XSLT szabványy

XSLT nyelv

Az előadás anyaga

Kovács László: Adatkezelés XML környezetbe,

XML technikák II. és további irodalom

alapján készült el

Témakör kérdései

1. XSLT szabvány áttekintése
2. XSLT nyelv
3. Az XSLT működési elve
4. Az XSLT parancsai
5. Mintafeladatok

Igényelt kompetenciák

- XSLT szabvány megismerése
- XSLT nyelv
- Az XSLT működési elvének elsajátítása
- Az XSLT parancsai
- Mintafeladatok
- Környezet: XML szerkesztő (Oxygen, EditIX, Eclypse, ….)

„XML nyelv - felhasználási területek

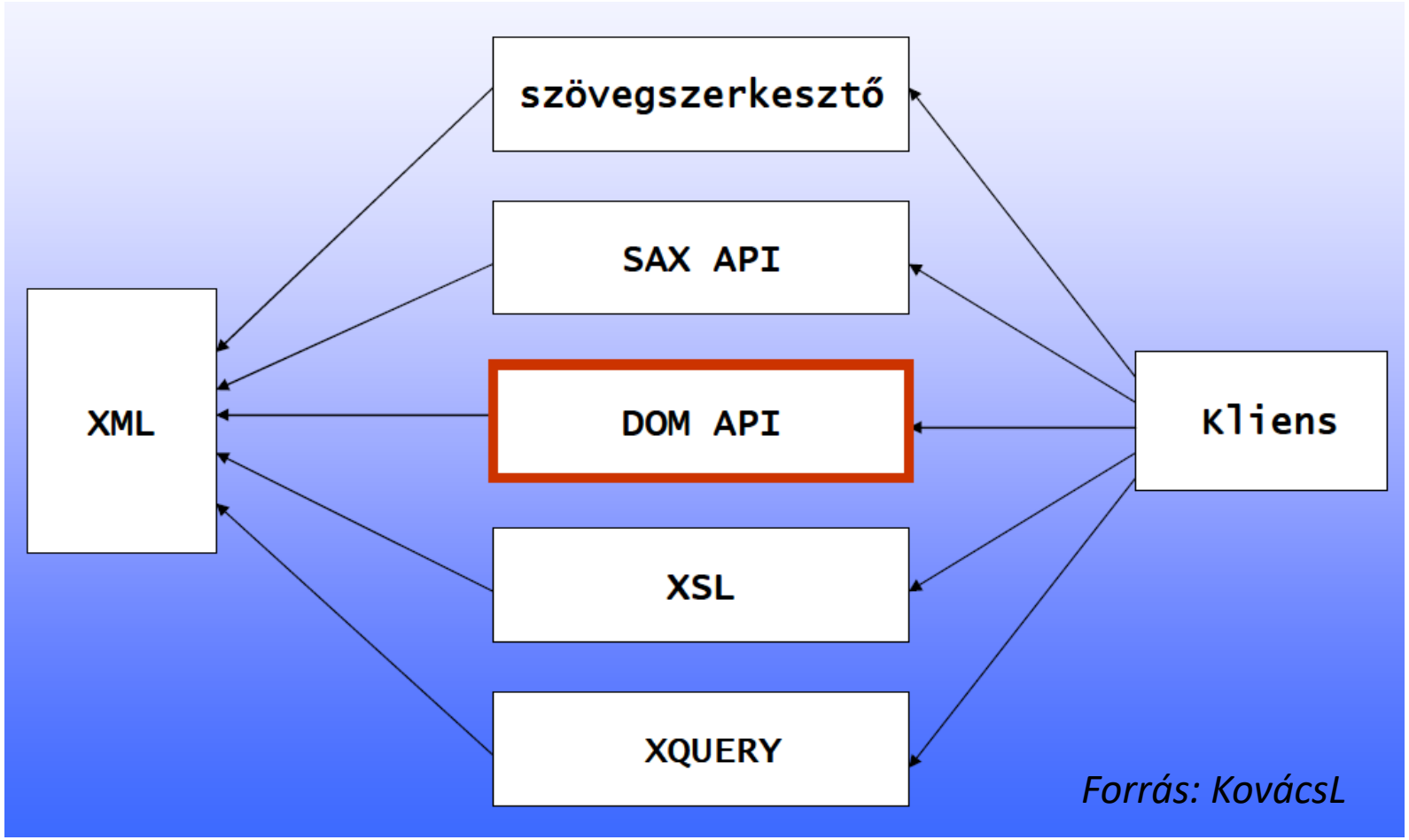
- szövegek reprezentációja és feldolgozása,
- *webes megjelenés* (szerver és kliens oldali transzformációk),
- *adatcsere (formátum, transzformáció)*,
- Web 2.0,
- technikai dokumentációk nyelvezete,
- *szoftverek konfigurálása*,
- felhasználói interfészek definiálása,
- EU önéletrajzok készítése (Europass).

XML adatok kezelési lehetőségei

Magával az XML-lel kapcsolatos specifikációk:

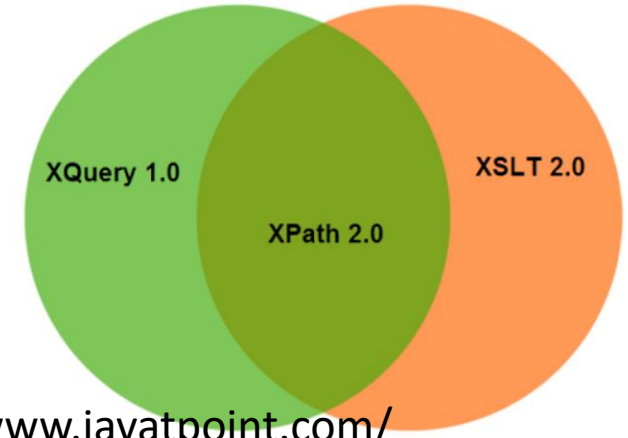
- Az XML lehetőségeit bővítik.
- Lehetővé teszik XML dokumentumok *szerkezetére* és *tartalmára vonatkozó megszorítások kifejezését* (*XML sémanyelvek).*
- Lehetővé teszik XML dokumentumokból *információ kinyerését (lekérdező nyelvek).*
- Lehetővé teszi XML dokumentumok *más formába alakítását (transzformációs nyelvek).*

XML adatok kezelési lehetőségei”

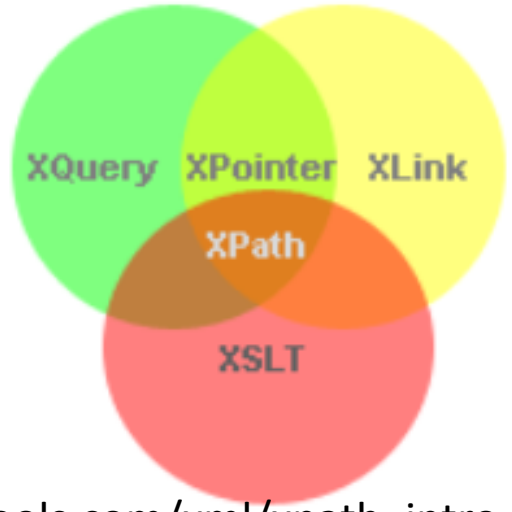


XSLT nyelv helye specializációk között

- „Az *XPath* különböző típusú *kifejezéseket* használ az információk lekérésére az XML dokumentumból.
- Az *XSLT parancsokat és* az *XPath kifejezéseit* használja fel egy XML dokumentum egyes részeinek meghatározásához.”



Forrás: https://www.javatpoint.com/



https://www.w3schools.com/xml/xpath_intro.asp

XML nyelv áttekintése

„Az XML formátum az általánosságából következően rendkívül *széles alkalmazási területtel* bír.

Az XML alkalmas arra, hogy *adatbázisként szolgáljon*, *paraméter adatokat tároljon*, *dokumentumokat* vagy *programokat* írjon le.

Pl.: egy *Neptun rendszerben az* XML szolgálhat a *hallgatók adatainak nyilvántartására*.

XML nyelv áttekintése – XSLT nyelv

Megjelent egy újabb igény: az XML dokumentumok *tartalmának hatékony konverziójára*.

Ez pedig az amikor *egy **forrás XML** dokumentumból* egy ***cél XML** dokumentumot állítunk* elő.

A *DOM modell*, alkalmas lehet ilyen konverzió elvégzésére, az *egyedi procedurális (*utasításokat eljárásokba csoportosítja) *konverzió kódolás* ellen a *nagy járulékos költség szól*.

XML - XSLT nyelv áttekintése

Minden *procedurális programfejlesztési* munkánál igen jelentős rész a *validálási, ellenőrzési fázis*.

Másrészt, egy *alacsony szintű leírást* sokkal *nehezebb módosítani, továbbfejleszteni*.

Tehát, célszerűbb egy *magasabb szintű, un.* **`imperatív`** (parancsoló (hang), kényszerű, elkerülhetetlen, kényszerítő) **`parancsnyelv használata`**, amelynél külön előnyt jelent az *XML formátumra való illeszkedés*.

XSLT nyelv - fogalma

Az XML dokumentum *konverziós nyelv* az - *XSLT elnevezést* kapta, mely az

Extensible Stylesheet Transformation Language (Bővíthető Stílus leírás Transzformációs Nyelve) elnevezésből származik.

Fogalma: az XML dokumentumok más XML dokumentummá való transzformálását teszi lehetővé.

Mi az XSLT?

- Az XSLT az *XSL Transformations* rövidítése.
- Az XSLT az XSL legfontosabb része.
- Az XSLT egy XML-dokumentumot egy másik XML-dokumentummá alakít át.
- Az XSLT *xPath segítségével* navigál az XML dokumentumokban.
- Az XSLT egy W3C ajánlás.”

XSLT nyelv története – 1.0

„XSL: eXtensible Stylesheet Language.
XSLT: XSL Transformation.

Az XSLT nyelv első verziója 1999-ben jelent meg a Word Wide Web Consortium (W3C) támogatásával.

James Clark (amerikai bankrabló volt) alkotta meg az XSLT 1.0 verziót, ami 1999. november 16-án lépett W3C ajánlás státuszba.

URL: https://www.w3.org/TR/xslt-10/

XSLT nyelv története – 1.0

A fejlesztés egyik fontos lépése volt, amikor a *tartalom* és a *forma szeparálhatósága*, függetlensége.

Ennek első lépései a stíluslapok (CSS) használatához kötődnek.

A *tartalom módosítására* irányuló *konverziós nyelvként* jött létre a *XSLT nyelv.*

Az XSLT rövid jellemzéseként azt mondhatjuk, hogy *egy XML* dokumentumot *másik XML* dokumentumba konvertál át.

XSLT nyelv története – 1.1

Az *XSL munkacsoport* a 2001-ben megpróbálta kifejleszteni az 1.1 verziót, de nem sikerült, majd összefogtak az *XQuery* munkacsoporttal és megalkották az *XPath 2.0-t* gazdagított *adat modellel* és *XML sémára* alapú *típusrendszerrel.*

Az *XSLT által* használt adatmodell megegyezik az *XPath által* használt adatmodell-el – némi kiegészítésekkel.

URL: https://www.w3.org/TR/2001/WD-xslt11-20010824/#data-model

XSLT nyelv története – 2.0

- Az XSLT 2.0 verzió 2007. január 23-án lépett W3C ajánlás státuszba.
- A *Michael Kay*-féle XSLT 2.0 verzió 2002 és 2007 között fejlődött ki.

URL: https://www.w3.org/TR/xslt20/

XSLT nyelv története – 3.0

Az XSLT 3.0 verzió 2017. június 8-án - W3C ajánlás státuszba.

Michael Kay, Saxonica http://www.saxonica.com/

URL: https://www.w3.org/TR/xslt-30/

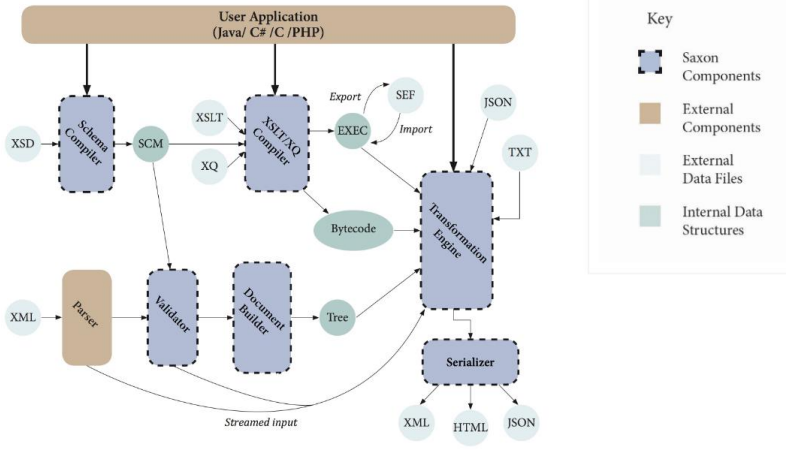
Két jelentős változás:

- Nem tárolódik *teljesen a memóriában* a *forrás* ill. *eredmény dokumentum.*
- *Másik:* a nagy *stíluslapok modularitásának javítása*, ami azt jelenti, hogy a stíluslapok önállóan fejleszthetők.”

XSLT nyelv története – 3.

„Saxon EE architecture: fő összetevői és adatfolyamok

Guide to terminology



- *Bytecode*: Java VM által értelmezett kód
- *EXEC:* Belső lefordított kifejezésfa
- *JSON:* File in JSON format
- *SCM:* Schema Component Model (összeállított séma)
- *SEF:* Stylesheet Export File (compiled stylesheet)

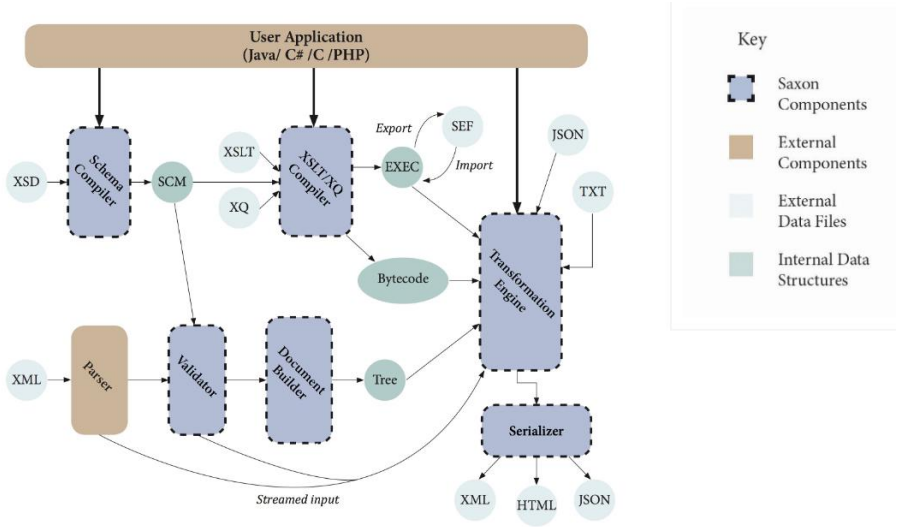
https://www.saxonica.com/technology/architecture.xml

XSLT nyelv története – 3.

Saxon EE architecture: fő összetevői és adatfolyamokat

Guide to terminology

- *Tree:* XML dokumentum faábrázolása
- *TXT:* Plain text file
- *XML:* eXtensible Markup Language
- *XQ:* XQuery query
- *XSD:* XML Schema Document
- *XSLT:* eXtensible Stylesheet Language Transformations”



https://www.saxonica.com/technology/architecture.xml

XSL nyelv - részei

„Az XSL az *eXtensible Stylesheet Language* és ez alkalmas arra, hogy *megformázzuk az XML-t*.

Valójában az XSL három (négy) részből áll:

- **XSLT** (XSL Transformations), XSL átalakítások - ezzel tudjuk az XML-t más formátummá alakítani, pl.: HTML, XML, TEXT.
- **XSL-FO** (XSL Formatting Objects): XML formázások (2013-tól a helyét átvette a CSS3),

XSLT nyelv - részei

- **XPath**: XML navigációra szolgál.
- **XQuery**: XML lekérdezéseket tudunk végrehajtani.

XSLT nyelv jellemzői

Dokumentum-fa modell (de nem DOM).

- egyszerű,
- nem módosítható (van forrás és cél külön).

Rugalmasabb *dokumentum-fa* feldolgozás.

- *rugalmas navigáció* (a fában több irányban is lehet haladni),
- *gazdagabb csomópont-kezelés*,
- az *elemek* mellett más *csomópont típusok* is vannak,

XSLT nyelv jellemzői

- a tartalom és struktúra szétválasztása,
- számított elemek használata.
- Az XPath szabványra épít.
- XML formátumú.
- Procedurális elemeket is tartalmaz.

XSLT nyelv - jellemzői

Az XSLT-hez kapcsolódó állományok:

- forrás XML dokumentum,
- a transzformációt leíró dokumentum (rendszerint XSL kiterjesztéssel),
- eredményt tartalmazó XML dokumentum.

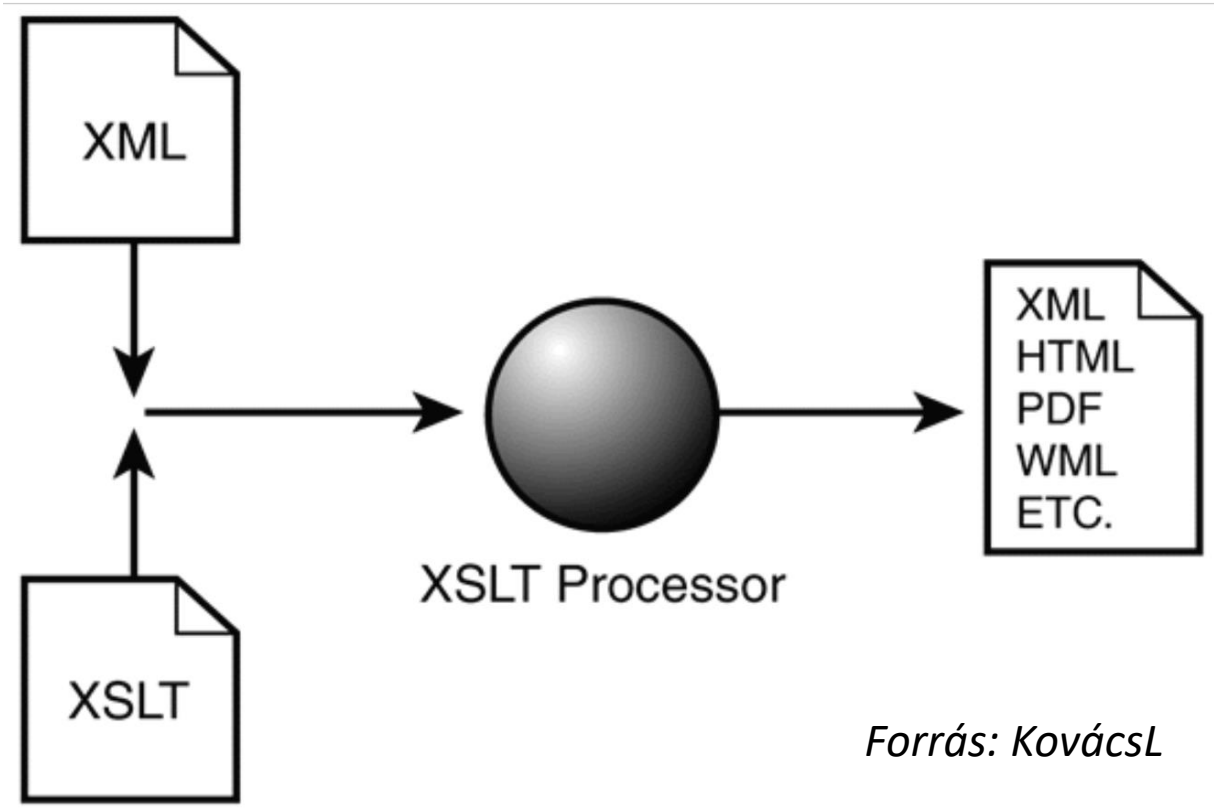
Az XSLT egy *imperatív nyelv* mivel *nem elemi algoritmusokat kell kidolgozni a transzformáció megadására*, hanem ún. *magasabb szintű* ***mintákkal*** lehet dolgozni.

XSLT nyelv - jellemzői

- Az XSLT nyelvet sokan a *funkcionális nyelvekhez (LISP, Scala, Erlang, Clean, F#)* sorolják, mert ott is *mintákat definiálunk* a programban.
- Az XSLT program *minták listája*, ahol minden minta megadja, hogy *egy adott dokumentumrészt milyen más alakra kell transzformálni*.
- A minták célja a dokumentum érintett részeinek kijelölése.

XSLT nyelv - kapcsolódó állományok

Az XSLT forrás és eredmény objektumai.



Forrás: KovácsL

XSLT - DOM eltérés

Az XSLT nyelv a dokumentumot - *dokumentum faként kezeli.*

Az *XSLT adatszemlélete* az *XDM modellen* alapul.

Eltérés a kétféle fa értelmezés között:

- az XSLT-fa egyszerűbb, kevesebb részletre tér ki,
- a DOM fa módosítható, az XSLT fa *csak olvasásra* vagy csak *írásra szolgál*,
- az XSLT-fa kezelésnél egyszerűbb *szinkronizálós feladatokra* van szükség.

XSLT nyelv - fejlesztőkörnyezet

Az XSLT használatához három elemre van szükség:

- forrás XML állomány,
- transzformációt leíró XSLT állomány,
- XSLT értelmező.

Fejlesztő környezet az *Oxygen XML szerkesztő*.

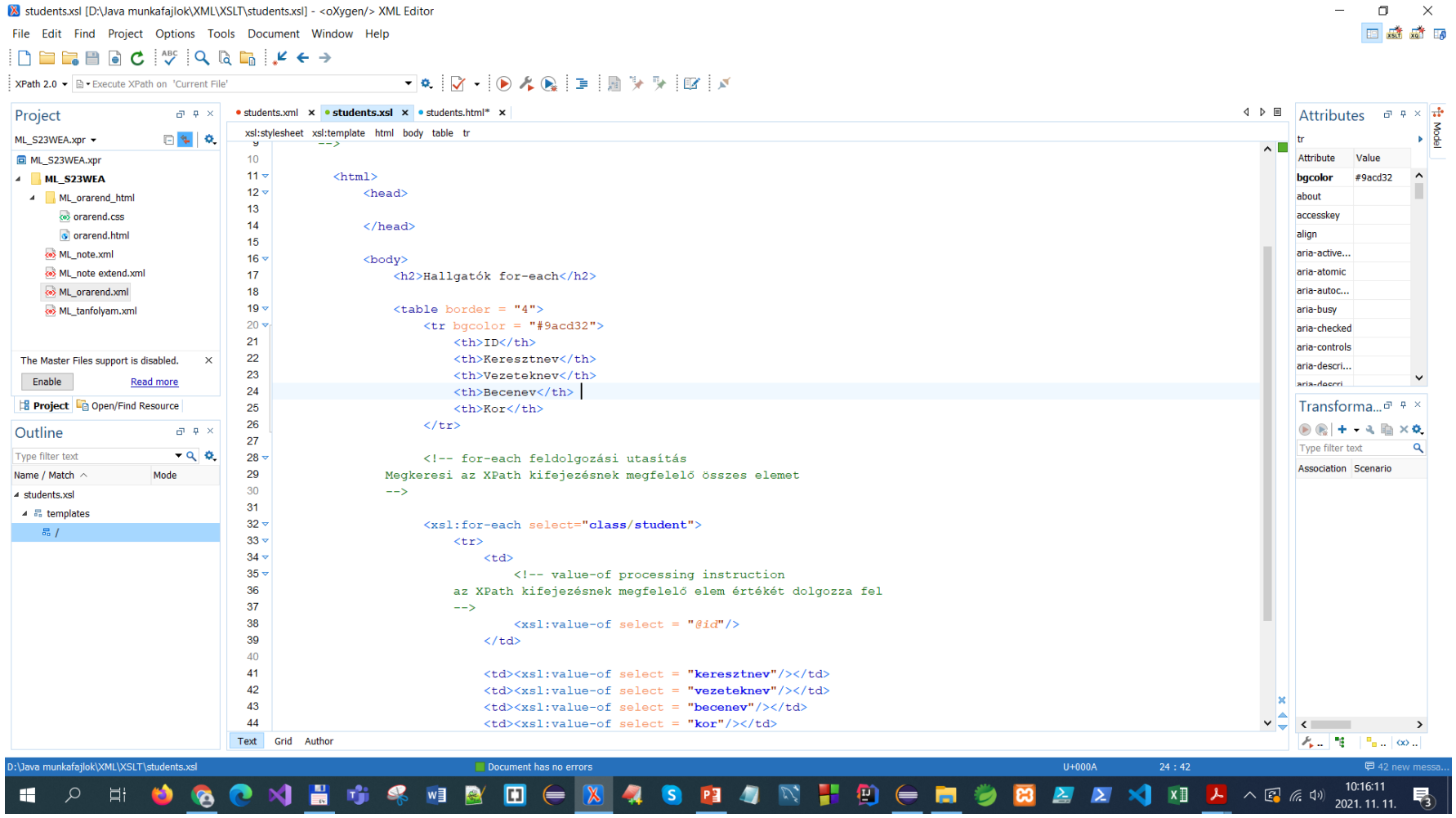
Előnye: hogy az XSLT program megírásánál nagy támogatást kapunk a *rendelkezésre álló parancsok* és a *nyelvhelyességet* illetően.

XSLT nyelv - fejlesztőkörnyezet

Az *Oxygen szerkesztő* esetében az alábbi lépéseken keresztül tudjuk kipróbálni az XSLT működését:

- *forrás állomány létrehozása: File* menüpont *New* (XML) alpontján keresztül,
- *transzformáció leíró állomány létrehozása*: *File* menüpont *New* (XSL Stylesheet) alpontján keresztül,
- *transzformáció végrehajtása*: *Document* menüpont *Transformation* alpontján keresztül.

Oxygen fejlesztőkörnyezet



URL: https://www.oxygenxml.com/download.html

Eclipse IDE fejlesztőkörnyezet

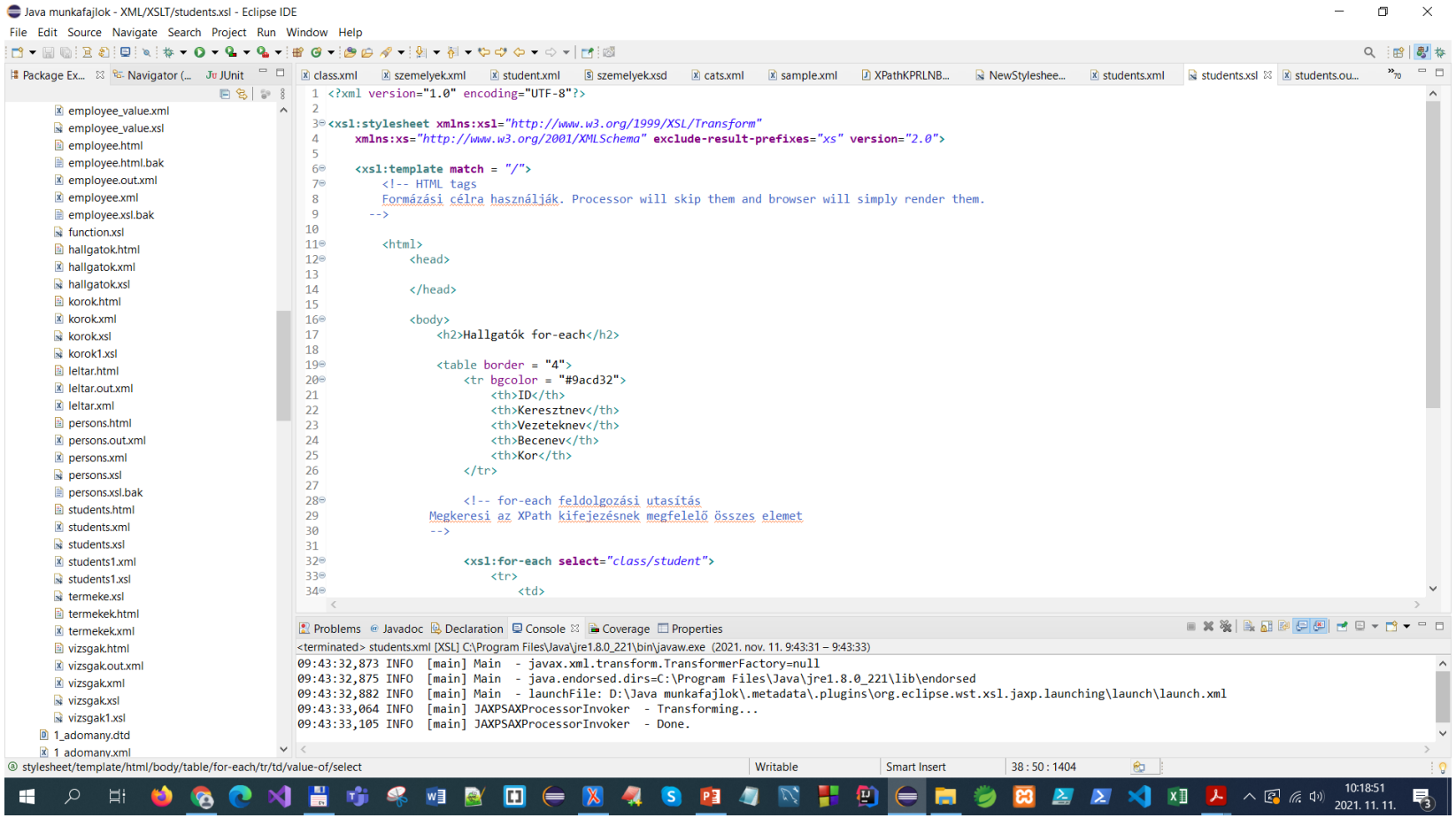
A világhálón több *ingyenes XSLT értelmező* is elérhető.

A legtöbbet használt termék ezek közül az *Eclipse IDE for Java Developers* keretrendszer és a *Microsoft Edge* böngészője.

Az Eclipse komplett *XML adat és séma szerkesztővel* bír, míg a böngésző *beépített XSLT értelmezővel* és *séma ellenőrzővel rendelkezik*, melyekkel az elkészült XML állományok kipróbálhatók.

URL: https://www.eclipse.org/downloads/packages/

Eclipse IDE fejlesztőkörnyezet



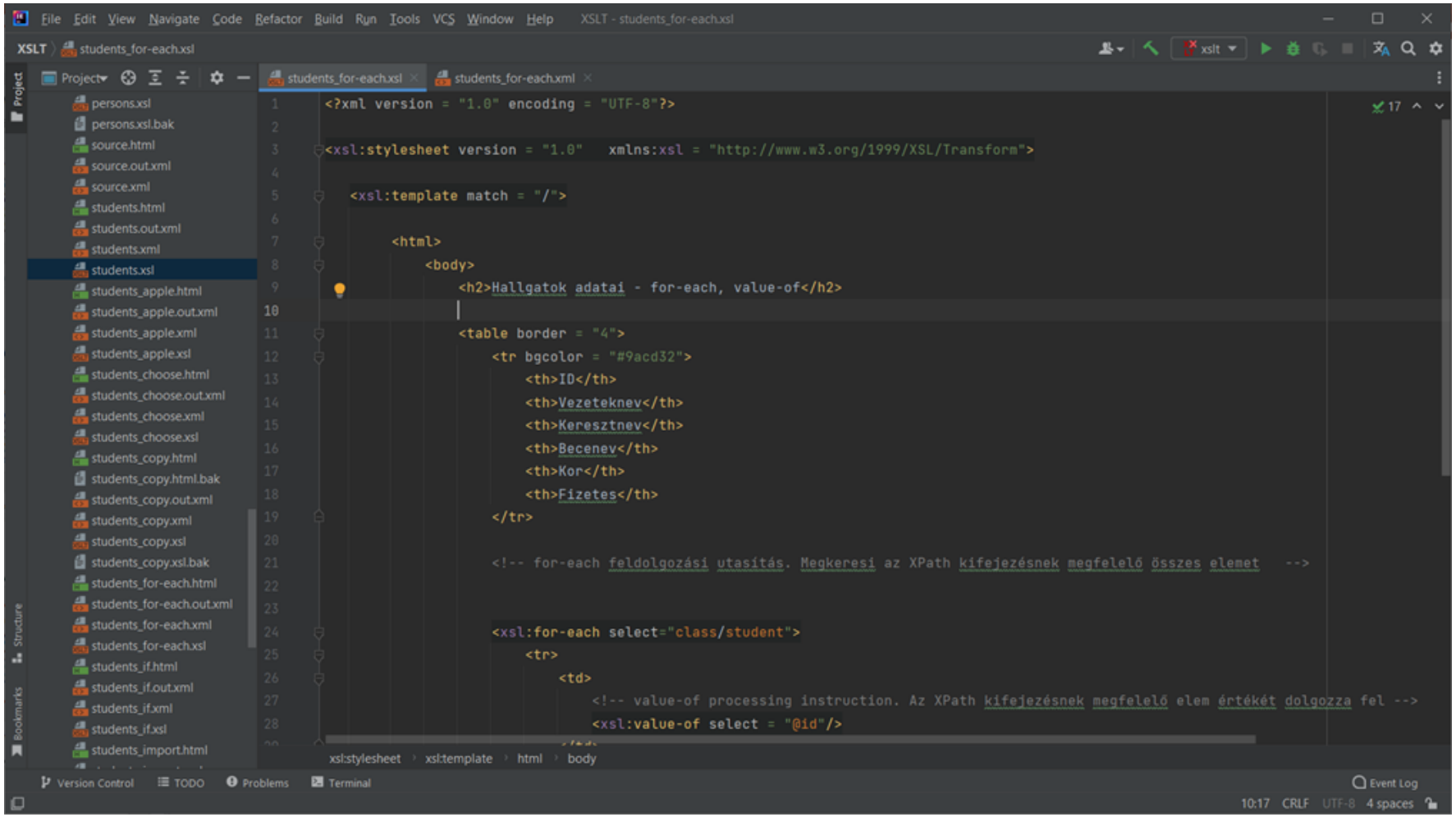
IntelliJ IDEA fejlesztőkörnyezet

Az *IntelliJ IDEA* az egyik leggyakrabban használt fejlesztői környezet a Java fejlesztők körében.

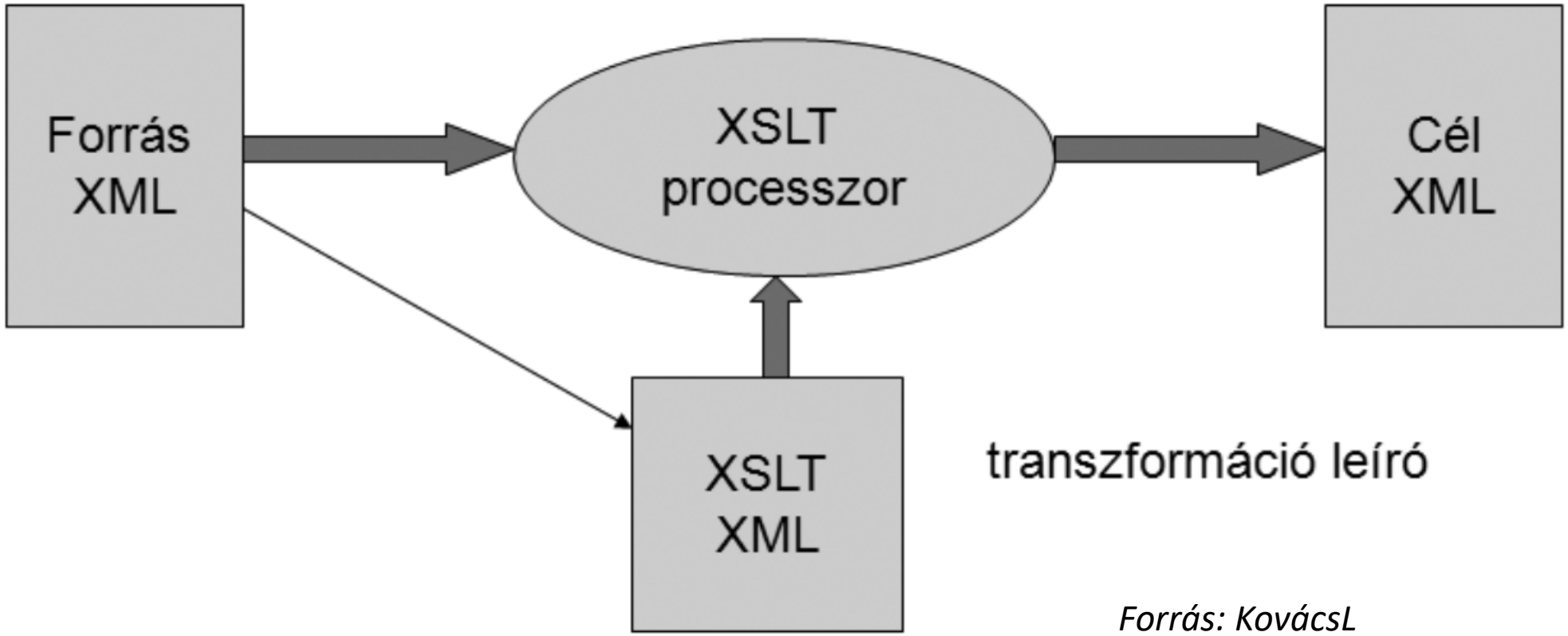
Kétféle termék változata ismert:

- *Community Edition:* Open-source változat, mely pluginok nélkül támogatja a Java, Kotlin, Groovy, Scala nyelveket.
- *Ultimate Edition:* Havidíjas / Évenként fizetendő változat, mely 1 év után örök licence-t ad a megvásárolt korábbi verzióra.”

IntelliJ IDEA fejlesztőkörnyezet

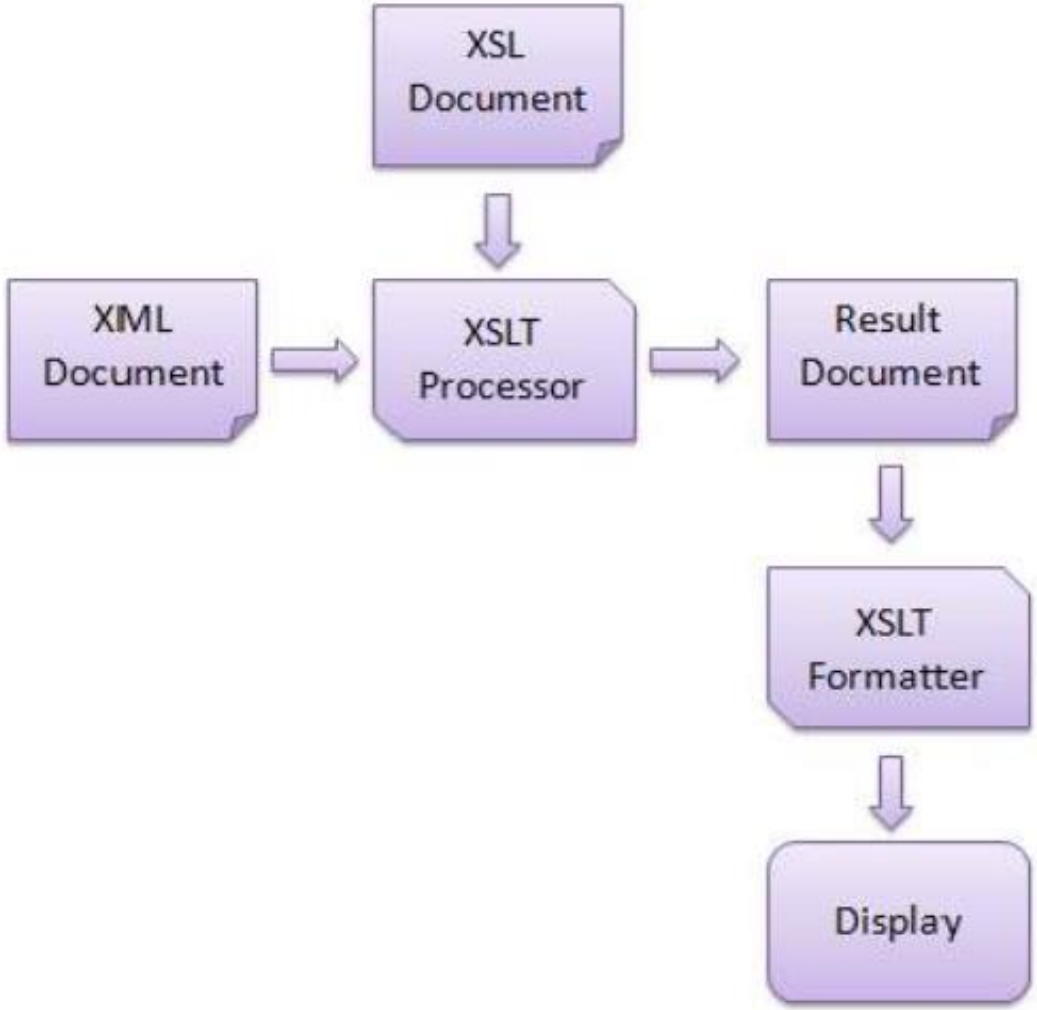


XSLT nyelv - modellje



Forrás: KovácsL

XSLT nyelv - modellje



Forrás: https://www.tutorialspoint.com/xslt/xslt_overview.htm

XSLT nyelv - forráskód

„Az *XSLT forráskódot* XML állományként tároljuk.

A forrásállomány *gyökér eleme* egy

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/199
9/XSL/Transform" version="2.0">
     <!-- transzformációs minták -->
</xsl:stylesheet>
```

elem.

`stylesheet` elemmel adjuk meg azt, hogy itt *stíluslapról* van szó.

Az XSLT transzformációs parancsok mindegyike a definiált `http://www.w3.org/1999/XSL/Transform` névtérhez tartoznak.

XSLT nyelv - forráskód

Használhatjuk az *stylesheet* vagy a *transform* elemet is gyökér elemként. A kettő teljesen megegyezik.

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

...

</xsl:stylesheet>
```

VAGY

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

...

</xsl:transform>
```

XSLT nyelv – transzformációs parancsok

A *transzformációs parancsok kiterjednek a következő műveletekre:*

- csomópontok *feldolgozási ciklusa*,
- *csomópontok rendezése* a feldolgozáshoz,
- *kifejezések megadása* az eredmény XML dokumentum előállításához,
- *változók létrehozása,*
- *új struktúra elem* létrehozatala,
- *feltételes transzformáció végrehajtása* etc.

XSLT nyelv - `üres transzformációs parancs`

A forrás *XML állományban jelöljük* ki, hogy az *adott állományon egy transzformációt kell végrehajtani*.

```
?xml version="1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="xx2.xsl" ?>
<autok>
 <auto rsz="r1">
  <tipus> Fiat </tipus> <ar> 21233 </ar>
 </auto>
 <auto rsz="r2">
  <tipus> Opel </tipus> <ar> 31233 </ar>
 </auto>
</autok>
```

Forrás: KovácsL

XSLT nyelv – üres transzformációs parancs

Üres transzformációs parancs

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <!-- üres -->
</xsl:stylesheet>
```

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

   Fiat  21233
   Opel  31233
```

XSLT hivatkozás megadása

A *transzformációs állomány kijelölése* egy

`<?xml-stylesheet type="text/xsl" href="forrás-xsl"?>` - elemmel történik.

Ez a feldolgozási direktíva a dokumentum gyökéreleme előtt szerepel.

A `'href'` attribútummal specifikáljuk a *transzformációt leíró XSLT állományt.*

XSLT nyelv – mintapélda

Mintapélda, amely bemutat egy:

- *X1.XML* forrás dokumentumot,
- *XS1.XSL* transzformációt megadó állományt és a
- transzformáció eredményét tartalmazó X2.XML állományt.

Forrás: KovácsL

XSLT nyelv - X1.XML mintapélda

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

<?xml-stylesheet type="text/xsl" href="xs1.xsl" ?>

<autok>
      <auto rsz ="r11"> <tipus> Fiat </tipus> <ar>21233</ar> </auto>

      <auto rsz ="r21"> <tipus> Opel </tipus> <ar> 41233</ar></auto>

      <auto rsz ="r31"> <tipus> Honda </tipus> <ar>71233</ar> </auto>

</autok>
```

XSLT nyelv - XS1.XSL mintapélda

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

<xsl:template match="auto">
     <xsl:value-of select ="./@rsz"/>:
     <b> <xsl:apply templates/> </b> <br/>
</xsl:template>

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

</xsl:stylesheet>
```

XSLT nyelv – X2.XML mintapélda

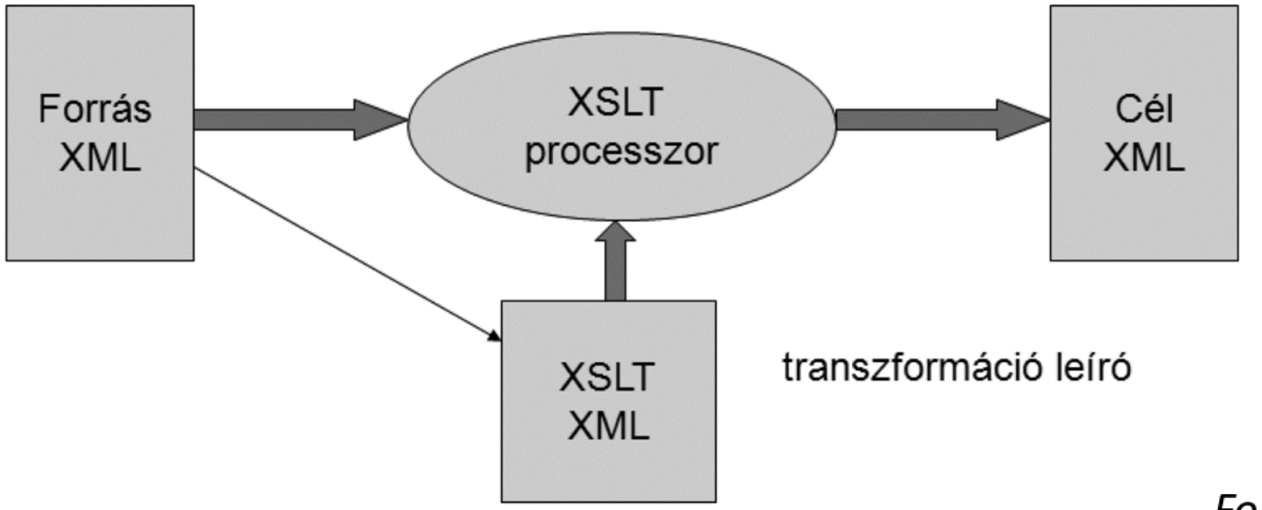
```
r11: <b> <i> Fiat </i> 21233 </b> <br/>

r21: <b> <i> Opel </i> 41233 </b> <br/>

r31: <b> <i> Honda </i> 71233 </b> <br/>”
```

XSLT működési elve

XSLT működési elve

„Az XSLT *feldolgozók bemenete* a feldolgozandó *XML dokumentumból* készített *dokumentumfa*.

A *feldolgozás kimenete* az ***eredményfa linearizálásával*** kapott *XML dokumentum* lesz.



Forrás: KovácsL

XSLT működési elve

A *dokumentumfa* feldolgozásának elve a *fa csomópontjainak bejárásán alapszik*.

A feldolgozó a *gyökér csomóponttól* kezdve bejárja a teljes fát, és ha egy csomóponthoz tartozik *átalakítási szabály,* ott előállítja a kért *XML dokumentum részletet.*

Alapesetben a feldolgozás a gyökértől halad a gyerekek felé, egy *top-down*, *deep-first* mélységi bejárás megközelítésben.

XSLT működési elve - a feldolgozás menete

1. A feldolgozó rááll a *gyökér csomópontra*.

2. Megnézi, hogy létezik-e *feldolgozási utasítás* erre a csomópontra.

Ha *nem létezik minta,* de vannak gyerek elemek, akkor sorra veszi a gyerek elemeket, és mindegyikre elvégzi ezt a *vizsgálatot*.

Ha nincs gyerek, *megáll a feldolgozási lánc*.

Előtte, ha *szöveg csomópontnál* tart a feldolgozás, akkor kiírja a *szövegelemek tartalmát*.

XSLT működési elve

3. Ha van *illeszkedő parancs* minta, akkor azt kiértékelve előáll egy *eredményfa részlet*.

Alapesetben a csomópont feldolgozása után *megáll a feldolgozási lánc.*

Lehet *tovább-haladási igényt* is adni, ekkor *a kijelölt elemekre megy tovább a feldolgozás - továbblépés* tetszőleges, *XPath formátumban* megadható célhelyekre történhet.

Az XSLT működési elve - szabályok

Az *alapértelmezett feldolgozási szabályok:*

- *minden elemnél* - ha az nem *szövegcsomópont - tovább lép a gyerek felé,*
- a *szövegcsomópont* esetén kiírja az eredménybe ezt a *csomópontot (a szövegtartalmat*),
- egyéb csomópontokat figyelmen kívül hagy.

XSL dokumentum gyökéreleme

Az XSL dokumentum gyökéreleme az alábbiak valamelyike kell, hogy legyen:

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

vagy

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

XSLT dokumentum kapcsolata az XML-hez

Az XSL dokumentumot az alábbi módon kapcsolhatjuk egy XML dokumentumhoz:

```
<?xml-stylesheet type="text/xsl"
href="xsl_dokumentum_neve"?>
```

XSLT parancsai - Sablonok

- Egy XSL stílusállomány *szabályok összességéből* épül fel – ezeket *sablonoknak (template*) nevezzük.
- Ezek a sablonok arra vonatkozó szabályokat tartalmaznak, hogy mi történjen, ha egy *illeszkedő csomópontot találunk*.

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

      ……

</xsl:stylesheet
```

XSLT parancsai - Sablonok

Az `<xsl:template />` *elem* segítségével sablonokat definiálhatunk a dokumentum *egyes részeinek átalakítására*.

Egy XSL az *átalakítási szabályokat* tartalmazza, melyeket **template**-eknek nevezünk.

Template-et a `<xsl:template>` elemmel hozhatunk létre, és ennek segítségével határozhatunk *meg egy sablont*.

A `match` attribútum segítségével választhatjuk ki, hogy *milyen elemekre alkalmazzuk majd a szabályt.*

XSLT parancsai - megadható elemjellemzők

- `name:` a sablon neve.
- `match:` *XPath kifejezés*, azoknak az *elemeknek a kijelölésére,* amikre a sablont alkalmazni kell.
- `priority:` egy XSL dokumentum *több sablont is tartalmazhat*, a sablonok közül a *legnagyobb prioritású fog érvényesülni.*
- `mode:` megadható, hogy egy *adott elem többször* is feldolgozható legyen.”

XPath expression

Példa:

XPath kifejezés	Leírás
nodename	Kiválaszt minden node-ot aminek 'nodename' a neve
/	Kiválasztja a gyökér elemet.
//	Az aktuális node-tól indítja a keresést és kiválasztja az összes egyezést.
.	Kiválasztja az aktuális node-ot
..	Kiválasztja a szülő node-ot
@	Attribútum kiválasztása

Az XSLT parancsai - mintaillesztés

„A dokumentumfa egy *megadott* ***csomóponthalmazának kijelölése*** a

```
<xsl:template match=kif1 name=kif2 >
    <!-- feldolgozó utasítások -->
</xsl:template>
```

paranccsal történik.

A **'kif1'** jellemző egy *illesztési mintát jelöl*, mely a *feldolgozandó csomópontokat jelöli* ki a *nevük alapján*.

Az XSLT parancsai - mintaillesztés

A **'kif2'** egy *nevet hordoz*, megadásával egy *azonosító név köthető a mintához.*

A *minta* feldolgozása után a *fa mélységi bejárása megáll*.

Például:

```
<xsl:template match="/"  >
      UDV!
  </xsl:template>
```

parancsállományt, amely a *gyökér elemre* illő *mintát tartalmaz*, melyben *egy szöveg konstans a generált kimenet*.

Az XSLT parancsai - mintaillesztés

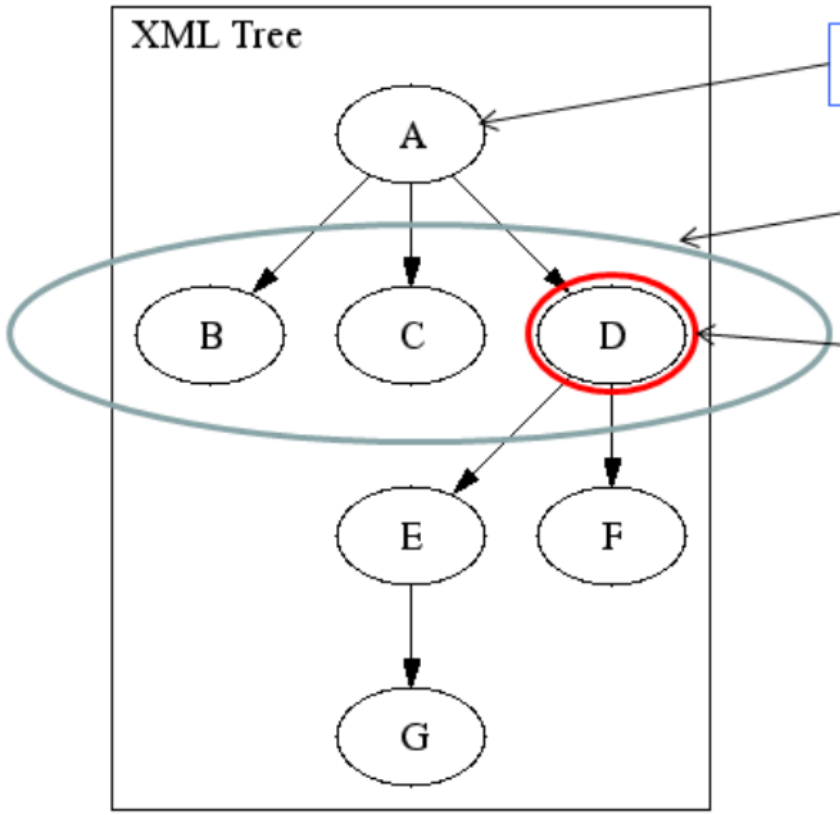
A feldolgozó a szöveg kiírása után leáll a fa feldolgozásával.

Az így kapott eredmény:

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

Az XSLT parancsai - mintaillesztés

Mintaillesztés XSLT parancsa



aktuális csomópont

továbblépési tartomány

illeszkedő elem

```
<xsl:template match=kif1 name=kif2 >
        <!-- feldolgozó utasítások -->
</xsl:template>
```

Forrás: KovácsL

Az XSLT parancsai - mintaillesztés

A minta egynél *több csomópontot* is tartalmazhat, ekkor mindegyiknél végrehajtódik az eredménygenerálás. Példa:

```
<xsl:template match="auto"  >
     Udv!
</xsl:template>
```

Az XML dokumentum *auto* nevű elemeinél lesz illeszkedés.

XSLT parancsai - kiértékelés algoritmusa

A *kiértékelés algoritmusa:*

- A kiértékelő motor elsőként elmegy a gyökér (dokumentum) elemhez.
- Megnézi, van-e az XSLT parancsfában *illeszkedő minta* a gyökérre:

  Ha nincs, akkor az *alapértelmezési parancsot* hajtja végre, tovább megy a gyerek csomópontok felé.

XSLT parancsai - kiértékelés algoritmusa

- A *szöveges csomópontok* esetében az alapértelmezési szabály a *szöveges tartalom kiírását* írja elő.

  Ha ez *levél csomópont,* a mélységi keresés is leáll.
- A feldolgozó motor az illeszkedés a minták vizsgálat során megkeresi, hogy mely minta illeszkedik a fa feldolgozás alatt álló csomópontjára.

XSLT parancsai – tipikus minták

Tipikus minták:

- `nev:` a fa összes olyan eleme, melynek azonosító neve `'nev'`, ekvivalens alakja `root(.)//nev`,
- `/:` a dokumentum gyökér csomópontja,
- `/nev:` a fa gyökér eleme, amely a dokumentum gyökér csomópont alatt helyezkedik el,
- `nev[2]:` azon elemek, melyek a szüleik második `'nev'` nevű gyerekei,

XSLT parancsai - tipikus minták

- `text()`: a dokumentum összes szövegtípusú eleme,
- `*`: a dokumentum összes *névvel rendelkező eleme* (szövegcsomópontok, jellemzők, névterek nem szerepelnek),
- `node()`: a dokumentum-fa összes *elem csomópontja* (elemjellemző, névtér nem szerepel benne),
- `attribute::nev`: a fa összes *'nev' elemjellemzője*,
- `nev1 | nev2[@nev3=kif3]`: az eredmény *két csomóponthalmaz* uniója.

XPath rövid áttekintése

Célja: az XML fa egy részhalmazának kijelölése

tengely:: csomópont_típus | elérési_útvonal [szűrési feltétel]

Tengelytípusok:

- self
- child
- descendant
- parent
- ancestor
- preceding
- following
- attribute
- …

Csomópont elérés:

- text()
- node()
- nev
- *
- /
- //
- ..
- .
- @

```
child::autok/auto[@ar > 1000 ]
/auto[tulaj]
/descendant::auto[1]
```

Forrás: KovácsL

XSLT parancsai

- *Iteráció* – ciklus utasítás,
- *Csomópontok (vagy* egyéb kifejezések) *kiértékelése,*
- Rendezés,
- Feltételvizsgálat,
- Sablonok explicit alkalmazása,
- Kimenet beállítása,
- Csomópontok másolása,

XSLT parancsai

- Csoportképzés,
- Új csomópontok létrehozása,
- Import,
- Változók,
- Függvények, eljárások.

Feladatok a következő URL címen találhatók:

Forrás: https://www.javatpoint.com/xslt-tutorial

Iteráció - `for-each` - Ciklus utasítás

Csomópontok halmazán való végig iteráláshoz az

```
<xsl:for-each />
```

elemet használhatjuk.

A `<xsl:for-each />` elemnek ***egy*** attribútuma van.

A `select` attribútumban XPath kifejezéssel *kijelölhetjük a csomópontok halmazát.*

Iteráció - for-each> - Ciklus utasítás

XSLT `<xsl:for-each>` Element

```
<xsl:for-each
  select = Expression>
</xsl:for-each>
```

Paraméter leírása:

A `select` attribútumban adhatjuk meg, hogy mely *elemeket válasszuk* ki, amelyen majd a ciklus végigmegy.

Pl.: **`<xsl:for-each select="class/student">`**

Pl.: `students_for-each.xml, students_for-each.xsl`

Iteráció - `for-each` - Ciklus utasítás

Az utasítás szintaktikája:

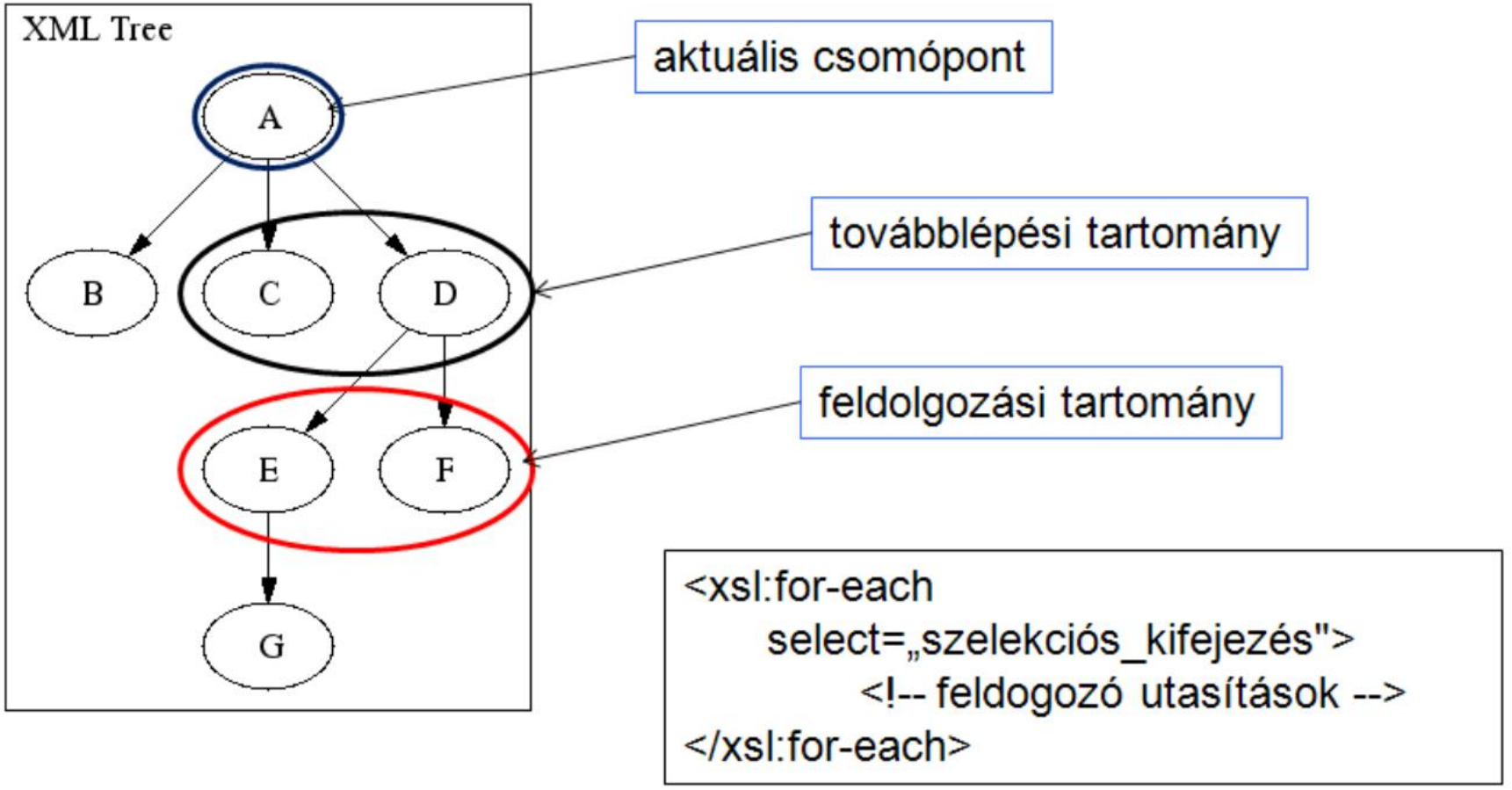
```
<xsl:for-each  select="kif1">
     <!-- feldolgozó utasítások -->
  </xsl:for-each>
```

A `'select'` rész **kijelöli** a *feldolgozandó csomópont halmazt.*

A feldolgozó utasítások rendszerint az *eredményfa tartalmát határozzák* meg.

Iteráció - `for-each` - Ciklus utasítás

Feldolgozási ciklus XSLT parancsa



XSLT parancsai - csomópontok kiértékelése

Csomópontok (vagy egyéb kifejezések) **kiértékelése**

Az `<xsl:value-of />` *elem segítségével* kinyerhető egy *adott csomópont szövegértéke* – azaz a kiválasztott elem *értékét is lekérhetjük*.

Két attribútuma van:

A *kiértékelendő csomópontot* egy *XPath kifejezéssel* adhatjuk meg a `select` attribútumban.

A `select` attribútumban megadhatjuk, hogy *melyik node-nak az értékét* szeretnénk lekérni.

XSLT parancsai - csomópontok kiértékelése

Value-of *elem*

```
<xsl:value-of
  select = Expression
  disable-output-escaping = "yes" | "no">
</xsl:value-of>
```

Paraméterek leírása:

Select: Megadja az aktuális környezetben kiértékelendő XPpath kifejezést.

Disable-output-escaping: Alapértelmezett *„no”.*

Ha *„yes*", a kimeneti szöveg nem hagyja ki az XML-karaktereket a szövegből.

XSLT parancsai - csomópontok kiértékelése

Csomópontok (egyéb kifejezések) *kiértékelése*

- A `'value-of'` utasítás segítségével lehet kifejezéseket kitenni a célállományba, azaz egy csomópont értékét kinyerjük.

A parancs alakja: `<xsl:value-of select="kif1" />`

A `'kif1'` jelöli ki a *kiíratandó értékeket*.

A *kifejezés jelölhet csomópontot* is, ebben az esetben a csomópont *szöveges tartalma* kerül bemásolásra az eredményfába.

Az XSLT parancsai - Csomópontok kiértékelése

```
<?xml version="1.0" ?>
  <xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999
/XSL/Transform"' version="1.0">
<xsl:template match="autok">

    <xsl:for-each select="auto">
        <xsl:value-of select="tipus"/>
      </xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

Forrás: KovácsL

Az XSLT parancsai - csomópontok kiértékelése

Eredményül az `'autok'` elem `'auto'` gyerekeinek `'tipus'` gyerekének szövegértékeit kapjuk meg, hiszen a `'value-of'` utasítás `'select'` paramétere egy *azonosító nevet* tartalmaz.

Ekkor a csomópont mögötti szöveges érték kerül át.

XSLT parancsai – `sort`–rendezés

Iterációnál az `<xsl:sort/>` elem segítségével megadhatjuk, hogy *milyen sorrendben legyenek a csomópontok feldolgozva.*

Az `<xml:sort>` elem az XSL fájl `<xsl:for-each>` elemébe kerül a kimenet rendezéséhez.

5 attribútuma van:

A `select`–*megadhatjuk* milyen n*ode*-ot használjunk a rendezéshez.

Az `order` attribútummal, hogy *csökkenő* vagy *növekvő* sorrendet akarunk, stb.

XSLT parancsai – `sort`–rendezés

XSLT `<xsl:sort>` Element

```
<xsl:sort
  select = string-expression
  lang = { nmtoken }
  data-type = { "text" | "number" | QName }
  order = { "ascending" | "descending" }
  case-order = { "upper-first" | "lower-first" } >
</xsl:sort>
```

A *lang* – ez a rendezési sorrend meghatározásához használt *nyelvi ábécét*.

A *data-type* - meghatározza az adattípusát.

A *case-order* - rendezési sorrend nagybetűk szerint.

XSLT parancsai – `sort`–rendezés

A ciklus esetében fontos az *elemek elérési sorrendje* is.

Ez a *rendezésre szolgáló utasítással* valósítható meg, amely egy csomóponthalmazhoz köthető, formátuma:

```
<xsl:sort select="kif1"  order="kif2„
          collation="kif3" />
```

A ‚`kif1`’ kifejezés megadja, hogy *mely csomópont tárolja a rendezés kulcsát* – ez egy *XPath kifejezés.*

XSLT parancsai – `sort`–rendezés

- Az `'order'` tulajdonság a rendezés irányát (csökkenő vagy növekvő) állítja be.
- A `,kif3'` tulajdonság a *rendezési sorrendet* határozza meg.

Ez adja meg az egyes karakterek közötti megelőzési relációt.

XSLT parancsai – `sort`–rendezés - mintapélda

Példa: típusnevek ABC sorrendben jelenjenek meg.

```
<?xml version="1.0" ?>
  <xsl:stylesheet xmlns:xsl="'http://www.w3.org/
1999/XSL/Transform"' version="1.0">
     <xsl:template match="autok">
        <xsl:for-each select="auto">
           <xsl:sort select="tipus"/>
           <xsl:value-of select="tipus"/>
        </xsl:for-each>
     </xsl:template>
  </xsl:stylesheet>
```

XSLT parancsai - if - feltételvizsgálat

Az `<xsl:if />` *elem* segítségével definiálhatunk *egy csomópontok tartalmára vonatkozó feltételt*.

A `test` attribútumban egy *logikai értékű XPath kifejezést* adhatunk meg.

Szintaxisa:

```
<xsl:if test="expression">
   ... valamilyen kimenet, ha a kifejezés igaz ..
</xsl:if>
```

XSLT parancsai – if mintapélda

```
<xsl:template match="/">
  <html><body><ul>
   <xsl:for-each select="book/authors/author">
   <xsl:sort select="@id" order="descending" />
   <xsl:if test="starts-with(., 'Ed')" > He is
   Ed,</xsl:if>
   <xsl:value-of select="." />
   </xsl:for-each></ul>
   </body>
  </html>
</xsl:template>
```

XSLT parancsai – `choose` - feltételvizsgálat

Az `<xsl:choose />` (a switch utasításhoz hasonló) elembe foglalva.

Az `<xsl:choose>` elemet együtt használjuk az `<xsl:when>` és `<xsl:otherwise>` elemekkel.

- `<xsl:when />` elemek segítségével *több feltételt megadhatunk*, vagy
- `<xsl:otherwise />` elemmel az alapértelmezett esetet.

XSLT parancsai – `choose` - feltételvizsgálat

Szintaxisa:

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>
</xsl:choose>
```

XSLT parancsai – `choose` - feltételvizsgálat

Ha több *különböző tevékenység* közül választunk egyet, akkor a *többszörös elágazás utasítását* használjuk. Az utasítás alakja:

```
<xsl:choose >
        <xsl:when test="kif1">
                <!-- tevékenységek -->
        </xsl:when>
        <xsl:when test="kif2">
                <!-- tevékenységek -->
        </xsl:when>
                ...
        <xsl:otherwise>
                <!--- tevékenységek -->
</xsl:choose>
```

XSLT parancsai – `choose` - feltételvizsgálat

A szerkezetben mindegyikághoz egy logikai kifejezés tartozik.

A vezérlés a *sorrendben első*, igaz értékű kifejezéshez tartozó ágat választja ki.

Ha egyetlen egy ilyen `'when'` ág sincs, akkor az `'otherwise'` ágra kerül a vezérlés.

XSLT parancsai – `choose` - feltételvizsgálat

```
<xsl:template match="/">
     <html><body><ul><xsl:for-each select="book/authors/author">
          <xsl:sort select="@id" order="descending" />
          <xsl:if test="starts-with(., 'Ed')">He is Ed,</xsl:if>
          <xsl:value-of select="." />
          <xsl:choose>
               <xsl:when test="@id = 1">The ID is 1.</xsl:when>
               <xsl:when test="@id = 2">The ID is 2.</xsl:when>
               <xsl:otherwise>
                    <xsl:value-of select="concat('ID:', @id)" />
               </xsl:otherwise>
          </xsl:choose>
     </xsl:for-each></ul></body></html>
</xsl:template>
```

XSLT parancsai – `choose` – feltételvizsgálat - mintapélda

Feladat: a vizsgaleíró dokumentumban a számmal megadott vizsgajegyet szöveges leírással is cseréljük le.

Az *induló forrásdokumentum*:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="x1.xsl" ?>
<vizsgak>
  <vizsga>
       <targy> Matematika </targy> <diak> K234 </diak> <jegy> 2 </jegy>
  </vizsga>
  <vizsga>
     <targy> Matematika </targy> <diak> K761 </diak> <jegy> 3 </jegy>
  </vizsga>
</vizsgak>
```

XSLT parancsai – `choose` – feltételvizsgálat - mintapélda

Ez a minta a gyökér csomópontot másolja át, gyerekeit a további minták határozzák meg. A kapcsolódó XSLT leírás:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="'http://www.w3.org/1999/XSL/Transform"'
version="1.0">
 <xsl:template match="/">
    <xsl:copy>
       <xsl:apply-templates/>
     </xsl:copy>
 </xsl:template>
```

XSLT parancsai – choose – feltételvizsgálat - mintapélda

Ez a minta a gyökérelemet ('vizsgak') teszi le az eredménybe.

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

XSLT parancsai – choose – feltételvizsgálat - mintapélda

Ez a minta végzi el a 'vizsga' elem átvitelét.

Ebben a ciklusban a gyerekek részfáját veszi sorra, s a típusoktól függően átmásol (otherwise) vagy átalakít ('jegy' ág).

Ezen esetben csak a csomópontot másoljuk át, majd a szöveges tartalmát a régi szövegtartalomtól függően határozzuk meg.

XSLT parancsai – choose – feltételvizsgálat - mintapélda

```
<xsl:template match="vizsga">
      <xsl:copy>
         <xsl:for-each select="*" >
            <xsl:choose>
 <xsl:when test="name()='jegy'">
    <xsl:copy>
       <xsl:choose>
         <xsl:when test=" text() = 1 ">
             <xsl:text> egy </xsl:text>
         </xsl:when>
         <xsl:when test=" text() = 2 ">
              <xsl:text> ketto </xsl:text>
         </xsl:when>
```

Forrás: KovácsL

XSLT parancsai – choose – feltételvizsgálat - mintapélda

```
<xsl:when test=" text() = 3 ">
            <xsl:text> harom </xsl:text>
        </xsl:when>
        <xsl:when test=" text() = 4 ">
            <xsl:text> negy </xsl:text>
        </xsl:when>
        <xsl:when test=" text() = 5 ">
            <xsl:text> ot </xsl:text>
        </xsl:when>
```

Forrás: KovácsL

XSLT parancsai – choose – feltételvizsgálat - mintapélda

```
</xsl:choose>
</xsl:copy>
   </xsl:when>
   <xsl:otherwise>
      <xsl:copy-of select="." />
    </xsl:otherwise>
</xsl:choose>
</xsl:for-each>
</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Forrás: KovácsL

XSLT parancsai – `kulcs-` elem

Az XSLT elem az XML dokumentum egy *adott eleméhez* rendelt *név-érték* pár megadására szolgál.

Szintaxisa:

<xsl:key>

name = QName

match = Pattern

use = Expression

</xsl:key>

XSLT parancsai – `kulcs`- elem

Paraméter leírása

Index	Név	Leírás
1.	Név	Megadja a használni kívánt *kulcs nevét.*
2.	Match	A mintát egy olyan **csomóponthoz kell illeszteni**, amely ezt a kulcsot tartalmazza.
3.	Use	*Megadja az XPath kifejezést*, hogy azonosítsa az xml dokumentum csomópontjainak értékét.

XSLT parancsai - `apply-template`

Sablonok explicit alkalmazása

Az `<xsl:apply-templates />` elem segítségével *expliciten alkalmazhatunk* sablonokat a `select` attribútumban megadott *XPath kifejezéssel kijelölt csomópontokra*.

Az `apple-templates` alkalmazása az *aktuális elemre* vagy *annak gyermekeire.*

A `selectben` megadhatjuk, hogy milyen gyerekekre alkalmazzunk `template`-et.

XSLT parancsai - apply-template

Sablonok explicit alkalmazása, szintaxisa:

```
<xsl:apply-template
    select = Expression
    mode = QName>
</xsl:apply-template>
```

Mode: arra használják, hogy egy elemet *a minősített nevei szerint többször feldolgozzanak*, minden alkalommal más eredményt adva.

XSLT parancsai - `import`

XSLT <xsl:import>

Az XSLT `<xsl:import>` elem az *egyik stíluslap* tartalmának egy *másik stíluslapba* történő *importálására* szolgál.

Az *importáló stíluslap* nagyobb prioritást élvez *az importált stíluslapokhoz* képest.

Szintaxis:

```
< xsl:import href = "uri" >
</ xsl:import >
```

XSLT parancsai – `output` – kimenet beállítása

Az `<xsl:output />` segítségével megadhatjuk, hogy milyen *formátumú kimenetet* akarunk kapni a transzformáció eredményeképp.

Megadható attribútumok:

`method, indent, encoding, exclude-result-prefixes, stb.`

Példa:

```
<xsl:output method="xml" indent="true" />
```

XSLT parancsai – `output` – kimenet beállítása

- `indent="yes"` – megadhatjuk, hogy a *kimeneti állomány tartalmazzon-e bekezdéseket*, vagy sem.

  Lehetséges értékei: `yes | no`, az alapértelmezett a `"no"`.

- `method="xml"` – megadhatjuk, hogy a kimeneti dokumentum milyen *formátumú legyen*.

  HTML kimenet esetében `"html"`, XML alapú kimenet esetében `"xml"` értéket kap, ill. „`text`”

  Lehetséges értékei: `xml | html | text`.

XSLT parancsai – `output` – kimenet beállítása

- `encoding="utf-8"` – megadhatjuk, hogy a kimeneti állományban milyen kódolást használunk.

XSLT parancsai – `copy`– csomópontok másolása

Az `<xsl:copy />` és `<xsl:copy-of />` segítségével a `select` attribútumban *megadott XPath kifejezésre illeszkedő csomópontokat* a kimenetre másolhatjuk.

A kettő közötti különbség, hogy

- az `<xsl:copy />` csak a megadott *fa gyökerét másolja* át üresen,
- az `<xsl:copy-of />` pedig az egész csomópontot *attribútumokkal, gyerekelemekkel* és *szövegtartalmakkal* együtt.

XSLT parancsai – `copy`– csomópontok másolása

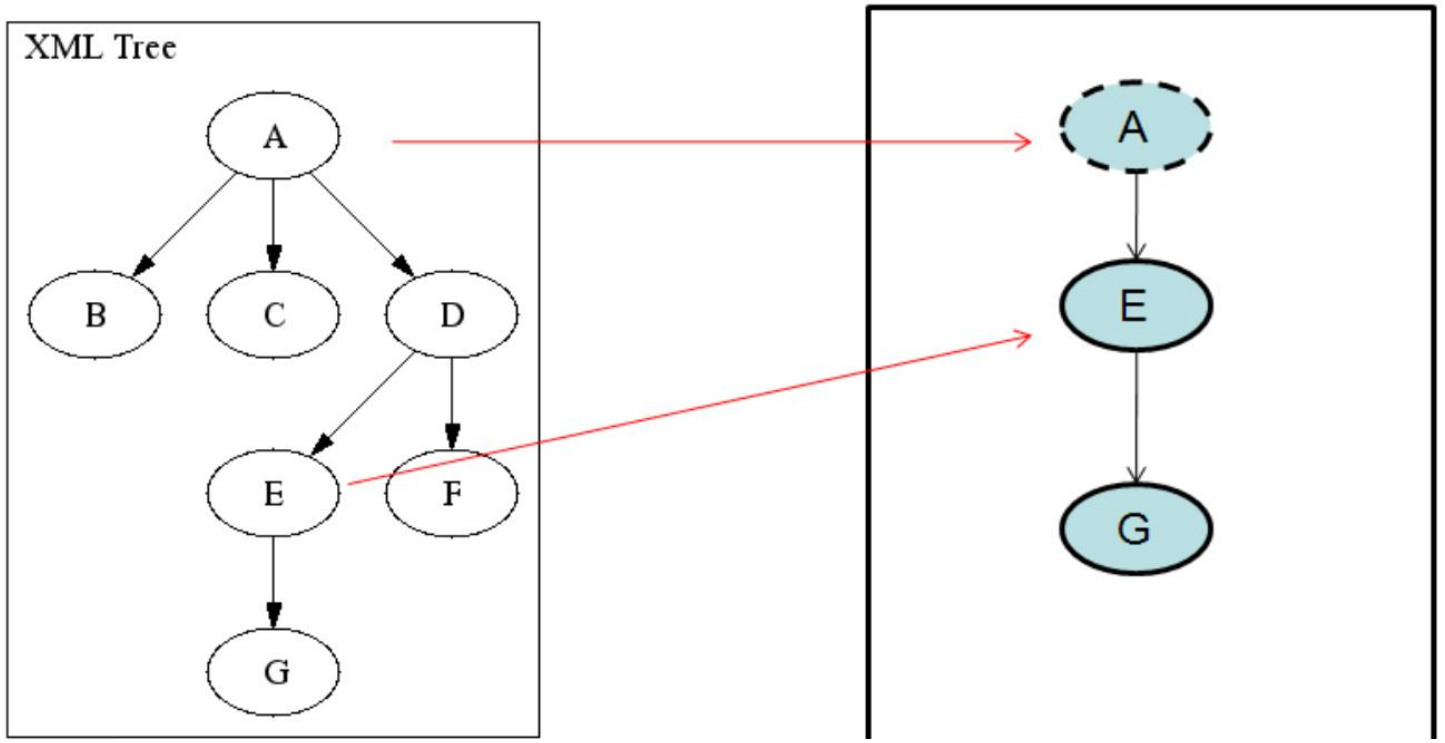
A teljes dokumentum átmásolása:

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


<xsl:template match="/">
        <xsl:copy-of select="." />
    </xsl:template>
 </xsl:stylesheet>
```

Forrás: KovácsL

XSLT parancsai – `copy`– csomópontok másolása

Elemek másolásának XSLT parancsa



<xsl:copy /> : csak az üres elem megy át

<xsl:copy-of select="kif1" /> : a teljes részfa átmegy

Forrás: KovácsL

XSLT parancsai – csoportképzés

Az adatbázis kezelésben megszokott dolog a *rekordok csoportosítása* és *aggregált adatok* képzése.

A csoportképzésnél meg kell adni a *feldolgozandó elemeket* és a *csoportképzési kifejezést*.

A feldolgozás során minden *csoportnál megáll*, s lehetőséget ad *csoport-szintű aggregációra* és *elem szintű részletezésre*.

Az XSLT-ben a csoport alkotó tagjai is elérhetők, külön-külön is.

XSLT parancsai – `group` csoportképzés

Az `<xsl:for-each-group />` használatával (XSLT 2.0) *csoportokat képezhetünk* `select`-tel a *kijelölt csomópontokból*, a `group-by` attribútumban megadott *kulcs* szerint.

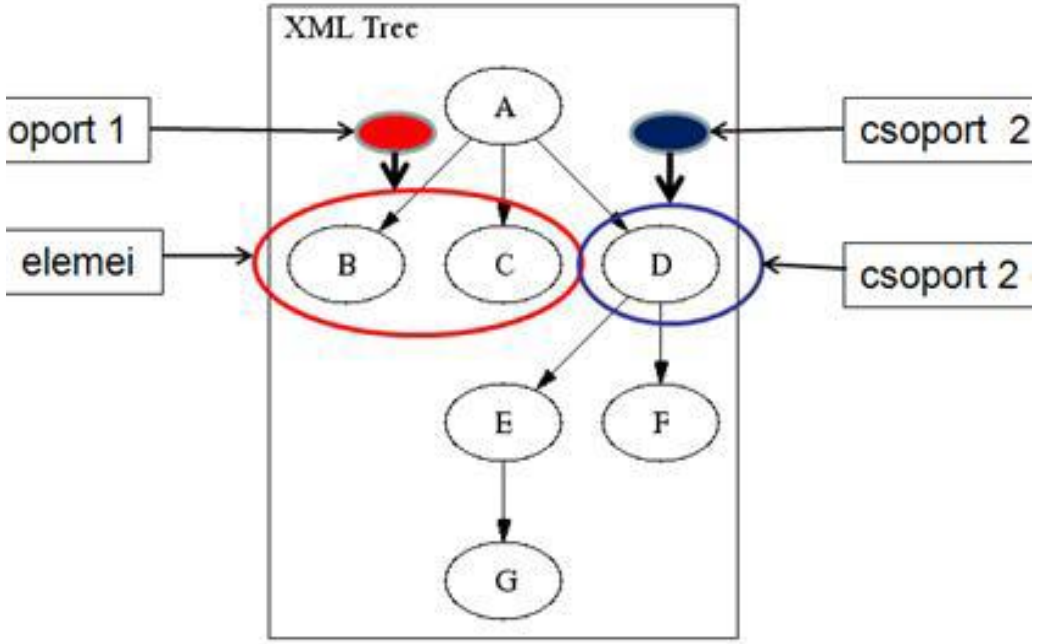
A csoportképzés parancsa:

```
<xsl:for-each-group select="elemek" group-by="csoportképzési_kifejezes" >...<..>
```

XSLT parancsai – `group` csoportképzés

XSLT parancsok áttekintése

Csoportképzés




```
each-group select="elemek"  group-by="csoportképzési_kife
<!-- tartalom …-->
each-group>
```

Forrás: KovácsL

XSLT parancsai – `group` csoportképzés

Csoportképzés - speciális szimbólum

A feldolgozó magban *két speciális szimbólum* használható a *csoport elemeinek elérésére:*

- *current-group*: az aktuális csoport *elemeinek szekvenciája*,
- *current-grouping*-key : az aktuális csoporthoz *tartozó csoportképzési kifejezés.*

XSLT parancsai – `group` csoportképzés - mintapélda

1. Adott a következő XML dokumentum, mely két tantárgy elemjellemzőit tartalmazza (vizsgak.xml):

- Határozza meg a csomópontok számát? *(vizsgak.xsl)*
- Határozza meg az XML és a Angol tárgyak átlagát? *(vizsgak1.xsl)*

XSLT parancsai - új csomópontok létrehozása

Az `<xsl:element />` segítségével létrehozhatunk *új elemcsomópontokat*, ebbe befoglalva

`<xsl:attribute />` és `<xsl:value-of />` segítségével pedig *hozzáadhatunk attribútumokat* és *szövegtartalmat*.

XSLT parancsai - új csomópontok létrehozása

Új elem létrehozása a csomópontok létrehozására szolgáló utasítás használható fel.

```
<xsl:element name="nev" >
      <!-- tartalom -->
   </xsl:element>
```

XSLT parancsai - új csomópontok létrehozása

Ha egy új elemjellemzőre van szükség, akkor a

```
<xsl:attribute name="nev" select="kif1" />
```

utasítást használhatjuk, melyben a `select` tulajdonság a létrehozott *elemjellemző értékét* adja meg.

Egyedi csomópont típusnak – *szövegcsomópont,* létrehozása:

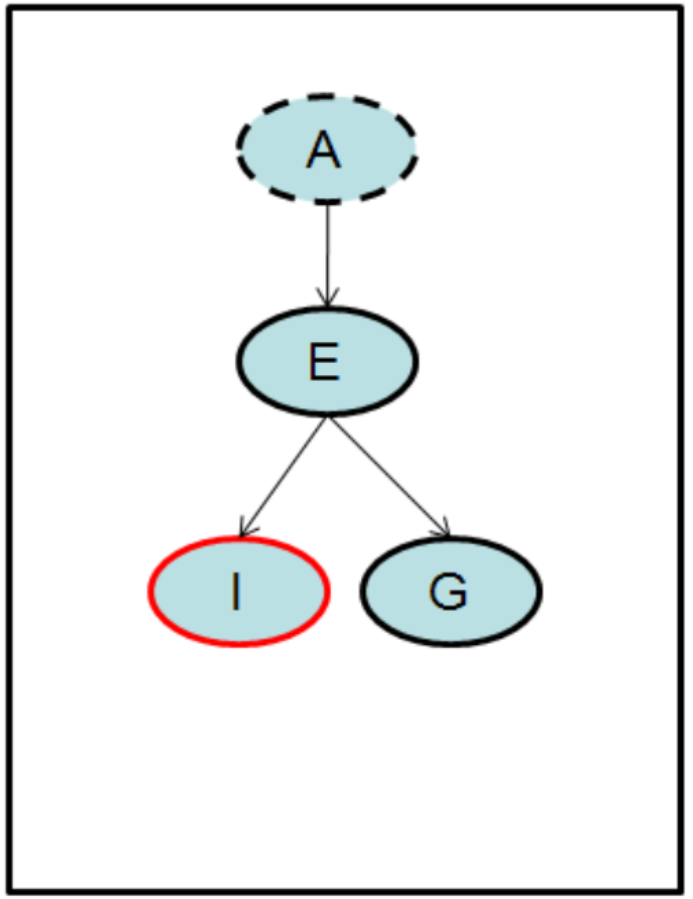
```
<xsl:text>
      <!-- tartalom -->
</xsl:text>
```

XSLT parancsai - új csomópontok létrehozása

```
<xsl:element name="nev" >
        <!-- tartalom -->
</xsl:element>
```

```
<xsl:attribute name="nev" select="kif1" />
```

```
<xsl:text>
        <!-- tartalom -->
</xsl:text>
```



A konstrukciós elemek egymásba ágyazhatók

Forrás: KovácsL

XSLT parancsai - változók

A *változók* az XSLT-ben *valójában konstansok*, egy részfát reprezentálnak, az értékük nem módosítható.

Az `<xsl:variable />` segítségével létrehozhatunk **globális** (az XSL dokumentum gyökéreleme alatt) és **lokális** (a dokumentum más eleme alatt) változókat (konstansokat).

A `name` attribútummal megadhatjuk a *változó* nevét, a `select`-tel pedig az *értékét*.

XSLT parancsai - változók

A változó értékére `$változónév` formában hivatkozhatunk XPath kifejezésekben. Példa:

```
<xsl:variable name="currency"
select="//price/@currency" />

<xsl:template match="/">

<xsl:value-of select="concat('Currency: ',
$currency)" />

</xsl:template>
```

XSLT parancsai - függvények, eljárások

XSLT-ben a **függvények** *skalár értéket* állítanak elő,

az **eljárások** pedig *XML részfát*.

Függvényekben *rekurziót* használunk.

Függvényt az

- `<xsl:function />` elem segítségével *definiálhatunk*,
- `<xsl:param />` elemekkel *paramétereket* adhatunk meg,
- **<**`xsl:sequence />` elemmel, pedig *visszatérési értéket*.

XSLT parancsai - függvények, eljárások

Az **eljárások** megadása megegyezik a *sablonokéval,* rendelkezhetnek *paraméterekkel* is.

Eljárásokat `<xsl:call-template />` elemek segítségével hívhatunk.

*Függvényeke*t *XPath kifejezésekben hívhatunk*.

XSLT parancsai - eljárások mintapélda

Példa: Adott két kör, amely a bemenő adatállomány adatait adja meg a *középpont* és egy *tetszőleges pont adatain* keresztül.*(korok1.xml)*

Eljárás során a *template* nem *mintához kötjük*, hanem *közvetlenül meghívhatjuk*, amihez egy *azonosító névvel* kell rendelkeznie.

Ekkor a séma *egy 'name*', azonosító nevet kijelölő jellemzővel rendelkezik:

```
<xsl:template  name="azonosito_nev" ...> ... </xsl:template>
```

XSLT parancsai - eljárások mintapélda

A *névvel rendelkező sémát* a nevük *megadása során lehet akt*ivizálni.

<xsl:call-template name="azonosito_nev"> .... </xsl:call-template>

A *formális paraméterek* kijelölése a sémában a '`param`' paranccsal történik:

<xsl:param name="parameter_nev" ...> .... </xsl:param>

XSLT parancsai - eljárások mintapélda

A *legfontosabb jellemzők*, amelyek a '`param`' elemnél előfordulhatnak:

- *as :* a paraméter adattípusának megadása,
- *select :* az alapértelmezési paraméterérték, ha nem szerepelne aktuális paraméter,
- *required :* a paramétert kötelező megadni a híváskor.

XSLT parancsai - eljárások mintapélda

A *hívásko*r az

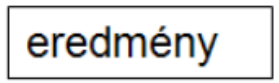
- aktuális paramétereket a '`call-template`' elem gyerekeként szerepeltetik,
- a paraméterátadás utasítása a '`with-param`' elem, melynek alakja:

<xsl:with-param name="parameter_nev" select="ertek"> .. </ >

XSLT parancsai - eljárások mintapélda

Paramétere

```
<xsl:template match="korok">
   <xsl:call-template name="m1">
      <xsl:with-param name="p1" select="'fenn'" />
   </xsl:call-template>
  <xsl:apply-templates />
 </xsl:template>

 <xsl:template match="kor">
     <xsl:call-template name="m1" >
        <xsl:with-param name="p1" select="'lenn'" />
     </xsl:call-template>
 </xsl:template>

 <xsl:template name="m1">
     <xsl:param name="p1" as="xs:string" />
     <xsl:value-of select="$p1" /> : XXXXXXXXXX
 </xsl:template>
```



Forrás: KovácsL

Felhasznált irodalom

- Kovács László: Adatkezelés XML környezetbe, ME
- Kovács László: XML technikák II., ME
- Jeszenszky Péter: XML, DE