## XML Schema’s (vervolg)

*(PS: Veel van de nu volgende informatie wordt uitgebreider toegelicht in de 2 PDF’s die je op BlackBoard onder DOCUMENTEN > VALIDATIE kunt vinden!)*

### Groeperen van elementen

In week 1 hebben we al de <xsd:sequence> constructie gezien om aan te geven dat er een groep elementen volgt. Er zijn 3 van dergelijke groeperingconstructies mogelijk:

* **xsd:all** => de volgorde waarin de elementen voorkomen is niet belangrijk, maar ieder element mag hooguit 1x voorkomen in de groep;
* **xsd:choice** => er moet ‘precies 1’ element uit de groep voorkomen;  
  het aantal keren dat (sub)elementen voorkomen is via minOccurs en maxOccurs te regelen;
* **xsd:sequence** => de elementen in de groep moeten exact volgens de opgegeven volgorde voorkomen; ook hier geldt dat het aantal voorkomens van een (sub)element via minOccurs en maxOccurs geregeld kan worden.

**xsd:all**

Stel dat we de volgende definitie van een *naamType* hebben en dat elders in het XSD-bestand is aangegeven dat er een element *naam* is dat van het type naamType is en dat dat meer dan 1x mag voorkomen:

...

<xsd:complexType name="naamType">

<xsd:all>

<xsd:element name="voornaam" type="xsd:string"/>

<xsd:element name="tussenvoegsel" type="xsd:string" minOccurs="0"/>

<xsd:element name="achternaam" type="xsd:string"/>

</xsd:all>

</xsd:complexType>

...

Dan zou de volgende XML-data toegestaan zijn:

...

<naam>

<voornaam>Rob</voornaam>

<achternaam>Willemsen</achternaam>

</naam>

<naam>

<achternaam>Seggelen</achternaam>

<tussenvoegsel>van</tussenvoegsel>

<voornaam>Frans</voornaam>

</naam>

...

Als echter de XSD-regel voor voornaam als volgt gewijzigd zou worden

<xsd:element name="voornaam" type="xsd:string" maxOccurs="5" />

om aan te geven dat iemand minimaal 1 en maximaal 5 voornamen kan hebben – hetgeen niet onmogelijk is – dan zul je merken dat de validator hiertegen bezwaar heeft. Die komt eigenlijk niet eens aan de echte validatie toe, maar klaagt al op voorhand over het feit dat de constructie met maxOccurs=”5” niet goed is.

**xsd:choice**

Het xsd:choice element is het schema-equivalent van “|” in DTDs, en geeft dus eigenlijk de ‘of’-optie weer. Als xsd:element elementen binnen een xsd:choice voorkomen, dan moet precies 1 van de keuze-elementen voorkomen in een bijbehorend XML-document.

...

<xsd:element name="groet">

<xsd:complexType mixed="true">

<xsd:choice>

<xsd:element name="hallo"/>

<xsd:element name="hi"/>

<xsd:element name="allerbeste"/>

</xsd:choice>

</xsd:complexType>

</xsd:element>

...

Dan zou bv. de volgende brief in XML-formaat gemaakt kunnen worden (er is natuurlijk maar een deel van de brief afgedrukt ;-):

<brief>

<groet><hi/> Rob!</groet>

...

</brief>

**PS1**: Let op het feit dat het hi-element zelf leeg is!

**PS2**: Let ook op het gebruik van het mixed-attribuut bij het groet-element om aan te geven dat groet qua inhoud zowel subelementen als tekst mag hebben!

Het xsd:choice element kan minOccurs en maxOccurs attributen hebben waarmee exact aangegeven kan worden hoeveel selecties gekozen kunnen worden. Stel dat je minOccurs=”1” en maxOccurs=”6” doet, dan geeft dat aan dat tussen 1 en 6 elementen uit de lijst van de xsd:choice gebruikt mogen worden. Dit kan verdeeld worden over alle elementen uit de xsd:choice. B.v. 6 verschillende elementen, 3 van de ene en 3 van de andere, of 6 van een en hetzelfde element.

**xsd:sequence**

Die kennen we intussen al wel … zie vorige week!

### Standaard SimpleTypes

Schema’s stellen je in staat om ‘simpele’ types behoorlijk precies te definiëren, ofwel door gebruik te maken van standaard types ofwel door zelf een preciezere definitie te maken.

Een Jaar-veld is meestal geen string maar vermoedelijk een integer, en vaak zelfs een positieve integer met 4 digits. Een prijs is ‘some sort of money’. Een liedje heeft een tijdsduur (duration) als waarde, enz.

Er zijn 44 ingebouwde simple types in de W3C XML Schema Language. Onofficieel kunnen deze in 7 groepen verdeeld worden:

* Numeric types
* Time types
* XML types
* String types
* Boolean type
* URI reference type
* Binary types

**Numeric data types**

De W3C XML Schema Language probeert iedereen tevreden te stellen door vrijwel elk numeric type dat je je kunt voorstellen te leveren, inclusief:

* Integer en floating point numbers
* Finite size numbers zoals in Java en C; infinitely precise, unlimited-size numbers zoals in Eiffel en in Java’s java.math package
* Signed en unsigned numbers

Vaak zul je slechts een subset hiervan gebruiken, b.v. ofwel het willekeurig grote xsd:integer type ofwel het 4 bytes grote xsd:int type.

XML kent de volgende numerieke types: *xsd:float, xsd:double, xsd:decimal, xsd:integer, xsd:int, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte, xsd:positiveInteger*.

Zie overige readers of andere referenties voor meer detail over de waardes die bij deze types horen.

**Time data types**

Time types kunnen tijdstippen per dag of tijdsduren weergeven. De formaten zoals in onderstaande tabel getoond, zijn alle gebaseerd op de ISO standaard 8601. Time zones worden gegeven als offsets van de Coordinated Universal Time (Greenwich Mean Time voor leken) of als de letter Z om Coordinated Universal Time aan te geven.

XML kent de volgende time types: *xsd:dateTime, xsd:date, xsd:time xsd:gDay, xsd:gMonth, xsd:gYear, xsd:gYearMonth, xsd:gMonthDay, xsd:duration*.

Zie overige readers of andere referenties voor meer detail over de waardes die bij deze types horen.

In alle date formats wordt eerst het jaar genoemd, dan de maand, de dag, het uur, enz. De grootste eenheid van tijd staat links, de kleinste rechts. Dit voorkomt de vraag of 2004–02–11 11 februari 2004 is of wellicht 2 november 2004.

Het type xsd:duration legt waardes als volgt vast: P2000Y10M31DT09H32M7.4312S, wat staat voor de duur van 2000 jaar, 10 maanden, 3 dagen, 9 uur, 32 minuten en 7.4312 seconden.

**XML data types**

De volgende groep data types zijn die types die direct met XML constructs zelf te maken hebben, en ook binnen DTD’s voorkomen, zoals NMTOKENS of IDREF. Waar ze binnen DTS’s betrekking hebben op attributen, kunnen ze binnen schema’s op attributen en elementen betrekking hebben. Dit zijn de volgende types: *xsd:ID, xsd:IDREF, xsd:ENTITY, xsd:NOTATION, xsd:IDREFS, xsd:ENTITIES, xsd:NMTOKEN. xsd:NMTOKENS*.

Er zijn ook 4 nieuwe types die met XML constructs te maken hebben: *xsd:language, xsd:Name, xsd:QName, xsd:NCName*.

**String data types**

Het *xsd:string* type is het meest generieke simple type. Het bevat een reeks Unicode karakters van willekeurige lengte. De 2 types: xsd:token en xsd:normalizedString zijn hier nauw mee verbonden. Bij *xsd:normalizedString* zijn alle tabs, CRs en LFs vervangen door spaties, en binnen een *xsd:token* worden dan ook nog eens meerdere spaties achter elkaar teruggebracht tot 1 spatie.

Geen van deze 3 types kent beperkingen voor de waardes die het type kan bevatten. Alle kunnen tabs, linefeeds, opeenvolgende spaties, enz. bevatten. Het verschil is dat de parser voor xsd:normalizedString en xsd:token (delen van) deze white space kan weglaten, terwijl dat niet gebeurt voor een xsd:string.

**Binary types**

Het is niet mogelijk om willekeurige binary files in XML documenten op te nemen, omdat die illegale karakters als form feeds of null characters zouden kunnen bevatten die het XML document malformed zouden maken. Zulke data moet dus eerst omgezet worden in legale karakters. De W3C XML Schema Language kent daarvoor 2 encodings: *xsd:base64Binary* en *xsd:hexBinary*.

Beide hebben hun eigen manier van data omzetten, maar eigenlijk wordt afgeraden om ze te gebruiken. Beter kun je naar die data linken m.b.v. XLink of via unparsed entities.

**Miscellaneous data types**

Er zijn nog 2 types over die niet netjes in een van de vorige categorieën passen: xsd:boolean en xsd:anyURI.

Het *xsd:boolean* type lijkt op C++’s bool data type. Het heeft 4 legale waardes: 0, 1, true, en false. 0 en false worden als hetzelfde beschouwd en idem voor 1 en true.

Het laatste simple type is *xsd:anyURI*. Een element van dit type bevat een relatieve of absolute URI, wellicht een URL, zoals

* urn:isbn:0764547607
* http://www.w3.org/TR/2000/WD-xmlschema-2-20000407/#timeDuration
* /javafaq/reports/JCE1.2.1.htm, /TR/2000/WD-xmlschema-2-20000407/
* ../index.html.

### Afleiden van SimpleTypes

Met deze standaardtypes kun je soms een heel eind komen bij het maken van definities, maar vaak toch nog niet ver genoeg.

Stel dat je b.v. een tijdsduur van 6 minuten en 20 seconden wilt vastleggen, dan wordt dat binnen het standaard type xsd:duration een waarde van P0Y0M0DT0H6M20S. Je mag dit weliswaar ook compacter schrijven door de nullen weg te laten (dus PT6M20S), maar nog steeds is dat niet erg handig. Gelukkig kun je ook je eigen types maken door deze af te leiden van de standaard types, zoals je gewend bent bij iedere OO-taal.

De meest gebruikte **afleiding (derivation)** is door een type te beperken tot een subset van zijn normale waardes. Je kunt b.v. een integer type definiëren dat alleen getallen tussen 1 en 20 mag bevatten door dit af te leiden van xsd:positiveInteger. Je kunt b.v. ook enumeratie-types definiëren die alleen waardes mogen aannemen uit een eindige lijst van vaste waardes. Je kunt ook nieuwe types definiëren die ranges van bestaande types samenvoegen door een union (b.v een type dat ofwel een xsd:date ofwel een xsd:int kan bevatten).

Nieuwe simple types worden gecreëerd m.b.v. **xsd:simpleType** elementen, net zoals nieuwe complex types gecreëerd worden m.b.v. xsd:complexType elementen. Het **name-**attribuut van xsd:simpleType bevat de naam van nieuwe type en daaraan kan gerefereerd worden via het **type**-attribuut van een xsd:element.

De toegestane inhoud van elementen en attributen van het nieuwe type kan gespecificeerd worden via 1 van de volgende 3 child elements:

* **restriction**: selecteert subset van de waardes van het basis type
* **union**: combineert meerdere types
* **enumeration**: specificeert een lijst van elementen van een bestaand simple type

**Afleiden door restrictie**

Om een nieuw type te maken door restrictie van een bestaand type moet je het xsd:simpleType element een xsd:restriction child element geven. Het base-attribute van dit element specificeert waarop je de restrictie toepast.

Onderstaand xsd:simpleType element creëert een nieuw type genaamd phonoYear dat afleidt van xsd:gYear:

<xsd:simpleType name="phonoYear">

<xsd:restriction base="xsd:gYear">

</xsd:restriction>

</xsd:simpleType>

Maar op deze manier is er natuurlijk nog niet veel verschil: ieder legaal xsd:gYear is ook een legaal phonoYear, en andersom. Je kunt phonoYear tot een subset van normale jaar-waardes beperken door een zgn. **facet** op te geven waarmee je specificeert welke waardes wel en welke niet zijn toegestaan. Zo definieert b.v. het **minInclusive-**facet de minimum waarde voor een type. Dit facet wordt toegevoegd aan de restrictie via een xsd:minInclusive child element. Het value attribuut van het xsd:minInclusive element bepaalt de minimaal toegestane waarde:

<xsd:simpleType name="phonoYearType">

<xsd:restriction base="xsd:gYear">

<xsd:minInclusive value="1877"/>

</xsd:restriction>

</xsd:simpleType>

PS: Het value-attribuut van xsd:minInclusive is op 1877 gezet, het jaar waarin Thomas Edison de phonograaf uitvond. Dus 1877 is het 1e legale phonoYear.

Nadat het phonoYearType gedefinieerd is, kun je het gebruiken als ieder built-in type:

<xsd:element type="phonoYearType"/>

Andere facetten die je op xsd:gYear kunt toepassen zijn:

* **xsd:minExclusive**: instanties moeten groter zijn dan deze waarde
* **xsd:maxInclusive**: instanties moeten kleiner/gelijk zijn dan deze waarde
* **xsd:maxExclusive**: instanties moeten kleiner zijn dan deze waarde
* **xsd:enumeration**: lijst van alle legale waardes
* **xsd:whiteSpace**: hoe wordt white space afgehandeld in het element
* **xsd:pattern**: regular expression waarmee instantie wordt vergeleken

Ieder facet wordt weergegeven via een leeg element binnen een xsd:restriction element. Ieder facet heft een **value**-attribute dat de waarde van het facet vastlegt.

Een restrictie kan meerdere facetten bevatten. Het volgende xsd:simpleType element definieert phonoYearType als een jaar tussen 1877 en2100, inclusief beide grenzen:

<xsd:simpleType name="phonoYearType">

<xsd:restriction base="xsd:gYear">

<xsd:minInclusive value="1877"/>

<xsd:maxInclusive value="2100"/>

</xsd:restriction>

</xsd:simpleType>

.

**Facetten**

Facetten worden gedeeld tussen verschillende types. Zo kan minInclusive eigenlijk bij ieder geordend type toegepast worden, dus niet alleen bij gYear maar ook bij gsd:byte, xsd:unsignedByte, xsd:integer, xsd:positiveInteger, enz. De complete lijst van bepeperkende facetten die bij verschillende types gebruikt kunnen worden, is als volgt:

* xsd:minInclusive: alle instanties moeten >= deze waarde zijn
* xsd:minExclusive: alle instanties moeten > deze waarde zijn
* xsd:maxInclusive: alle instanties moeten <= deze waarde zijn
* xsd:maxExclusive: alle instanties moeten < deze waarde zijn
* xsd:enumeration: lijst van toegestane waardes
* xsd:whiteSpace: how wordt white space binnen het element behandeld
* xsd:pattern: reguliere expressie waaraan instantie moet voldoen
* xsd:length: exacte aantal characters in een string, items in een list of bytes in binary data
* xsd:minLength: minimum lengte
* xsd:maxLength: maximum lengte
* xsd:totalDigits: maximum aantal toegestane digits in het element
* xsd:fractionDigits: maximum aantal toegestane digits in fractionele deel van het element

Niet alle facetten zijn zinvol bij alle types. Het is niet zinnig b.v. om te spreken over de minimum waarde van een xsd:NMTOKEN of over het aantal fractionele digits in een xsd:gYear.

De 3 lengte-facetten, **xsd:length**, **xsd:minLength** en **xsd:maxLength**, specificeren het aantal eenheden in een waarde. Voor xsd:string and subtypes zijn de eenheden karakters en regelen de facetten dus de lengte van de string. Voor list types regelen ze het aantal items in de lijst en voor de 2 binary types het aantal bytes in de gecodeerde waarde.

In het volgende voorbeeld wordt een nieuw type Str255 gemaakt dat afleidt van xsd:string:

<xsd:simpleType name="Str255">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1"/>

<xsd:maxLength value="255"/>

</xsd:restriction>

</xsd:simpleType>

Het **xsd:whiteSpace** facet is wat anders als de andere. Het beperkt niet de toegestane inhoud van een element, maar suggereert wat de applicatie zou moeten doen met de white space die voorkomt in het XML-document door aan te geven hoe ‘significant’ de white space is.

Het xsd:whiteSpace facet kent 3 mogelijke waardes:

* **preserve**: white space in het input document wordt niet gewijzigd
* **replace**: iedere tab, CR of LF wordt vervangen door 1 spatie
* **collapse**: iedere tab, CR of LF wordt vervangen door 1 spatie en daarna worden alle series van spaties teruggebracht tot 1 spatie. Ook worden leading en trailing white space opgeruimd.

Dit zijn dus enkel hints voor de lezende applicatie. Geen van deze opties hebben enig effect op de validatie.

Het whiteSpace facet kan alleen worden toegepast op xsd:string, xsd:normalizedString, en xsd:token types, en ook alleen op elementen. XML 1.0 vereist dat parsers alle white space in attributen opruimen van wie het DTD type iets anders is als CDATA, wat een schema daarover ook mag zeggen.

De facetten voor decimale getallen, **xsd:totalDigits** en **xsd:fractionDigits**, helpen bij het formatteren van getallen door aan te geven hoeveel digits er moeten zijn voor het getal in zijn geheel, in het integer deel ervan en in het fractionele deel (achter de komma).

Het xsd:totalDigits facet bepaalt het maximum aantal decimale digits in een getal en werkt voor de meeste numerieke types. De waarde ervan moet een positieve integer zijn.

Het xsd:fractionDigits facet specificeert het maximum aantal decimale digits rechts van de komma. Dit kan alleen op xsd:decimal’s worden toegepast. De waarde moet een niet-negatieve integer zijn.

Het **xsd:enumeration** facet somt eenvoudigweg alle toegestane waardes op. Dit kan voor iedere simple type gedaan worden m.u.v. xsd:boolean. De syntax is wat ongewoon. Iedere waarde krijgt nl. een eigen xsd:enumeration element als child van het xsd:restriction element.

Hieronder wordt als voorbeeld een PublisherType afgeleid van een xsd:string en dit nieuwe type wordt vervolgens toegepast:

<xsd:simpleType name="PublisherType">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="Warner-Elektra-Atlantic"/>

<xsd:enumeration value="Universal Music Group"/>

<xsd:enumeration value="Sony Music Entertainment, Inc."/>

<xsd:enumeration value="Capitol Records, Inc."/>

<xsd:enumeration value="BMG Music"/>

</xsd:restriction>

</xsd:simpleType>

<xsd:complexType name="songType">

<xsd:sequence>

. . .

<xsd:element name="PUBLISHER" type="PublisherType"

minOccurs="0"/>

. . .

</xsd:sequence>

</xsd:complexType>

De opgesomde waardes moeten alle legale instanties zijn van het basistype.

Als je niet uit de voeten kunt met een afleiding van een van de standaard types, dan kan het **xsd:pattern** facet wellicht nog uitkomst brengen. Via het value-attribuut geef je een reguliere expressie op die het patroon weergeeft waaraan alle legale waardes moeten voldoen..

De reguliere expressies in schema’s zijn vergelijkbaar met die in andere talen. Via statements als “[A-Z]+” geef je b.v. aan dat een string 1 of meer van de hoofdletters A tot Z moet bevatten, en met een “\*” bedoel je 0 of meer voorkomens.

### Afleiden van ComplexTypes

**Afleiden door overerving**

Een mooie functie van schema’s is dat je ook gebruik kunt maken van **overerving**, d.w.z. dat een vorm van OO (Object Oriëntatie) beschikbaar is. Je maakt dan gebruik van het **<xsd:extension>** element.

Vorige week hebben we al even het starWarsEntity datatype gezien, dat er als volgt uitzag:

<xsd:complexType name="starWarsEntity">

<xsd:sequence>

<xsd:element name="name" type="xsd:string"/>

<xsd:element name="species" type="xsd:string"/>

<xsd:element name="language" type="xsd:string"/>

<xsd:element name="home" type="xsd:string"/>

</xsd:sequence>

</xsd:complexType>

Van dit type kan op de volgende manier worden geërfd:

<!−− extend base to derive a new sub−type −−>

<xsd:complexType name="wookieType">

<xsd:complexContent>

<xsd:extension base="starWarsEntity">

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

Het **base** attribuut bevat dan de naam van het parent type. Wookie heeft nu alle karakteristieken van starwarsEntity geërfd. Het is natuurlijk mogelijk om extra functionaliteit toe te voegen aan het wookieType, want tot nu toe heeft de overerving natuurlijk nog niet veel zin.

<!−− extend base to derive a new sub−type −−>

<xsd:complexType name="wookieType">

<xsd:complexContent>

<xsd:extension base="starWarsEntity">

<xsd:sequence>

<xsd:element name="battlecry" type="xsd:string"/>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

Hier is dus een extra element “battlecry” toegevoegd aan wookieType.

Stel dat we deze types in een schema met de naam STARWARS.XSD hebben gestopt, hoe kunnen we nu in XML-data gebruik maken van deze types?

Welnu, dat kan b.v. als volgt:

<?xml version="1.0" encoding="UTF−8"?>

<gallery xmlns:xsi="http://www.w3.org/2001/XMLSchema−instance"

xsi:noNamespaceSchemaLocation="starwars.xsd">

<character xsi:type="starWarsEntity">

<name>Luke Skywalker</name>

<species>Human</species>

<language>Basic</language>

<home>Tatooine</home>

</character>

<character xsi:type="wookieType">

<name>Chewbacca</name>

<species>Wookie</species>

<language>Shyriiwook</language>

<home>Kashyyyk</home>

<battlecry>AARGH!</battlecry>

</character>

</gallery>

Het lijkt dus net echte OO (object-oriëntatie) :-) !!

### Tussenopdracht 1

* Maak het bestand “starwars.xsd” aan waarin beide types “starWarsEntity” en “wookieType” voorkomen (met de laatste dus als een extensie op de eerste).
* Maak ook een XML-bestand “gallery.xml” aan waarin je van beide types meerdere instanties opneemt.
* Valideer vervolgens gallery.xml tegen starwars.xsd.

Bovenstaande kun je overigens op meerdere manieren aanpakken c.q. oplossen.

Kijk hierbij ook eens of je op de plek waar je een element opneemt in het XML bestand gebruik kunt maken van het type-attribuut volgens de volgende syntax:

<character xsi:type="…">

***Vraag****: Waarom staat de prefix “xsi:” voor beide type-attributen bij het character-element?*

## Introductie XSLT

In dit practicum zullen we aan het werk gaan met XSLT. XSLT staat voor **eXstensible Stylesheet Language for Transformations**. XSLT is een taal waarin de conversie van het ene (XML)documentformaat naar een ander documentformaat kan worden gespecificeerd. Hierbij is XSLT vooral gericht op het transformeren van tekst naar tekst en dus niet van tekst naar binaire output. Het bronbestand is daarbij een XML bestand. XSLT maakt een volledige scheiding van inhoud (content) en vorm mogelijk, zodat gegevens niet hoeven te worden aangepast wanneer de vorm verandert (en andersom). Het is natuurlijk ook mogelijk om hiervoor een programma te schrijven. XSLT biedt echter een standaard interface voor het omzetten van XML naar het gewenste formaat.

Een mogelijke toepassing van een transformatie m.b.v. XSLT is van XML naar HTML. Als voorbeeld kunnen we de omzetting van het XML adresboek nemen naar HTML formaat.

In deze les zullen we XML transformeren naar HTML maar ook naar XML. Dit laatste klinkt misschien gek, maar het is het meest voorkomende praktijkvoorbeeld in het bedrijfsleven. Waarom zouden we dit eigenlijk willen? Stel je nu eens voor dat twee bedrijven (A en B) beide hun data opslaan in XML en dat ze die data willen uitwisselen. Dat wordt lastig, omdat ze beiden hun eigen XML structuur aanhouden. Gelukkig kunnen we nu XSLT gebruiken om bijv. de XML bestanden van bedrijf A te converteren naar het gehanteerde XML formaat van bedrijf B.

Informatie over XSLT op internet is o.a. te vinden op:

* <http://www.ibiblio.org/xml/books/bible2/chapters/ch17.html>
* <http://www-128.ibm.com/developerworks/views/xml/libraryview.jsp?type_by=Tutorials>
* <http://www.w3.org/tr/xslt>
* <http://www.xml.com>
* <http://msdn.microsoft.com>
* [http://www.w3schools.com](http://www.w3schools.com/)
* <http://xsl.startkabel.nl/>

En natuurlijk in de readers op BB!

## De XSLT-Syntax

### De basis

XML is, zoals je nu weet, als volgt opgebouwd:

<?xml version="1.0"?>

<top-level-element>

<element>Blabla</element>

</top-level-element>

Maar ook XSLT is XML, dus het moet well-formed en zelfs valide XML zijn. De basis van een XSLT-document is als volgt:

<?xml version="1.0"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

</xsl:template>

</xsl:stylesheet>

### XML naar HTML

Als eerste ga je nu een XML document maken met “de wekelijkse aanbieding van je webshop”. Plaats onderstaande XML in het bestand PRODUCTEN.XML.

*NB: Onderstaande tekst is gekopieerd uit Word … (waarom zou die dat nou toch zeggen?)*

<?xml version="1.0"?>

<?xml-stylesheet href="producten.xsl" type="application/xml"?>

<producten>

<boek>

<titel>Patty Brard De naakte waarheid</titel>

<schrijver>Ploeg</schrijver>

<isbn>9058602222</isbn>

<omschrijving>

Patty Brard is een vrouw van uitersten. Ze zit met prinses Caroline van Monaco aan de rand van het zwembad in Zuid-Frankrijk, woont in een van de duurste wijken van Los Angeles, krijgt van haar man een Ferrari Mondial op haar verjaardag.

En leeft ook jarenlang als bijstandsmoeder op een flatje. Patty is recht voor z'n raap, neemt nooit een blad voor de mond. Ze zegt wat ze vindt en ze weet wat ze zegt. Ook in dit boek, waarin ze haar opmerkelijke levensverhaal vertelt.

</omschrijving>

</boek>

<boek>

<titel>Liegangst</titel>

<schrijver>Youp van 't Hek</schrijver>

<isbn>9060054164 </isbn>

<omschrijving>

Liegangst is geen boek dat stukgelezen op het nachtkastje van onze koningin zal liggen. Ook bij Maxima en Willem-Alexander zal het niet pront in de boekenkast prijken.

Bij Edwin de Roy van Zuydewijn en zijn Margarita dan wel? Absoluut niet. Ook veel topmanagers met hun buitensporige inkomens zullen het niet aanschaffen, terwijl ook de meeste politici het liever in de boekhandel laten liggen.

Bij Youp van 't Hek komt iedereen aan de beurt in zijn columns in NRC Handelsblad. Zonder aanziens des persoons schiet hij zijn pijlen recht op zijn slachtoffers. Of je Ahold naar de Filistijnen helpt, de schoonzoon van je zus illegaal laat screenen of als pianist teveel toegiften geeft, je komt aan de beurt. Kort en fel geeft de columnist zijn eigen mening. Goed voor een glimlach. En meestal meer dan dat. Elke week weer.

</omschrijving>

</boek>

<dvd>

<titel>Kopspijkers </titel>

<artiest>Kopspijkers </artiest>

<speelduur>360 Min.</speelduur>

<beschrijving>

2 DVDs met ruim zes uur hoogtepunten plus 60 minuten nieuw materiaal en extra's! Op 11 september 2001 boorden twee vliegtuigen zich in het World Trade Center in New York. De wereld zou nooit meer hetzelfde zijn. Op 12 september 2001 kwamen vier jongens bijeen om een begin te maken met de eerste aflevering van een nieuw humoristisch bedoeld onderdeel van het programma Kopspijkers. Het werden imitaties. Ruim 6 uur fragmenten, ruim 40 minuten speciaal voor deze DVD gemaakte afleveringen van Zomergasten, Andere Tijden, Andermans Veren en Wintertijd, plus een kijkje achter de schermen van een goed half uur.

Hoogtepunten van het Cabaret van Kopspijkers uit de periode najaar 2001 tot en met voorjaar 2003. DISC 1/BAND 1 Serie 1 (najaar 2001) met als extra: 'Zomergasten'. Serie 2 (voorjaar 2002) met als extra: 'Andere tijden'. Extra: de Making of van Kopspijkers. Een uniek kijkje achter de schermen. Owen Schumacher registreert de achterkant van Kopspijkers. DISC 2 Serie 3 (najaar 2002) met als extra: 'Andermans Veren' Serie 4 (voorjaar 2003) met als extra: ‘Wintertijd' .

</beschrijving>

</dvd>

</producten>

Zoals ieder (goed) XML document is ook dit XML document “human readable”. Deze producten wil je, naast dat ze op je eigen website verschijnen, ook doorgeven aan je “partners” die ook een webshop hebben.

De 2e regel van dit XML-document

<?xml-stylesheet href="producten.xsl" type="application/xml"?>

roept het XSL-document aan.

*NB: In de sommige versies van FireFox en Internet Explorer (b)lijkt het noodzakelijk om (soms) verschillende waarde aan het type-attribuut mee te geven (en wie houdt zich daarbij niet aan de standaard … inderdaad: IE :-( ):*

* *voor FF: type="application/xml"*
* *voor IE: type="text/xsl" (hoewel FF hier ook mee overweg lijkt te kunnen)*

Als eerste willen we het XML-document PRODUCTEN.XML omzetten in HTML-code, zodat de aanbieding op ‘onze’ site gepubliceerd kan worden. Je kunt hierbij gebruik maken onderstaande code die verderop toegelicht wordt. Sla onderstaande XSLT-code alvast op in een bestand PRODUCTEN.XSL en plaats dit in dezelfde map als waar PRODUCTEN.XML staat.

<?xml version="1.0"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="producten">

<html>

<head>

<title>De Aanbiedingen van Deze Week</title>

</head>

<body bgcolor="lightblue">

<xsl:apply-templates/>

</body>

</html>

</xsl:template>

<xsl:template match="boek">

<p>

<p>-<b>Titel:<xsl:value-of select="titel"/></b><br>

-Schrijver:<xsl:value-of select="schrijver"/><br>

-ISBN:<xsl:value-of select="isbn"/><br>

-Omschrijving:<xsl:value-of select="omschrijving"/></p>

</p>

<hr></hr>

</xsl:template>

<xsl:template match="dvd">

<p>

<p>-<b>Titel:<xsl:value-of select="titel"/></b><br>

-Artiest:<xsl:value-of select="artiest"/><br>

-Speelduur:<xsl:value-of select="speelduur"/><br>

-Beschrijving:<xsl:value-of select="beschrijving"/></p>

</p>

<hr></hr>

</xsl:template>

</xsl:stylesheet>

Laten we eerst even deze XSL-code bekijken, dan zul je vermoedelijk snel begrijpen wat er gebeurt.

Het document begint, na de vaste XML-definitie, met het xsl:stylesheet element, waarmee aangegeven wordt dat deze XML een stylesheet bevat.

Vervolgens komt het eerste template-element om de hoek kijken:

<xsl:template match="producten">

Alles wat binnen dit element opgesomd staat geeft de bewerkingen weer die toegepast moeten worden op het producten-element van het XML document.

<html>

<head>

<title>De Aanbiedingen van Deze Week</title>

</head>

<body bgcolor="lightblue">

<xsl:apply-templates/>

</body>

</html>

We zien dat er de nodige HTML-elementen staan en in het midden een nieuw xsl-element, nl. <xsl:apply-templates>.

Alles wat er niet aan xsl-spullen staan in zo’n template voorkomt, wordt (grofweg) 1-op-1 overgenomen naar de output, en wat er wel aan xsl-elementen e.d. in staat, die worden natuurlijk verwerkt door de XSLT-parser. In dit geval geeft <xsl:apply-templates> aan dat de templates die voor alle child elements die vanuit het huidige element gevonden kunnen worden, moeten worden uitgevoerd.

Het volgende template in het XSL-bestand is:

<xsl:template match="boek">

Hierin worden de gewenste bewerkingen opgesomd voor alle boek-elementen, enz.

Binnen het boek-template worden klaarblijkelijk de verschillende stukjes boek-informatie in een HTML-paragraaf gezet, waarbij de titel vet afgedrukt wordt.

Je kunt de waarde van een betreffend sub-element van boek ophalen via een xsl:value-of element:

<xsl:value-of select="element-naam"/>

Aan het select-attribuut geef je de naam van het betreffende element mee waarvan je de waarde wilt gebruiken. Als je niet de waarde van een subelement van boek wilt ophalen maar de waarde van een attribuut, dan gaat dat met de volgende constructie:

<xsl:value-of select="@attribuut-naam"/>

<xsl:value-of> is één van de vele instructies die gebruikt kunnen worden bij XSLT.

En zo gaat het verder…

NB: Binnen een XSL-document moet je alle elementen netjes afsluiten. Het is tenslotte XML, dus moet je ervoor zorgen dat het wellformed is!

## Tussenopdracht 2a:

Maak, als je dat nog niet gedaan hebt, de bestanden PRODUCTEN.XML en PRODUCTEN.XSL aan en controleer beide bestanden op wellformed-ness.

Hiervoor moet je in het XML-bestand wellicht eerst de regel die verwijst naar het XSL-bestand even weghalen, want anders probeert de browser meteen het XML-bestand door de XSL-transformatie heen te halen!

## Tussenopdracht 2b:

Plaats evt. de XSL-verwijzing weer terug in PRODUCTEN.XML en probeer nu de resulterende HTML-weergave in de browser te laten zien.

***NB: Bekijk ook eens de source van het getoonde (via View Source o.i.d.). Wat zie je dan?!?***

## Tussenopdracht 2c:

Probeer nu eens of je de HTML-weergave van het bestand wat sprekender kunt maken, zodat b.v. de titel van het product wat meer opvalt.

### Huiswerkopgave voor week 2

*NB: de huiswerkopgave moet aan het begin van de volgende workshop draaien! Zorg ervoor dat deze op de W-drive van beide studenten van het duo draait!*

*Ook deze week geldt dat voor delen van de hieronder vermelde opgaves je nog wat extra informatie moet zien te bemachtigen, b.v. via de readers op Blackboard, om alles voor elkaar te kunnen krijgen.*

**Deelopdracht 1:**

Vorige week heb je een XML schema gemaakt voor ledenbestand c.q. je adresboek.

1. Breid deze definitie uit met minstens 2 afgeleide SimpleTypes die je ook toepast op elementen en/of attributen. Maak hierbij gebruik van 2 verschillende facetten.
2. Als het goed is, heb je een type gemaakt dat “persoonType” o.i.d. heet.  
   Creëer nu een type “alienType” dat dezelfde elementen heeft als “persoonType” maar als extra element kleur heeft. Hiervoor gebruik je natuurlijk een type dat afgeleid wordt van “persoonType”. Bovendien moet de kleur van een derived simple type zijn waarin alleen de kleuren van de regenboog zijn toegestaan.
3. Maak ook een XML-bestand waarin je beide types persoonType en alienType gebruikt, en valideer dit XML-bestand met de XML-validator.

**Deelopdracht 2:**

Vorige week heb je een adresboek c.q. ledenbestand gemaakt. Maak een XSLT bestand dat het adresboek omzet naar HTML en waarin aan de volgende voorwaarden voldaan wordt:

* Iedere contactpersoon komt op een nieuwe rij in een (HTML-)tabel te staan.
* Geef de tabel ook een header-rij.
* Zorg er voor dat je minstens 5 rijen kunt laten zien.
* Zorg ervoor dat je de *email-adressen* als een *hyperlink* opneemt in het HTML-document, zodat je natuurlijk meteen kunt mailen!  
  *Hiervoor moet je misschien wat extra zoekwerk doen, omdat de constructie die je hiervoor nodig hebt, nog niet besproken is.*

**Deelopdracht 3:**

Borduur voort op deelopdracht 2 door de volgende functionaliteit toe te voegen:

* Zorg ervoor dat je op minstens 2 van de kolom-headers (b.v. achternaam en telefoonnummer) kunt klikken en dat dan een HTML-bestand getoond wordt, waarin de rijen in volgorde van de betreffende kolom getoond worden.
* Maak hiervoor gebruik van 2 verschillende XML-bestanden die weliswaar dezelfde data-inhoud hebben maar naar verschillende XSL-bestanden linken. Dus geen Javascript toepassen o.i.d.

*PS: Het mag duidelijk zijn dat dit vanuit onderhoudsoogpunt niet verstandig is, maar dat heeft niet de focus van deze oefening.*

**Deelopdracht 4:**

Tot nu toe hebben we het XSL-bestand aangeroepen vanuit het XML-bestand met:

<?xml-stylesheet href="bestandsnaam.xsl" type="application/xml"?>

Bij het laden van een XML-bestand in de browser zorgde deze ‘automatisch’ voor de transformatie. Dit is echter een handigheidje om te oefenen. De browser heeft nl. een ingebouwde XSLT-transformator.

Normaal gesproken komt deze regel niet voor in het XML-bestand. In plaats daarvan maak je gebruik van een losse XSLT-engine, ofwel handmatig ofwel via code.

De AltovaXML-tool die we eerder al hebben gebruikt voor validatie, beschikt over zo’n XSLT-engine. Die gaan we nu toepassen.

Jij hebt een eigen adresboek-formaat (zie vorige opdrachten), maar jouw vriend(in) houdt er zo zijn/haar eigen smaak van een XML-adresboek op na. Dat ziet er als volgt uit, en is wellicht wat grover van opzet dan de manier waarop jij het hebt vormgegeven:

<?xml version="1.0"?>

<xvriendenboek>

<xvriend>

<xnaam>Joske Vermeulen</xnaam>

<xbijnaam/>

<xgebdat>19-5-1986</xgebdat>

<xtel>016/589652</xtel>

<xemail>joske@vermeulen.com</xemail>

</xvriend>

</xvriendenboek>

Jullie willen gegevens uitwisselen. Met name wil jouw vriend(in) jouw gegevens inlezen, en jij helpt hem/haar daarbij een handje door jouw XML-adresboek om te vormen tot een formaat dat overeenkomt met zijn/haar ‘systeem’. Je moet dus een XSLT bestand maken dat jouw XML-adresboek omzet naar een XML-bestand dat overeenkomt met het formaat van je vriend(in).

Gebruik de XSLT-engine van AltovaXML voor het uitvoeren van de feitelijke transformatie!

*NB: Je zult dus zelf moeten proberen te achterhalen wat je in het XML-bestand precies nodig hebt om de gewenste structuur te laten opbouwen!*

**Algemene criteria:**

* Er wordt gelet op de netheid van de XML/XSD/XSL-code!