;potrivirea cu intregul document XML se face folosind match="/"
Exemplu :

```
<xsl:template match="/">
        <html>
            <body>
            .......
</xsl:template>,
```

'match="/"' face asocierea intre elementul radacina al documentului XML cu
acest template, iar ca actiune se realizeaza tot ce se gaseste in corpul <xsl:template>: se trimit catre output tagurile pe care nu le identifica (<html>, <body>, etc.), se executa celelalte instructiuni XSL care apar in acest template.

<xsl:value-of>

Are 2 atribute si nu contine alte elemente <xsl:value-of .../>.Dintre atribute cel mai important este "select", care utilizeaza ca si match o sintaxa conforma cu XPath.Acest element XSL are ca actiune implicita returnarea (afisarea) a ceea ce se gaseste in elementul identificat prin "select".
Exemplu: <xsl:value-of select="nume"/> realizeaza afisarea informatiilor care se gasesc in elementul <nume> in XMLul pe care vrem sa-l XSL- tam. De exemplu documentul XML contine "<nume>Gigi</nume>", iar <xsl:value-of select="nume"> returneaza "Gigi".

<xsl:for-each>

Este folosit pentru bucle (for), are ca atribut "select" si

asigura aplicarea actiunilor specificate in interiorul sau de un numar de ori egal cu numarul de potriviri care au loc cu expresia Xpath din atributul "select".

Exemplu:

```
<table border="1">
    <xsl:for-each select="grupa/student">
        <tr>
            <td><xsl:value-of select="nume"/></td>
            <td><xsl:value-of select="varsta"/></td>
        </tr>
    </xsl:for-each>
</table>,
```

In exemplu de mai sus pentru fiecare element <student> care apare intr-un element <grupa>, se realizeaza actiunile din elementul <xsl:for-each>, mai exact afisarea in celulele unui tabel a valorilor elementelor <nume> si <varsta> - puteti adauga in atributul "select" informatie de filtrare a potrivirilor, astfel incat sa se potriveasca doar elemente. exemplu:

```
<xsl:for-each select="grupa/student[varsta!='2 ani']">
    ....
</xsl:for-each>
```

In exemplu de mai sus sunt identificati doar studentii care au in interior un element cu valoarea (<xsl:value-of>) diferita de '2' - mai se pot folosi pentru comparare valori functiile:

=, < (mai mic), > (mai mare)

<xsl:sort>

Nu contine alte elemente,are un atribut: "select" si e folosit pentru sortarea iesirii (rezultatului) dupa valoarea elementului specificat in atributul "select"

```
<table border="1">
    <xsl:for-each select="grupa/student">
        <xsl:sort select="nume"/>
        <tr>
            <td><xsl:value-of select="nume"/></td>
            <td><xsl:value-of select="varsta"/></td>
        </tr>
    </xsl:for-each>
</table>,
```

realizeaza sortarea dupa nume a studentilor listati in tabel

<xsl:if>

Este un element xsl ale carui actiuni (elementele din interiorul sau) au loc doar la indeplinirea conditiei specificata cu atributul "test"

```
<table border="1">
    <xsl:for-each select="grupa/student">
```

```
                <xsl:if test="varsta!='2 ani'">
                  <tr>
                    <td><xsl:value-of select="nume"/></td>
                    <td><xsl:value-of select="varsta"/></td>
                  </tr>
                </xsl:if>
              </xsl:for-each>
            </table>,
```

Exemplu de mai sus asigura afisarea doar a studentilor care au varsta diferita de '2 ani'.Ceea ce poate apare in atributul "test" este similar cu ceea ce apare ca informatie de filtrare in atributul "select" din <xsl:for-each> intre "[" si "]";

<xsl:choose>

Este similar cu "switch" din C sau Java si se utilizeaza alaturi de <xsl:when> ("case" in C) si <xsl:otherwise> ("default" in C);

```
        <table border="1">
            <xsl:for-each select="grupa/student">
                <tr>
                    <td><xsl:value-of select="nume"/></td>
                    <td>
                        <xsl:choose>
                            <xsl:when test!='2 ani'>
                              <B><xsl:value-of
                select="varsta"/></B>
                            </xsl:when>
                            <xsl:otherwise>
                              <I><xsl:value-of
                select="varsta"/></I>
                            </xsl:otherwise>
                        </xsl:choose>
                    </td>
                </tr>
            </xsl:for-each>

        </table>,
```

Exemplu de mai sus are ca rezultat afisarea varstei studentilor fie Bold, fie Italics.

<xsl:apply-templates/>

Permite aplicarea unui template asupra elementului curent sau a fiilor acestuia. Elementul curent este elementul cu care s-a facut potrivirea template-ului in care apare <xsl:apply-templates/>

De asemeni nu contine alte elemente iar cel mai important atribut este "select", care asigura faptul ca se incearca potrivirea unui template numai asupra elementelor fii ai elementului curent, care au numele specificat de prin "select".Daca nu apare "select", se incearca aplicarea de template-uri asupra elementului curent si a tuturor fiilor

acestuia (elementele din el)

Exemplu:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
          <html>
              <body>
                  <h2>Studenti</h2>
                  <table border="1">
                        <tr>
                           <th
              align="left"><big>Nume</big></th>
                           <th
              align="left"><big>Varsta</big></th>
                        </tr>
                         <xsl:apply-templates/>
                    </table>
              </body>
          </html>
       </xsl:template>

        <xsl:template match="grupa/student">
           <tr>
              <td><xsl:value-of select="nume"/></td>
              <td><xsl:value-of select="varsta"/></td>
           </tr>
        </xsl:template>
  </xsl:stylesheet>
```

5.Alte elemente si functii XSLT

<xsl:include>

Este element de nivel 1, adica e fiu al lui "xsl:stylesheet" sau "xsl:transform".Include continutul unui stylesheet in altul.Styleshetul inclus si cel in care se include au aceeasi precedenta

exemplu: <xsl:include href="URI"/>

<xsl:import>

Este element de nivel 1, adica e fiu al lui "xsl:stylesheet" sau "xsl:transform".Importa continutul unui stylesheet in altul.Styleshetul inclus are precedenta mai mica decat cel in care se include

exemplu: <xsl:import href="URI"/>

<xsl:apply-imports>

Aplica un template dintr-un stylesheet importat .Sintaxa sa este similara cu cea din apply-template.

<xsl:number>

E folosit pentru a afisa un numar intr-un anumit format, de exemplu poate afisa pozitia nodului curent in document.

Sintaxa :

```
<xsl:number count="expression" level="single|multiple|any"
   from="expression" value="expression" format="formatstring"
   lang="languagecode" letter-value="alphabetic|traditional"
   grouping-separator="character" grouping-size="number"/>
```

Are ca atribute:

- count: [optional]: expresie XPath ce identifica nodurile ce vor fi numarate
- from: [optiona]: expre XPath: specifica de unde incepe numararea
- value: [optional]: un numar definit de user, in locul numarului generat prin numarare
- format: "1" = "1 2 3 .."; "01" = "01 02 03 .."; "a" = "a b c .." "I" = "I II III IV ..."; specifica cum se va face numararea
- groupping-separator: caracter de separare grupuri de digitzi
- grouping-size: dimensiunea grupurilor de digiti

Exemplu 1

```
<xsl:number value="250000" grouping-separator="."/>
```

Output:
250.000

Exemplu 2

```
<xsl:number value="250000" grouping-size="2"/>
```

Output:
25,00,00

Exemplu 3

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
  <p>
  <xsl:for-each select="grupa/student">
    <xsl:number value="position()" format="1" />
    <xsl:value-of select="nume" /><br />
  </xsl:for-each>
  </p>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

<xsl:message>

Acest element scrie un mesaj la output ;e folosit ptr tratare cazuri eroare.Are ca atribut "terminate='yes|no'" pentru cazul in care se doreste oprirea procesarii dupa mesaj.

Eemplu:

```
<xsl:if test="nume=''">
  <xsl:message terminate="yes">
    Studentul nu campul nume vid!
  </xsl:message>
</xsl:if>
```

<xsl:text>

Folosit pentru a scrie text la output.

<xsl:call-template>

Apeleaza un template cu nume.

Ex:

```
<xsl:template name="templatename" match="student">
</xsl:template>
<xsl:call-template name="templatename">
   <!-- Content:xsl:with-param* -->
</xsl:call-template>
```

In "xsl:call-template" pot aparea "xsl:with-param" folosit pentru a transmite parametrii in templateul apelat.

<xsl:copy>

Creaza o copie a nodului curent, nodurile fii si atributele nu sunt copiate automat.

Exemplu:

```
<xsl:template match="message">
  <xsl:copy>
    <xsl:apply-templates/>
  </xsl:copy>
</xsl:template>
```

Exemplu copiaza la output nodul message.

<xsl:copy-of>

Creaza o copie a nodului curent, nodurile fii si atributele sunt copiate automat.

Atribut: select="expression" = specifica ce sa fie copiat

Exemplu:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:variable name="header">
  <tr>
  <th>Nume</th>
  <th>Prenume</th>
  </tr>
</xsl:variable>
<xsl:template match="/">
  <html>
  <body>
  <table>
    <xsl:copy-of select="$header" />
    <xsl:for-each select="grupa/student">
    <tr>
      <td><xsl:value-of select="nume"/></td>
      <td><xsl:value-of select="prenume"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Exemplu de mai sus afiseaza intr-un tabel dupa nume si prenume.Se defineste o variabila "header" care se apeleaza cu sintaxa "$header".

<xsl:variable>

Defineste o variabila, locala sau globala (dc e pe nivelu 1 e globala).

Valoarea acestui element este data de continutul elementului "variable", fie prin valoarea atributului "select".Ca atribute amintim "name" si "select".Dupa ce a fost definita valoarea unei variabile, nu mai poafi modificata.O variabila fara "select" si care nu contine nimic in ea este echivalenta cu un sir gol "".O variabila se acceseaza folosind $nume_varb

Exemplu
```
<xsl:variable name="sir_vid"/>
<xsl:variable name="color" select="'red'" />
 <xsl:variable name="color" select='"red"' />
```

<xsl:param>

Declara un parametru local sau global iar tributele si declararea sunt similare cu <xsl:variable>.

<xsl:with-param>

Defineste valoarea unui parametru care sa fie transmis unui template.Atributul nume trebuie sa aiba valoare identica cu numele unui element <xsl:param>.Aceste element poate fi folosit intr-un <xsl:apply-template> si <xsl:call-template> ;atributele si declararea sunt similare cu <xsl:variable>.
Valoarea parametrului e data fie de continutul elementului <xsl:with-param> fie de valoarea atributului "select".

<xsl:comment>

Introduce un comentariu.

<xsl:element>

Creaza un nod element pentru a fi trimis catre output.

Syntax

```
<xsl:element name="name" namespace="URI" use-attribute-sets="namelist">
  <!-- Content:template -->
</xsl:element>
```

<xsl:attribute>

Adauga un atribut unui element iar valoarea atributului va fi data de continutul lui <xsl:attribute >.

Exemplu:

```
<picture>
  <xsl:attribute name="source">
    <xsl:value-of select="images/name" />
  </xsl:attribute>
</picture>
```

<xsl:attribute-set>

Defineste o multime de atribute, sub un singur nume

Exemplu:

```
<xsl:attribute-set name="font">
  <xsl:attribute name="fname">Arial</xsl:attribute>
  <xsl:attribute name="size">14px</xsl:attribute>
  <xsl:attribute name="color">red</xsl:attribute>
</xsl:attribute-set>
```

<xsl:fall-back>

Specifica codul xslt care va fi rulat in cazul in care procesorul XSL nu suporta un anumit element XSL.

Exemplu:

```
<xsl:loop select="title">
```

```
    <xsl:fallback>
      <xsl:for-each select="title">
        <xsl:value-of select="."/>
      <xsl:/for-each>
    </xsl:fallback>
  </xsl:loop>
```
  Daca nu se poate executa <xsl:loop> (care oricum nu exista) atunci se executa codul din interiorul <xsl:fallback>

<xsl:key>

  Exemplu de mai sus declara o cheie cu nume care poate fi folosita cu functia "key()". E folosit in special pentru cautari de elemente

Syntax: <xsl:key name="name" match="pattern" use="expression"/>

Atribute: match = indica nodurile asupra carora cheia va fi aplicata

        use = valoarea cheii ptr fiecare dintre noduri

Exemplu:

Presupunem documentul xml "persons.xml":

```
<persons>
  <person name="Tarzan" id="050676"/>
  <person name="Donald" id="070754"/>
  <person name="Dolly" id="231256"/>
</persons>
```

Definim in XSL cheia "preg" care are ca valoare IDul "person"ului

```
<xsl:key name="preg" match="person" use="@id"/>
```

Pentru a gasi "person"ul cu id="050676":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:key name="preg" match="person" use="@id"/>
<xsl:template match="/">
  <html>
  <body>
  <xsl:for-each select="key('preg','050676')">
    <p>
    Id: <xsl:value-of select="@id"/><br />
    Name: <xsl:value-of select="@name"/>
    </p>
  </xsl:for-each>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

5.1.Functii:

current()

Returneaza nodul current.

document(uri_fisier,optional_select_noduri)

Acceseaza noduri intr-un document extern.
Ex:
<xsl:value-of
select="document('celsius.xml')/celsius/result[@value=$value]"/>

element-available(nume_element)

Testeaza daca un element este suportat de procesorul XSLT. Se refera doar la elementele ce pot apare intr-un <xsl:template>.

format-number(nr, format, optional_decimal_format)

Converteste numarul intr-un sir conform cu formatul.

function-available(nume)

Verifica daca functia data ca param e suportata de procesorul XSLT

generate-id()

Genereaza un id unic.

key(nume_element_key, sir_de_cautat)

Returneaza o multime de noduri din document, folosind indexul dat de elementul <xsl:key> identificat de primul parametru.

system-property(string)

Returneaza valoarea unei proprietati a sistemului.Proprietatile disponibile sunt : 'xsl:version', 'xsl:vendor', 'xsl:vendor-uri'.

unparsed-entity-uri(nume_entity)

Returneaza URIul unei entitati externe.

Exemplu:
in DTD:
<!ENTITY pic SYSTEM "http://www.w3schools.com/picture.jpg"
NDATA JPEG>
in XSL:
unparsed-entity-uri('pic')
intoarce http://www.w3schools.com/picture.jpg

5.2.Remarci

–Variabilele (declarate cu <xsl:variable>) au valori imutabile (valoarea asociata nu se schimba din moment ce a fost evaluata) .

<xsl:variable name="varsta" select="7 + 12"/>

– Parametrii (declarati prin <xsl:param>) isi pot modifica ulterior valoarea

– Apelarea (invocarea) unui sablon se poate realiza cu pasarea de valori ale parametrilor via <:xslwith-param>

6.Transformari XSLT

XSLT defineste un vocabular, pentru transformar ea documentelor XML, care include
tag-uri XML pentru arbori, noduri, template-uri si alte elemente necesare pentru ajustarea si transformarea documentelor XML într-un alt vocabular markup (sau acelasi, cu o alta ordine).

6.1.Transformari XSL la nivel programatic

(procesoare XSLT)

–Apache Cocoon & Xalan (Xerces) – C, C++, Perl, Java: **http://xml.apache.org/**

–libxslt (C/C++): **http://xmlsoft.org/XSLT/**

–Oracle-J

–Sablotron (C++): **http://www.gingerall.org**

- PHP (functiile xslt_*)
- Perl (modulul XML::Sablotron)

–Saxon (Java): **http://saxon.sourceforge.net/**

–MSXML – invocare prin COM (ActiveX)

–.NET Framework + *managed languages*(clasa XslTransform din spatiul de nume System.Xml)

–TrAX (parte din JAXP): **http://java.sun.com/**

–XML::XSLT (modul Perl): **http://www.cpan.org/**

–XT (Java)

–Suport (limitat) in cadrul actualelor navigatoare (clienti) Web – via JavaScript

Detalii:
http://xml.coverpages.org/xslSoftware.html

7. Java si XSLT

TrAX acronimul de la *Transformation API for XML* . Este parte integranta din distributia JAXP.

XML sursa→ Arbore DOM sursa → XSLT → Arbore DOM destinatie → XML destinatie

Pachetele care ofera support pentru transformari sunt :

javax . xml . transform

Javax . xml .transform . dom

Javax . xml . transform . stream

7.1.Etapele trasformarii :

1.Stabilirea sursei si a rezultatului
2.Crearea unui obiect *Transformer*
3.Transfomarea propriu-zisa
Interfata *Source* descrie sursa unei transfomari.
Implementari:*DOMSource , SaxSource, StreamSource .*
Interfata *Result* descrie destinatia unei transformari.Implementari :*DOMResult ,SaxResult , StreamResult .*

7.2.Secventa de transformare

```
// Stabilim sursa si destinatia
Source source = new DOMSource(sourceDoc);
Result result = new DOMResult(resultDoc);

// Specificam fisierul cu regulile
Source style = new StreamSource("fisier.xsl");

// Cream obiectul Transformer
TransformerFactory transFactory
=TransformerFactory.newInstance();
Transformer transformer =transFactory.newTransformer(style);
transformer.transform(source, result);
```

7.3.Afisarea unui arbore DOM

```
String filename = "fisier.xml";
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(new File(filename));
DOMSource source = new DOMSource(doc);
StreamResult result = new StreamResult(System.out);
TransformerFactory transFactory =
TransformerFactory.newInstance();
Transformer transformer = transFactory.newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
```

```
//transformer.setOutputProperty("indent", "yes");
transformer.transform(source, result);
```

7.4.Afisarea unui subarbore

```
// Selectam primul nod 'persoana' din arbore
NodeList list = document.getElementsByTagName("persoana");
Node node = list.item(0);
// Sursa o contruim folosind nodul selectat
DOMSource source = new DOMSource(node);
StreamResult result = new StreamResult(System.out);
...
transformer.transform(source, result);
```

7.5.Uilitarul Process

Utilitarul *Process* din distributia JAXP permite efectuarea de transformari de la linia de comanda:

```
java org.apache.xalan.xslt.Process
-IN intrare.xml
-XSL tranformare.xsl
-OUT rezultat.xml
```

8.Tool-uri

Stylus Studio® 2006 este o unealta de dezvoltare XSL ce include : XSLT Editor , XSLT Mapper , Open XSLT Architecture , XSLT Debugger
Sursa : http://www.stylusstudio.com/xslt.html

eXist este o bază de date nativă XML ce permite o eficienta procesare şi indexare automată XQuery. Suportul XQuery in eXist face să fie posibilă scrierea de aplicaţii întregi doar cu XQuery şi XSLT.
Sursa: http://exist.sourceforge.net/

Altova a produs o serie de utilitare pentru dezvoltarea XQuery. **XMLSpy® 2006 este un produs œ permite dezvoltarea şi testarea codului XSLT si XPath. Alte utilitare produse de Altova : StyleVision® 2005 , MapForce® 2006, Altova® XML Suite 2006 .**
Sursa : http://www.altova.com/dev_portal_xslt.html

9.Concluzii

XSL este o componenta importanta a tehnologiei XML, iar cunoasterea sa reprezinta
un element esential în dezvoltarea aplicatiilor bazate pe XML. Transformarea documentelor XML se realizeaza cu ajutorul stylesheet-urilor XSLT care permit procesari complexe oferind dezvoltatorilor o multitudine de elemente si functii în acest sens.

Prezentarea într-o viziune originala a conceptelor si elementelor cu adevarat importante a avut în vedere crearea unei imagini cât mai clare referitoare la tehnologia XSL, tehnologie care are o importanta deosebita în procesul de proiectare si implementare a aplicatiilor complexe, bazate pe tehnologia XSP, care utilizeaza biblioteci de tag-uri pentru implementarea logicii aplicatiei.

Bibliografie : http://serghei.net/docs/programming/XmlBible/pdf
http://w3.org
http://w3schools.com