IK WIL

XML deel 2
XSLT en XPath

vijfhart
IT-OPLEIDINGEN

Introductie

- Voorstelrondje
- Agenda
- Dagindeling
- Cursusmateriaal
- Doel van deze cursus

Agenda

Dag 1	H1	Inleiding
	H2	XSLT en XPath 1.0
Dag 2	H3	XSLT en XPath 2.0

Dagindeling

- 8:45 -..... Begin cursusdag
  - Introductie
  - .....
  - Hoofdstuk
    - theorie
    - opdrachten maken
    - opdrachten bespreken
- 10:30 -10:45 Koffiepauze
- 12:00 -12:45 Lunchpauze
- 14:30 -14:45 Koffiepauze
- ..... -16:00 Einde cursusdag

Cursusmateriaal

- Map met sheets
- Cursistenmap
  - Samenvatting theorie
  - Bijlagen
  - Opdrachten
- Computer
  - XML editor (XMLBlueprint)
  - Cursusbestanden

Doel van deze cursus

- XML transformaties uitvoeren met XSLT 1.0 en XSLT 2.0
- Verschillen tussen XSLT 1.0 en XSLT 2.0 kennen
- Werken met XPath

Hoofdstuk 1 Inleiding

Leerdoelen:

Werken met XML Blueprint

Werken met stylesheets

Inleiding

- We willen tegenwoordig al onze gegevens opslaan
- Het opslaan kan op veel verschillende manieren
  - Databases
  - Bestanden in verschillende bestandsformaten
- Gegevens moeten regelmatig met anderen worden uitgewisseld
- De vele vormen van opslag kunnen hierbij voor problemen zorgen
- Tekst kan eenvoudig worden verzonden en gelezen
- Met XML worden gegevens als tekst opgeslagen
- Dit maakt XML erg geschikt voor het uitwisselen van gegevens
- Er weinig applicaties waar XML geen rol speelt

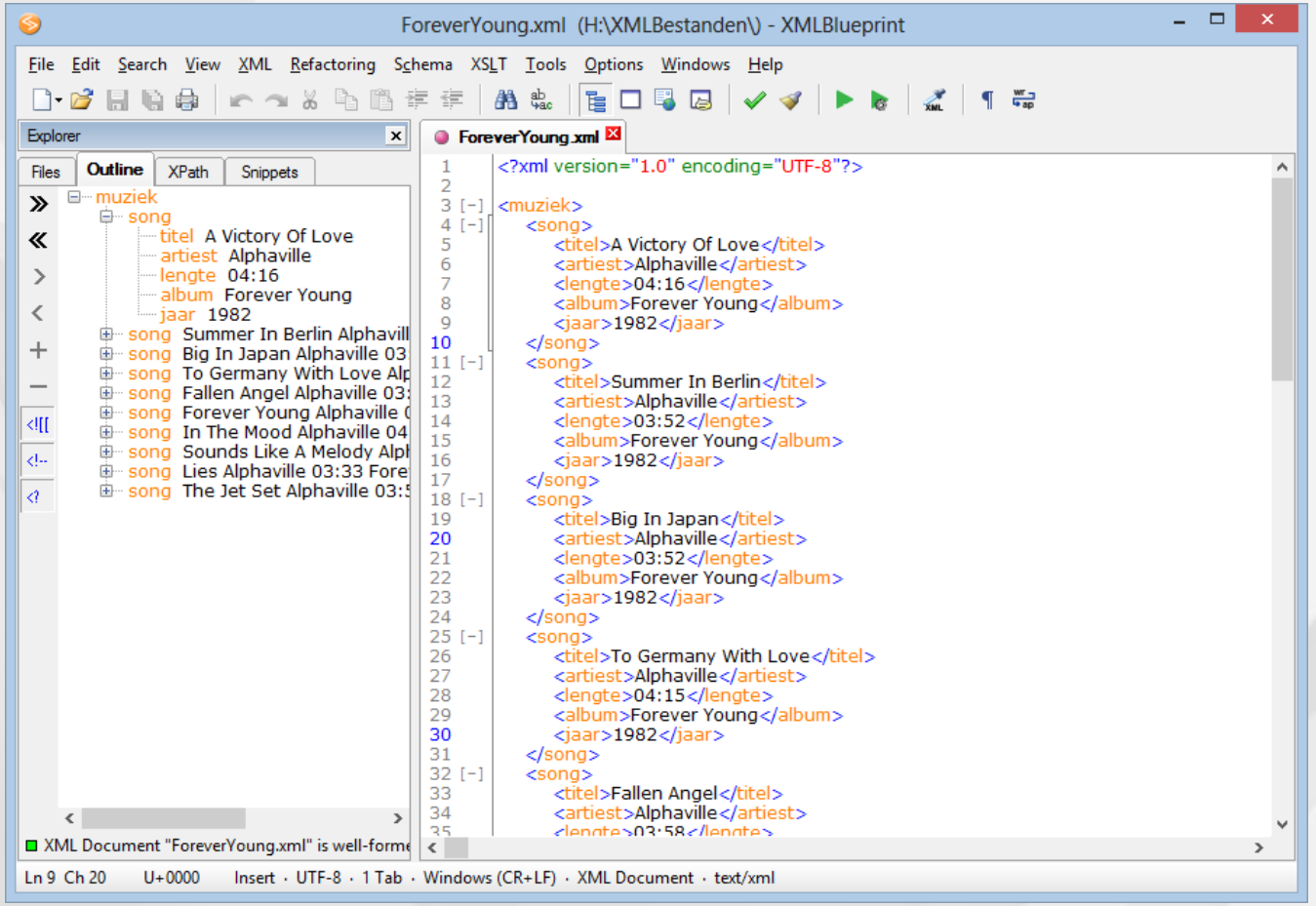
XML editor

- XML documenten maken of wijzigen kan in elke teksteditor
  - Notepad
  - Word
- Maar en specifieke XML editor biedt veel meer mogelijkheden
  - Eenvoudige Opmaak
  - XML documenten controleren
  - XML documenten transformeren
- Er zijn veel XML editors met elk hun eigen mogelijkheden
  - Wij gebruiken tijdens deze cursus XMLBlueprint

XML Blueprint

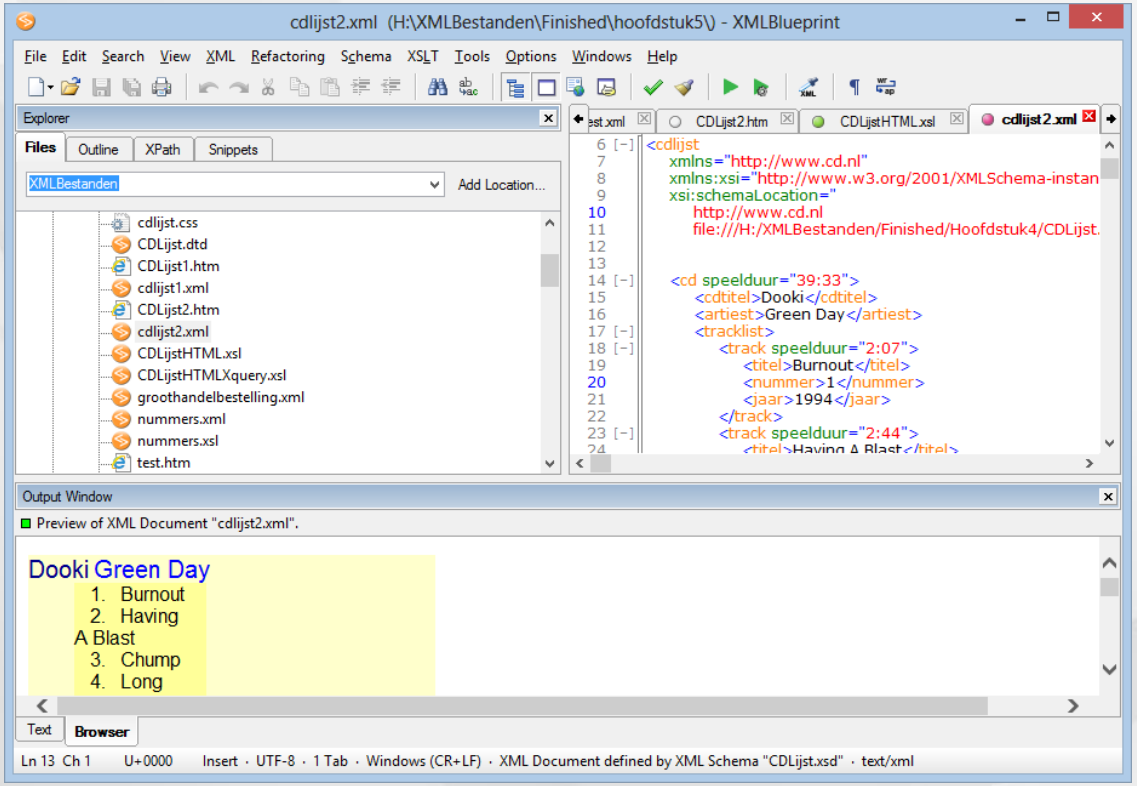
- XML Blueprint is een uitgebreide editor
- Zo kunnen we o.a.
  - Met een klik XML in een overzichtelijke structuur tonen
  - De XML structuur controleren (is het geldige XML)
  - XML documenten valideren (voldoet het aan de voorwaarden)
  - XML omzetten in ander document (XSLT-transformatie)
  - XPath expressies evalueren

XMLBlueprint



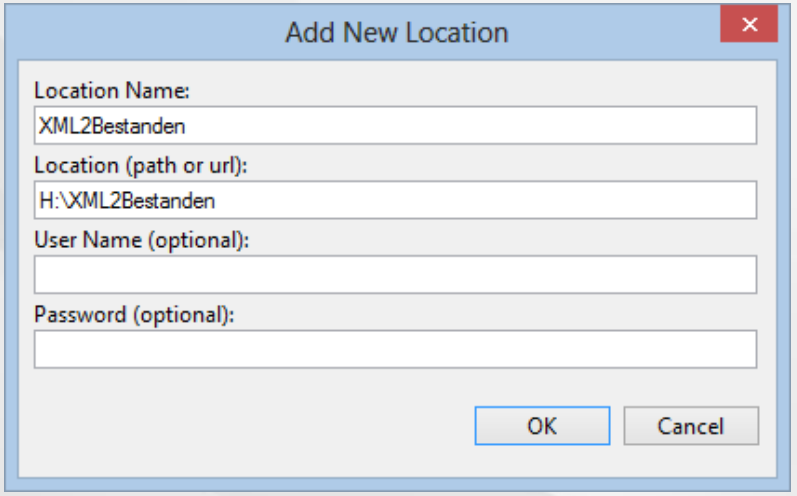
XMLBlueprint

- Output op twee tabs Tekst en Browser
- zichtbaar maken met Preview in Output Window 



XMLBlueprint

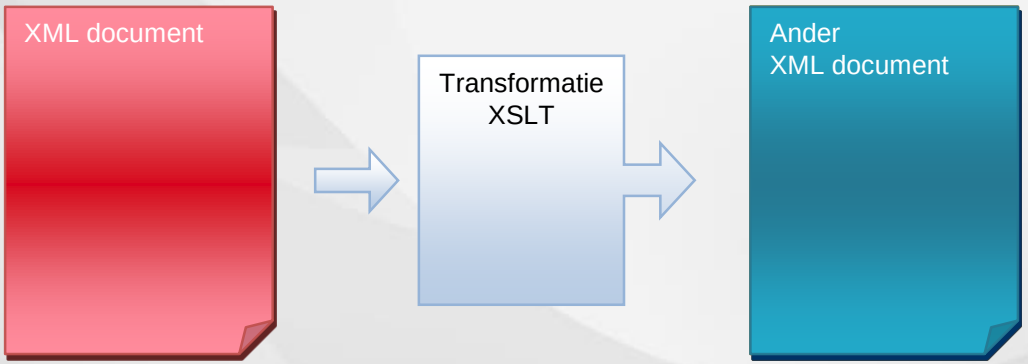
- Opdracht
  - Start XMLBlueprint
  - Klik bij Explorer op de knop op Add Location (tabblad Files)
  - Vul het venster in als volgt in en klik op OK



  - Klik in het invul vak en kies XML2Bestanden uit de lijst
  - De oefenbestanden zijn nu te openen vanuit de Explorer

XSLT Stylesheets

- Niet iedere organisatie kan met één uniform (XML) document werken
- Ook verschillende afdelingen kunnen verschillende XML gebruiken
- Bij gebruik van XML als communicatie zijn vaak vertaalslagen nodig
- Voorbeelden XML naar ander XML, XML naar PDF of XML naar HTML
- Voor transformaties worden XSLT Stylesheets gebruikt



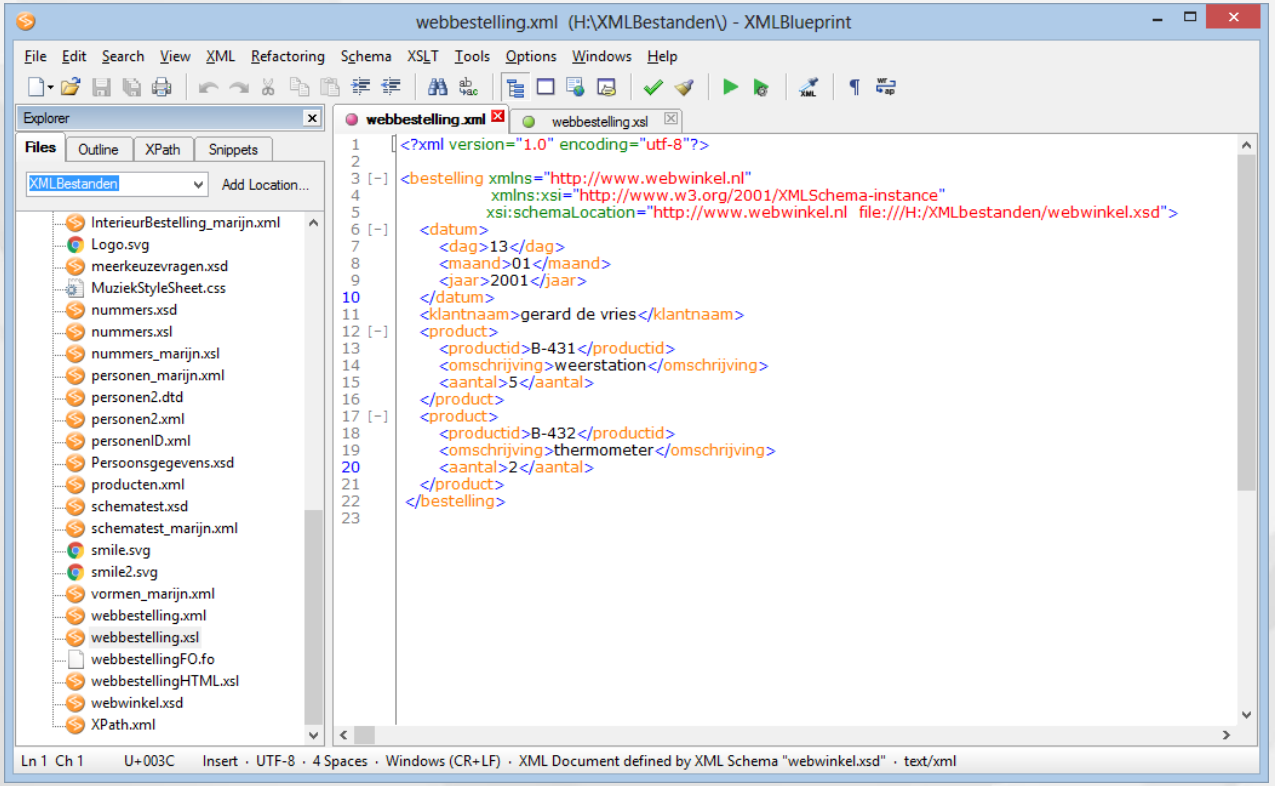
XSLT Stylesheets

- In een XSLT document worden regels vastgelegd op basis waarvan een nieuw document wordt gegenereerd
  - Hoe komt het nieuwe document er uit te zien
  - Welke bestaande objecten worden gebruikt
- Een minimaal XSLT document:
  Het resultaat is een leeg XML document

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

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

    </xsl:template>

</xsl:stylesheet>
```

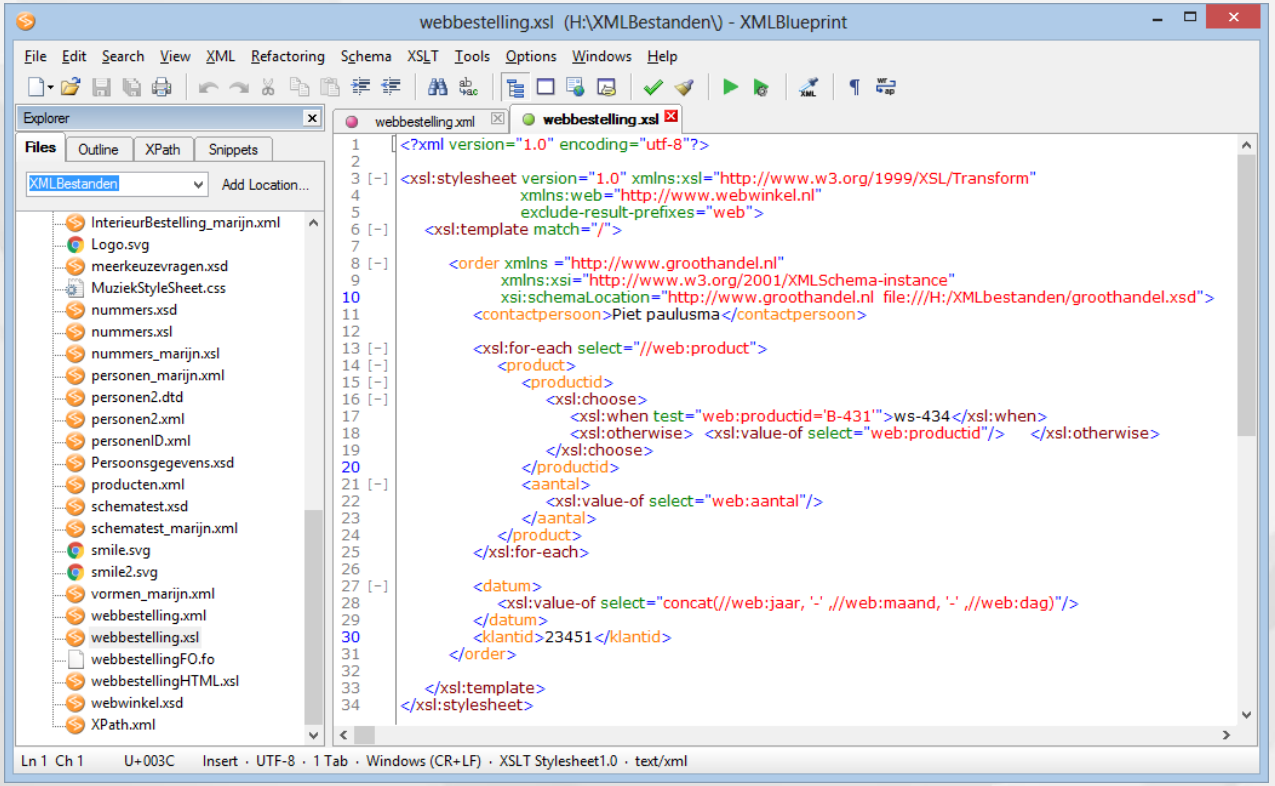
XML stylesheets

- Het document webbestelling.xml moet verzonden worden
- De groothandel gebruikt andere xml tags



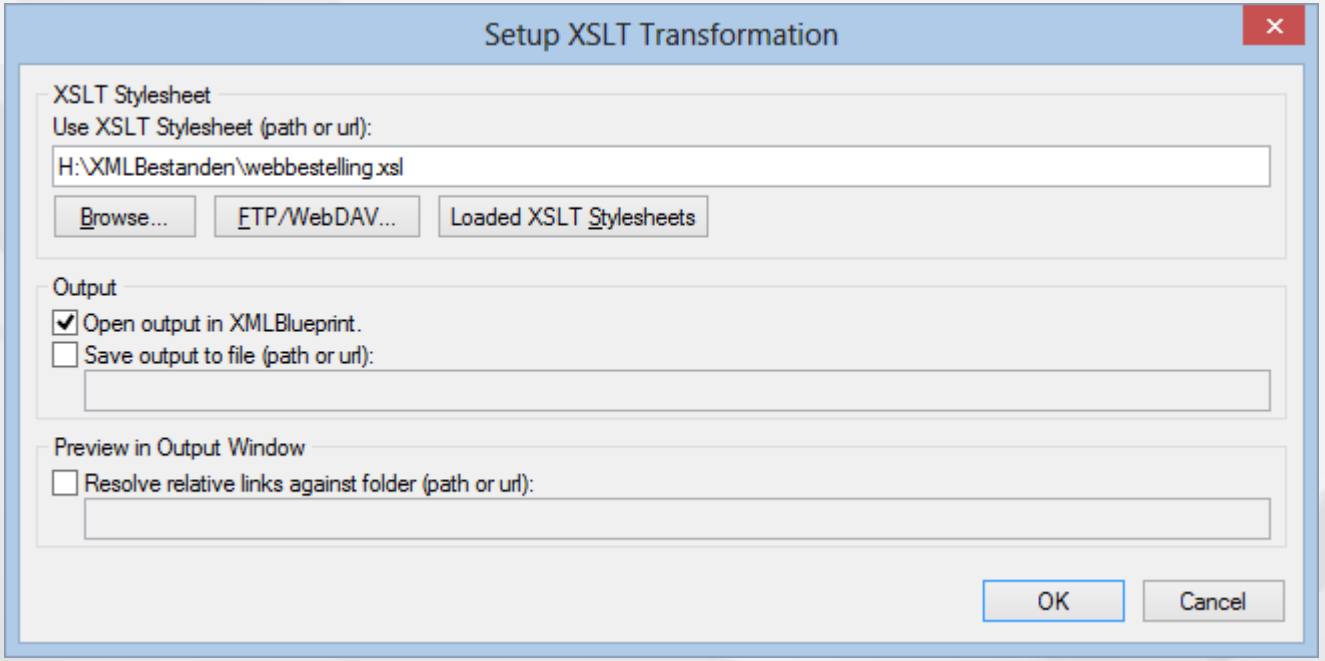
XML stylesheets

- Het stylesheet webbestelling.xsl transformeert onze xml naar een xml document dat de groothandel kan begrijpen



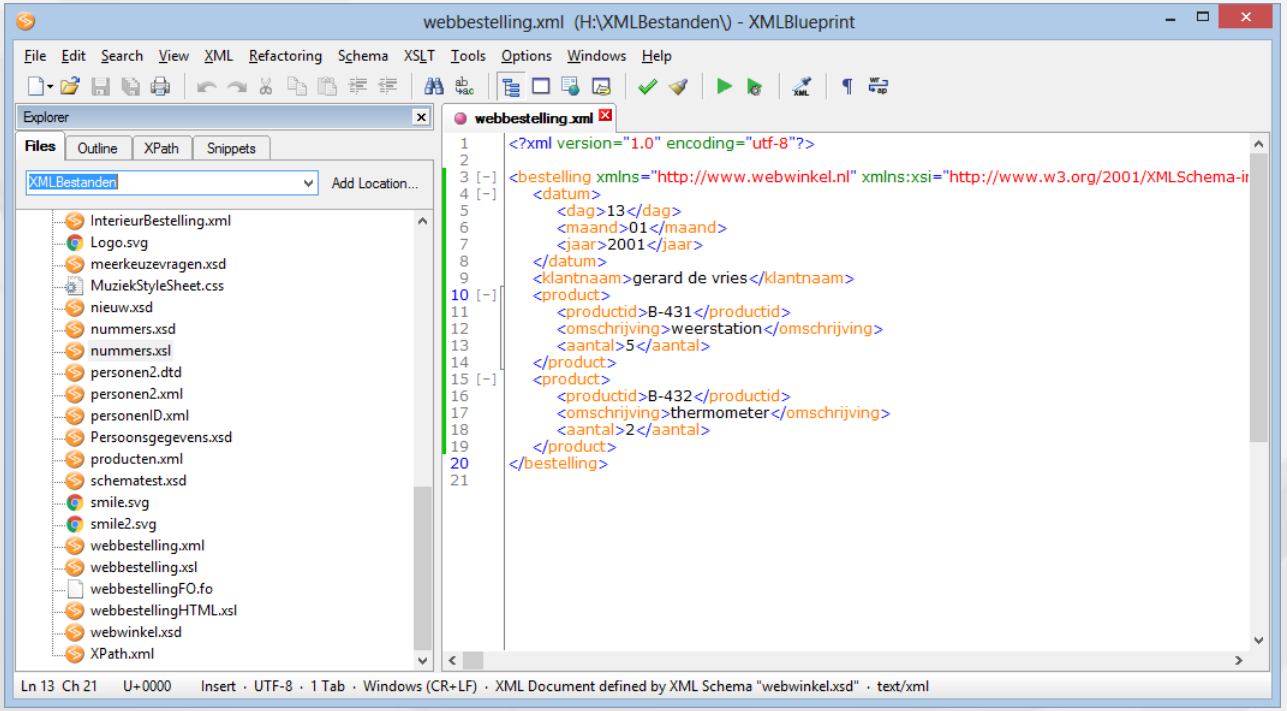
XML stylesheets

- Transformatie uit te voeren met de knop Run XSLT transformatie



XML stylesheets

- De uitvoer is een nieuw XML document
- Leesbare opmaak met de knop Document Format 



Hoofdstuk 2 XSLT en XPath 1.0

Leerdoelen:

Kunnen werken met XSLT transformaties en XPath

Kunnen werken met templates

Gebruik kunnen maken van XPath Axes

XSLT functies kunnen gebruiken en toepassen

XPath functies kunnen gebruiken

XSLT Stylesheets

- XSLT Stylesheets zijn zelf well-formed XML documenten
- Een stylesheet heeft een root element **<stylesheet>**
  - Komt uit de namespace *http://www.w3.org/1999/XSL/Transform*
  - Namespace wordt opgeven bij het attribuut *xmlns*
    - In deze cursus gebruiken we voor deze namespace steeds de prefix *xsl*
  - Ook het attribuut *version* is verplicht op te geven
- Optioneel kunnen nog een tal attributen worden opgegeven
  - *exclude-result-prefixes*
  - *extension-element-prefixes*
  - *id*
- Het element **<xsl:stylesheet>** mag ook **<xsl:transform>** zijn

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="web"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:web="http://pers.nl">
```

Het element <xsl:stylesheet>

- Binnen `<xsl:stylesheet>` kunnen de volgende child elementen voorkomen
  - `<xsl:template match="...">...</xsl:template>`
  - `<xsl:import href="..."/>`
  - `<xsl:output method="..."/>`
  - `<xsl:include href="..."/>`
  - `<xsl:variable name="...">...</xsl:variable>`
  - `<xsl:param name="...">...</xsl:param>`
  - `<xsl:attribute-set name="...">...</xsl:attribute-set>`
  - `<xsl:decimal-format name="..."/>`
  - `<xsl:strip-space elements="..."/>`
  - `<xsl:preserve-space elements="..."/>`
  - `<xsl:key name="..." match="..." use="..."/>`
  - `<xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>`

Het element <xsl:template>

- Het meest gebruikte element binnen stylesheet is `<xsl:template>`
- Dit element bepaalt wat er moet gebeuren zodra een match gevonden wordt met het attribuut *match*
- In `<xsl:template match="/">` verwijst de **/** naar het root element van het brondocument
  - Bij match mag ook naar andere elementen verwezen worden
- Binnen het template wordt de structuur voor het nieuwe document opgesteld.
- Binnen de structuur kan content uit het brondocument gemengd worden met de nieuwe structuur

Het element <xsl:value-of>

- Om waarden uit het brondocument toe te voegen moeten deze eerst worden opgezocht
- Hiervoor kan **<xsl:value-of>** gebruikt worden
- Dit element heeft een attribuut *select* waarmee wordt bepaald wat er uit het brondocument wordt opgehaald
- De waarde van dit attribuut is een XPath expressie

```
<leden>
  <naam>
    <xsl:value-of select="//pers:voornaam"/>
    <xsl:value-of select="//pers:achternaam"/>
  </naam>
</leden>
```

- In dit voorbeeld worden uit het brondocument de waarden van de elementen voornaam en achternaam opgehaald
  - De **//** in de XPath expressie zoekt naar alle elementen

Het element <xsl:for-each>

- Met `<xsl:value-of>` wordt alleen de eerste node opgehaald
- Om alle nodes te doorlopen kan `<xsl:for-each>` gebruikt worden
- Met het attribuut *select* kan weer de juiste node geselecteerd worden

```
<leden>
  <xsl:for-each select="/pers:personen/pers:persoon">
    <naam>
      <xsl:value-of select="pers:voornaam"/>
      <xsl:value-of select="pers:achternaam"/>
    </naam>
  </xsl:for-each>
</leden>
```

- Binnen `<xsl:for-each>` kan het element `<xsl:sort>` gebruikt worden om de nodeset te sorteren
- Ook `<xsl:sort>` heeft een attribuut *select*

Het element <xsl:if>

- Met het element **<xsl:if>** kan een als… dan… gebouwd worden
- Dit element bevat een attribuut *test*
- Het gedeelte binnen *if* wordt alleen uitgevoerd als de *test* waar is

```
<leden>
  <xsl:for-each select="/pers:personen/pers:persoon">
    <xsl:if test="pers:nationaliteit = 'Nederlands'">
      <naam>
        <xsl:value-of select="pers:voornaam"/>
        <xsl:value-of select="pers:achternaam"/>
      </naam>
    </xsl:if>
  </xsl:for-each>
</leden>
```

Het element <xsl:choose>

- Met **<xsl:if>** kan maar één test gedaan worden
- Bijeen keuze uit meerdere opties wordt **<xsl:choose>** gebruikt
- De verschillende opties worden opgegeven met **<xsl:when>**
- Elke *when* heeft weer een attribuut *test*
- Met **<xsl:otherwise>** kunnen overige situaties worden opgevangen

```
<kleur>
  <xsl:choose>
    <xsl:when test="pers:color ='blue'">blauw</xsl:when>
    <xsl:when test="pers:color ='red'">rood</xsl:when>
    <xsl:when test="pers:color ='green'">groen</xsl:when>
    <xsl:when test="pers:color ='yellow'">geel</xsl:when>
    <xsl:when test="pers:color ='orange'">oranje</xsl:when>
    <xsl:when test="pers:color ='purple'">paars</xsl:when>
    <xsl:otherwise>paars met groene stippen</xsl:otherwise>
  </xsl:choose>
</kleur>
```

DEMO

XPath expressies

- Een XPath expressie beschrijft de weg door de boomstructuur van het brondocument
- Hiervoor moet de structuur van het brondocument goed bekend zijn
- Met XPath expressie kunnen zowel elementen als attributen ophalen
- De term node is de verzamelnaam voor elementen en attributen
  - Een node kan zowel een element als een attribuut zijn
- In een brondocument worden vaak namespaces gebruikt
- Deze namespaces zullen dan ook in het stylesheet moeten worden opgenomen
- Dit geldt ook voor de default namespace van het brondocument
  - Deze krijgt in het stylesheet wel een prefix
  - Het stylesheet heeft zelf al een andere default namespace

XPath expressies

- Binnen XPath zijn de volgende expressies mogelijk

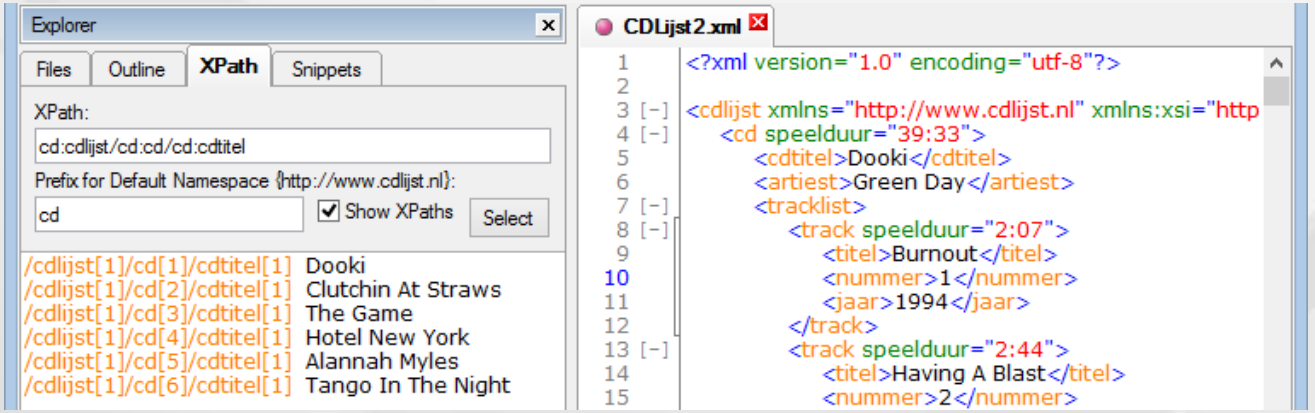
Expressie	beschrijving
nodename	Selecteert alle nodes van met deze naam op deze plek in het pad
/	Selecteert de root (waarvan het root element het enige child element is).
//nodename	Selecteert alle nodes in het document die aan de opgegeven naam voldoen het maakt niet uit waar ze in de structuur staan.
.	Selecteert de huidige node.
..	Selecteert de parent van de huidige node.
@attributename	Selecteert een attribuut met deze naam.
*	Een wildcard voor elke element node.
@*	Een wildcard voor elke attribute node.
node()	Een willekeurige node, niet de root node of een attribuut
text()	Een tekst node

Voorbeelden XPath expressies

Voorbeeld pad expressie	Resultaat
`bestelling/product[2]`	Selecteert het tweede *product* dat een child node is van *bestelling*.
`bestelling/product[last()]`	Selecteert het laatste *product* dat een child node is van *bestelling*.
`//bestelling/product[last()=1]`	Selecteert het product van bestellingen met maar één product (het laatste product is het eerste product).
`bestelling/product[position() <= 2]`	Selecteert de eerste twee producten (child nodes) van *bestelling*.
`//product[2]/@kleur`	Selecteert de waarde van het attribuut *kleur*, van het tweede *product* van elke reeks van producten
`//product[@kleur]`	Selecteert alle nodes *product* met een attribuut *kleur*.
`//product[@kleur='chroom']`	Selecteert alle nodes *product* die een attribuut *kleur* met de waarde '*chroom*' hebben.
`//product[@kleur='chroom']/omschrijving`	Selecteert de node *omschrijving* van alle producten in de kleur '*chroom*'.
`//product[prijs > 700]`	Selecteert alle producten waarvan de *prijs* boven de 700 ligt
`bestelling/product[2]`	Selecteert het tweede *product* dat een child is van *bestelling*.

XPath evaluator

- Met een XPath evaluator kan een XPath expressie getest worden
- In XML blueprint is een XPath evaluator aanwezig
  - Tabblad XPath in de Explorer
- Vul een XPath expressie in en klik op Select
- Vul in het onderste vak een prefix voor de default namespace in
  - Gebruik hier de prefix die ook in de xsl wordt gebruikt



DEMO

Het element <xsl:text>

- Bij een transformatie wordt alle witruimte verwijderd
- Het resultaat van de volgende transformatie koppelt de voornaam en de achternaam aan elkaar

```
<naam>
  <xsl:value-of select="pers:voornaam"/>
  <xsl:value-of select="pers:achternaam"/>
</naam>
```

- Alleen een spatie er tussen levert ook een gekoppeld resultaat

```
<naam>
  <xsl:value-of select="pers:voornaam"/> <xsl:value-of select="pers:achternaam"/>
</naam>
```

- Dit is op te lossen met het element **<xsl:text>**

```
<naam>
  <xsl:value-of select="pers:voornaam"/>
  <xsl:text> </xsl:text>
  <xsl:value-of select="pers:achternaam"/>
</naam>
```

Het element <xsl:text>

- Het element is ook goed te gebruiken om < en > op te nemen
- Het attribuut *disable-output-escaping* moet dan wel *yes* zijn

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">a &gt; b en b &lt; c</xsl:text>
</xsl:template>
```

- Het resultaat ziet er als volgt uit

```
a > b en b < c
```

- Het attribuut *disable-output-escaping* heeft default de waarde *no*

DEMO

Het element <xsl:element>

- Een nieuw element kan ook met **<xsl:element>** gemaakt worden
- De naam moet dan opgegeven worden bij het attribuut *name*
- De volgende 2 transformaties leveren het zelfde resultaat op

```
<naam>
  <xsl:value-of select="pers:voornaam"/>
</naam>
```

```
<xsl:element name="naam">
  <xsl:value-of select="pers:voornaam"/>
</xsl:element>
```

Het element <xsl:element>

- Gebruik van **<xsl:element>** is dynamischer
- De waarde van een node kan als elementnaam worden opgenomen

```
<personen>
  <xsl:for-each select="pers:personen/pers:persoon">
    <xsl:element name="{pers:mv}">
      <xsl:value-of select="pers:voornaam"/>
    </xsl:element>
  </xsl:for-each>
</personen>
```

- Afhankelijk van de waarde wordt nu de tag man of vrouw gemaakt

```
<personen>
  <man>Alex</man>
  <man>Simon</man>
  <vrouw>Marith</vrouw>
</personen>
```

DEMO

Het element <xsl:attribute>

- Om aan een element attributen toe te voegen kan **<xsl:attribute>** gebruikt worden
- Bij *name* kan de naam van het attribuut worden opgegeven

```
<personen>
  <xsl:for-each select="pers:personen/pers:persoon">
    <naam>
      <xsl:attribute name="gebdat">
        <xsl:value-of select="pers:geboortedatum"/>
      </xsl:attribute>
      <xsl:value-of select="pers:voornaam"/>
    </naam>
  </xsl:for-each>
</personen>
```

- Dit element heeft ook nog een optioneel attribuut *namespace*

Het element <xsl:attribute-set>

- Als een groep van attributen aan verschillende elementen toegekend moet worden kan deze als groep gedefinieerd worden
- Hiervoor wordt het element **<xsl:attribute-set>** gebruikt
- Dit element is een child element van **<xsl:stylesheet>**
- Hierbinnen worden de attributen gedefinieerd met **<xsl:attribute>**

```
<xsl:attribute-set name="gegevens">
  <xsl:attribute name="gebdat">
    <xsl:value-of select="pers:geboortedatum"/>
  </xsl:attribute>
  <xsl:attribute name="oogkleur">
    <xsl:value-of select="pers:ogen"/>
  </xsl:attribute>
</xsl:attribute-set>
```

- Een attribuut set kan als volgt aan een element worden toegekend

```
<xsl:element name="naam" use-attribute-sets="gegevens">
```

Variabelen

- Soms moet een opgehaalde waarde even bewaard worden om later weer naar te kunnen verwijzen
- Hiervoor kan het element **<xsl:variable>** worden gebruikt
- Dit element heeft een verplicht attribuut *name*
- Naar de variabele verwijzen kan met **$variabelenaam**

```
<xsl:template match="/">
  <cds>
   <xsl:for-each select="//cd">
     <xsl:variable name="id" select="@id"/>
     <cd titel="{cdtitel}">
       <xsl:for-each select="//track[@cd=$id]">
         <track titel="{titel}"/>
       </xsl:for-each>
     </cd>
   </xsl:for-each>
  </cds>
</xsl:template>
```

DEMO

Copy en copy-of

- Om een node met inhoud uit het brondocument te kopiëren kunnen `<xsl:copy>` en `<xsl:copy-of>` gebruikt worden
  - `<xsl:copy-of>` kopieert de huidige node met child nodes
    - Met *select* is de juiste node te selecteren
    - Namespace van de bron wordt opgenomen in het element
  - `<xsl:copy>` kopieert de huidige node zonder child nodes
    - Geen attribuut select dus eerst naar de juiste node navigeren
    - Namespace van de bron wordt opgenomen in het element

DEMO

Pull benadering

- De elementen *for-each* en *value-of* doorlopen zelf gekozen elementen
- Dit wordt de pull benadering genoemd:
- De stylesheet maakt gebruik van kennis over het xml document om benodigde informatie eruit te trekken
- Voordeel
  - Code redelijk eenvoudig
  - Bekende (vertrouwde) manier van programmeren
- Nadeel
  - Weinig flexibel structuur moet van te voren bekend zijn
  - Alleen de nodes die je uit het document trekt worden verwerkt
  - Overige nodes wordt niets mee gedaan
- De pull benadering is goed te gebruiken als de structuur van het document bekend is en niet verandert

Push benadering

- Tegenover pull benadering staat de push benadering
- Hierbij worden verschillende templates gedefinieerd die aangeven hoe de verschillende nodes verwerkt moeten worden
- Templates worden toegepast op nodes die het stylesheet tegenkomt
- Dit wordt de push benadering genoemd omdat het xml document als het ware door templates van de stylesheet heen geduwd wordt
- Voordeel
  - Beter toepasbaar als de structuur van het document mogelijk uitgebreid wordt (XML = **Extensible** Markup Language)
- Nadeel
  - Minder overzichtelijk programmeren

Templates

- De Push benadering werkt met verschillende templates
- Tot nu toe is steeds gewerkt met stylesheets met één template
- Maar er zijn meerdere templates te definiëren
- Templates worden toegepast met `<xsl:apply-templates>`

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

- Als er geen template wordt toegepast zal er niets gebeuren

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

</xsl:template>
```

Templates

- Als er geen extra templates gedefinieerd zijn, zal een standaard template door `<xsl:apply-templates>` worden uitgevoerd
- Standaard worden alleen alle elementen en tekst nodes opgehaald
  - Attributen worden dus niet behandeld
- De volgende code zal alle tekst nodes vanaf het root element tonen

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

- Merk op dat waarden van eventuele attributen niet getoond worden

Templates

- Om attributen te verwerken is een extra template nodig

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

- Merk op dat de *match* van dit template * (alle nodes) is
- De *select* van geeft aan dat dit op zowel child elementen, tekst nodes als attributen toegepast moet worden
- In plaats van *|tekst() mag ook node() gebruikt worden
  - node() ziet geen attributen deze worden wel apart toegevoegd

DEMO

Recursieve templates

- Met `<xsl:copy-of>` kan een hele structuur gekopieerd worden
  - Geen afhandeling voor individuele nodes mogelijk

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

- Met een recursief template is een aparte afhandeling wel mogelijk

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

- Elke node wordt gekopieerd.
- Voor elke child node wordt het template opnieuw aangeroepen
- Specifieke nodes kunnen met extra templates verwerkt worden

Templates importeren

- Bestaande templates kunnen geïmporteerd worden
- Stylesheets kunnen zo eenvoudig opnieuw gebruikt worden
- Hiervoor wordt het element **<xsl:import>** gebruikt
- Het attribuut *href* bevat de verwijzing naar het betreffende stylesheet

```
<xsl:import href="identity.xsl/>

<xsl:template match="pers:adres">
  <adres>
    <xsl:value-of select="pers:straat"/>, <xsl:value-of select="pers:plaats"/>
  </adres>
</xsl:template>
```

- Hier worden alle nodes afgehandeld door het stylesheet identity.xsl
  - Bevat de code van het recursieve template (vorige sheet)
- Alleen het element adres wordt apart afgehandeld

DEMO

Opdracht 1 Hoofdstuk 2 (succes)

Templates afhandelen

- Een nodes kan door verschillende templates afgehandeld worden
- In dat geval wordt het meest specifieke template uitgevoerd
- Een specifieker template heeft een hogere prioriteit
- Als twee templates even specifiek zijn wordt de laatste gebruikt
- Geïmporteerde templates hebben altijd een lagere prioriteit dan lokale templates
- Een geïmporteerd template krijgt voorrang als `<xsl:apply-imports/>` gebruikt wordt in plaats van `<xsl:apply-templates/>`
- Als een stylesheet met `<xsl:include>` wordt ingevoegd wordt deze als een lokale template afgehandeld de prioriteit is dan dus hoger
  - Positie van `<xsl:include>` kan van belang zijn als templates even specifiek zijn

DEMO

Template mode

- Als een node op verschillende manier bewerkt moet worden zit de prioriteit ons in de weg
- In dat geval kan in verschillende modes gewerkt worden
- Op te geven bij attribuut *mode* van **<xsl:template>**
- Bij het aanroepen met **<xsl:apply-templates>** ook *mode* opgegeven

```
<xsl:template match="/">
  <namen>
    <hoofdletters>
      <xsl:apply-templates mode="upper"/>
    </hoofdletters>
    <klein>
      <xsl:apply-templates mode="lower"/>
    </klein>
  </namen>
</xsl:template>
```

DEMO

Templates aan roepen

- Ook in stylesheets met de Pull benadering kunnen verschillende templates gebruikt worden
- Deze templates worden aangeroepen met `<xsl:call-template>`
- Het template moet dan wel een naam hebben
- Naar deze naam wordt verwezen bij `<xsl:call-template>`

```
<xsl:choose>
  <xsl:when test="//cd:albumhoes">
    <xsl:call-template name="met"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:call-template name="zonder"/>
  </xsl:otherwise>
</xsl:choose>
```

DEMO

Opmaak van getallen

- In XML worden getallen standaard in de Amerikaanse notatie
  - Decimale . en , als 1000 tallen scheiding
- Met de elementen `<xsl:number>` en `<xsl:decimal-format>` kan een andere opmaak gedefinieerd worden

Het element <xsl:number>

- Het element **<xsl:number>** kan gebruikt worden voor de opmaak van gehele getallen
- Het attribuut *value* bevat een XPath Expressie naar het juiste element
  - Haalt de waarde ook gelijk op er is geen **<xsl:value-of>** nodig
- Met het attribuut *format* kan een opmaak gedefinieerd worden
  - format="1" geeft de getallen (default)
    eventueel zijn voorloop nullen op te geven
  - format ="a" geeft getallen in letters 1=a 26=z 27=aa
    Hoofdletters door A als waarde op te geven
  - format ="i" geeft getallen in romeinse cijfers
    Hoofdletter I geeft waarde in hoofdletters
- Met *grouping-seperator* is een groeperingsymbool op te geven
  - Alleen zinvol als *grouping-size* ook een waarde heeft

DEMO

Nummeren

- Het element `<xsl:number>` kan ook gebruikt worden voor het nummeren van elementen
- Met het attribuut *count* geven we aan welke nodes geteld worden
  - Meerdere nodes op te geven kan gescheiden door |
- Met het attribuut *level* is een nummering op meerdere niveaus mogelijk
  - Mogelijke waarden: single (default), multiple en any
  - Met *format* is voor de verschillende niveaus een eigen opmaak te definiëren

Nummeren

- Het volgende voorbeeld nummert de hoofdstuk en paragraaf elementen op verschillende niveaus

```
<xsl:template match="/">
  <inhoud>
    <xsl:for-each select=".//hoofdstuk|.//paragraaf">
      <xsl:element name="{name()}">
        <xsl:number count="hoofdstuk|paragraaf" level="multiple"/>
      </xsl:element>
    </xsl:for-each>
  </inhoud>
</xsl:template>
```

DEMO

Functie format-number

- Het element **<xsl:number>** is alleen te gebruiken voor de opmaak van gehele getallen zoals valuta of percentage opmaak
- Met de XSLT functie *format-number* hebben we meer mogelijkheden
  - `<xsl:value-of select="format-number(pro:prijs,'€ #.00)'">`
- Deze functie krijgt een XPath expressie als eerste parameter
- De tweede parameter is een opmaak masker op te geven tussen ' '
  - een aparte masker voor negatieve getallen op te geven achter ;

Functie format-number

- Standaard worden decimalen achter de . gegeven en de duizendtalgroepering is een ,
- In het masker punt en komma wisselen is nodig maar niet voldoende
- Er moet een element `<xsl:decimal-format>` opgenomen worden
- Dit element heeft 3 belangrijke attributen
  - *name* om het format een naam te geven
  - *decimaal-seperator* definitie van decimaal scheidingssymbool
  - *grouping-seperator* definitie van duizendtal groeperingssymbool
- Het element `<xsl:decimal-format>` moet altijd buiten het template opgenomen worden
  - Wel meerdere keren op te nemen
  - Derde parameter in format-number verwijst naar de juiste naam

DEMO

XSLT functies

- Naast *format-number* zijn er nog meer XSLT functies

Functie	Beschrijving
current`()`	Geeft de huidige node terug. Merk op dat select="current()" vergelijkbaar is met select= ".".
document`(string [,node-set])`	Functie om de node set van een extern xml document te benaderen.
element-available`(string)`	Test of een opgegeven XSLT element wordt ondersteund door de XSLT processor.
format-number`(number,format [,decimalformat])`	Converteert een getal naar een tekst in de opgegeven notatie.
function-available`(string)`	Test of een opgegeven functie wordt ondersteund door de XSLT processor.
generate-id`([node-set])`	Deze functie geeft een unieke tekstwaarde waarmee een node kan worden geïdentificeerd.
key`(string, value)`	Deze functie geeft een node-set bij de opgeven waarde. Er wordt gerefereerd aan een <xsl:key> element.
system-property`(string)`	Deze functie geeft een aantal systeem eigenschappen. Mogelijk parameter-waarden zijn: xsl:version, xsl:vendor en xsl:vendor-url.
unparsed-entity-uri`(string)`	Deze functie geeft de URI van een niet geparste entity.

De XSLT functie document()

- De functie document kan gebruikt worden om gegevens uit een ander xml document in te voegen
- De functie krijgt een bestandsnaam als parameter
- Eventueel gevolgd door een XPath expressie om naar de juiste node te navigeren

```
<xsl:template match="/">
  <cdlijst xmlns="http://www.cd.nl">
    <xsl:copy-of select="document('cdlijst1.xml')/cd:cdlijst/cd:cd"/>
    <xsl:copy-of select="document('cdlijst2.xml')/cd:cdlijst/cd:cd"/>
  </cdlijst>
</xsl:template>
```

- Beide documenten worden nu samengevoegd
- Merk op dat deze transformatie met elk willekeurig xml document uitgevoerd kan worden

DEMO

De XSLT functie key()

- Om elementen snel te vinden kan de functie *key()* gebruikt worden
- Een key moet eerst gedefinieerd met het element <xsl:key> worden
- Dit element heeft 3 attributen
  - *name* naam van de key
  - *match* een XPath expressie voor het zoeken van de juiste node
  - *use* het element of attribuut dat wordt gebruikt als sleutel om de gezochte nodes te vinden
    plaats in de boomstructuur is hier niet van belang
- Het key element maakt in het geheugen een extra node-set aan van sleutelwaarden met bijbehorende nodes
- Bij het aanroepen van de functie worden de naam van de key en de zoekwaarde als parameters mee gegeven

De XSLT functie key()

- Voorbeeld

```
<xsl:key match="pers:persoon" name="pers_key" use="pers:achternaam"/>

<xsl:template match="/">
  <xsl:for-each select="key('pers_key', 'Scholten')">
    <xsl:copy-of select="pers:adres"/>
  </xsl:for-each>
</xsl:template>
```

- Het key element pers_key zoekt naar personen
- De achternaam wordt als sleutel gebruikt
- De key wordt in dit geval gebruikt bij de select van `<xsl:for-each>`
- De functie key() haalt alle personen op met de achternaam Scholten
- Van deze personen wordt het adres getoond.

DEMO

XPath expressies met relatieve paden

- Tot nu toe zijn alleen XPath expressies met absoluut pad besproken
- We navigeren heel precies door de boomstructuur
- Soms is het makkelijker om een relatief pad te gebruiken
  - Om alle child nodes te selecteren
  - Om alle nodes op het zelfde niveau te selecteren
- Voor een relatief pad worden axes (assen) gebruikt

XPath Axes

Axis	Resultaat
ancestor	Selecteert alle bovenliggende nodes (parent, grandparent, etc.) van de huidige node
ancestor-or-self	Selecteert de huidige node en alle bovenliggende nodes (parent, grandparent, etc.).
attribute	Selecteert alle attributen van de huidige node.
child	Selecteert alle child elementen van de huidige node.
descendant	Selecteert alle onderliggende nodes (children, grandchildren, etc.) van de huidige node.
descendant-or-self	Selecteert de huidige node en alle onderliggende nodes (children, grandchildren, etc.).
following	Selecteert alle nodes na de huidige node.
following-sibling	Selecteert alle nodes na de huidige node die op hetzelfde niveau als de huidige node staan.
namespace	Selecteert alle namespace nodes van de huidige node.
parent	Selecteert de bovenliggende node van de huidige node.
preceding	Selecteert alle nodes voor de huidige node.
preceding-sibling	Selecteert alle nodes voor de huidige node die op hetzelfde niveau als de huidige node staan.

XPath Axes

- Een axis moet op de juiste plaats in een XPath worden opgenomen
- De axis wordt altijd gevolgd door ::
- Achter de laatste dubbele punt staat de naam van een node of een *
- Voorbeelden
  - */bestelling/product/descendant::**
    selecteert alle child nodes van elk *product*
  - */bestelling/product/attribute::**
    selecteert alle attributen van het element *product*
  - */bestelling/product/attribute::type*
    selecteert van elk *product* met een attribuut *type* dit attribuut

DEMO

XPath operatoren

- In XPath expressies worden operatoren gebruikt voor bewerkingen

+	optellen	*prijs + btw*
-	aftrekken	*prijs - btw*
*****	vermenigvuldigen	*prijs * 1.19*
div	delen	*aantal div 4*
mod	modulus (rest van een deling)	*aantal mod 4*
|	verwerkt twee nodesets	*//boek | //cd*

- Daarnaast zijn er vergelijkingsoperatoren te gebruiken in XPath expressies

=	is gelijk aan	*aantal=10*
!=	is ongelijk aan	*aantal!=10*
<	kleiner dan	*aantal<10*
<=	kleiner of gelijk aan	*aantal<=10*
>	groter dan	*aantal >10*
>=	groter of gelijk aan	*aantal >=10*
or	logische of	*aantal =10 or aantal =20*
and	logische en	*aantal >10 and aantal <20*

XPath operatoren

- Bij het vergelijken van nodesets moeten we goed opletten
- XPath operatoren werken met bijzondere principes
  - Twee nodesets zijn gelijk als in beide nodesets een node met dezelfde waarde voorkomt
  - Twee nodesets zijn ongelijk als er minimaal één node niet in beide set voorkomt
- //aantal = 3 is waar als minimaal één node aantal de waarde 3 heeft
- //aantal !=3 is waar als minimaal één node aantal ongelijk aan 3 is
- Voor één nodeset kan dus tegelijk gelden //aantal=3 en //aantal != 3

XPath operatoren

- Voorbeeld XPath operatoren

```
<xsl:template match="bes:aantal">
  <xsl:copy-of select="."/>
    <xsl:if test="../bes:aantal > 1">
      <totaal>
        <xsl:value-of select="../bes:aantal * ../bes:prijs"/>
      </totaal>
    </xsl:if>
</xsl:template>
```

- In de test wordt de XPath operator > gebruikt, dit mag ook > zijn
  - Bij een test kleiner mag geen < maar moet < gebruikt worden

DEMO

XPath functies

- Naast XPath operatoren zijn er ook XPath functies
- Met functies kunnen nodes of nodesets bewerkt worden
- Een functie geeft een waarde terug
- Aan functies kunnen 1 of meerdere parameters worden meegegeven
  - Parameters worden achter de functie tussen haakjes opgenomen
  - Meerdere parameters scheiden door een komma
  - Bij functies zonder parameters wel de haakjes opnemen
  - Parameters kunnen verplicht of optioneel zijn
- Een aantal functies zijn we al eens tegen gekomen
  - name()
  - concat()
  - last()
  - position()

XPath functies

- XPath functies worden in verschillende categorieën ingedeeld
  - Boolean functies
    - worden gebruikt voor testen
    - Geven *true* of *false* terug
  - Node-set functies
    - Geven informatie over nodesets en hun nodes terug
    - Sommige functies kunnen een node-set mee krijgen als parameter.
  - Numerieke functies
    - Numerieke functies worden gebruikt om berekeningen uit te voeren.
    - Deze functies geven altijd een getal terug
  - String functies
    - voeren bewerkingen uit op tekstwaarden en tekst nodes
- Per categorie zullen we de belangrijkste functies noemen
  - Als een parameter wordt gevolgd door een ? is deze optioneel

XPath nodeset functies

position()	Deze functie geeft de positie van de huidige node terug als getal.
last()	Deze functie geeft de positie van de laatste node terug als getal.
namespace-uri(node-set?)	Deze functie geeft de namespace uri van de huidige node (default)\ als string terug, of van de eerste node van de opgegeven node-set.
id(object)	Deze functie selecteert elementen met het opgegeven id.
local-name(node-set?)	De functie local-name() geeft een string terug met de het lokale deel van de naam van de huidige node (default), of van de eerste node van de opgegeven node-set.
name(node-set?)	De functie name doet bijna het zelfde als local-name() alleen wordt er nu een qname (qualfied name) terug gegeven. Dit is handig wanneer nodes met dezelfde naam in verschillende namespaces voorkomen.

DEMO

Boolean functies

- boolean(object) Deze functie converteert het opgegeven argument naar een boolean.
  - Een getal wordt "false" als het 0 of NaN is.
  - Een tekst wordt "false" als de tekst lengte 0 heeft.
  - Een node set wordt false als deze leeg is .
- not(boolean) De functie not() geeft het tegengestelde van een boolean expressie.
- true() Deze functie heeft geen argumenten en geeft altijd "true" terug.
- false() Deze functie heeft geen argumenten en geeft altijd "false" terug.

Numerieke functies

- round(number) Deze functie krijgt als argument een getal mee en geeft als resultaat de afgeronde waarde van het getal als integer terug.
- floor(number) Deze functie krijgt als argument een getal mee en geeft als resultaat de naar beneden afgeronde waarde van het getal als integer terug.
- ceiling(number) Deze functie krijgt als argument een getal mee en geeft als resultaat de naar boven afgeronde waarde van het getal als integer terug.
- sum(node-set) Deze functie krijgt als argument een node-set mee. Het resultaat is de optelling van de waarden van de opgegeven node in deze node-set.
- count(node-set) Deze functie krijgt als argument een node-set mee. Het resultaat is aantal keren dat de opgegeven node in deze node-set voorkomt.
- number(object?) Deze functie converteert het opgeven argument naar een getal. Als dit niet mogelijk is wordt NaN terug gegeven. Het argument kan een string, een boolean of een node-set of zijn. Van een node-set wordt eerst de string weergave bepaald. Een boolean geeft een "1" terug bij "true" en een "0" bij "false".

String functies

string(object?)	Deze functie converteert het opgeven argument naar een tekst. Het argument kan een number, een boolean of een node-set zijn.
concat(string, string, ..?)	De functie concat zet alle opgeven argumenten achter elkaar zodat ze samen 1 tekstwaarde vormen.
starts-with(string, string)	Deze functie geeft een boolean waarde terug. Het resultaat is "true" als het eerste argument begint met het tweede argument.
contains(string, string)	Deze functie geeft een boolean waarde terug. Het resultaat is "true" als het tweede argument in het eerste argument gevonden wordt.
normalize-space(string?)	Deze functie haalt bij het opgegeven argument alle spaties aan het begin en eind weg. Tussen twee woorden wordt het aantal spaties teruggebracht tot één. Het resultaat wordt als een string terug gegeven.
translate(string, string, string)	Deze functie vervangt in het eerste argument de tekens van het tweede argument door de overeenkomstige tekens van het derde argument.

String functies

- substring(string, number, number?)

  De functie substring geeft een deel van het eerste argument terug beginnend op de positie die wordt opgegeven bij het tweede argument. Het derde argument is optioneel; hiermee kan de lengte van de substring worden opgegeven.
  - voorbeeld: substring("123456", 3) wordt 3456.
  - substring("123456",3,2) wordt 34.
- substring-before(string, string)

  De functie substring-before geeft een deel van het eerste argument terug dat voor het tweede argument staat.
- substring-after(string, string)

  De functie substring-after geeft een deel van het eerste argument terug dat na het tweede argument staat.
- string-length(string?)

  Deze functie telt het aantal karakters in het argument. en geeft dit als een getal terug.

DEMO

Opdracht 2 en 3 Hoofdstuk 2 (succes)

Parameters

- Soms willen we aan een template parameters meegeven
- De transformatie wordt dan uitgevoerd met de waarde van de parameter
- Een parameter definiëren we met `<xsl:param>`
  - Dit element heeft een verplicht attribuut *name*
  - Ook is optioneel een attribuut *select* te gebruiken

Parameters

- Een parameter wordt gevuld met het element **<xsl:with-param>**
  - Dit is een childelement van **<xsl:call-template>** of **<xsl:apply-templates>**
- Dit element heeft dezelfde 2 attributen
  - *name* is verplicht en verwijst naar de naam van de parameter
  - *select* is weer optioneel
- In een XPath expressie kan naar een parameter verwezen worden met *$parameternaam*

Parameters

- Het volgende templaten gebruikt parameters om een totaal te berekenen

```
<xsl:template name="bereken">
  <xsl:param name="artikelen"/>
  <xsl:choose>
    <xsl:when test="$artikelen">
      <xsl:variable name="recursief_resultaat">
        <xsl:call-template name="bereken">
          <xsl:with-param name="artikelen" select="$artikelen[position()>1]"/>
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="number($artikelen[1]/prijs)* number($artikelen[1]/aantal)
                            + $recursief_resultaat"/>
    </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="0"/>
      </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

- Het template wordt met twee parameters aangeroepen:
  - een lijst met producten *artikelen* en een resultaat *result*
- Het template wordt recursief (herhaald) aangeroepen, zolang er artikelen zijn
- Het resultaat wordt uiteindelijk teruggeven d.m.v. *$result*

Parameters

- Het template wordt voor de eerste keer aangeroepen bij het maken van het element totaal

```
<totaal>
   <xsl:call-template name="bereken">
      <xsl:with-param name="artikelen" select="artikel"/>
      <xsl:with-param name="result" select="0"/>
   </xsl:call-template>
</totaal>
```

- Hier wordt de parameter artikelen gevuld met een nodeset artikel
- De parameter *result* krijgt in het begin de waarde 0

DEMO

Overige opdrachten Hoofdstuk 2 (succes)

Hoofdstuk 3 XSLT en XPath 2.0

Leerdoelen:

XSLT 2.0 elementen kennen en gebruiken

Verschil tussen XSLT 1.0 en XSLT 2.0 kennen

Bewerkingen met XSLT 2.0 functies kunnen uitvoeren

Nieuwe XPath 2.0 functionaliteit kunnen gebruiken

Zelf nieuwe functies kunnen maken

XSLT en XPath 2.0

- Sinds januari 2007 heeft w3c XSLT 2.0 vastgelegd
- Daar aan gekoppeld is ook de nieuwe standaard voor XPath 2.0
- Veel van XSLT 1.0 gewoon in XSLT 2.0 te gebruiken
- Maar XSLT 2.0 is meer dan alleen een uitbreiding
- Ook een verandering in een aantal concepten
- En verandering in het onderliggende datamodel

XSLT en XPath 2.0

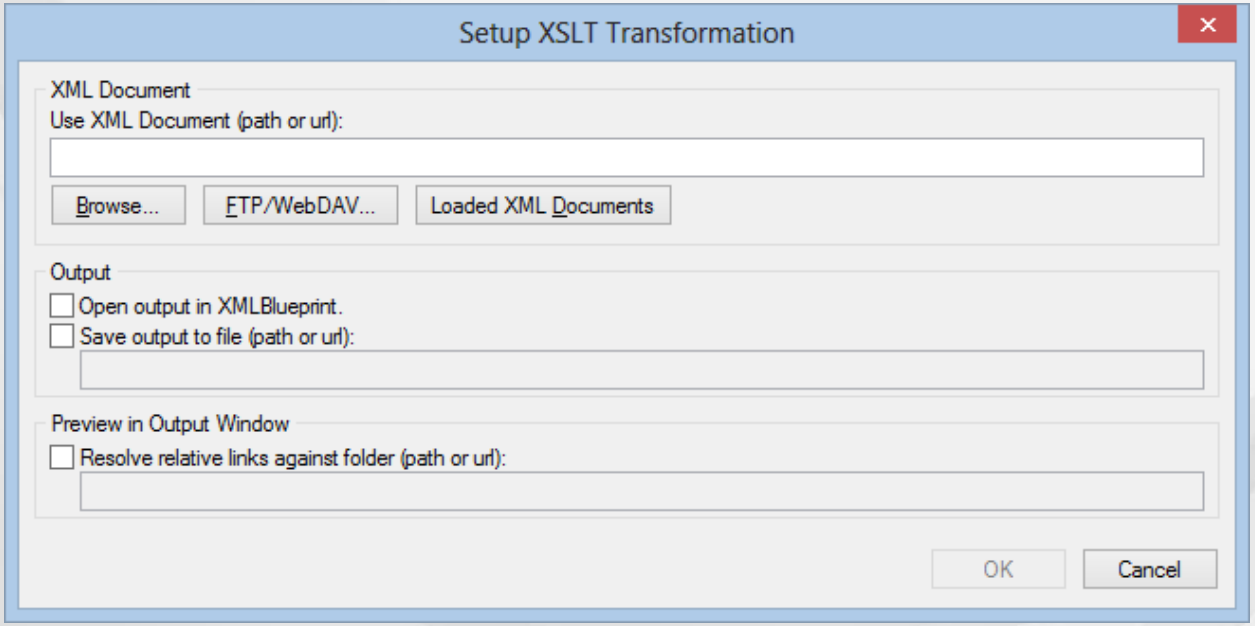
- Voor XSLT 2.0 stylesheets is het volgende van belang
- XSLT 1.0 en XSLT 2.0 elementen komen uit dezelfde namespace
- Het attribuut *version* moet de waarde 2.0 hebben

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- Het stylesheet kan nu niet door de standaard parser verwerkt worden
- Er is een XSLT 2.0 parser nodig
- In deze cursus gebruiken we XSLT 2.0 parser *Saxon 8B*

Parser Saxon 8B

- Voor een transformatie met Saxon 8B moet het stylesheet eerst opgeslagen worden
- Daarna kan de transformatie uitgevoerd worden via *Tools , Saxon 8B*
- Wel moet eerst een XML document gekoppeld zijn met *Setup*



DEMO

Element selectie

- Het selecteren van elementen is in XSLT 2.0 anders dan in XSLT 1.0

```
<xsl:template match="/">
  <cd>
    <xsl:value-of select="//cd:cdtitel"/>
  </cd>
</xsl:template>
```

- Met XSLT 1.0 levert dit template de titel van de eerste cd op
- In XSLT 2.0 levert dit template ene lijst van alle cd-titels
- De lijst wordt standaard gescheiden door spaties
- Met het attribuut *separator* mag een ander scheidingsteken opgegeven worden

```
<xsl:value-of select="//cd:cdtitel" separator=", "/>
```

Nieuwe XSLT 2.0 elementen

- In XSLT 2.0 zijn een heel aantal nieuwe elementen toegevoegd
  - Schema validatie in het stylesheet kan met `<xsl:import-schema>`
- Daarnaast is er aan bestaande elementen functionaliteit toegevoegd
  - Zo hebben `<xsl:copy-of>` , `<xsl:element>` en `<xsl:document>` nu een attribuut *validation* om een validatiemethode af te dwingen
  - Ook zijn er elementen met een attribuut type of collation
- Een aantal van deze nieuwe elementen zullen we hier bespreken

Het element <xsl:analyze-string>

- Met **<xsl:analyze-string>** kan een tekst gecontroleerd worden
- Tekst(delen) die voldoen aan de opgegeven reguliere expressie worden afgehandeld volgens **<xsl:matching-substring>**
- De tekst (delen) die niet aan de reguliere expressie voldoen worden afgehandeld volgens **<xsl:non-matching-substring>**

```
<xsl:analyze-string regex="^(\w)+(\.(\w)+)?@(\w)+\.([a-z]){{2,}}$" select=".">
  <xsl:matching-substring>
    <correct> <xsl:value-of select="."/> </correct>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <incorrect> <xsl:value-of select="."/> </incorrect>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

- Let op dat **<xsl:matching-substring>** alleen op de gevonden substring wordt toegepast

DEMO

Het element <xsl:character-map>

- Met **<xsl:character-map>** kan een karakter vervangen worden door een andere tekst
- Dit element is een child element van **<xsl:stylesheet>**
- Het element **<xsl:output>** zorgt dat de **<xsl:character-map>** wordt toegepast

```
<xsl:character-map name="html_char">
  <xsl:output-character character="&#xA;" string="&lt;br&gt;"/>
</xsl:character-map>

<xsl:output indent="yes" use-character-maps="html_char" method="html"/>
```

- &#xA is de hexadecimale weergave van een harde return
-
 is de tekst
  - Dit is html code voor een nieuwe regel

DEMO

Het element <xsl:next-match>

- Bij meerdere templates wordt meest specifieke template uitgevoerd
- Andere templates worden genegeerd
- Meerdere templates uitvoeren kan met **<xsl:next-match>**
- Met het attribuut *priority* kan een volgorde worden afgedwongen
  - Normaal ligt de prioriteit tussen -0.5 en +0.5

```
<xsl:template match="lid[@datum='2003-10-03']">
  <xsl:next-match/> (vanaf begin)
</xsl:template>

<xsl:template match="lid[substring(@datum,1,4) = '2010']">
  <xsl:next-match/> (nieuw)
</xsl:template>

<xsl:template match="lid" priority="1">
  <tr><td><xsl:next-match/></td></tr>
</xsl:template>
```

DEMO

Het element <xsl:perform-sort>

- Variabelen kunnen een waarde of een nodeset zijn
- In XSLT 1.0 is een nodeset (result tree) vrij statisch
  - Kan met `<xsl:copy-of>` of `<xsl:value-of>` benaderd worden
- XSLT 2.0 nodesets worden in het geheugen geladen (temporary tree)
  - Deze kan als een gewone nodeset doorlopen worden
- Een nodeset kan gesorteerd worden met `<xsl:perform-sort>`
  - Er is geen `<xsl:for-each>` nodig

```
<xsl:variable name="lijst">
  <xsl:perform-sort select="cd:cd">
    <xsl:sort select="cd:cdtitel"/>
  </xsl:perform-sort>
</xsl:variable>
```

Het element <xsl:result-document>

- Met **<xsl:result-document>** is het mogelijk om het resultaat van een transformatie naar verschillende documenten weg te schrijven

```
<xsl:template match="/cd:cdlijst">

 <xsl:for-each select="cd:cd">
  <xsl:variable name="bestandsnaam">
   <xsl:value-of select="concat('///H:/artiest/' , cd:artiest, '.xml')"/>
  </xsl:variable>

  <xsl:result-document href="{$bestandsnaam}">
   <xsl:copy-of select="."/>
  </xsl:result-document>
 </xsl:for-each>

</xsl:template>
```

DEMO

Het element <xsl:sequence>

- Met `<xsl:sequence>` wordt een sequence in het geheugen gemaakt
- Een sequence is een serie waarden, of nodes zonder parent-node
- Hiermee kunnen we bijvoorbeeld een variabele vullen

```
<xsl:template match="/">
  <xsl:variable name="nummers" as="xs:integer *">
    <xsl:sequence select="1 to 4"/>
    <xsl:sequence select="(11, 13, 17)"/>
    <xsl:for-each select="1 to 3">
      <xsl:sequence select="20 + 2 * ."/>
    </xsl:for-each>
  </xsl:variable>

  <nummers>
    <xsl:value-of select="$nummers" separator=", "/>
  </nummers>

</xsl:template>
```

Variabelen van een opgegeven type

- Op de vorige sheet werd een variabele aangemaakt
- Hier is een nieuw attribuut *as* aan toegevoegd
  - `as="xs:integer *"`
- In XSLT 2.0 kan een variabele een XMLSchema (atomic) type krijgen
  - xs:integer, xs:date, xs:time, xs:float, xs:boolean of xs:string
  - De schema namespace moet dan wel toegevoegd worden
- Bij het uitvoeren van de transformatie wordt gecontroleerd of de inhoud wel voldoet aan het type
- Achter het type mag opgegeven worden hoeveel waarden de variabele kan bevatten.
  - * de variabele mag nul of meer waarden bevatten
  - + de variabele mag één of meer waarden bevatten
  - ? de variabele mag nul of één waarden bevatten
  - Zonder aanduiding mag de variabele één waarde bevatten

Variabelen van een opgegeven type

- Het type van een variabele hoeft niet altijd een atomic type te zijn
- Het kan ook een ander type zijn zoals een nodeset
- Voorbeelden
  - element()+ 1 of meer elementen
  - node() een node
  - attribute()? een optioneel attribuut
  - element(cd:cd)* 0 of meer elementen van het type cd
  - item()* 0 of meer nodes of atomic types

Sequence eigenschappen

- Een sequence met nodes van een node-type maakt een verwijzing naar de oorspronkelijke node in de boomstructuur
- Geen kopie in het geheugen zoals bij **<xsl:copy>** en **<xsl:copy-of>**
- Met XPath expressies kan naar ander node worden verwezen

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

<xsl:variable name="seq_voorbeeld" as="element()">
<xsl:sequence select="cd:cdlijst/cd:cd[1]/cd:cdtitel"/>
</xsl:variable>

<sequence>
  <titel><xsl:value-of select="$seq_voorbeeld"/></titel>
  <artiest><xsl:value-of select="$seq_voorbeeld/../cd:artiest"/></artiest>
</sequence>

</xsl:template>
```

- Hier wordt naar de parent node van de variabele verwezen

DEMO

Groeperen

- Groeperen is een belangrijke toevoeging binnen XSLT 2.0
- Met `<xsl:for-each-group>` worden nodes per waarde gegroepeerd
- Het attribuut *select* geeft aan welke waarde gegroepeerd wordt

```
<xsl:for-each-group select="cd" group-by="land">
```

- Van groepen kunnen we m.b.v. functies extra informatie tonen
  - Huidige node met current-group()
  - Huidige waarde van de groep met current-grouping-key()
  - Aantal van de groep met count()
  - Totaal van de groep met sum()
  - Gemiddelde van de groep met avg()
  - Minimum van de groep met min()
  - Maximum van de groep met max()

Groeperen

- Voorbeeld groeperen van cd's per land

```
<xsl:template match="lijst">
  <landen>
    <xsl:for-each-group select="cd" group-by="land">
      <land>
        <code>
          <xsl:value-of select="current-grouping-key()"/>
        </code>
        <artiesten>
          <xsl:copy-of select="current-group()/artiest"/>
        </artiesten>
        <aantal>
          <xsl:copy-of select="count(current-group())"/>
        </aantal>
        <totaal>
          <xsl:copy-of select="sum(current-group()/prijs)"/>
        </totaal>
      </land>
    </xsl:for-each-group>
  </landen>
</xsl:template>
```

DEMO

Anders groeperen

- Het element `<xsl:for-each-group>` kent nog een aantal attributen
- Deze attributen komen in de plaats van *group-by*
  - group-by
    Elementen worden eerst gesorteerd en dan gegroepeerd
  - group-adjacent
    Elementen worden niet gesorteerd. Bij elke andere waarde begint een nieuwe groep
  - group-starting-with
    Nieuwe groep beginnen als opgegeven element gevonden is
  - group-ending-with
    Eindigt de groep als opgegeven element gevonden is

DEMO

Opdracht 1 t/m 3 Hoofdstuk 3 (succes)

XPath 2.0

- Met XSLT 2.0 is er ook een nieuwe versie van XPath ontwikkeld
- XPath 2.0 is vooral een uitbreiding op XPath 1.0
  - Alle XML schema datatypen kunnen in XPath gebruikt worden
  - Ook eigen datatypen kunnen nu in XPath gebruikt worden
  - Veel meer functies beschikbaar
  - En zelf functies maken is ook mogelijk
  - Variabelen zijn nu toegestaan in XPath expressies

XPath sequences

- XPath 2.0 werkt met sequences in plaats van nodesets
- Een sequence kan waarden bevatten maar ook een lijst van nodes
- Dus meer controle en flexibiliteit bij de verwerking van expressies
- Bijzondere sequences zijn
  - empty sequence bevat geen items
  - singleton sequence bevat precies één item
- Waarden uit een sequence worden opgehaald met `<xsl:value-of>`
- Er zijn ook diverse functies en operatoren beschikbaar om sequences te bewerken

Set operatoren voor sequences

- Om verschillende sequences te combineren zijn een aantal set operators beschikbaar
- Set operators hebben alleen betrekking op sequences van nodes, niet op atomic types

operator	beschrijving	voorbeeld	uitvoer
| of ,	geeft de union van twee sequences	*((item1, item2) | (item3))* *of ((item1, item2), (item3))*	*item1, item2, item3*
intersect	geeft de overlap tussen twee sequences	*(item1, item2, item3) intersect (item1, item2, item4)*	*item1, item2*
except	geeft het verschil tussen twee sequences	*(item1, item2, item3) except (item1, item2, item4)*	*item3, item4*

Functies voor sequences

- De volgende functies kunnen op sequences van nodes en van atomic types worden toegepast

functie	beschrijving	voorbeeld	uitvoer
index-of	geeft de posities aan van gezocht item in een sequence	*index-of((1,3,5,3),3)*	*(2,4)*
remove	verwijdert item op aangegeven positie	*remove((1,3,5),3)*	*(1,3)*
empty	test of een sequence leeg is	*empty(1,3)*	*false*
exists	test of een sequence *niet* leeg is	*exists(1,3)*	*true*
distinct-values	geeft de unieke waarden uit een sequence	*distinct-values(1,3,3)*	*(1,3)*
reverse	draait de volgorde van items in een sequence om	*reverse(2,4,6)*	*(6,4,2)*
subsequence	haalt een deel van een sequence op, vanaf een startpositie, met een bepaalde lengte	*subsequence((1,2,3,4,5), 2, 3)*	*(2,3,4)*
unordered	retourneert de items in applicatie-afhankelijke (willekeurige) volgorde	*unordered(//titel)*	*een reeks titels*

Groepsfuncties voor sequences

- De groepsfuncties kunnen ook op een XPath sequence worden toegepast

functie	beschrijving	voorbeeld	uitvoer
count	geeft het aantal items van de opgegeven sequence	*count((1,3,5,3))*	*4*
min	geeft de laagste waarde van de opgegeven sequence	*min((1,3,5,3))*	*1*
max	geeft de hoogste waarde van de opgegeven sequence	*max((1,3,5,3))*	*5*
sum	geeft het totaal van de opgegeven sequence	*sum((1,3,5,3))*	*12*
avg	geeft het gemiddelde van de opgegeven sequence	*avg((1,3,5,3))*	*3*

XPath Functies

- De besproken functies hebben betrekking op een sequence
- Naast de functies voor sequence zijn er veel (nieuwe) XPath functies
- De XPath 2.0 functies zijn aan een nieuwe namespace gekoppeld
- Deze namespace heeft standaard de prefix fn
  - `xmlns:fn="http://www.w3.org/2005/xpath-functions"`

DEMO

Constructors

- In de vorige demo is een functie uit de schema namespace gebruikt
  `xs:date('2011-05-31')`
- Deze functie is een zogenaamde constructor
- Met een constructor wordt een waarde van een bepaald datatype gemaakt
- We kennen constructors voor atomic data typen zoals :
  - `xs:integer('waarde')`
  - `xs:float('waarde')`
  - `xs:string('waarde')`
  - `xs:date('waarde')`
  - `xs:time('waarde')`
  - `xs:boolean('waarde')`
- Als de waarde niet van het juiste type is zal er een fout optreden

Andere XPath Functies

- Numerieke XPath functies

functie	beschrijving	voorbeeld	uitvoer
number	geeft de numerieke waarde	number('100')	100
abs	geeft de absolute waarde	abs(-100)	100
ceiling	geeft het getal naar boven afgerond	ceiling(3.6)	4
floor	geeft het getal naar beneden afgerond	floor(3.6)	3
round	geeft het getal afgerond naar dichtstbijzijnde gehele getal	round(3.5)	4
round-half-to-even	geeft het getal afgerond naar dichtstbijzijnde even getal	round-half-to-even(2.6)	2

Andere XPath Functies

- XPath string functies

functie	beschrijving	voorbeeld	uitvoer
concat	plakt strings aan elkaar	concat('a','b','c')	'abc'
string-join	als concat, maar met optionele string als separator	string-join(('a','b','c'),' - ')	'a - b - c'
substring	geeft het deel van de string vanaf een beginpositie met een bepaalde lengte (als lengte niet wordt meegegeven: tot het eind)	substring('abcdefg',2,3)	'bcd'
string-length	geeft de lengte van de string	string-length('abcdefg')	7
upper-case	geeft de string in hoofdletters	upper-case('abcdefg')	'ABCDEFG'
lower-case	geeft de string in kleine letters	lower-case('ABCDEFG')	'abcdefg'
normalize-space	verwijdert whitespace (spaties, tabs e.d.) voor en na de string, en vervangt whitespace tussen strings door een spatie.	normalize-space (' hier staan veel spaties ')	'hier staan veel spaties'
substring-before	geeft het deel van de string voor een opgegeven string	substring-before('abcdefg','d')	'abc'

Andere XPath Functies

- XPath string functies

functie	beschrijving	voorbeeld	uitvoer
substring-after	geeft het deel van de string na een opgegeven string	substring-after('abcdefg','d')	'efg'
starts-with	test of een string begint met de opgegeven string	starts-with('abcdefg','abc')	true
ends-with	test of een string eindigt met de opgegeven string	ends-with('abcdefg','abc')	false
contains	test of de opgegeven string voorkomt in een string	contains('abcdefg','abc')	true
string-pad	herhaalt de string een opgegeven aantal keer	string-pad('xyz',2)	'xyzxyz'
matches	controleert of het opgegeven patroon voorkomt in een string	matches(telefoon,'\d{3,5}-,\d*')	true
translate	vervangt in een string de tekens uit de ene string door overeenkomstige tekens uit de andere string	translate('12:30','0123','abcd')	'bc:da'
tokenize	breekt een string op in delen, met het opgegeven patroon als separator	tokenize('dit is een voorbeeld','/s+')	('dit', 'is', 'een', 'voorbeeld)

Andere XPath Functies

- Nieuw in XPath 2.0 zijn datum functies zijn

functie	beschrijving	voorbeeld	uitvoer
current-date	huidige datum (inclusief tijdzone)	current-date()	2011-05-11 +02:00
current-dateTime	huidige datum en tijd	current-dateTime()	2011-05-11 T10:54:32.792 +02:00
current-time	huidige tijd	current-time()	10:54:32.792 +02:00
dateTime	opgegeven datum inclusief tijd	dateTime("xs:date('2011-05-11'), xs:time('11:00:00+02:00')")	2011-05-11 T11:00:00 +02:00

XPath datum functies

- Er zijn nog veel meer datum functies binnen XPath beschikbaar
- Functies om het verschil tussen twee data te berekenen
  - `years-from-duration(datum1- datum2)`
  - `months-from-duration (datum1- datum2)`
  - `days-from-duration(datum1- datum2)`
  - `minutes-from-duration(datum1- datum2)`
  - `seconds-from-duration(datum1- datum2)`

XPath datum functies

- Functies om een gedeelte van de datum of tijd op te halen
  - `year-from-dateTime(datum met tijd)`
  - `month-from-dateTime(datum met tijd)`
  - `day-from-dateTime(datum met tijd)`
  - `minutes-from-dateTime(datum met tijd)`
  - `seconds-from-dateTime(datum met tijd)`
  - `year-from-date(datum)`
  - `month-from-date(datum)`
  - `day-from-date(datum)`
  - `minutes-from-time(tijd)`
  - `seconds-from-time(tijd)`

XPath datum functies

- Constructor functies om een tijdseenheid bij een datum op te tellen
  - `xs:dayTimeDuration ('waarde')`
  - `xs:yearMonthDuration('waarde')`
- Waarde bestaat opgeven in het volgende format
  - `<jaar>Y><maanden>M<dagen>DT<uren>H<minuten>M<seconden>S`
  - `P0Y0M6DT22H14M30S` (6 dagen 22 uur 14 minuten en 30 seconden)
  - `P5D` (5 dagen)
  - `PT6H30M` (6 uur en 30 minuten)
- Voorbeeld
  - **`fn:current-dateTime() + xs:dayTimeDuration('P2DT4H30M')`**

XPath datum functies

- Met opmaak functies kan datum en tijd in een opgeven opmaak worden getoond
  - `format-dateTime('datum met tijd', 'opmaakmasker')`
- Er is ook een `format-date` en `format-time` variant
- Voorbeeld

```
fn:format-dateTime(fn:current-dateTime(),'[D]-[M]-[Y]')
```

- In het masker mogen de volgende aanduidingen tussen [ ] staan
  - Y jaar
    M maand van het jaar
    D dag van de maand
    F dag van de week
    W week van het jaar
    H uur van een dag (24 uurs)
    P am/pm markering
    m minuten in een uur
    s seconden in een minuut
    f fractionele seconden

    d dag van het jaar

    w week van de maand
    h uur van een halve dag (12 uurs)

DEMO

XPath 2.0 operatoren

- In XPath geldt
  - Twee nodesets zijn gelijk als in beide nodesets een node met dezelfde waarde voorkomt
  - Twee nodesets zijn ongelijk als er minimaal één node niet in beide set voorkomt
- In XPath 2.0 zijn extra operatoren toegevoegd die alleen true of false opleveren als één node met één waarde wordt vergeleken
  - Als de sequence uit verschillende nodes bestaat geeft dit een fout

Operator	beschrijving	Voorbeeld
eq	Is gelijk aan	aantal eq 10
ne	Is ongelijk aan	aantal ne 10
lt	Kleiner dan	aantal lt 10
le	Kleiner of gelijk aan	aantal le 10
gt	Groter dan	aantal gt 10
ge	Groter of gelijk aan	aantal ge 10

XPath 2.0 uitbreidingen

- In XPath 2.0 zijn een aantal mooie uitbreidingen
- In de XPath expressie wordt een variabele gemaakt waar later in de expressie weer naar verwezen kan worden
- Zo kan in een XPath expressie met een for loop gewerkt worden
  - `for $variabele in <items> return $variabele/<pad>`
- Eerst wordt een variabele aangemaakt op basis van een expressie.
- Met return wordt aangegeven wat hiervan moet worden uitgevoerd

XPath 2.0 uitbreidingen

- Testen of alle items in een sequence voldoen aan een voorwaarde
  - `every $variable in <items> satisfies <expressie>`
- Testen of sommige items in een sequence voldoen aan een voorwaarde
  - `some $variabele in <items> satisfies <expressie>`

DEMO

Zelf functies maken

- Ondanks het grote aantal functie is niet overal een functie voor
- In XSLT 2.0 is het mogelijk om zelf nieuwe functies te maken
- Hiervoor is het element `<xsl:function>` beschikbaar
- Met `<xsl:param>` kunnen we aan een functie parameters mee geven
- Een functie moet iets terug geven, dit kan met `<xsl:value-of>`
- Eigen gemaakte functies moet in een namespace staan
  - Mag een bestaande of een nieuwe namespace zijn
- Meestal wordt er een aparte namespace voor eigen functies gebruikt

Zelf functies maken

- De functie round(waarde) rond een waarde af op helen
- Geen mogelijkheid voor afronden op decimalen
- Hier kunnen we zelf een functie voor maken
- Eerst de namespace `xmlns:mf="www.myfunctions.nl"` toevoegen
- Functie toevoegen als childelement van `<xsl:stylesheet>`

```
<xsl:function name="mf:rondaf_2dec">
  <xsl:param name="getal"/>
  <xsl:value-of select="round($getal * 100) div 100"/>
</xsl:function>
```

- De functie kan nu gebruikt worden

```
<totaal>
  <xsl:value-of select="mf:rondaf_2dec(sum(current-group()/prijs))"/>
</totaal>
```

DEMO

Een functie stylesheet

- Eigen gemaakte functies zijn alleen binnen het stylesheet beschikbaar
- Daarom wordt vaak een apart functie stylesheet gemaakt
- Dit stylesheet kan dan in andere stylesheets geïmporteerd worden

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:mf="www.mijnfuncties.nl">

  <xsl:function name="mf:rondaf_2dec">
    <xsl:param name="getal"/>
    <xsl:value-of select="format-number(round($getal*100) div 100,'€ #.00')"/>
  </xsl:function>

</xsl:stylesheet>
```

- Dit stylesheet kan dan in andere stylesheets geïmporteerd worden

```
<xsl:import href="///H:/XML2Bestanden/mijnfuncties.xsl"/>
```

- Daarna kan de functie gebruikt worden

DEMO

Overige opdrachten Hoofdstuk 3 (succes)

Bedankt voor uw aandacht!