XSLT

dr. Paller Gábor

XSL szabványsorozat

- XSL (eXtensible Stylesheet Language) szabványsorozat: http://www.w3.org/Style/XSL/
  - XPath - már ismerjük
  - XSL Transformations - most jön.
  - XSL-FO (Formatting Objects) - csak felületesen megyünk bele.
- Vízió:
  - XML dokumentum: nyers adat
  - XSLT stíluslap transzformálja XSL-FO objektumokat tartalmazó formázott dokumentummá.
  - Az XSL-FO formátumelemekkel dúsított XML dokumentumot a célformátummá konvertálják (pl. PDF).
  - A kulcs: könnyen szerkeszthető, scriptszerű konverternyelv. Ez az XSLT.

XSL-FO

- Gazdag lapformázást lehetővé tevő elemkönyvtár. A CSS2-höz hasonló bonyolultságú.
- <fo:block break-before="page">
  <fo:block text-align="center" space-after="8pt" space-before="16pt"
  space-after.precedence="3">
  Chapter title
  </fo:block>
  <fo:block text-indent="0pc" space-after="7pt" space-before.minimum="6pt"
  space-before.optimum="8pt" space-before.maximum="10pt">
  Section one's first paragraph.
  </fo:block>
  <fo:block text-align="center" space-after="6pt" space-before="12pt"
  space-before.precedence="0"
  space-after.precedence="3">Second section title
  </fo:block>
  </fo:block>

XSLT

- XML dokumentumok más XML dokumentumokká való transzformálására való.
- Önmaga is XML alapú nyelv.
- Elvileg nem általános célú, hanem az XSL-FO céljait szolgálja. Valójában szinte minden célra használható.
- Szoros kapcsolatban van az XPath nyelvvel.

Template

- A transzformáció alapegysége a template.
- A template egy csomópontok bizonyos csoportjára vonatkozó feldolgozási szabály.
- A csomópont-csoport kijelölésére XPath kifejezést használunk.
- A template egy eredményfa-részletet (result tree fragment) is tartalmaz, amelyet a template illeszkedése esetén a generált eredményfába szúrunk.
- A forrás XML feldolgozásának lépései
  - Kezdet: aktuális csomópont-lista: a forrásdokumentum összes csomópontja, aktuális csomópont: ezek közül egy.
    - Dokumentumsorrendben előről kezdjük.
  - Vesszük a forrás XML egy csomópontját
  - Megkeressük az összes rá illeszkedő template-et
  - Kiválasszuk a legjobbat (a prioritásokról később beszélünk)
  - Ebből a template-ből példányt hozunk létre (instantiate)
    - Ennek eredményeképpen a template eredményfa-részlete a generált eredményfába szúródik és a template sok esetben új aktuális csomópont-listát definiál.
  - Mindez rekurzívan folytatódik, amíg az aktuális csomópont-listát el nem fogyasztottuk.
- Az XSLT feldolgozó másként is működhet, ha az eredmény megegyezik a fenti algoritmus által produkált eredménnyel.

Template illesztés

- Forrás XML dokumentum:

```
<?xml version="1.0"?>
<html>
  <body>
    Text <emph>emphasized text</emph> more text
  </body>
</html>
```

- XSLT stíluslap:

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

<xsl:template match="emph">
  <b/>
</xsl:template>

</xsl:stylesheet>
```

Template illesztés, 2.

- Eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>

Text <b/> more text
```

- Érdekességek:
  - Honnét az <?xml ...> sor? Az XSLT feldolgozó automatikusan generálja az eredményfa Document elemét.
  - Hová lettek a tag-ek? Csak az <emph> illeszkedett, a többire nem volt illeszkedő szabály.
  - Akkor hogy jutottunk el az <emph>-ig? És miért kerültek át a szöveges csomópontok? Később beszélünk a beépített template-ekről.
  - Hová lett az <emph> tartalma? Az emph-re illeszkedő template nem adta ezeket hozzá az aktuális csomópont-listához, tehát elnyelődtek.

Beépített template-ek

- Az XSLT feldolgozó néhány template-et beépítve tartalmaz. Ezek prioritása olyan alacsony (prioritásról később), hogy bármilyen, felhasználó által definiált template felülbírálja őket. Ha azonban nincs a csomópontra illeszkedő template, akkor ezek jutnak érvényre.
  - Ha a csomópont elem csomópont, az összes gyerekét az aktuális csomópontlistához adja (a feldolgozást továbbküldi a gyerekek felé). Ezért érte el a feldolgozás az emph elemet az előző példában.
  - Ha a csomópont attribútum vagy szöveg csomópont, akkor változatlanul kiküldi az eredményfába (attribútumot persze csak akkor, ha az anyaelemét is átmásolta).

apply-templates

- Az apply-templates paranccsal tudunk hozzáadni csomópontokat az aktuális csomópont-listához. Legegyszerűbb, paraméter nélküli formájában az aktuális csomópont összes gyerekét hozzáadja az aktuális csomópont-listához.
- Korábbi példánkat használva:

```
<xsl:template match="emph">
  <b>
    <xsl:apply-templates/>
  </b>
</xsl:template>
```

- Ez már a várt eredményt adja:

```
<?xml version="1.0" encoding="UTF-8"?>

Text <b>emphasized text</b> more text
```

apply-templates, 2.

- ```
  <xsl:apply-templates
    select = node-set-expression
    mode = qname>
    <!-- Content: (xsl:sort | xsl:with-param)* -->
  </xsl:apply-templates>
  ```
- select attribútum: milyen csomópontokat adjon az aktuális csomópont-listához (alapértelmezett: az aktuális csomópont összes gyereke).
- mode attribútum: milyen template-eket vegyen figyelembe (a template mode attribútumának illeszkednie kell az apply-templates mode attribútumához).
  ```

apply-templates, 3.

- Forrás XML dokumentum:

```
<?xml version="1.0"?>
<books>
  <book>
    <title>Book title #1</title>
    <publish_info>
      <publisher>Book publisher #1</publisher>
      <published>1999</published>
    </publish_info>
  </book>
  <book>
    <title>Book title #2</title>
    <publish_info>
      <publisher>Book publisher #2</publisher>
      <published>2001</published>
    </publish_info>
  </book>
</books>
```

apply-templates, 4.

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

<xsl:template match="publisher|published|title">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="book">
  <b>Title: <xsl:apply-templates select="title"/></b>
  <i>(<xsl:apply-templates
select="publish_info/publisher"/>,
          <xsl:apply-templates
select="publish_info/published"/>)
  </i>
</xsl:template>

</xsl:stylesheet>
```

apply-templates, 5.

- Eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>
  <b>
    Title: Book title #1</b><i>
    (Book publisher #1
    ,1999)
  </i>
  <b>
    Title: Book title #2</b><i>
    (Book publisher #2
    ,2001)
  </i>
```

- Miért?
  - A publisher, published, title elemekre illeszkedő template továbbküldi a feldolgozást a gyerekek felé->szöveges csomópontok az eredményfába kerülnek.
  - A book elemekre illeszkedő template célzott XPath kifejezésekkel kiválasztja a publisher, published és title elemeket, így a publish_info elemek sose kerülnek az aktuális csomópont-listába.

apply-templates, 6.

- "mode" attribútum: lehetővé teszi, hogy az apply-templates a template-ek egy csoportjára vonatkozzon. A "mode" attribútum értéke egy QName, aminek illeszkedni kell a template-ek "mode" attribútumának értékére. Csak az illeszkedő "mode" attribútumú template-eket lehet kiválasztani.
- ```
  <xsl:apply-templates mode="amode"/>
  ...
  <xsl:template match="..." mode="amode">
  ...
  </xsl:template>
  ```
  ```

apply-templates, 7.

- mode példa: az apply-templates a mode attribútumon keresztül kijelöli, melyik template-et kívánja kiválasztani. Ha nem lenne "mode", akkor a stíluslap hibás lenne (két, azonos prioritású template illeszkedne az "emph" elemre).
- Forrásdokumentum: az ismert XHTML példa. Stíluslap:

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

<xsl:template match="html">
  <xsl:apply-templates mode="b"/>
</xsl:template>

<xsl:template match="emph" mode="b">
  <b><xsl:apply-templates/></b>
</xsl:template>

<xsl:template match="emph" mode="i">
  <i><xsl:apply-templates/></i>
</xsl:template>

</xsl:stylesheet>
```

Template-ek prioritása

- Ha egy csomópontra több template is illeszkedik, a legnagyobb prioritású template-et választja ki az XSLT feldolgozó. Ha ez után is több lehetséges template marad (több, egyenlő prioritású template létezik legnagyobb prioritással), az **hiba** és a feldolgozás megszakad.
- A prioritás lehet explicit vagy implicit. Az explicit prioritást a template-ben definiáljuk.

```
<xsl:template match="..."  priority="5">
...
</xsl:template>
```

- Implicit prioritás: a template mintaillesztő kifejezéséből számítódik.
  - Ha az XPath kifejezés csomópont-kiválasztó része QName (pontos név névterülettel), a prioritás 0.
  - Ha az XPath kifejezés NCName:* (névterület-azonosító lokális név helyett wildcard), a prioritás -0.25.
  - Ha az XPath kifejezés csak iránykijelölésből áll (pl.: child::*), a prioritás -0.5.
  - Különben a prioritás 0.5.

Template-ek prioritása, 2.

- Forrás: ismert XHTML dokumentumunk. Stíluslap (az xsl:copy lemásolja az aktuális csomópontot, bővebben róla később):

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

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

<xsl:template match="emph">
  <b><xsl:apply-templates/></b>
</xsl:template>

</xsl:stylesheet>
```

- Eredmény:

```
<html><body>Text <b>large text</b> more text</body></html>
```

- Magyarázat: az "emph" template-nek nagyobb a prioritása, mivel ő egy QName, míg a * csak egy iránykijelölés:* minta, ezért az "emph" elemre a második template hajtódik végre, míg minden más elemre az első.

Template-ek hívása

- A template-et általában az XSLT feldolgozó hívja meg, ha a hozzá rendelt XPath kifejezés illeszkedik az aktuális csomópontra.
- Lehetséges a template explicit meghívása egy másik template-ből. Ehhez a template-et névvel kell ellátni és a call-template paranccsal meghívni.
- Template definíció:
```
<xsl:template match="..." name="template_name">
...
</xsl:template>
```
- A template-nek match és name attribútuma egyaránt lehet.
- Névvel ellátott template-et meg lehet hívni a call-template paranccsal.
```
<xsl:call-template name="template_name"/>
```
- Az aktuális csomópont és az aktuális csomópont-lista változatlan marad a hívás során.

Template hívás paraméterei

- A template hívásának paraméterei lehetnek.
- A paramétert a meghívandó template oldalán a "param" elem definiálja az alapértelmezett értékkel együtt. Erről többet a változóknál.
- Hívásnál a "with-param" paraméternél definiáljuk az átadandó paramétert és értékét. Az értékekről ugyancsak a változóknál beszélünk többet.
- Az átadott paramétert (a változókhoz hasonlóan) XPath változókkal hivatkozhatjuk ($változónév). A {} szintakszisról később.
- ```
  <xsl:template name="numbered-block">
    <xsl:param name="format">1. </xsl:param>
    <fo:block>
      <xsl:number format="{$format}"/>
      <xsl:apply-templates/>
    </fo:block>
  </xsl:template>

  <xsl:template match="ol//ol/li">
    <xsl:call-template name="numbered-block">
      <xsl:with-param name="format">a. </xsl:with-param>
    </xsl:call-template>
  </xsl:template>
  ```
  ```

Attribute value template

- Előző példa: `format="{$format}"`
- Ezt attribute value template-nek hívják, a szintaxisa: {*XPath kifejezés*}
- Az XPath kifejezés kiértékelődik, string típussá konvertálódik (string függvény) és az XSLT stíluslapba helyettesítődik.
- Sok XSLT direktíva megengedi attribútumaiban.

Eredményfa: a stylesheet elem

- Eredményfába kerül minden elem
  - Ami nem az xslt (http://www.w3.org/1999/XSL/Transform) névtérbe tartozik.
  - Ami nem az XSLT-t bővítő elem (később)
- Az eredményfa gyökerét (Document elem) az XSLT elemző maga generálja.
- A generált elemek névtér-specifikációját az xsl:stylesheet elem névtér-specifikációiból veszi.
  - Az XSLT-hez tartozó névtér nem kerül át, minden egyéb igen.

XSLT eredményfa névtér példa

- Forrás dokumentum:

```
<?xml version="1.0"?>
<xhtml:html xmlns:xhtml="http://www.w3c.org/1999/xhtml">
  <xhtml:body>
    Text <xhtml:emph>emphasized text</xhtml:emph> more text
    This is more <xhtml:emph>emphasized text.</xhtml:emph>
  </xhtml:body>
</xhtml:html>
```

- XSLT stíluslap:

```
<xsl:stylesheet  version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:xhtml="http://www.w3c.org/1999/xhtml"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsl:template match="xhtml:emph">
  <b><xsl:apply-templates/></b>
</xsl:template>

</xsl:stylesheet>
```

XSLT eredményfa névtér példa, 2.

- Eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>

Text <b xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xhtml="http://www.w3c.org/1999/xhtml">emphasized
text</b> more text
This is more <b
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xhtml="http://www.w3c.org/1999/xhtml">emphasized
text.</b>
```

- A stylesheet elemben definiált összes névtér átkerült az egyedi elemekbe, kivéve az XSLT-hez tartozó.
- Ez azt jelenti, hogy céldokumentumban használt összes névterületet definiálni kell az xsl:stylesheet elemben. Ezen felül ugyanitt deklarálni kell a forrásdokumentum összes névterét, amit címzésre (XPath kifejezés) használunk fel.

Névterület-definíciók vezérlése

- xsl:stylesheet exclude-result-prefixes attribútuma vagy xsl:exclude-result-prefixes bármelyik generált (kimeneti) elemen: értéke szóközökkel elválasztott listája azoknak a prefixeknek, amelyek nem kerülnek át az eredményfába. Ezzel el lehet nyomni a csak a forrásdokumentum feldolgozásakor használt névterület-kijelöléseket.
- Példa (előző példára építve):

```
...
<b xsl:exclude-result-prefixes="xsd">
    <xsl:apply-templates/>
</b>
```

- xsl:stylesheet elem attribútumaként:

```
<xsl:stylesheet  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xhtml="http://www.w3c.org/1999/xhtml"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xsd"> ...
```

- Az eredmény mindkét esetben: `<b xmlns:xhtml="http://www.w3c.org/1999/xhtml">` ...

Névterület-definíciók vezérlése, 2.

- xsl:namespace-alias: megadja, hogy az XSLT stíluslapban levő egyik névterület helyett az eredményfában egy másikat kell használni. Különösen hasznos, ha XSLT stíluslapot akarunk generálni, mert a http://www.w3.org/1999/XSL/Transform névteret közvetlenül nem használhatjuk.
- Példa:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xhtml="http://www.w3c.org/1999/xhtmlalias"
 xmlns:html="http://www.w3c.org/1999/xhtml">

<xsl:namespace-alias stylesheet-prefix="xhtml" result-
prefix="html"/>

<xsl:template match="html:emph">
  <xhtml:b><xsl:apply-templates/></xhtml:b>
</xsl:template>

</xsl:stylesheet>
```

- Eredmény: `<html:b xmlns:html="http://www.w3c.org/1999/xhtml">` ...

Literális eredmény-elemek

- Literális eredményelem az, amely a template-ek belsejében van és nem tartozik az XSLT névterülethez vagy az XSLT-t kiterjesztő névterülethez (erről később).
- A literális eredményelemre az előzőekben említett névterület-transzformációk vonatoznak.
- Ezen felül attribútumai értékeként tartalmazhat attribute value template-eket.

Literális eredmény-elemek, 2.

- Forrás dokumentum:

```
<?xml version="1.0"?>
<html>
  <largesize>9</largesize>
  <body>Text <large>large text</large> more text</body>
</html>
```

- Stíluslap:

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

<xsl:template match="largesize">
</xsl:template>

<xsl:template match="large">
  <font size="{/child::html/child::largesize}">
    <xsl:apply-templates/>
  </font>
</xsl:template>

</xsl:stylesheet>
```

- Eredmény: `<font size="9">` ...

xsl:value-of

- Szöveges csomópont eredményfába szúrása XPath kifejezés alapján. Az XPath kifejezés kiértékelődik, string típussá konvertálódik a string() függvénnyel, majd az eredmény karaktersorozat mint szöveges csomópont az eredményfába szúródik.
- Forrásdokumentum:

```
<?xml version="1.0"?>
<employees>
  <employee name="John Doe" id="123456"/>
  <employee name="Karl Smith" id="654321"/>
</employees>
```

- Stíluslap:

```
<xsl:stylesheet  version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="employee">
  <p><xsl:value-of select="attribute::name"/>,
        <xsl:value-of select="attribute::id"/></p>
</xsl:template>
</xsl:stylesheet>
```

- Eredmény: `<p>John Doe,123456</p><p>Karl Smith,654321</p>`

Szöveg, feldolgozásvezérlő, komment generálása

- xsl:text - szöveges csomópont generálása. Példa: <xsl:text>Text ...</xsl:text>. Jól jön, ha fontosak a szóközök, az xsl:text az egyedüli elem, amelyik megőrzi a szóközöket ("fehér karaktereket").
- xsl:processing-instruction - feldolgozásvezérlő generálása. Példa:
  Stíluslap:
  ```
  <xsl:processing-instruction name="xml-
  stylesheet">href="book.css"
  type="text/css"</xsl:processing-instruction>
  ```
  Eredmény:
  ```
  <?xml-stylesheet href="book.css" type="text/css"?>
  ```
- xsl:comment - komment generálása.
  Stíluslap:
  ```
  <xsl:comment>This file is automatically generated. Do
  not edit!</xsl:comment>
  ```
  Eredmény:
  ```
  <!--This file is automatically generated. Do not edit!-
  ->
  ```

Gyakorlat

- Írjon XSLT stíluslapot, ami a korábban szerepelt "books" dokumentum alapján kiírja a kiadókat és a kiadási éveket "kiadó,kiadási év" formában.

Megoldás

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

  <xsl:template match="publish_info">
    <xsl:text>
    </xsl:text>
    <xsl:value-of select="publisher"/>,<xsl:value-of
  select="published"/>
  </xsl:template>


  <xsl:template match="text()">
  </xsl:template>
  </xsl:stylesheet>
  ```
- Eredmény:
  <?xml version="1.0" encoding="UTF-8"?>
  Book publisher #1,1999
  Book publisher #2,2001
  ```

xsl:element

- Kiszámított nevű elemet szúr az eredményfába. A direktíva attribútumai meghatározzák az elem nevét, névterületét és lehetőség van hasonlóan számított attribútumok megadására.
- Forrásdokumentum:

```
<?xml version="1.0"?>
<html>
<emphelem>b</emphelem>
<body>
  Text <emph>emphasized text</emph> more text
</body>
</html>
```

- Stíluslap:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="emphelem"></xsl:template>
<xsl:template match="emph">
  <xsl:element name="{/child::html/child::emphelem}">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

- Eredmény: `Text <b>emphasized text</b>` ...

xsl:attribute

- Kiszámított nevű attribútumok eredményfába szúrására szolgál. Az xsl:attribute direktíva lehet literális vagy xsl:element direktívával létrehozott elem belsejében. Az attribútumokat a gyerekelemek hozzáadása előtt kell az elemhez adni.
- Példa: a literális eredmény-elemnél bemutatott funkció megoldása másképpen. Ugyanaz a forrásdokumentum. Stíluslap:

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

<xsl:template match="largesize"></xsl:template>


<xsl:template match="large">
  <font>
    <xsl:attribute name="size">
      <xsl:value-of select="/child::html/child::largesize"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </font>
</xsl:template>
</xsl:stylesheet>
```

- Eredmény: `<font size="9">large text</font>`

Attribútumkészletek elnevezése

- xsl:attribute-set: Attribútumok egy csoportjának név adható és erre a névre való hivatkozással az egész készletet az elemhez lehet adni. Az attribútumkészletre az xsl:element, xsl:copy vagy xsl:attribute-set use-attribute-sets attribútumával lehet hivatkozni, illetve literális elemhez a xsl:use-attribute-sets attribútummal lehet a készletet hozzáadni.
- Az attribútumkészlet makrónak felel meg: a készlet felhasználásakor mindig kiértékelődik (pl. az attribute value template-ek kiértékelődnek újra).
- Ugyanazon névvel létrehozott attribútumkészletek összefésülődnek, attribútum névegyezés esetén a magasabb prioritású stíluslapból (erről később) származó attribútun jut érvényre.

Attribútumkészletek elnevezése, 2.

- Forrásdokumentum:

```
<?xml version="1.0"?>
<html>
<body>
  Text <emph>large text</emph> more text
</body>
</html>
```

- Stíluslap:

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

<xsl:template match="emph">
  <font xsl:use-attribute-sets="emph-style">
    <xsl:apply-templates/>
  </font>
</xsl:template>

<xsl:attribute-set name="emph-style">
  <xsl:attribute name="size">8</xsl:attribute>
   <xsl:attribute name="color">red</xsl:attribute>
</xsl:attribute-set>

</xsl:stylesheet>
```

- Eredmény: `Text <font size="8" color="red">large text</font> ...`

xsl:copy

- xsl:copy - az eredményfába másolja az aktuális csomópontot.
- Ha a csomópont elem csomópont, a névterület-deklarációkat átmásolja, de az gyerekelemeket és attribútumokat nem.
- Az xsl:copy elem tartalma határozza meg az átmásolt elem attribútumait és gyerekelemeit.
- Példa: ez az identitás-transzformáció (módosítás nélküli másolás az attribútumokkal és gyerekelemekkel együtt)

```
<xsl:template match="@*|node()">
  <xsl:copy>
     <xsl:apply-templates select="@*|node()"/>
   </xsl:copy>
</xsl:template>
```

- Copy elem tartalmazhat attribútumokat (xsl:attribute) és hivatkozhat attribútumkészletekre. Ez azonban csak akkor megengedett, ha az aktuális csomópont elem csomópont.

Számozás

- Az xsl:number direktívával egy sorszámot szúrhatunk az eredményfába, mint szöveges csomópontot. A sorszám alapértelmezett értéke a "position()" XPath kifejezés értéke.
- Példa: forrásdokumentum

```
<?xml version="1.0"?>
<doc>
  <item/>
  <item/>
  <otherelement/>
  <item>
    <subitem/>
    <subitem/>
  </item>
</doc>
```

Számozás, 2.

- Stíluslap:

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

<xsl:template match="item">
  <xsl:number/>. <xsl:value-of select="name()"/>
</xsl:template>

</xsl:stylesheet>
```

- Eredmény:

```
<?xml version="1.0" encoding="UTF-8"?>
  1. item
  2. item
  3. item
```

Számozás, 3.

- Forrásdokumentum:

```
<?xml version="1.0"?>
<doc>
  <item/>
  <item/>
  <item>
    <item>
      <item/>
       <item/>
    </item>
    <item/>
  </item>
</doc>
```

- Stíluslap:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="item">
  <xsl:number level="level"/>. <xsl:value-of select="name()"/>
  <xsl:apply-templates/>
</xsl:template>
</xsl:stylesheet>
```

Számozás, 4.

- level -> single. Eredmény:

```
1. item
2. item
3. item
  1. item
    1. item
    2. item
  2. item
```

- level->multiple. Eredmény:

```
1. item
2. item
3. item
  3.1. item
    3.1.1. item
    3.1.2. item
  3.2. item
```

- level->any. Eredmény:

```
1. item
2. item
3. item
  4. item
    5. item
    6. item
  7. item
```

Számozás, 5.

- count attribútum: XPath kifejezés, ami megmondja, milyen csomópontokat kell számlálni. Alapértelmezésben az aktuális csomóponttal megegyező nevű elemeket választ ki.
- Forrásdokumentum:

```
<?xml version="1.0"?>
<doc>
  <section>
    This is section <sec/>.
  </section>
  <section>
    This is section <sec/>.
  </section>
</doc>
```

Számozás, 6.

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

  <xsl:template match="*">
    <xsl:copy><xsl:apply-templates/></xsl:copy>
  </xsl:template>
  <xsl:template match="sec">
    <xsl:number count="section"/>
  </xsl:template>

  </xsl:stylesheet>
  ```
- Eredmény:
  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <doc>
    <section>
      This is section 1.
    </section>
    <section>
      This is section 2.
    </section>
  </doc>
  ```
  ```

Számozás, 7.

- A value attribútum értéke egy XPath kifejezés és a szám értékét meghatározó alapértelmezett kifejezést (position()) lehet vele felülírni. Ennek akkor van jelentősége, ha az xsl:number-t nem sorszámozásra, hanem formázásra használjuk.
- Formátum: a format attribútummal állíthatjuk be a kimeneti formátumot.
  format="1", value=124 -> 124
  format="01" value=9 -> 09
  format="a" value=2 -> b
  format="a" value=27 -> aa
  format="A" value=28 -> AB
  format="i" value=8 -> viii
  format="I" value=9 -> IX
- Előző level="multiple" példánk format="1.A.I." formátumspecifikációval:

```
1.item
2.item
3.item
    3.A.item
      3.A.I.item
      3.A.II.item
    3.B.item
```

Számozás, 8.

- grouping-size, grouping-separator: elválasztókarakterek számba szúrása.
- grouping-size="3" grouping-separator="," value="1250000" -> 1,250,000

Gyakorlat

- Adva van a következő formátumú XML dokumentum:

```
<?xml version="1.0"?>
<doc>
  <number format="1">124</number>
  <number format="01">9</number>
  <number format="a">2</number>
  <number format="a">27</number>
  <number format="A">28</number>
  <number format="i">8</number>
  <number format="I">9</number>
</doc>
```

- Írjon XSLT stíluslapot, ami kiírja a "number" elem belsejében levő számokat a "number" elem "format" attribútumának megfelelő formátumban!

Megoldás

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

<xsl:template match="number">
  <xsl:number value="." format="{@format}"/>
</xsl:template>

</xsl:stylesheet>
```

Ciklus

- Az xsl:for-each ciklus szervezésére való.
- Szintaxis:

```
<xsl:for-each
   select = node-set-expression>
   <!-- Template -->
</xsl:for-each>
```

- "select" attribútuma egy XPath kifejezés. Az ebben szereplő minden csomópontra külön példányosítja a template-et. Vagyis végigiterál a "select" kifejezés által kiválasztott csomópontokon úgy, hogy az aktuális csomópont a template számára az xsl:for-each által kiválasztott csomópontokon fut végig.

Ciklus, 2.

- Oldjuk meg a 11-12. dián szereplő feladatot az xsl:for-each felhasználásával!
- Megoldás:

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

<xsl:template match="child::books">
  <xsl:for-each select="child::book">
   <b><xsl:value-of select="child::title"/></b>
    (<i><xsl:value-of
select="child::publish_info/child::publisher"/></i>,
    <i><xsl:value-of
select="child::publish_info/child::publisher"/></i>)
  </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Gyakorlat

- Nevezetes "books"dokumentumunkban lehetséges lesz több "publish_info" elem megadása egyetlen könyvhöz.
- Példa:

```
<books>
  <book>
    <title>Book title #1</title>
    <publish_info>
      <publisher>Book publisher #1</publisher>
      <published>1999</published>
    </publish_info>
    <publish_info>
      <publisher>Book publisher #1</publisher>
      <published>2005</published>
    </publish_info>
  </book>
...
</books>
```

- Feladat: írja ki a könyvek adatait HTML táblában, egy könyv-egy tábla formában! Generáljon teljes XHTML dokumentumot!

Megoldás

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

<xsl:template match="/">
  <html>
  <body>
    <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>


<xsl:template match="book">
  <table>
    <th><xsl:value-of select="title"/></th>
    <xsl:for-each select="publish_info">
      <tr>
        <td><xsl:value-of select="publisher"/></td>
        <td><xsl:value-of select="published"/></td>
      </tr>
    </xsl:for-each>
  </table>
</xsl:template>

</xsl:stylesheet>
```

Feltételes végrehajtás: xsl:if

- Elemek feltételes eredményfába szúrása: xsl:if
- ```
  <xsl:if
    test = boolean-expression>
    <!-- Template  -->
  </xsl:if>
  ```
- A "test" attribútum egy XPath kifejezés boolean értékkel (ha nem az, a boolean() függvény hívásával ilyen típussá konvertálódik).
- Ha a "test" értéke "true", az xsl:if belsejében levő template az eredményfába szúródik, egyébként nem.
  ```

Feltételes végrehajtás: xsl:if, 2.

- Feladat: van egy ilyen dokumentumunk:

```
<?xml version="1.0"?>
<doc>
  <number grouping-size="3">1250000</number>
  <number>9</number>
</doc>
```

- Jelezze ki a "number" elemek tartalmát az xsl:number direktívával. Használjon az xsl:number direktívában grouping-size attribútumot, ha a forrás "number" eleme is tartalmazza!

Feltételes végrehajtás, 3.

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

<xsl:template match="number">
  <xsl:if test="@grouping-size">
    <xsl:number value="."
        grouping-size="{@grouping-size}"
            grouping-separator=","/>
  </xsl:if>
  <xsl:if test="not( @grouping-size )">
    <xsl:number value="."/>
  </xsl:if>
</xsl:template>

</xsl:stylesheet>
```

Esetválasztás, xsl:choose

- Az xsl:choose a népszerű programozási nyelvek switch szerkezetének felel meg. Jelentős eltérés, hogy minden ágnak külön tesztkifejezése lehet. Általános formája:

```
<xsl:choose>
  <xsl:when test="XPath boolean kifejezés1">
    ... template1 ...
  </xsl:when>
 <xsl:when test="XPath boolean kifejezés2">
    ... template2 ...
  </xsl:when>
  <xsl:otherwise>
    ... template ...
  </xsl:otherwise>
</xsl:choose>
```

- Feladat: írjuk át az előző stíluslapot úgy, hogy megspóroljuk a tesztkifejezés és negáltjának kiértékelését!

Esetválasztás, xsl:choose, 2.

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

<xsl:template match="number">
  <xsl:choose>
    <xsl:when test="@grouping-size">
      <xsl:number value="."
              grouping-size="{@grouping-size}"
              grouping-separator=","/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:number value="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Rendezés

- xsl:apply-templates vagy xsl:for-each műveletnél rendezés definiálható. Ez azt jelenti, hogy a fenti műveletek eredményhalmaza megadott kulcsok szerint rendezhető. (Emlékeztető: az xsl:apply-templates eredményhalmaza alapértelmezésben az aktuális csomópont gyerekeinek listája, ha a select attribútum adott ill. az xsl:for-each-nél az XPath kifejezés által megadott csomópontlista).
- A rendezési műveletet úgy kell felfogni, mintha a forrásdokumentum Infoset elemein zajlana le. Tehát a rendezett elem gyerekelemeire illeszkedő template-ek már a rendezett listát fogják látni.
- Alapszintaxis:
  `<xsl:sort select="`*`XPath kifejezés`*`"/>`
- Jelentése: vedd a szülődirektíva (apply-templates vagy for-each) eredményhalmazát, értékeld ki minden elemre az xsl:sort select attribútumában megadott XPath kifejezést és rendezd őket sorba. Az eredményhalmazt feldolgozó többi template ezek után már rendezett sorrendben látja az elemeket.
- Egymás után több xsl:sort is megadható, ekkor sorrendben több kulcs szerint is lehet rendezni (első xsl:sort: elsődleges kulcs, második xsl:sort: másodlagos kulcs).

Rendezés, 2.

- Adva van a következő forrásdokumentum.

```
<?xml version="1.0"?>
<modules>
  <module>
    <name>Module #1</name>
    <version>
      <major>0</major>
      <minor>2</minor>
     </version>
  </module>
  <module>
    <name>Module #2</name>
    <version>
      <major>2</major>
      <minor>4</minor>
     </version>
  </module>
  <module>
    <name>Module #3</name>
    <version>
      <major>0</major>
      <minor>1</minor>
     </version>
  </module>
</modules>
```

Rendezés, 3.

- Feladat: írjuk ki a modulok neveit verziószám szerint növekvő sorrendben.
- Stíluslap:

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

<xsl:template match="modules">
   <xsl:apply-templates select="module">
      <xsl:sort select="version/major"/>
      <xsl:sort select="version/minor"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="module">
  <xsl:text>
  </xsl:text>
  <xsl:value-of select="name"/>
</xsl:template>

</xsl:stylesheet>
```

Rendezés, 4.

- order attribútum: "ascending" - növekvő sorrend, "descending" - csökkenő sorrend, az "ascending" alapértelmezett.
- data-type attribútum: a rendező kulcs típusa. Lehetőségek:
  - text - ábécésorrendben rendez. Ez az alapértelmezett.
  - number - a kulcsot számként kezeli és e szerint rendez. Igazából ezt kellett volna használni az előző példában.
  - QName - speciális adattípus amit a QName azonosít, az XSLT specifikáció nem foglalkozik vele.
- case-order: "text" adattípus esetén meghatározza, hogy a nagybetűk vagy a kisbetűk legyenek előbb a rendezési sorrendben. Lehetőségek: upper-first, lower-first, az alapértelmezés függ a nyelvtől (angol: kisbetű előbb).

Változók

- Két típusa van:
  - xsl:param - már láttuk, az xsl:call-template-hez használatos.
  - xsl:variable - általános változó.
- Az xsl:param csupán annyiban különbözik, hogy az ő értéke csupán alapértelmezett érték és a template-hívás során felülíródik az aktuális paraméterekkel (ha átadtak ilyen nevű paramétert az xsl:with-param direktívával).
- Mindkét direktívának kötelező "name" attribútuma van.
- Értékadás két módja:
  - <xsl:variable name="varname" select="*XPath kifejezés*"/> - a "varname" nevű változó értéke az XPath kifejezésnek megfelelő értékű és típusú lesz.
  - <xsl:variable name="varname">
    <!-- template -->
    </xsl:variable>
    A "varname" változó értéke a template-nek megfelelő *eredményfa részlet* lesz, ami az XPath típusokhoz képest új.

Változók, 2.

- Eredményfa részlet: a string és csomópont-halmaz különös hibride. Ilyen típusokon csak a string típuson megengedett műveleteket lehet használni (pl. a /,//,[] nem megy rajta), ha viszont az eredményfába másoljuk a megfelelő direktívával, XML részlet keletkezik (nem csak szöveges csomópontok).
- Ügyeljünk a különbségre!

```
<xsl:variable name="varname">2</xsl:variable>
<xsl:variable name="varname" select="2"/>
```

  Az első esetben a változó egy eredményfa-részletet tartalmaz, amely egyetlen szöveges csomópontból áll, melynek értéke "2". A második esetben típusa XPath number típus, értéke numerikus 2. Az automata konverzió miatt a különbség nem mindig látható, de néha bajba kerülhetünk vele.
- Eredményfa-részlet eredményfába szúrása: xsl:copy-of direktíva. Ennek select attribútuma egy csomópont-halmazt produkál, ami XML elemekként szúródik a fába. Figyelem: az xsl:value-of szöveges elemeket hoz létre, tehát az eredményfa-részletet szövegessé konvertálja, úgy szúrja a fába. Az xsl:copy-of ekvivalens az xsl:value-of-fal, ha a select attribútuma szöveges vagy szám típusú értéket ad eredményül.

Változók, 3.

- Illesszük bele a lenti dokumentum <item> elemeibe <itemnum> elemeket, melyeknek tartalma legyen az <item> elem sorszáma!
- Forrásdokumentum:

```
<?xml version="1.0"?>
<items>
  <item/>
  <item/>
  <item/>
</items>
```

- Stíluslap:

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

<xsl:template match="item">
  <xsl:variable name="treefrag">
    <itemnum>
      <xsl:number format="1." level="single"/>
    </itemnum>
  </xsl:variable>
  <xsl:copy>
    <xsl:copy-of select="$treefrag"/>
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Változók, 4.

- A változók és paraméterek két szinten lehetnek a stíluslapokban.
  - Stíluslap szinten, az xsl:stylesheet gyerekeiként. Ekkor a változó vagy a paraméter az összes template-ben látszik. Stíluslap szintű paraméter értékének megadására a szabvány nem tartalmaz rendelkezést. (Tipikusan az XSLT feldolgozó hívásakor adódnak át).
  - Template szinten, ekkor csak az adott template-ben láthatóak.
- Template szintű változó elfedhet azonos nevű stíluslap szintű változót, de általános esetben azonos nevű változók használata tilos.
- Ez azt jelenti, hogy az XSLT változó-koncepciója szerint a változónak csak egyszer (az inicializálásakor) lehet értéket adni.

Kimenet formázása

- Az XSLT feldolgozó három eredménytípust képes generálni: XML, HTML és egyszerű szöveges típust.
- Az alapértelmezett eredménytípus az XML. A feldolgozó azonban automatikusan HTML kimeneti formátumra vált, ha a következők mindegyike teljesül:
  - A gyökérelemnek van legalább egy gyerekeleme.
  - A gyerekelem előtt levő esetleges szöveges csomópontok csak "fehér karaktereket" tartalmaznak.
  - A gyökérelem első gyerekelemének lokális neve "html".
- Egyébként az xsl:output direktívával állítjuk be a kimeneti formátumot. Ennek "method" attribútuma megfelel a kimeneti formátumnak ("xml", "html", "text" értelem szerint). Ezen felül a "method" értéke tetszőleges QName is lehet. Ez a QName a szabványban nem specifikált kimeneti formátumot állít be.
- Az xsl:output csak stíluslap-szintű elem lehet.

Kimenet formázása, 2.

- Példa: "book" gyakorlat a 49-50. dián.
- <xsl:output method="xml"/>; eredmény:
```
<?xml version="1.0" encoding="UTF-8"?><html><body> ...
</body></html>
```
- <xsl:output method="html"/>; eredmény:
```
<html>
<body>

<table>
<th>Book title #1</th>
...
</html>
```
- <xsl:output method="text"/>; eredmény:
```
Book title #1Book publisher #11999Book publisher #12005
Book title #2Book publisher #22001
```
- Ebben a példában kimeneti formázás beállítása nélkül a "html" az alapértelmezett, mert az eredményformátum megfelel a "html"-re alapértelmezésben átváltó formátumnak.

Kimenet formázása, 3.

- XML eredménytípus:
  - encoding attribútum - az eredménydokumentum karakterkészlete. UTF-8 és UTF-16 kötelezően támogatandó, a többi az implementációtól függ.
  - indent attribútum - ha az értéke "yes", a kimeneti XML dokumentumhoz "fehér karaktereket" lehet adni, hogy a kinézete "szép" (nice, XSLT szabvány) legyen. Az alapértelmezés "no" és gond lehet kevert tartalom (elemek és szöveges csomópontok keverve) esetén. Előző példánkban indent="yes":

```
<?xml version="1.0" encoding="UTF-8"?>
<html>
<body>
  <table>
<th>Book title #1</th> ...
```

  - cdata-section-elements atribútum: értéke azoknak a neveknek szóközökkel elválasztott felsorolása, amelyeknek tartalmát CDATA formában kell kiírni. A kimeneti elemekre vonatkozik. Példa: <xsl:output method="xml" cdata-section-elements="th"/>; eredmény:

```
<?xml version="1.0" encoding="UTF-8"?><html><body>
  <table><th><![CDATA[Book title #1]]></th> ...
```

Kimenet formázása, 4.

- HTML eredménytípus:
  - A HTML eredménytípus nem generál zárótag-et az ismert üres HTML elemeknek. Ezek HTML 4.0 esetén: area, base, basefont, br, col, frame, hr, img, input, isindex, link, meta és param. Ebben az esetben az eredményfába szúrt <br/> elem a kimeneten egyszerű
-ként jelenik meg zárótag nélkül. Az XSLT feldolgozó ezeket a speciális elemeket kis- és nagybetűs formában is fel kell ismerje (pl.
, <BR/>).
  - script és style HTML-elemekre az escape-elés ki van tiltva. Pl.
    Eredményfa: <script>if (a < b) foo()</script>
    Kimenet: <script>if (a < b) foo()</script>
  - Az encoding attribútum támogatott és ha a dokumentum tartalmaz HEAD elemet, az XSLT feldolgozó hozzáadja a kódolás specifikációját a következő formában:

```
<HEAD>
<META http-equiv="Content-Type" content="text/html;
charset=EUC-JP"> ..
```

Kimenet formázása, 5.

- Szöveges eredménytípus:
  - Az eredményfa minden szöveges csomópontjának (tehát nem az elemeinek és attribútumainak!) szöveges értékét kiírja az eredménydokumentumba minden további escape-elés nélkül.
  - encoding attribútum támogatott.
- Kimeneti dokumentum escape-elése
  - Normálisan az XML dokumentumtípus a kimeneti dokumentumot escape-eli, tehát egy literális "<" karakter <-ként íródik az eredménydokumentumba.
  - A disable-output-escaping attribútummal ez a viselkedés xsl:text és xsl:value-of direktívákra megtiltható (csak egy direktívára). Példa: `<xsl:text disable-output-escaping="yes">&lt;</xsl:text>` -> egy darab "<" karaktert generál.

"Fehér karakterek" kezelése

- Főszabályok:
  - "Fehér karaktereket" sose távolít el az XSLT elemző a forrásdokumentumból.
  - A stíluslapból a "fehér karaktereket" kiszedi a lentebb leírt szabályok szerint.
  - "Fehér karaktereket" adhat az eredménydokumentumhoz az xsl:output indent attribútuma.
- "Fehér karakterek" eltávolítása stíluslapokból:
  - Ilyen karaktereket megőrző elemekben mindig megmaradnak. Alapértelmezésben ez kizárólag az xsl:text direktíva.
  - A teljesen fehér karaktereket tartalmazó szöveges csomópontokat eltávolítják a stíluslapból. Emlékeztető: az XSLT feldolgozó összeolvasztja az egymás mellett levő szöveges csomópontokat.
  - Fehér karaktereket megőrző (xml:space="preserve" attribútummal rendelkező) elemekből sose távolít el szóközt.
- Ezen felül definiálni lehet szóköz-eltávolító és szóköz-megőrző elemeket az xsl:strip-space és az xsl:preserve-space direktívákkal.

"Fehér karakterek" kezelése, 2.

- Példa: szóközök forrás XML dokumentumban, másoló stíluslap:
  Forrás:
```
<?xml version="1.0"?>
<doc>  <item/>  </doc>
```
  Eredmény:
```
<?xml version="1.0" encoding="UTF-8"?><doc>  <item/>  </doc>
```
- 2. példa: "fehér karakterek" stíluslapban (megmaradnak, mert nem fehér karakterek is vannak az összeolvasztott szöveges csomópontban):
```
<p>
  Hello
</p>
```
- 3. példa: "fehér karakterek" stíluslapban, ebből egyetlen <p/> tag lesz.
```
 <p>


</p>
```

Import és include

- Egy stíluslap kétféle módon használhat fel egy másik stíluslapot.
  - xsl:include direktíva - egyszerűen beszúrja a felhasználó stíluslap XML fájába. Pl. <xsl:include href="otherst.xml"/>
    - Az xsl:stylesheet-et eltávolítja a beszúrt stíluslapból, csak az xsl:stylesheet gyerekei lesznek beszúrva az xsl:include-ot használó stíluslapba.
    - Ha a beszúrt stíluslap ugyanolyan nevű template-eket vagy stíluslap-szintű változókat használ, mint az xsl:include-ot tartalmazó stíluslap, az hiba.
  - xsl:import direktíva - majdnem ugyanaz, mint az xsl:include, de template vagy változónév egyezés esetén az importáló stíluslapnak előnye van az importálttal szemben.
    - Pl. ha mindkét stíluslap tartalmaz egy <template match="*ˇ"> ...template-et, akkor az importálóstíluslap definíciója lesz érvényes.

current(), document()

- Az XSLT szabvány újabb függvényekkel bővíti az XPath függvénykészletét.
- `node-set current()` - Visszaadja az aktuális csomópontot, mint az eredmény csomópont-halmaz egyetlen elemét. Ez nem mindig egyezik meg a kontextus-csomóponttal, pl. ha a template belsejében egy XPath kifejezésben használjuk, ami már megváltoztatta a kontextus csomópontot. Példák:
  ```
  <xsl:apply-templates
  select="//glossary/item[@name=current()/@ref]"/>
  ```
  nem egyenlő a
  ```
  <xsl:apply-templates select="//glossary/item[@name=./@ref]"/>
  ```
  kifejezéssel. Az utóbbi a
  ```
  <xsl:apply-templates select="//glossary/item[@name=@ref]"/>
  ```
  kifejezéssel egyezik meg.
- `node-set document( object )` - Az aktuális forrásdokumentumtól eltérő XML forrásdokumentum feldolgozását teszi lehetővé. Paramétere egy URI, ahonnan beolvassa a forrásdokumentumot és csomópont-lista formájában teszi elérhetővé.
  - A visszaadott csomópont-halmaz a forrásdokumentum legfelső szintű elemétől indul, de tartalmazza az összes csomópontot, mint a legfelső elem gyerekeit (útvonalkijelölőkkel a szokásos módon feldolgozható).
  - Ez relatív URI az aktuális stíluslaphoz képest. Pl. document("") maga a stíluslap.

current(), document(), 2.

- Feladat: adott egy forrásdokumentum, amelynek legfelső szintű eleme "doc".
- Adva van egy másik XML dokumentum, amelynek neve searchandreplace.xml, amely megadja, milyen elemeket milyen másik elemre kell kicserélni. Példa:

```
<?xml version="1.0"?>
<searchandreplace>
  <item>
   <sritem rep="attr"/>
  </item>
  <item2>
    <sritem2 rep="attr2"/>
  </item2>
</searchandreplace>
```

  A példában pl. az <item> elemet <sritem rep="attr"/> elemre kell kicserélni (az <item> attribútumai és tartalma elvesznek).
- Az elemek, melyek ebben a fájlban nincsenek felsorolva, nem kerülnek át a kimenetre.
- Írjunk stíluslapot, ami ezt a konverziót elvégzi!

current(), document(), 3.

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

<xsl:variable name="sr"
select="document('searchandreplace.xml')/searchandreplace/*"/>

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

<xsl:template match="*">
   <xsl:variable name="tnode" select="current()"/>
   <xsl:for-each select="$sr">
      <xsl:if test="name( current() ) = name( $tnode )">
          <xsl:copy-of select="./*"/>
       </xsl:if>
    </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

xsl:message

- A leghasznosabb direktíva: példányosításakor a tartalmát kijelzi valamiképpen a felhasználónak (pl. konzol, üzenetablak, stb.) Példa:
  ```
  <xsl:message><xsl:value-of
  select="name(current())"/></xsl:message>
  ```
  (kiírja az aktuális csomópont nevét a konzolra)
- terminate attribútum - ha az értéke "yes", az üzenet kiküldése után a feldolgozás megszakad. Alapértelmezés: no.

Gyakorlat

- Bővítse ki az előző példát általános keresés-és-kicserélés alkalmazássá! Tehát a stíluslap végezze el a kicserélést a forrásdokumentum akármelyik elemszintjén és a searchandreplace.xml fájlban nem szereplő elemeket hagyja érintetlenül!

Megoldás

- Első lépésben stíluslapot generálunk a searchandreplace.xml fájlból.
- Második lépésben ezt a stíluslapot alkalmazzuk a forrásdokumentumra.

Megoldás, 2.

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

<xsl:namespace-alias stylesheet-prefix="axsl" result-prefix="xsl"/>
<xsl:output method="xml" indent="yes"/>

<xsl:template match="searchandreplace">
  <xsl:comment>This file is automatically generated. Do not
edit!</xsl:comment>
  <axsl:stylesheet
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <axsl:template match="*">
    <axsl:copy>
      <axsl:apply-templates/>
    </axsl:copy>
  </axsl:template>
  <xsl:for-each select="./*">
    <axsl:template match="{name( current() )}" priority="1">
      <xsl:copy-of select="./*"/>
    </axsl:template>
  </xsl:for-each>
  </axsl:stylesheet>
</xsl:template>

</xsl:stylesheet>
```

Megoldás, 3.

- Generált stíluslap:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This file is automatically generated. Do not edit!-
->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="*">
<xsl:copy>
<xsl:apply-templates/>
</xsl:copy>
</xsl:template>
<xsl:template match="item" priority="1">
<sritem rep="attr"/>
</xsl:template>
<xsl:template match="item2" priority="1">
<sritem2 rep="attr2"/>
</xsl:template>
</xsl:stylesheet>
```

Megoldás, 4.

- Forrásdokumentum:

```
<?xml version="1.0"?>
<doc>
  <item2/>
  <item/>
  <item/>
  <item3/>
</doc>
```

- Erre alkalmazva a generált stíluslapot az eredmény:

```
<?xml version="1.0" encoding="UTF-8"?><doc>
<sritem2 rep="attr2"/>
  <sritem rep="attr"/>
  <sritem rep="attr"/>
  <item3/>
</doc>
```

Héder Mihály megoldása: egy lépésben

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="sr" select="document('searchandreplace.xml')/*"
/>
  <xsl:template match="*">
       <xsl:choose>
            <!-- ezt cserelni kell -->
            <xsl:when test="$sr/*[name() = name(current())]">
                <!-- uj elem letrehozasa -->
                <xsl:element name=
                    '{name($sr/*[name() = name(current())]/*[1])}'>
                    <!-- attributumok masolasa -->
                    <xsl:copy-of select=
                     "$sr/*[name() = name(current())]/*[1]/@*"/>
                    <!-- a gyermek elemeket is feldolgozzuk -->
                    <xsl:apply-templates/>
                </xsl:element>
            </xsl:when>
            <!-- ez marad az eredeti -->
            <xsl:otherwise>
                <xsl:copy>
                    <!-- Az elem es az attributumok lemasolasa majd
tovabb a gyermekek fele -->
                    <xsl:copy-of select="@*"/>
                    <xsl:apply-templates/>
                </xsl:copy>
            </xsl:otherwise>
       </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```