CURSISTENMAP

XML introductie

Revisie: 1.3

Versie 1.2, mei 2012

Inhoudsopgave

Inhoudsopgave

1 Inleiding

In de maatschappij van dit moment draait alles om informatie. Waar vind ik zo snel mogelijk de informatie die ik nodig heb? Hoe haal ik uit een stroom aan informatie de gegevens die voor mij belangrijk zijn en hoe kan ik deze gegevens opslaan, benaderen en doorgeven? Eén van de manieren om de stroom aan gegevens door te geven is XML. In de cursus XML Introductie gaan we kijken naar verschillende mogelijkheden van XML. We zullen bespreken hoe een XML document is opgebouwd. Wanneer is een XML document bruikbaar om gegevens te verwerken. Hoe kunnen we een XML document netjes op het scherm tonen. Hoe kan een XML document vertaald worden naar een bruikbare vorm. In het eerste hoofdstuk zullen we ons vooral richten op de plaats en functie van XML binnen het spectrum van de IT.

1.1 Wat is XML?

XML is een standaard manier om gegevens te beschrijven en gestructureerd vast te leggen. Eigenlijk is XML niets meer dan een serie regels die de structuur van een document bepalen, vergelijkbaar met de grammatica van een taal. Als we ons aan deze grammatica houden kunnen we zelf de inhoud verder invullen. We kunnen als het ware zelf de woorden bij deze grammatica bedenken. Deze woorden noemen we ook wel *tags*. De tags vormen als het ware het vocabulaire van XML.

Gegevens die in XML zijn opgeslagen kunnen eenvoudig door applicaties worden gelezen en verwerkt. Daarnaast zijn XML-gegevens ook eenvoudig door mensen te lezen en eventueel bij te werken. XML tags zijn namelijk 'zelfbeschrijvend'. Bijvoorbeeld:

```
<cursus>
    <naam>XML</naam>
    <dagen>2</dagen>
</cursus>
```

Dit voorbeeld beschrijft een cursus, met de naam XML, die 2 dagen duurt.

Een ander belangrijk voordeel van XML is dat de inhoud uit alleen tekst bestaat. En tekstgegevens kunnen eenvoudig via HTTP verstuurd worden over het internet. Het is dan ook niet verwonderlijk dat XML juist op het internet zo'n centrale plaats inneemt. Zo zijn bijvoorbeeld RSS-feeds in XML gecodeerd.

Wat het gebruik van XML verder aantrekkelijk maakt, is dat we zelf kunnen bepalen wat voor onze toepassing een 'geldig' document is. XML documenten kunnen worden gecontroleerd op geldigheid met een DTD (Doc Type Definition) of met een XML-schema. Zo kunnen we bijvoorbeeld vastleggen dat het aantal dagen van een cursus een geheel getal moet zijn, en dat dit niet negatief mag worden.

De afkorting XML staat *voor eXtensible Markup Language*. Met een markup taal worden elementen van een document gemarkeerd. Dit markeren gebeurt door middel van de tags. Het woord *extensible* kan vertaald worden met uitbreidbaar. Een XML document is namelijk uit te breiden met nieuwe tags en/of met een nieuwe inhoud. De applicaties die het XML document gebruiken hoeven dan niet te worden aangepast. Hierdoor ontstaat een hoge mate aan flexibiliteit.

1 Inleiding

In een XML document is een tag een stukje code tussen een '<' (kleiner dan teken) en een '>' (groter dan teken). Om een element te markeren gebruiken we een begintag *<code>* en een eindtag *</code>*.
Bijvoorbeeld:

```
<product>laptop</product>
```

of

```
<naam>Gerard</naam>
```

De begintag en eindtag hebben de zelfde naam, alleen staat er voor de code van de eindtag een slash '/'.
Door deze markeringen met tags wordt de structuur van data in een document vastgelegd. Aan de hand van deze structuur kunnen gegevens op de juiste manier in het document worden ingevoegd maar ook weer worden weergegeven, geïmporteerd en benaderd.

Met XML valt ook vaak de term *SGML*. SGML staat voor *Standard Generalized Markup* Language. Met SGML kunnen we net als met XML de structuur van een document vastleggen. XML is afgeleid van het veel complexere SGML en is breder toepasbaar, onder meer voor gebruik op het internet.

XML is niet ontworpen voor een speciaal platform als Windows of Linux. XML is platformonafhankelijk. Samen met de hoge mate van flexibiliteit zorgt dit er voor dat XML wereldwijd geaccepteerd is. Zo is XML niet alleen geschikt om gegevens in op te slaan, maar wordt XML ook veel gebruikt om gegevens via het internet of tussen programma's te versturen, zodat applicaties met elkaar kunnen communiceren.

1.2 XML en HTML

XML wordt vaak in één adem genoemd met HTML. Ook HTML is een markup taal. HTML staat voor Hyper Text Markup Language. Net als XML is HTML afgeleid van SGML. Met HTML kunnen documenten worden opgemaakt om bekeken te worden met een webbrowser. Hiervoor worden tags gebruikt. Met een HTML tag kan een bepaalde opmaak aan een document worden toegekend. Zo wordt `<i>dit</i>` in een webbrowser cursief weergegeven, en `<b>dit</b>` vet gedrukt. De tags voor HTML zijn vastgesteld door het World Wide Web Consortium (*W3C*).

Hoewel HTML en XML beide gebruik maken van tags, zijn de verschillen groot. XML is niet bedoeld voor opmaak, maar om de structuur van gegevens te beschrijven. Hoe deze gegevens vervolgens worden gebruikt, hangt af van de toepassing. Eén van de mogelijke toepassingen is het beschikbaar maken voor het internet. De gegevens kunnen dan worden getransformeerd naar HTML, of via CSS van opmaak voorzien. Er zijn nog tal van andere toepassingen van XML.

Een ander belangrijk verschil is het definiëren van tags. De HTML tags zijn vastgelegd door het W3C. Dit maakt HTML niet echt flexibel. Als we een tag nodig hebben die niet bestaat hebben we gewoon pech. We kunnen geen nieuwe tags definiëren. Daar is HTML niet voor bedoeld. In XML kunnen we wel zelf de tags definiëren die we nodig hebben. Hierbij moeten we wel rekening houden met bepaalde principes en regels maar er is een hoge mate aan vrijheid en flexibiliteit. Hierdoor kunnen gegevens gestructureerd in een document opgeslagen worden en is het resultaat zowel door de mens als door een computerprogramma te lezen.

We bekijken bijvoorbeeld het volgende stukje code:

```
<song>
    <titel>Forever Young</titel>
    <artiest>Alphaville</artiest>
    <lengte>03:47</lengte>
    <album>Forever Young</album>
    <jaar>1982</jaar>
</song>
```

Iemand die niets van XML weet, kan hieruit al vrij snel concluderen dat hier het nummer *Forever Young* wordt beschreven. Dat het nummer gezongen wordt door de groep *Alphaville*, 3 minuten en 47 seconden duurt, van het album *Forever Young* komt en dat het in 1982 is uitgebracht. Dit komt doordat de tags duidelijk benoemd zijn en al aangeven om welke gegevens het gaat.

1.2.1 World Wide Web Consortium

In het voorgaande is het W3C al even genoemd. Het W3C is een internationale organisatie die de webstandaarden voor het World Wide Web ontwerpt, zoals HTML, XML en CSS. Het W3C heeft hierin een adviserende rol maar heeft geen bevoegdheid om standaarden vast te stellen, zoals de ISO. De HTML-specificaties van het W3C zijn daarom formeel niet meer dan aanbevelingen. In de praktijk worden de W3C-aanbevelingen echter als standaarden gezien, mede omdat belangrijke ICT bedrijven als IBM en Microsoft in het W3C samenwerken.

1.3 XML-Editor

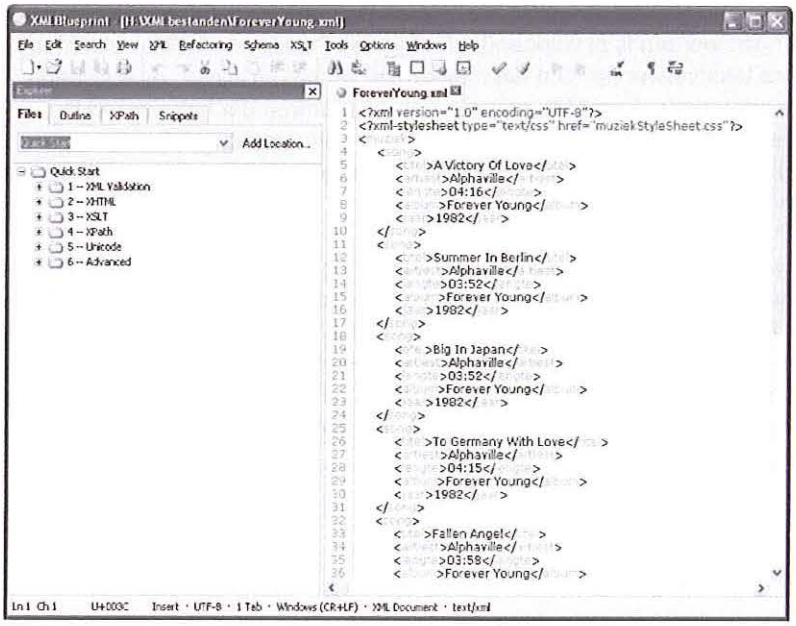
Voor het schrijven van XML is weinig nodig. Een programma waarmee tekstbestanden gemaakt kunnen worden is al voldoende. Dit kan het Windows kladblok zijn maar ook Word of een andere tekstverwerker. Om het makkelijker te maken zijn er verschillende XML-editors ontwikkeld. Een XML editor is een programma dat helpt om XML documenten te ontwikkelen. Een editor kan een XML document controleren, geeft de structuur overzichtelijk weer en heeft verschillende andere hulpmiddelen voor het ontwikkelen van een XML document.

Er zijn veel verschillende XML-editors. Elke editor heeft zijn eigen mogelijkheden en onmogelijkheden. De ene werkt als een eenvoudige teksteditor waarin alle code handmatig kan worden ingetypt. De andere laat bijna niets van XML zien en werkt veel meer grafisch. Elke editor heeft zo zijn eigen sterke en zwakke punten. Het is van belang om een editor te kiezen die voor de gebruiker de juiste mogelijkheden heeft. Tijdens de cursus XML Introductie zal XMLBlueprint als editor gebruikt worden.

1.3.1 XMLBlueprint

XMLBlueprint is een editor waarin we XML documenten kunnen maken en bekijken. Het bekijken van een XML document kan met XMLBlueprint op verschillende manieren.

Als er een bestaand document geopend is zal het venster van XMLBlueprint er als volgt uitzien:



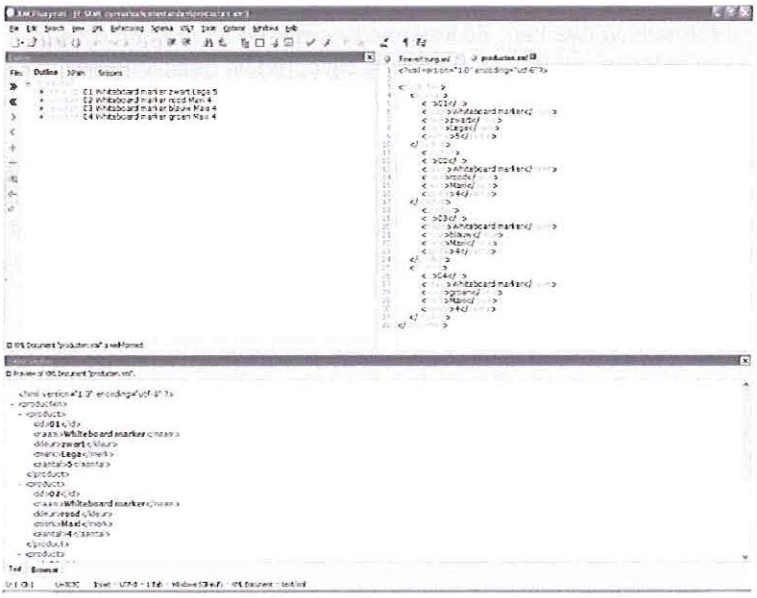
Links zien we de Explorer. De Explorer bevat een aantal tabbladen: *Files, Outline, Xpath* en *Snippets*. Het middengebied is het werkgebied.

Om een bestaand XML-document te openen kan de knop *Open File* of het menu *File, Open* gebruikt worden. In het venster dat verschijnt kunnen alle soorten XML documenten worden gevonden.

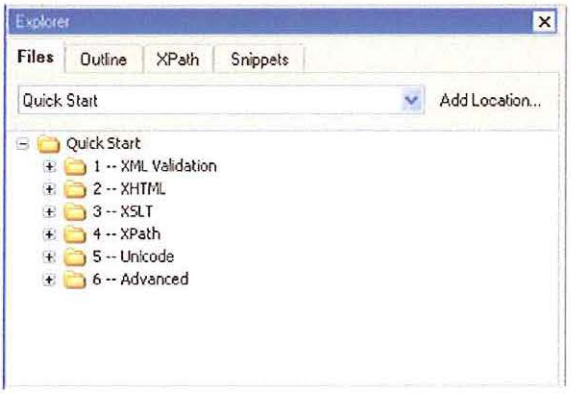
1 Inleiding

Het bewerken van een document kan eenvoudig door de tekst aan te passen in het werkgebied. Als er meerder documenten geopend zijn, kan door middel van de tabbladen in het werkgebied van document worden gewisseld.

In het *Output Window* op het tabblad *Browser*. Kunt u zien hoe het document er in de webbrowser uitziet.



In het *Explorer* gedeelte van het scherm vindt u nog een aantal tabbladen:



Het tabblad *Files* kunt u gebruiken om bestaande documenten te openen. Het is handig om hier de locatie met uw XML bestanden op te nemen zodat u daar altijd snel bij kunt.

Bestanden die bij het afsluiten van XMLBlueprint nog geopend waren worden direct weer geladen als u XMLBlueprint opnieuw start.

1.4 XML Parser

Een XML parser is een programma dat het XML document leest en verwerkt zodat andere programma's ermee om kunnen gaan. De XML parser controleert eerst of het bestand een geldig XML document is. Daarna kan de parser de structuur van de elementen verwerken en naar een ander programma, zoals bijvoorbeeld een webbrowser, sturen. Hierna kan het programma de informatie verwerken. Zo kan een browser de elementen op een overzichtelijke manier tonen, terwijl een database bijvoorbeeld dezelfde structuur als gegevens kan importeren.

Er zijn verschillende XML parsers. De ene parser leest de hele boomstructuur van elementen, de andere doorloopt een XML document als een reeks begin- en eindtags, met de inhoud daartussen. Deze verschillende manier van benaderen zorgt ervoor dat er per situatie gekeken moet worden welke parser het meest geschikt is. In het laatste hoofdstuk worden XML parsers nader toegelicht.

2 XML documenten

2.1 Inleiding

In dit hoofdstuk behandelen we de structuur van verschillende soorten XML documenten. We bekijken uit welke onderdelen een XML document is opgebouwd, en aan welke eisen een geldig XML document moet voldoen. Vervolgens zullen we zelf een XML document maken en bewerken.

2.2 Elementen en Tags

Een XML document bestaat uit elementen. Met elementen bedoelen we alles waarmee we de pagina structureren. Elementen kunnen we herkennen aan de tags. Een tag is een code tussen een groter-dan teken en een kleiner-dan teken: `<code>`. Een element bestaat meestal uit een begintag; `<code>` en een eindtag: `</code>`, met daartussen de inhoud.
De code van de eindtag wordt altijd vooraf gegaan door een slash (/) en is verder gelijk aan de code van de begintag. Tussen de begintag en de eindtag vinden we de inhoud of waarde van het element. De inhoud van een element kan tekst zijn maar kan ook bestaan uit nieuwe elementen. We spreken dan wel van child elementen of *subelementen.*
Een element *adres* kan er bijvoorbeeld als volgt uitzien:

```
<adres>Bovenstraat 5 1234 AB Overdam</adres>
```

De inhoud van het element *adres* is nu een tekst met het hele adres als waarde.

We kunnen het element *adres* ook met behulp van child elements weergeven.

```
<adres>
    <straatnaam>Boventraat</straatnaam>
    <huisnummer>5</huisnummer>
    <postcode>1234 AB</postcode>
    <plaats>Overdam</plaats>
</adres>
```

Afhankelijk van wat er met de gegevens gedaan moet worden is de eerste of de tweede manier handiger.

2.2.1 Lege elementen

Een element hoeft niet altijd gevuld te zijn. Als een element geen inhoud heeft kunnen we het gewoon weglaten. We kunnen er ook voor kiezen om het element toch op te nemen maar geen inhoud te geven. Voor de verwerking zal dit over het algemeen geen gevolgen hebben.

Een leeg element kunnen we als `<adres></adres>` schrijven. Er is ook nog een andere manier om een leeg element weer te geven. Door aan het eind van de begintag een slash op te nemen geven we aan dat een element geen inhoud krijgt:

```
<adres/>
```

Een persoon waarvan het adres niet bekend is kunnen we dus op de volgende manieren weergeven:

```
<persoon>
    <naam>Merijn de Vries</naam>
</persoon>
```

of

```
<persoon>
    <naam>Merijn de Vries</naam>
    <adres></adres>
</persoon>
```

of

```
<persoon>
    <naam>Merijn de Vries</naam>
    <adres/>
</persoon>
```

2.2.2 Begin- en eindtags

We hebben al gezien dat een element wordt gemarkeerd door een begin- en een eindtag. De naam van de eindtag is altijd gelijk aan de naam van de begintag. Bij XML is het markeren van een element met een tag aan een aantal regels gebonden.

- Een element wordt altijd gedefinieerd door een begintag. De eindtag sluit het element af. Begintag: `<adres>` Eindtag: `</adres>`
- De eindtag moet altijd worden voorzien van een slash na het `<` teken.
- Een eindtag kan in sommige gevallen worden weggelaten. Als het element leeg is kan worden volstaan met alleen een begintag met een slash aan het eind: `<plaats/>`
- Een begintag moet altijd gelijk zijn aan de eindtag. Dit is ook hoofdlettergevoelig dus de tag `<muziek>` kan niet worden gesloten met `</MUZIEK>`. De eindtag zal in dit geval `</muziek>` moeten zijn.
- De naam van een tag moet altijd beginnen met een letter of een underscore (_) en mag alleen bestaan uit een combinatie van letters, cijfers, streepjes, underscores en punten.

Er mogen geen spaties worden opgenomen in de naam van een tag. Om binnen de naam van een tag toch uit meerdere woorden te laten bestaan worden de woorden wel gescheiden door punten of underscores.

Bijvoorbeeld:

```
<dit_is_een_tag> inhoud </dit_is_een_tag>
```

of

```
<nog.een.tag> nog meer inhoud </nog.een.tag>
```

Ook wordt er vaak gebruikt gemaakt van hoofdletters om de verschillende woorden te markeren.

2.2.3 Betekenis van een XML tag

Tags kunnen verschillende betekenissen hebben. HTML tags hebben betekenis voor de opmaak. De tag bepaalt dan hoe een element wordt weergegeven. Bijvoorbeeld:

```
<b>Dit wordt vet gedrukt (bold) weergegeven in de browser</b>
<i>en dit cursief (italic)</i>
```

XML tags beschrijven gegevens. De tag geeft dus alleen aan om wat voor soort gegevens het gaat, niet hoe dat moet worden weergegeven in een browser. Bijvoorbeeld:

```
<naam>
    <voornaam>Anita</voornaam>
    <achternaam>de Wit</achternaam>
</naam>
```

Tags hebben dus een *semantische* betekenis. Ze zorgen als het ware voor een koppeling met de echte wereld. Aan de tags kunnen mensen afleiden om wat voor gegevens het gaat. Nemen we bijvoorbeeld de onderstaande tags, dan is niet gelijk duidelijk waar het hier om gaat:

```
<b> <n> </n> </b>
```

Deze code had er ook als volgt uit kunnen zien. De tags zijn nu voor mensen veel beter te begrijpen.

```
<boodschap> <naam> </naam> </boodschap>
```

Het is nu duidelijk dat deze code de structuur van een persoonlijke boodschap vormt. We kunnen er zelf vrij eenvoudig een inhoud voor de elementen bij bedenken.

```
<boodschap>goedemorgen <naam>Michiel</naam> succes met de XML-cursus </boodschap>
```

Tags kunnen ook betekenis hebben voor de applicatie. De applicatie bepaalt afhankelijk van de tag wat er met een element gedaan moet worden. De semantische betekenis is voor een computersysteem niet van belang. Het is om het even of de tag *b* of *boodschap* heet, als de structuur en inhoud maar gelezen en verwerkt kunnen worden. Voor mensen die met het XML document moeten werken is deze betekenis wel van belang. Ze kunnen beter begrijpen om wat voor informatie het gaat. Voor een XML ontwikkelaar is het zaak om de tags zo te kiezen dat de gebruikers er betekenis aan kunnen geven. Zo kunnen gebruikers het document makkelijker ontwikkelen en onderhouden.

Een XML ontwikkelaar kan zelf de set aan tags ontwerpen zodat er voor elk XML document en elke XML applicatie een passende set tags gebruikt wordt.

2.2.4 Het root element

Elk geldig XML document bevat in elk geval één element. Dit is een soort basis element. Alle andere elementen zijn child elements van dit basiselement. Het basis element wordt ook wel het *root element* genoemd. In het bovenstaande voorbeeld is *muziek* het root element. Het document begint met de begintag `<muziek>` en wordt ook afgesloten met de eindtag `</muziek>` . Binnen het element *muziek* vinden we een aantal *song* elementen terug die zelf elk ook weer uit verschillende elementen bestaan. Dit wordt wel het nesten van elementen genoemd. Zo krijgen we een boomstructuur van elementen waarvan *muziek* het basiselement is.

2.3 Well-formed

We zijn al een aantal regels tegengekomen waaraan een XML document moet voldoen. Voldoet het bestand niet aan die regels dan kan het bestand bijvoorbeeld niet verwerkt worden door een applicatie. Het is dus belangrijk om een XML document op de juiste manier op te bouwen. Een XML document moet *well-formed* zijn. Een goede editor controleert of een document well-formed is. Een well-formed XML document voldoet aan de volgende voorwaarden.

- Een XML document bevat tenminste 1 element (het root element)
- Het root element bevat alle andere elementen
- De begin- en eindtags moeten gelijk zijn (hoofdlettergevoelig)
- De elementen moeten op de juiste manier genest worden

Bij het nesten van elementen moet een element altijd volledig binnen een bovenliggend element liggen. De onderstaande structuur is dan ook geen geldige structuur:

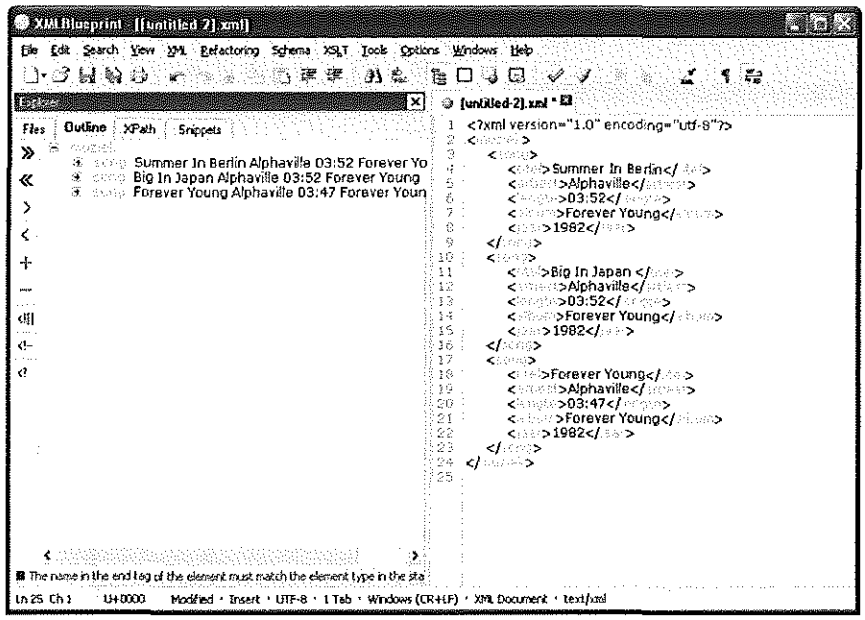
```
<muziek>
    <album> album1
       <artiest>artiestnaam
           <song>titel1</song>
       </album>
    </artiest>
</muziek>
```

In HTML kan deze vorm van nesten wel gebruikt worden. Daarom is het vaak niet mogelijk om een HTML pagina in één keer over te zetten naar XML.

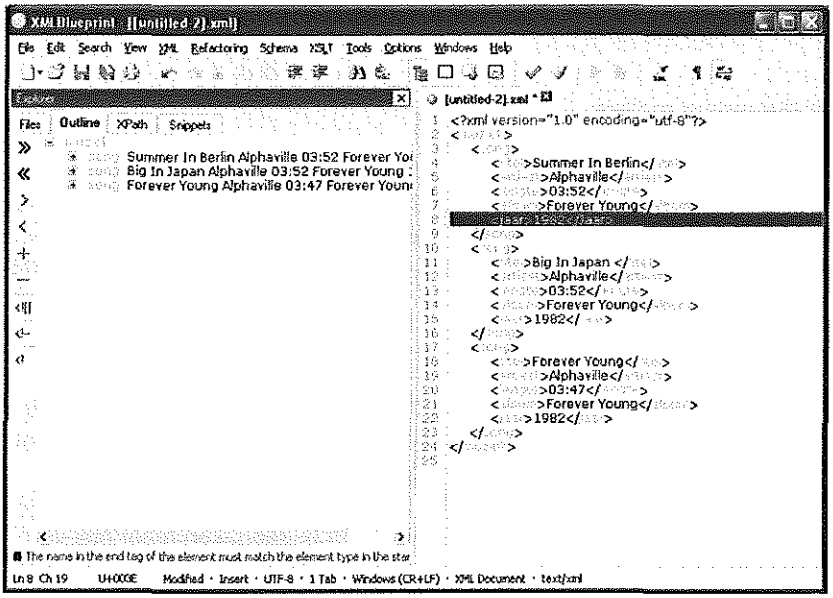
Het element *artiest* wordt geopend binnen het element *album*, maar wordt pas buiten het *album* element gesloten. In een geldig XML document moet deze code als volgt genest worden.

```
<muziek>
    <album> album1
       <artiest>artiestnaam
           <song>titel1</song>
       </artiest>
    </album>
</muziek>
```

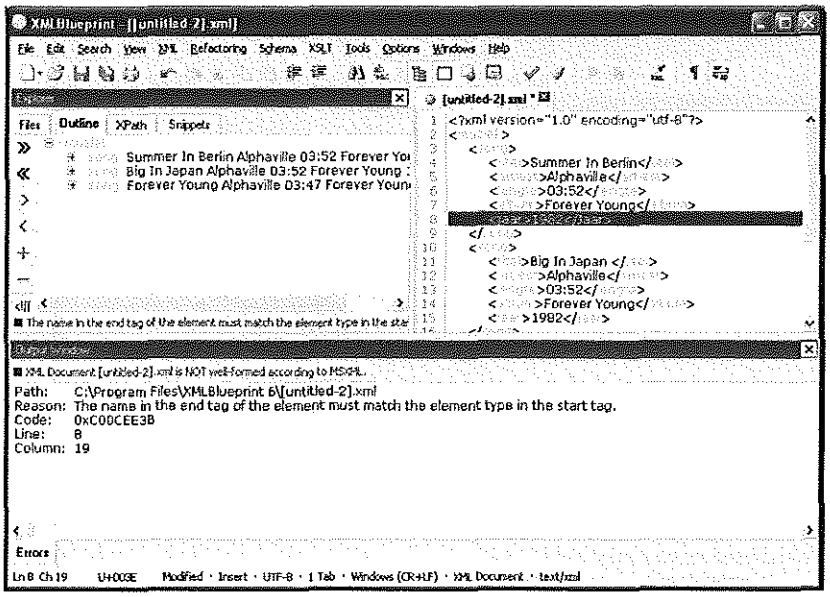
XMLBlueprint geeft door middel van een rode tekst onderin het venster aan, wanneer een document niet well-formed is. In het voorbeeld hieronder zijn de begin- en eindtag van een element niet gelijk. XMLBlueprint geeft dit aan met een duidelijke melding:



Aan de melding is te zien dat dit bestand niet well-formed is: de begin- en de eind tag zijn niet gelijk. Als op deze foutmelding wordt geklikt, markeert XMLBlueprint de foutieve regel:



Een document kan ook met de [F7] (*Check Well-formedness*) gecontroleerd worden. Er verschijnt dan een melding in het *Output Window*. Fouten worden inclusief regelnummer weergegeven. Bovendien wordt de foute regel dan gemarkeerd.



De melding geeft aan dat de begin en eindtag niet overeenkomen. De hoofdletter *J* van de eindtag moet een kleine *j* zijn.
Als we dit aanpassen zal XML Blueprint de melding geven dat het document well-formed is. In het Output Window blijft de foutmelding wel staan. Als we deze met [F7] verversen zal ook hier de foutmelding verdwijnen.

2.4 Een XML document maken

We hebben gezien dat een XML document uit verschillende elementen bestaat. Deze elementen vormen een boomstructuur. Aan de basis staat het root element met daarbinnen een verzameling van andere elementen. Bovenaan het document geven we door middel van een XML declaratie aan dat we met een XML document te maken hebben. De XML declaratie ziet er als volgt uit:

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

Zodra een applicatie deze tekst boven een document vindt, weet de applicatie dat het om een XML document gaat. De applicatie weet dan hoe het document behandeld moet worden. De XML declaratie is zelf ook een tag maar is geen element.
De tag begint met een `<?` en eindigt met een `?>`. Dergelijke tags worden *processing instructions (PI)* genoemd. Het is aan te raden een PI te gebruiken maar niet verplicht.

Om een geldig XML declaratie te maken is het verplicht om in de tag een version op te nemen. Deze moet eigenlijk altijd 1.0 zijn: een latere versie, versie 1.1, wordt door de meeste applicaties nog niet ondersteund.
In het tweede stuk van de XML declaratie wordt de karakterset gedefinieerd. Vooral als er afwijkende karaktersets worden gebruikt is het noodzakelijk om deze hier te benoemen. Wordt er geen karakterset opgegeven dan wordt standaard UTF-8 genomen.
Het is niet noodzakelijk om een XML declaratie toe te voegen, maar het is wel aan te raden om elk document met een XML declaratie te beginnen.
Bij eenvoudige editors moet de XML declaratie worden ingetypt. Bij editors met meer mogelijkheden kan de XML declaratie ook op andere manieren worden ingevoegd.

Zo zet XMLBlueprint de XML declaratie bij het maken van een nieuw document al in het document.

2.5 Attributen

In het voorgaande hebben we gezien dat we een hele boomstructuur aan elementen kunnen opzetten. Een element kan meerdere child elementen bevatten. Het is ook mogelijk om een eigenschap aan een element toe te kennen via een *attribuut*.

Een attribuut wordt binnen de begintag opgenomen. Om een attribuut toe te voegen wordt eerst de naam van het attribuut opgenomen met een is-gelijk teken (=) teken en daarachter volgt de waarde van het attribuut tussen dubbele quotes " ".
De tag gaat er dan als volgt uitzien:

```
<tag attribuut="waarde">
```

Bekijk de code voor de volgende song

```
<song>
    <titel>Sounds Like A Melody</titel>
    <lengte>04:45</lengte>
    <jaar>1982</jaar>
</song>
```

Binnen het song element zijn drie child elementen gedefinieerd. Deze child elementen zouden we ook als attributen kunnen opnemen, zoals in de code hieronder:

```
<song jaartal="1982" lengte ="04:45">
    Sounds Like A Melody
</song>
```

We zien dat *jaartal* en *lengte* als attributen aan het *song* element zijn toegevoegd.
Er is nog een verschil met het vorige voorbeeld. Er is geen child element *titel* meer. De titel is nu de inhoud van het song element zelf geworden. Dit is een keuze; het titel element had ook gewoon kunnen blijven staan.

Inhoudelijk is er weinig verschil tussen een attribuut en een child element: beide zeggen iets over het bovenliggende element. Aangeraden wordt om de relevante gegevens in child elementen onder te brengen. Attributen worden gebruikt voor metadata, die voor de applicatie van belang kunnen zijn, maar niet iets wezenlijks zeggen over het betreffende element. Neem bijvoorbeeld een artikel in de supermarkt: relevante gegevens die het artikel beschrijven zijn bijvoorbeeld het merk, de productsoort, het gewicht en de prijs. Deze informatie kan het beste in child elementen worden ondergebracht.

Voor het afrekenen bij de kassa is ook de streepjescode van het artikel van belang, maar het voegt geen informatie toe aan het artikel zelf. De streepjescode zou daarom beter als attribuut opgenomen kunnen worden.

Attributen zijn eigenlijk een speciaal soort elementen. Een belangrijk verschil is dat een attribuut alleen een tekstwaarde als inhoud heeft, terwijl een element ook child elementen als inhoud kan hebben.

De onderstaande structuur is een geldige structuur

```
<persoon>
    <naam>
       <voornaam>Marcel</voornaam>
       <achternaam>Veldhuis</achternaam>
    </naam>
</persoon>
```

Met attributen lukt dit niet. Het volgende is namelijk niet toegestaan:

```
<persoon naam= "<voornaam>Marcel</voornaam>
       <achternaam>Veldhuis</achternaam>"
</persoon>
```

Willen we dit met attributen oplossen dan zouden we dit in twee afzonderlijke attributen op moeten vangen:

```
<persoon voornaam= "Marcel" achternaam="Veldhuis">
</persoon>
```

In een element kunnen we ook niet twee keer hetzelfde attribuut gebruiken. Het onderstaande voorbeeld toont een persoon met meerdere voornamen.

```
<persoon>
    <naam>
       <voornaam>Marcel</voornaam>
       <voornaam>Erik</voornaam>
       <achternaam>Veldhuis</achternaam>
    </naam>
</persoon>
```

We kunnen niet een dubbele voornaam als twee attributen toevoegen. Het onderstaande stukje code is geen geldige XML code:

```
<persoon voornaam="Marcel" voornaam="Erik" achternaam="Veldhuis">
</persoon>
```

Een attribuut kan maar één keer in een element voorkomen. Dit zou opgelost kunnen worden door beide namen samen te voegen, maar of dat de gewenste oplossing is moet de ontwerper zelf gaan bepalen:

```
<persoon voornaam="Marcel Erik" achternaam="Veldhuis">
</persoon>
```

Voor de verwerking maakt het niet uit of de manier met child elementen of de manier met attributen wordt gebruikt. Wel zijn er enkele principes die helpen bij het bepalen of informatie beter als element of als attribuut opgenomen kan worden:

Principe van kern en inhoud	Gegevens die mede bepalen wat het element precies is, worden in childelementen onder gebracht. Extra beschrijvende (meta-)gegevens komen in attributen.
Principe van gestructureerde informatie	Gestructureerde gegevens staan in childelementen: deze kunnen verder worden opgesplitst.
Principe van leesbaarheid	Als de gegevens zijn bedoeld om te worden gelezen door een persoon, gebruik dan elementen. Als het voor een applicatie is bedoeld (id's, URL's en dergelijke), gebruik dan attributen.
Principe van element/attribuut binding	Neem alleen attributen op die iets zeggen over het element, niet over een ander attribuut van hetzelfde element. Dit is wel toegestaan, maar kan beter niet worden gedaan.

Een voorbeeld bij het laatste principe:

```
<product>
    <prijs eenheid="euro">1.20</prijs>
    <naam>brood</naam>
</product>
```

2.6 Gereserveerde karakters

Om een XML document te maken hebben een aantal karakters een speciale betekenis. Deze karakters kunnen we niet zomaar gebruiken voor de inhoud van een element of attribuut. Zo kunnen we bijvoorbeeld niet een < teken gebruiken. Volgens de regels van XML geeft dit het begin van een nieuwe tag aan. Willen we toch deze karakters gebruiken dan kunnen we gebruik maken van *Built-in Entities*.

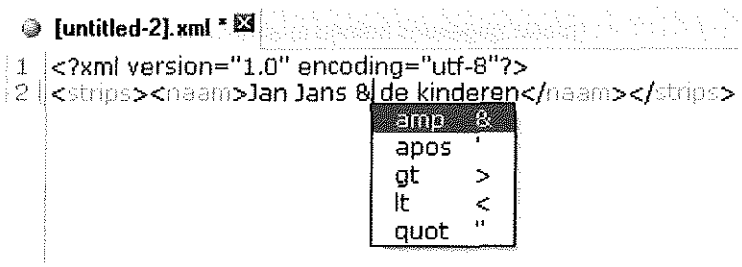
2.6.1 Entities

Een entity is een tekstweergave (stringweergave) van een gereserveerd teken. In een document kunnen we een entity opnemen door deze vooraf te laten gaan door een ampersand (&) en af te sluiten met een puntkomma (;). De ampersand geeft aan dat de tekens die volgen een karakter voorstellen.
Voor de volgende gereserveerde karakters bestaat er een entity.

Teken	entity
<	<
>	>
'	'
"	"
&	&

Door het werken met entities is de ampersand zelf ook een gereserveerd karakter geworden en bestaat er zoals hierboven te zien is voor de ampersand zelf ook een entity.
Willen we bijvoorbeeld de tekst Jan, Jans & de kinderen als naam van een strip in een XML document zetten, dan zien we dat het document niet well-formed is. De ampersand wordt als een ongeldig teken gezien.

XML BluePrint geeft bij het typen van & een keuze lijstje met vijf mogelijke opties



Het Outputwindow bevat twee tabbladen Text en Browser. Om de weergave te zien als in een browser moet het tabbald Browser atvief zijn.

2.6.2 CDATA

Als er veel gereserveerde karakters in een stukje tekst staan is het niet handig om elk teken met een entity weer te geven. Het geheel wordt dan minder goed leesbaar. We kunnen dan een CDATA sectie toevoegen. CDATA staat voor Character Data: alle gegevens binnen een CDATA sectie worden als tekst gezien, niet als gestructureerde inhoud. Binnen een CDATA sectie kunnen we alle tekst toevoegen die we willen; ook de gereserveerde karakters kunnen gewoon gebruikt worden. Een CDATA sectie begint met `<![CDATA[` en eindigt met `]]>`.

In het onderstaande stukje code ziet u een voorbeeld van het gebruik van CDATA:

```
<wiskunde>
    <vergelijking><![CDATA[y<2x]]></vergelijking>
    <vergelijking><![CDATA[y<2x & 3y<5]]></vergelijking>
</wiskunde>
```

Indien een CDATA sectie tags bevat, worden de < en > tekens geïnterpreteerd alsof ze als entities zijn opgenomen, en worden ze niet als elementen herkend.

3 XML in de praktijk

3.1 Inleiding

XML doet van zichzelf niks: het beschrijft alleen gestructureerde gegevens. Deze informatie kan vervolgens worden verstuurd, gelezen, en/of verwerkt, zowel door applicaties als door personen. De afgelopen jaren blijkt XML in tal van toepassingen met veel succes gebruikt te kunnen worden

In dit hoofdstuk gaan we kijken naar een aantal voorbeelden van XML in de praktijk. We zullen XML documenten op verschillende manier verwerken, en kijken wat er dan mee gedaan wordt. We letten in dit hoofdstuk vooral op het resultaat. U hoeft nog niet te begrijpen hoe dit allemaal in zijn werk gaat. In de latere hoofdstukken zal dit verder uitgewerkt worden.

3.2 Communiceren via XML

Via een webbestelling krijgt een webwinkel een XML document met een bestelling binnen. De webwinkel moet het product op zijn beurt bestellen bij een groothandel. Het XML document van de klant wordt hierbij met een aantal aanpassingen doorgestuurd naar de groothandel. De persoonsgegevens van de klant hoeft de groothandel niet te weten. In dit geval zal de webwinkel de klant van de groothandel zijn. Ook kan het zijn dat de webwinkel met een andere product omschrijving of een ander product-ID werkt dan de groothandel. In dat geval zal er een aanpassing nodig zijn. Natuurlijk wil de webwinkel het XML document niet handmatig aanpassen. Er wordt XML gebruikt om deze vertaalslag te maken zodat de groothandel het XML document zal kunnen verwerken. Deze vertaalslag wordt ook wel transformatie genoemd.

```
<xsl:for-each select="document('H:\XMLbestanden\webbestelling.xml')/web:bestelling/web:product">
   <product>
      <productid>
         <xsl:choose>
            <xsl:when test="web:productid='B-431'">ws-434</xsl:when>
            <xsl:when test="web:productid='B-432'">th-111</xsl:when>
            <xsl:when test="web:productid='B-433'">rg-212</xsl:when>
            <xsl:when test="web:productid='B-434'">wm-212</xsl:when>
            <xsl:otherwise>??-???</xsl:otherwise>
         </xsl:choose>
      </productid>
      <aantal>
         <xsl:value-of select="web:aantal"/>
      </aantal>
   </product>
</xsl:for-each>
```

3.3 Stylesheets

Bij het verwerken van XML documenten wordt vaak gebruik gemaakt van stylesheets. Stylesheets worden onder meer gebruikt om een document een leesbare opmaak te geven. Een stylesheet kan een document leesbaar maken voor mensen, bijvoorbeeld door het als een webpagina te representeren. Het kan ook zijn dat een stylesheet het XML document leesbaar maakt voor een applicatie door er een ander XML document van te maken.

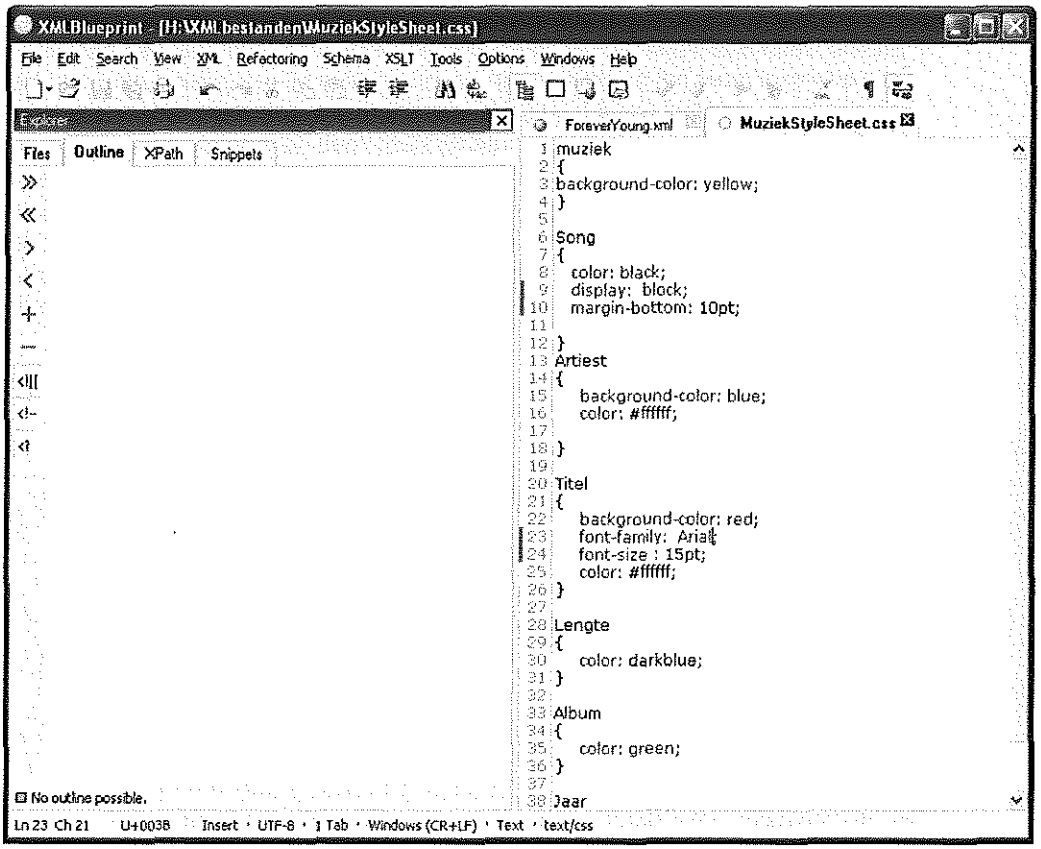
3.3.1 Cascading Style Sheets

De gegevens in een XML document kunnen worden gebruikt om als webpagina te presenteren. De eenvoudigste manier is door gebruik te maken van CSS (Cascading Style Sheets). CSS stylesheets worden meestal gebruikt voor het opmaken van HTML pagina's, maar zijn ook geschikt voor eenvoudige opmaak van XML documenten. In een CSS stylesheet kan voor elk element in een XML document worden aangeven hoe het opgemaakt moet worden. In een XML document wordt een CSS stylesheet als volgt gekoppeld:

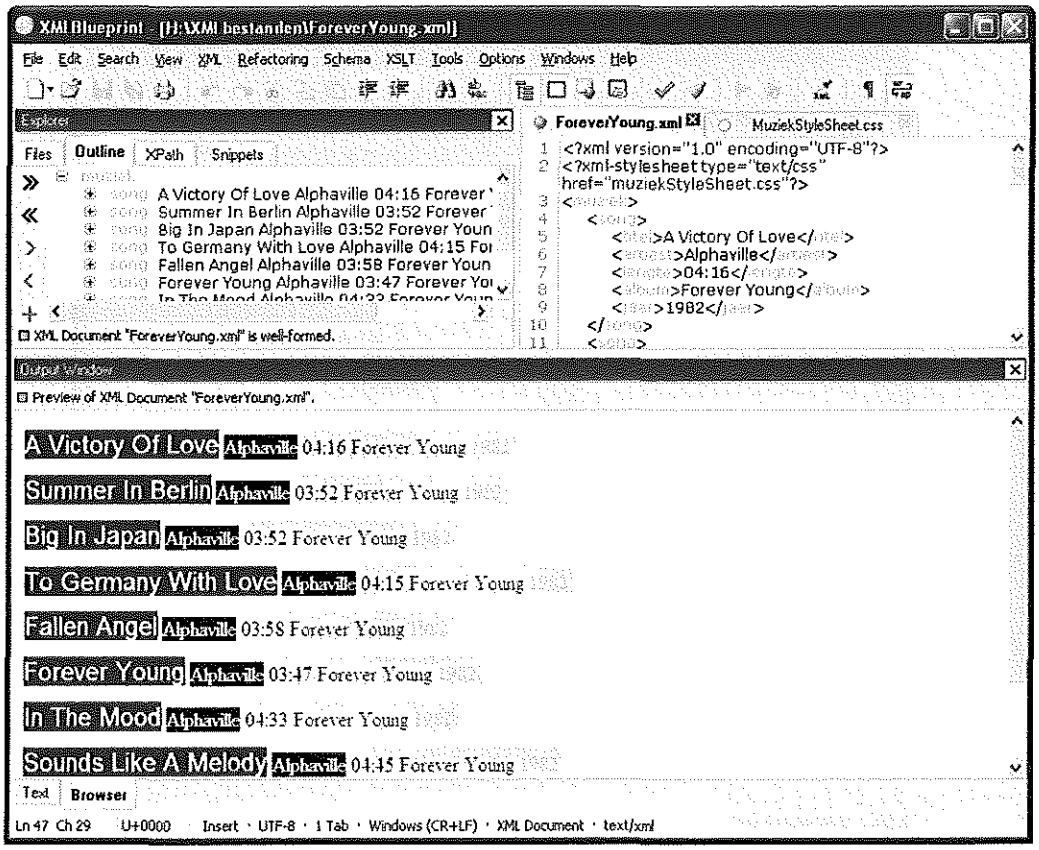
```
<?xml-stylesheet type="text/css" href="MuziekStyleSheet.css"?>
```

Eerst wordt het type stylesheet aangegeven en vervolgens de naam van het bestand. Eventueel kan ook het pad naar dit bestand worden opgegeven.

In het volgende voorbeeld bekijken de CSS stylesheet *MuziekStyleSheet.css*. In dit bestand wordt per element de opmaak aangegeven, bijvoorbeeld de kleur of een deel dat op een nieuwe regel moet beginnen:



Hieronder ziet u een voorbeeld van een XML document waaraan de bovenstaande CSS stylesheet is gekoppeld. We hebben dit in XMLBlueprint weergegeven met de optie Preview in Ouput Window (F10), zodat er een voorbeeld verschijnt van het XML document zoals dit in een webbrowser getoond zou worden:



In de CSS stylesheet wordt per tag een aantal waarden opgeven met betrekking tot de opmaak. Hier gaat het bijvoorbeeld om de tekstkleur en de achtergrondkleur. Maar ook lettertype, tekengrootte en andere opmaakkenmerken kunnen hier worden opgegeven.

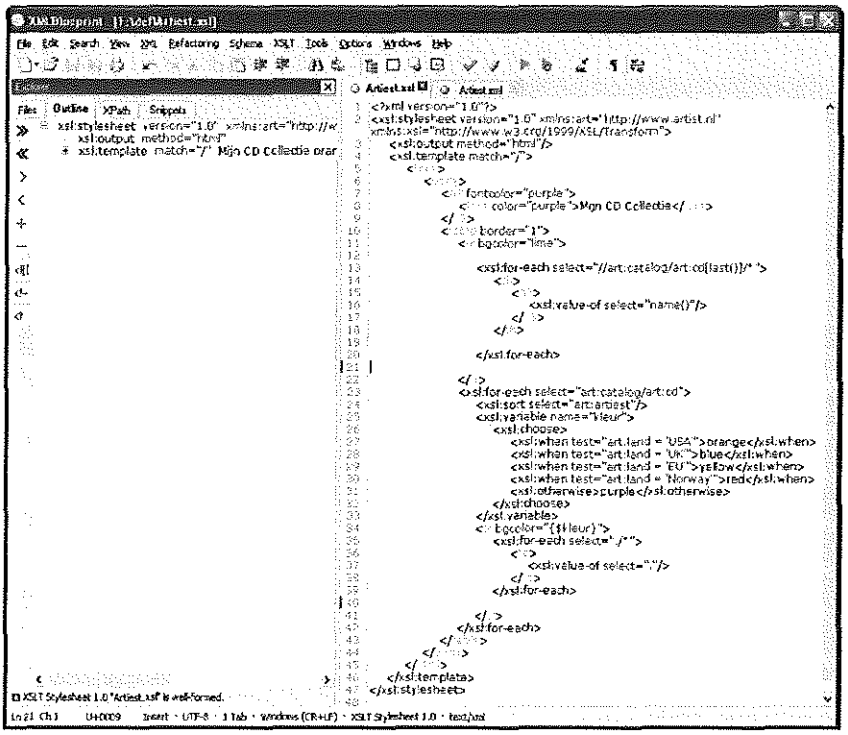
Ook in de CSS zijn de elementnamen hoofdlettergevoelig. De namen moeten worden geschreven, zoals in het XML document.

3.3.2 XSLT StyleSheets

Als XML documenten gepubliceerd worden met behulp van een CSS stylesheet dan zijn de mogelijkheden behoorlijk beperkt. Een aantal opmaaktaken kan wel worden uitgevoerd, maar er is nog heel veel wat niet met CSS stylesheets geregeld kan worden. Zo kan met behulp van een CSS stylesheet de opmaak niet afhankelijk van de inhoud worden gemaakt. Denk bijvoorbeeld aan een banksaldo dat in het rood wordt getoond als het saldo negatief is, terwijl een positief saldo in het zwart wordt getoond.

Met XSLT stylesheets hebben we hiervoor wel heel veel mogelijkheden. XSLT staat voor *Extensible Stylesheet Language Transformations.* Het XSLT stylesheet kan XML documenten lezen, en bewerken naar nieuwe XML-, HTML-, PDF- of andere tekstgeoriënteerde documenten.
We kunnen bijvoorbeeld aan de hand van een voorwaarde de opmaak van een element vastleggen. Ook kunnen er sorteringen worden toegepast. Net als bij het eerste voorbeeld van dit hoofdstuk vindt er hier weer een vertaalslag, een transformatie, plaats.

Het XSL document is een stylesheet. Met deze stylesheet wordt het XML-document naar HTML vertaald:



3.4 XML verwerken

De gegevens uit een XML document kunnen we op verschillende manieren verwerken. De voorbeelden met stylesheets zijn vooral gericht op een nette uitvoer. Maar XML documenten worden niet alleen gebruikt om gegevens netjes te presenteren. Het kan ook zijn dat op basis van een XML document gegevens worden opgeslagen in een database of dat ze naar een applicatie worden verstuurd. In de applicatie is een stuk programmacode aanwezig zodat het document verwerkt kan worden. Deze code zal meestal heel specifiek voor de applicatie zijn ontwikkeld. Over het algemeen zal het erop neer komen dat in een XML document gecontroleerd wordt welke tags er aanwezig zijn en wat de inhoud van deze tags is.

3.5 SVG

Met XML kunnen we ook afbeeldingen beschrijven. Hiervoor wordt een SVG document gebruikt. SVG is een standaard voor vectorafbeeldingen, goedgekeurd door het W3C. SVG staat voor *Scalable Vector Graphics*. Net als een standaard XML document bestaat een SVG document uit elementen en attributen. Door middel van deze elementen en attributen wordt een afbeelding in tekst geschreven.
De meeste nieuwe versies van webbrowsers ondersteunen SVG afbeeldingen. Zo ondersteund Microsoft Internet Explorer SVG vanaf versie 7.

```
<g id="a">
   <circle cx="500" cy="300" r="200" fill="yellow" stroke="black" stroke-width="2"/>
   <circle cx="400" cy="250" r="20" fill="black"/>
   <circle cx="600" cy="250" r="20" fill="black"/>
   <path d="M400 400 C400 400 500 450 600 400C600 400 500 440 400 400" fill="black"/>
</g>
```

Er kan ook beweging aan de afbeelding worden toegevoegd. We gaan de afbeelding aanpassen zodat de smile door het venster schuift.

In principe kunnen we een afbeelding volledig in tekst programmeren maar over het algemeen worden verschillende programma's gebruikt om een afbeelding te tekenen en om te zetten naar SVG.

3.6 Nog een aantal voorbeelden

Tot slot nog noemen we nog een aantal voorbeelden waarin XML wordt toegepast:

Web services:
XML maakt het mogelijk om delen van de bedrijfslogica beschikbaar te maken via het internet. Web services kunnen door externe applicaties, via het internet, gelezen en gebruikt worden. Denk hierbij bijvoorbeeld aan opzoeken van beurskoersen, valutaconversies, adressen en dergelijke.

Uitwisseling van gegevens via internet:
Het beschikbaar stellen van gegevens via web services heeft vaak nogal wat voeten in de aarde. Het is complexe materie, die in de praktijk vooral door grote bedrijven wordt gebruikt. Op kleinere schaal komt het vaker voor dat XML gegevens periodiek worden uitgewisseld, bijvoorbeeld via FTP. Een leverancier kan bijvoorbeeld zijn producten via XML aan een afnemer sturen. In dit soort gevallen is er vaak een vertaalslag nodig: het "dialect" dat bij de afnemer wordt gebruikt, is vaak net even anders dan de taal van de leverancier. Hier kunnen onderling afspraken over worden gemaakt. Vaak is er toch nog een controle en transformatie van de toegeleverde gegevens nodig, voordat de applicatie van de afnemer ermee kan praten. Daarvoor zijn twee talen ontwikkeld, die zelf ook weer in XML zijn genoteerd: XML-schema (voor de controle van het document) en XSL-stylesheet (voor de transformatie van het document naar de gewenste vorm). Deze talen komen in deze cursus uitgebreider aan de orde.

Gegevens beschikbaar maken voor diverse soorten output:
Veel bedrijven en particulieren willen gegevens beschikbaar stellen voor diverse media. Gegevens uit bijvoorbeeld relationele databases kunnen in XML worden omgezet. Technologieën als CSS en XSL kunnen vervolgens gebruikt worden om dezelfde gegevens op te maken voor verschillende media, zoals een webbrowser, printer, of mobiele telefoon.

XML vocabulaires:
Talen specifiek voor een bepaald domein. XML wordt hier gebruikt om de entiteiten en relaties daartussen binnen een kennisgebied te beschrijven. Deze maken het mogelijk om applicatieonafhankelijk informatie uit te wisselen binnen een bepaald kennisgebied. Was het vroeger noodzakelijk om de taal te gebruiken die een gekozen applicatie voorschrijft; tegenwoordig worden applicaties gebouwd die de afgesproken taal kunnen lezen. Voorbeelden van XML vocabulaires zijn MathML (wiskunde), Open Office XML (office applicaties), MusicXML (voor bladmuziek), LegalXML (juridisch domein).

4 DTD's en schema's

4.1 Inleiding

In het voorgaande hoofdstuk heeft u een aantal voorbeelden gezien van XML documenten en hoe ze worden gebruikt. In dit hoofdstuk behandelen we documenttypen en schema's.

Daarnaast zullen we leren werken met namespaces. Een namespace zouden we kunnen vergelijken met een woordenboek van een taal. Als we een brief in het Nederlands krijgen moeten we deze brief kunnen lezen. Behalve het kunnen lezen moeten we ook de Nederlandse taal beheersen. Als we een brief in een andere taal krijgen kunnen we de karakters wel lezen maar dat betekent nog niet dat we begrijpen wat er staat. Zelfs als we er een woordenboek bij willen gebruiken zullen we eerst moeten weten om welke taal het gaat. Zo is het met XML ook. Wordt in een XML document bijvoorbeeld de tag `<titel>` gebruikt, dan kan dit verschillende betekennissen hebben. Er kan een titel van een CD bedoeld worden, maar het zou ook om een titel van een song, van een boek of van een persoon kunnen gaan.

In dit hoofdstuk gaan we kijken hoe we ervoor kunnen zorgen dat alle applicaties die met een XML document werken dezelfde taal spreken. We gaan controleren of een XML document aan de juiste structuur voldoet door een document te valideren.

4.2 DTD's

Als een XML document well-formed is wil dit nog niet zeggen dat het document ook verwerkt kan worden. Om een document te verwerken moet eerst gecontroleerd worden of de ontvangen informatie voldoet aan de structuur die verwacht wordt. We zouden dit kunnen vergelijken met een spellings- en grammaticacontrole. Bij een XML document noemen we dit *valideren*. Dit valideren is van belang bij geautomatiseerde processen. Veel applicaties kunnen alleen documenten met een bepaalde structuur verwerken. Pas als een document valide is voldoet het aan die structuur.

Het vastleggen van de structuur kan op verschillende manieren. We gaan eerst kijken naar een DTD, de meest eenvoudige vorm om de structuur van een XML document vast te leggen.

DTD staat voor Document Type Definition. Een DTD is een tekstbestand waarin de regels voor de structuur van een XML document worden vastgelegd. De code van een DTD kan ook in het XML document zelf worden vastgelegd. Voordat XML werd ontwikkeld werden DTD's al gebruikt binnen SGML. Omdat XML van SGML is afgeleid zijn DTD's ook binnen XML te gebruiken.

Een DTD beschrijft de elementen die voor kunnen komen, maar ook of ze verplicht zijn of optioneel. De verwijzing naar een DTD wordt bovenin een document gemaakt in de DOCTYPE Declaration, die als tag wordt opgenomen. De meeste XML editors hebben een mogelijkheid om een XML document te valideren aan de hand van een DTD of een XML Schema.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE personen[
3  <!ELEMENT personen (persoon)+>
4  <!ELEMENT persoon (naam?, adres, plaats)>
5  <!ELEMENT naam (#PCDATA)>
6  <!ELEMENT adres (#PCDATA)>
7  <!ELEMENT plaats (#PCDATA)>
8  ]>
9
10 <personen>
11   <persoon>
12     <naam>Vincent</naam>
13     <adres>Perkstraat 5</adres>
14     <plaats>Hogerdam</plaats>
15   </persoon>
16   <persoon>
17     <naam>Marith</naam>
18     <adres>Grasstraat 18</adres>
19     <plaats>Hogerdam</plaats>
20   </persoon>
21 </personen>
22
```

Een DOCTYPE Declaration begint altijd met `<!DOCTYPE` en daar achter de naam van het root element. Hierna volgt tussen twee blokhaken een beschrijving van alle elementen die in het XML document kunnen voorkomen. Elk element wordt vervolgens apart beschreven, te beginnen met het root element. Tussen haakjes zien we de elementen die het root element kan bevatten. In dit geval is dit alleen het element persoon. Het plusteken achteraan geeft aan dat het element persoon één of meer keer voor kan komen binnen personen. We komen hier later op terug. Vervolgens wordt het element *persoon* beschreven. Dit element kan drie elementen bevatten: *naam*, *adres* en *plaats*. Vervolgens worden deze 3 elementen beschreven. Deze elementen bevatten tekst. Dit wordt aangegeven met de aanduiding `#PCDATA`.

Om te controleren of een XML document valid is gebruiken we de knop **Validate** ✓ of de toets **[F8]**. Er verschijnt een melding dat het document valid is.

Als het document niet valid is zal dit ook gemeld worden en zal er aangegeven worden waar het mis gaat.

In het bovenstaande voorbeeld spreken we van een inline DTD. De DTD is in de DOCTYPE Declaration van het XML document opgenomen. De DOCTYPE Declaration bevat zelf de regels waar het XML bestand aan moet voldoen.

4.3 Een DTD maken

De DOCTYPE Declaration kan ook een verwijzing naar een extern bestand bevatten. De regels waaraan het XML document moet voldoen worden in een apart bestand opgeslagen. Dit bestand noemen we een DTD, en het wordt ook met de extensie *dtd* opgeslagen.
Dit is handig als meerdere documenten dezelfde DTD moeten gebruiken. De DTD hoeft dan niet meer in zijn geheel bovenaan elk document vermeld te worden.
De beschrijving van de elementen komt dan in de DTD te staan zoals ze ook in de DOCTYPE Declaration stonden. In de DOCTYPE Declaration zelf verwijzen we nu naar de DTD door de tekst: `SYSTEM "bestandsnaam.dtd"` in de tag op te nemen.

4.3.1 Elementen definiëren

Als we een DTD maken, moeten we weten welke elementen er in een document voor moeten of kunnen komen, en wat voor soort elementen dit zijn. In principe hebben we twee soorten: een element kan tekst bevatten of het kan bestaan uit child elementen. Als een element tekst bevat, vermelden we dit met #PCDATA:

```
<!ELEMENT naam (#PCDATA)>
```

Dit staat voor *Parsable Character DATA.*
Willen we dat een element leeg blijft dan kunnen we dit aangeven met EMPTY:

```
<!ELEMENT naam EMPTY>
```

Bestaat een element uit 1 of meerdere child elementen, dan worden deze child elementen als een opsomming achter elkaar gezet:

```
<!ELEMENT naam (voorletters, voornaam, achternaam)>
```

De volgorde van de elementen tussen de haakjes is bepalend voor de volgorde van de elementen in het XML document.
Met behulp van komma's, haken en pipes (|) kan de structuur van een element worden vastgelegd.

Operator	beschrijving	voorbeeld	uitleg voorbeeld
,	Wordt gebruikt om de volgorde van child elementen vast te leggen	`<!ELEMENT adres (straat, huisnummer, postcode, woonplaats)>`	Het element *adres* bevat 4 child elementen: *straat*, *huisnummer*, *postcode* en *woonplaats*
|	Wordt gebruikt om een keuze weer te geven	`<!ELEMENT betaling (PIN | creditcard )>`	Het element *betaling* bevat het child element *PIN*, of het child element *creditcard.*
( )	Er kan ook gebruik worden gemaakt van groepen: de elementen binnen een groep worden dan weer tussen haakjes opgenomen	`<!ELEMENT persoon ( naam, (geboortedatum | leeftijd))>`	Het element *persoon* bevat nu altijd twee child elementen: *naam* en *geboortedatum*, of *naam* en *leeftijd.*
		`<!ELEMENT adres ((straat,huisnummer, postcode, woonplaats) | (postcode, huisnummer))>`	Het element *adres* bevat of vier child elementen: *straat*, *huisnummer*, *postcode* en *woonplaats*, of het element *adres* bevat maar twee elementen, namelijk *postcode* en *huisnummer.*

We kunnen ook nog wat extra voorwaarden stellen. We kunnen aangeven of een element optioneel is en of een element vaker dan één keer mag voorkomen. Hiervoor voegen we een extra teken achter het element toe. We hebben de keuze uit drie mogelijkheden:

teken	beschrijving	Voorbeeld	Uitleg voorbeeld
?	Het vraagteken gebruiken we als een element optioneel is. Het element komt 0 of 1 keer voor.	`<!ELEMENT persoon (naam, geboortedatum?)>`	Van een *persoon* mag de *geboortedatum* worden geregistreerd maar dit is niet noodzakelijk. De *naam* moet wel worden ingevuld.
+	Met + geven we aan dat een element meerdere keren mag voorkomen. Wel wordt afgedwongen dat het element minimaal 1 keer moet voorkomen Het element komt dus 1 of meer keren voor.	`<!ELEMENT naam (voornaam+, achternaam)>`	Een *naam* bestaat uit minimaal 1 *voornaam* en 1 *achternaam* Er mag wel meer dan 1 *voornaam* benoemd worden.
*	Het element is optioneel, maar mag ook vaker dan 1 keer voorkomen. Het element komt dus 0 of meer keer voor.	`<!ELEMENT email (to, cc* , from, tekst)>`	Een *email* is aan iemand gericht, van iemand afkomstig en heeft een inhoud Anderen kunnen een kopie van het bericht krijgen. Dat kunnen er 0 , 1 of meer zijn .

*De tekens +, * en ? kunnen we ook voor een groep van elementen gebruiken. De groep geven we weer aan met haakjes.*

4.3.2 Attributen definiëren

Tot nu toe hebben we alleen elementen zonder attributen bekeken. Als een element attributen heeft, moeten we dit ook in de DTD benoemen.
Per attribuut gebruiken we daarvoor de volgende structuur:

```
<!ATTLIST element attribuutnaam CDATA Default_value>
```

ATTLIST geeft aan dat het om een attribuut gaat. De naam van het element moet worden gegeven om aan te geven van welk element het een attribuut is. Hierna volgt de naam van het attribuut met daar achter het type. Met CDATA geven we aan dat het attribuut een tekst inhoud is. Tenslotte kunnen we nog een standaardwaarde opgeven.
Stel bijvoorbeeld dat we een attribuut geslacht van een element persoon willen beschrijven:

```
<persoon geslacht="V">
    <naam>Aniek</naam>
</persoon>
```

We hebben nu een aantal mogelijkheden om met dit attribuut om te gaan:

- het attribuut moet verplicht in het element worden opgenomen en ingevuld
- het attribuut mag worden weggelaten, en krijgt dan een standaardwaarde (bijvoorbeeld `"M"` of `"V"`)
- het attribuut mag worden weggelaten, en geldt dan als leeg.

In de DTD geven we dit aan door als standaardwaarde een van de volgende mogelijkheden te kiezen.

waarde	Beschrijving	Voorbeeld	Uitleg voorbeeld
`"waarde"`	De standaard waarde voor het attribuut wordt tussen " " opgegeven.Dit kan een tekst of een getal zijn.	`<!ATTLIST bestelling CDATA aantal "1")>`	Het element *bestelling* heeft een attribuut *aantal*. Dit attribuut krijgt de waarde *1* als er in het XML document niets wordt opgegeven. `<bestelling>` *aantal* heeft de waarde 1 `<bestelling aantal="3">` *aantal* heeft de waarde 3.
`#REQUIRED`	Er wordt geen standaard waarde gegeven. In het XML document moet voor dit attribuut altijd een waarde worden opgegeven.	`<!ATTLIST bestelling aantal CDATA #REQUIRED)>`	Het attribuut *aantal* moet nu verplicht worden opgegeven. Binnen het XML document zal `<bestelling/>` een probleem op leveren.

`#IMPLIED`	Ook hier wordt geen standaard waarde gegeven. In het XML document hoeft dit attribuut geen waarde te krijgen.	`<!ATTLIST bestelling aantal CDATA #IMPLIED)>`	Het attribuut *aantal* is optioneel. De volgende notaties in het XML document zijn toegestaan:
			`<bestelling>`
			`<bestelling aantal="3">.`
			Aantal heeft in het eerste geval geen waarde in het tweede geval de waarde 3.

`#FIXED`	#Fixed gebruiken we om een attribuut een vaste waarde te geven. Deze waarde kan niet door een XML document worden aangepast.	`<!ATTLIST getallen pi CDATA #FIXED "3.1416">`	Het element *getallen* bevat een attribuut *pi*. Dit getal verandert nooit. Er wordt altijd 3.1416 gebruikt.
			In het XML document mag het element *getallen* op de volgende manieren gebruikt worden:
			`<getallen pi="3.1416"/>`
			`<getallen/>.`
			In beide gevallen wordt met de waarde *pi*=3,1416 gewerkt.
			Het is nu niet toegestaan om de volgende notatie te gebruiken:
			`<getallen pi="3.14"/>.`

Merk op dat niet mogelijk is aan te geven dat een attribuut 1 of meer keer of 0 of meer keer gebruikt mag worden. Een attribuut kan namelijk maximaal 1 keer worden gebruikt binnen een element.

Indien een element meer attributen heeft, worden er voor hetzelfde element meerdere ATTLIST declaraties in de DTD opgenomen. Bijvoorbeeld een element bestelling, met een attribuut datum en een attribuut aantal:

```
<!ATTLIST bestelling datum CDATA #REQUIRED>
<!ATTLIST bestelling aantal CDATA "1">
```

4.3.3 Attribuuttypen

In de voorbeelden hebben we tot nu toe steeds gebruik gemaakt van attributen van het type *CDATA*. Attributen kunnen ook van een ander type zijn. In deze cursus zal vooral het *CDATA* als attribuut type gebruikt worden, maar voor de volledigheid worden de verschillende typen hieronder genoemd:

CDATA
CDATA geeft aan dat het attribuut als inhoud een tekst heeft:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personen [
<!ELEMENT personen (persoon+)>
<!ELEMENT persoon (#PCDATA)>
<!ATTLIST persoon persid CDATA #REQUIRED>
]>

<personen>
    <persoon persid="1234">Hans</persoon>
    <persoon persid="1235">Anouk</persoon>
</personen>
```

ID
Het type *ID* wordt gebruikt om naar attributen te kunnen verwijzen. De waarde van een *ID* attribuut is uniek voor het hele XML document (en dus niet alleen voor het desbetreffende element). Een element kan maar één attribuut van het type *ID* hebben.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personen [
<!ELEMENT personen (persoon+)>
<!ELEMENT persoon (#PCDATA)>
<!ATTLIST persoon persid ID #REQUIRED>
]>

<personen>
    <persoon persid="p1">Hans</persoon>
    <persoon persid="p2">Anouk</persoon>
    <persoon persid="p3">Gert</persoon>
    <persoon persid="p4">Kirsten</persoon>
    <persoon persid="p5">Anouk</persoon>
</personen>
```

Door het *ID* kan bijvoorbeeld onderscheid gemaakt worden tussen twee personen met dezelfde naam. Geen enkel ander element in het hele XML document kan met hetzelfde *ID* wordt benoemd.

IDREF
Als we gebruik maken van het type *ID* kunnen we hier met een ander attribuut naar verwijzen met attributen van het type *IDREF*.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE personen [
<!ELEMENT personen (persoon+)>
<!ELEMENT persoon (#PCDATA)>
<!ATTLIST persoon persid ID #REQUIRED>
<!ATTLIST persoon vader IDREF #IMPLIED>
<!ATTLIST persoon moeder IDREF #IMPLIED>
]>

<personen>
    <persoon persid="p1">Hans</persoon>
    <persoon persid="p2">Anouk</persoon>
    <persoon persid="p3" vader="p1" moeder="p2">Gert</persoon>
    <persoon persid="p4">Kirsten</persoon>
    <persoon persid="p5" vader="p3" moeder="p4">Anouk</persoon>
</personen>
```

Door bij de derde persoon bij vader te verwijzen naar *p1* en bij moeder naar *p2*, weten we dat Hans de vader en Anouk de moeder van Gert is. Merk op dat we niet kunnen aangeven naar welk *ID* we willen verwijzen. *IDREF* dwingt alleen af dat de waarde van het attribuut moet voorkomen als *ID* waarde in het document.

4.3.3.1 Weinig gebruikte attribuut typen

Binnen XML kennen we nog een aantal attribuut typen waarvan het gebruik tegenwoordig wordt afgeraden. Zonder hier al te diep op in te gaan willen we deze typen kort noemen.

NMTOKEN/ NMTOKENS
Het type *NMTOKEN* kan gebruikt worden om aan te geven dat de waarde van een attribuut moet voldoen aan de eisen die ook voor XML namen gelden. Dit kan bijvoorbeeld handig zijn als het XML document met behulp van *java* of *javascript* wordt verwerkt.

ENTITY/ ENTITIES
Het type *ENTITY* kan worden gebruikt om een externe verwijzing te maken. De externe gegevens worden niet als XML gezien. Dit kan bijvoorbeeld gebruikt worden om naar een afbeelding te verwijzen.

NOTATION
Als er in een XML document gegevens voorkomen die geen XML zijn dan kan met behulp van *NOTATION* aangeven wat voor soort gegevens dit wel zijn.

Bij de beschrijving van de elementen en attributen is de naamgeving hoofdlettergevoelig. De elementen en attributen zullen binnen XML dus exact overeenkomen met de namen in de DTD. De overige onderdelen van de declaratie moeten normaal gesproken in hoofdletters worden opgenomen.

4.4 Namespaces

Bij het ontwikkelen van XML is iedereen vrij om namen voor de elementen te kiezen. Het zou kunnen dat twee bedrijven met een XML document werken waarvan het root element <bestellingen> heet. En misschien wordt in beide XML documenten wel een datum geregistreerd in het element <datum>. Alleen wordt bij het ene datum element de besteldatum vastgelegd terwijl bij het andere element de geboortedatum van de besteller moet worden opgeven. Zo kan bijvoorbeeld ook het element <naam> in beide XML documenten voorkomen. Maar bij het ene document gaat het om de naam van een product terwijl in het andere geval de naam van de klant bedoeld wordt. Soms worden XML documenten uit verschillende applicaties gecombineerd.

Daarbij kan het dus gebeuren dat dezelfde elementnamen met een verschillend soort inhoud door elkaar komen te staan. Om problemen hiermee te voorkomen worden namespaces gebruikt.

Een namespace is een uitbreiding van de naam van een element, die aangeeft in welk 'domein' een element thuis hoort. Dit is een beetje te vergelijken met een achternaam: over welke Piet hebben we het? Piet Hein of Piet Visser? Met behulp van de achternaam zijn we zekerder dat we het over dezelfde persoon hebben. Zo ook met namespaces. Als we weten welke namespace bij een element hoort, weten we ook hoe we het element moeten interpreteren.

Om een namespace te kunnen gebruiken moet deze worden gedeclareerd op of boven het niveau waar de namespace wordt gebruikt. Meestal wordt de namespace daarom in het root element gedeclareerd.

Een namespace bestaat uit twee delen: een prefix en een naam.
Het declareren van een namespace gebeurt als een attribuut in een element: xmlns:prefix="URI". De prefix mag de ontwikkelaar zelf bedenken. De naam van de namespace is meestal een URI (Uniform Resource Identifier) en geeft een identificatie van de namespace. Een URI moet uniek zijn. Vaak worden er op het web extra gegevens van de namespace opgeslagen, maar dit hoeft niet. De URL van deze plek is een unieke naam en wordt vaak als URI voor de namespace gebruikt. Het feit dat een URL toch al uniek is maakt het daar heel geschikt voor.

De declaratie van een namespace kan in elk element gebeuren en is dan voor alle onderliggende elementen geldig.

Om aan te geven dat een element tot een namespace behoort, wordt meestal een prefix gebruikt voor de naam van het element. De prefix en de naam van het element worden gescheiden door een dubbele punt. Bijvoorbeeld:
<super:bestellijst> of <markt:bestellijst> of <xsl:template>

In een voorbeeld ziet dat er als volgt uit:

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

<bestelling xmlns: super="http://super.nl/namespaces/">
    <super:product>
       <super:naam>koffie</super:naam>
       <super:aantal>10</super:aantal>
    </super:product>
    <super:product>
       <super:naam>koffiemelk</super:naam>
       <super:aantal>8</super:aantal>
    </super:product>
    <super:product>
       <super:naam>suiker</super:naam>
       <super:aantal>5</super:aantal>
    </super:product>
</bestelling>
```

De namespace is in het element bestelling gedeclareerd en kan daarom in alle onderliggende elementen gebruikt worden.

In het voorbeeld hieronder zien we dat de namespace in het element product is gedefinieerd. Dit is ook een geldige structuur. De namespace is voor alle onderliggende elementen te gebruiken en is dus ook voor de elementen naam en aantal geldig.

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

<bestelling >
    <super:product xmlns:super="http://super.nl/namespaces/">
       <super:naam>koffie</super:naam>
       <super:aantal>10</super:aantal>
    </super:product>
</bestelling>
```

Hebben we in deze structuur meerdere producten dan gaat er iets fout:

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

<bestelling >
    <super:product xmlns:super="http://super.nl/namespaces/">
       <super:naam>koffie</super:naam>
       <super:aantal>10</super:aantal>
    </super:product>
    <super:product>
       <super:naam>koffiemelk</super:naam>
       <super:aantal>8</super:aantal>
    </super:product>
    <super:product>
       <super:naam>suiker</super:naam>
       <super:aantal>5</super:aantal>
    </super:product>
</bestelling>
```

De namespace is nu alleen bekend voor het product koffie en niet meer voor de andere producten, want deze staan in de hiërarchie naast het eerste product en niet eronder.

Willen we de namespace niet in het element bestelling opnemen, dan moet ons voorbeeld er als volgt uit zien:

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

<bestelling >
    <super:product xmlns:super="http://super.nl/namespaces/">
       <super:naam>koffie</super:naam>
       <super:aantal>10</super:aantal>
    </super:product>
    <super:product xmlns:super="http://super.nl/namespaces/">
       <super:naam>koffiemelk</super:naam>
       <super:aantal>8</super:aantal>
    </super:product>
    <super:product xmlns:super="http://super.nl/namespaces/">
       <super:naam>suiker</super:naam>
       <super:aantal>5</super:aantal>
    </super:product>
</bestelling>
```

4.4.1 Meerdere namespaces in een document

Het is ook mogelijk meerdere namespaces in één document te gebruiken. Hiervoor kan een extra namespace inclusief prefix in een element worden opgenomen

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

<bestelling xmlns:super="http://super.nl/namespaces/"
xmlns:markt="http://markt.nl/namespaces/">
    <super:product>
       <super:naam>koffie</super:naam>
       <super:aantal>10</super:aantal>
    </super:product>
    <markt:product>
       <markt:naam>koffiemelk</markt:naam>
       <markt:hoeveelheid>8</markt:hoeveelheid>
    </markt:product>
    <super:product>
       <super:naam>suiker</super:naam>
       <super:aantal>5</super:aantal>
    </super:product>
</bestelling>
```

Dit is een veelgebruikte manier om problemen met interpretatie te voorkomen. In bovenstaand voorbeeld zien we dat in beide namespaces een element *product* voorkomt, maar dat het product in de ene namespace een element *aantal* kan bevatten, en dat daarvoor in de andere namespace een element *hoeveelheid* wordt gebruikt. Zonder namespaces zou dit een probleem opleveren bij het controleren met behulp van een DTD of XML Schema (waarover verderop meer). Met namespaces worden beide elementen *product* terecht als verschillend herkend.

4.4.2 Standaard namespace

Als we meerdere namespaces in een document gebruiken, kunnen we één namespace als standaard namespace (default namespace) definiëren. Dit kan door een verwijzing naar de namespace op te nemen zonder een prefix te gebruiken. Het voordeel is dat de prefix voor de elementen uit deze namespace weggelaten kan worden.

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

<bestelling xmlns="http://super.nl/namespaces/"
xmlns:markt="http://markt.nl/namespaces/">
    <product>
       <naam>koffie</naam>
       <aantal>10</aantal>
    </product>
    <markt:product>
       <markt:naam>koffiemelk</markt:naam>
       <markt:hoeveelheid>8</markt:hoeveelheid>
    </markt:product>
    <product>
       <naam>suiker</naam>
       <aantal>5</aantal>
    </product>
</bestelling>
```

4.4.3 Namespaces en XML talen

Namespaces komen we vaak tegen bij standaard XML talen. Er bestaan verschillende XML talen, ook wel vocabulaires, voor verschillende kennisgebieden. Een applicatie die documenten in zo'n taal moet verwerken, controleert eerst of het document wel aan de verwachte structuur voldoet. We zullen een aantal talen in deze cursus nader bestuderen. We noemen alvast de namespaces die bij deze talen horen.

Taal	Beschrijving	Namespace
XML Schema	om een XML document mee te beschrijven en te valideren.	http://www.w3.org/2001/XMLSchema
XSLT	om een XML document mee te transformeren	http://www.w3.org/1999/XSL/Transform
XHTML	HTML die voldoet aan XML regels	http://www.w3.org/1999/xhtml
XSL-FO	om de layout van een XML document voor print-output geschikt te maken (zoals PDF)	http://www.w3.org/1999/XSL/Format

4.5 XML Schema's

Eerder in dit hoofdstuk hebben we een document gevalideerd met een DTD. We hebben gezien dat dit in sommige gevallen voldoet. Maar in de praktijk blijken DTD's vaak beperkt te zijn. XML schema's kunnen dan uitkomst bieden. In een XML schema leggen we net als in een DTD vast hoe een XML document eruit moet zien. en aan welke andere voorwaarden de inhoud moet voldoen.

Er zijn met XML Schema's veel meer mogelijkheden dan met een DTD. Zo kunnen we in een schema bijvoorbeeld aangeven:

- hoe vaak een element mag/moet voorkomen;
- van welk type de inhoud moet zijn;
- wat de maximale of minimale lengte voor de inhoud is;
- welke waarden een element mag hebben.

Dit kan allemaal niet met DTD's.
Een schema zelf is een XML bestand met de extensie *xsd*. De extensie xsd van een XML Schema bestand staat voor *XML Schema Definition.* Omdat het een XML document is moet het ook aan dezelfde regels voldoen. Een schema bestaat dus ook uit elementen waarvan één het root element is. Een XML schema moet dan ook gecontroleerd worden op well-formedness.
Net als met DTD's kan een XML document gevalideerd worden met een XML schema.
Als we gebruik maken van een XML schema moet dit in het root element worden aangeven In onderstaande afbeelding wordt in het root element *cdlijst* een verwijzing naar het schema gemaakt.

```
<cdlijst
   xmlns="http://www.artist.nl"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.artist.nl file:///H:/XMLbestanden/Artiest.xsd">
```

Als eerste wordt de standaard namespace gedefinieerd. Daaronder wordt de namespace xsi gedefinieerd, dit is een verwijzing naar de namespace XMLSchema-instance van W3C. Door deze namespace kunnen we gebruik maken van het attribuut *schemaLocation*. Hier wordt de locatie van het gebruikte schema opgegeven. Als eerste wordt de naam van de namespace genoemd en vervolgens de locatie van het schemabestand voor deze namespace.

De prefix xsi wordt vaak gebruikt voor de verwijzing naar XMLSchema-instance. XMLBlueprint gebruikt deze prefix ook als er een document op basis van een schema wordt aangemaakt. Indien gewenst mag er een andere prefix voor deze namespace gekozen worden.

In een XML schema leggen we voorwaarden voor de elementen en attributen vast.
Ook in het root element van het schema bestand vinden we een aantal verwijzingen naar verschillende namespaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.artist.nl"
   targetNamespace="http://www.artist.nl"
   elementFormDefault="qualified">
```

Het root element van de namespace *xsd* verwijst weer naar XMLSchema-instance van W3C. Merk op dat het root element schema zelf ook uit deze namespace komt. Daaronder wordt de default namespace aangegeven. Ook zien we het attribuut *targetNamespace* terug.

4.5.1 De targetNamespace

Met een schema worden elementen beschreven. De target namespace is bedoeld om aan te geven bij welke namespace deze elementen horen. Anders gezegd: het schema beschrijft elementen uit de target namespace.
Vaak is de target namespace van het schema gelijk aan de default namespace van het XML document. Het XML document hoeft echter geen default namespace te hebben: het kan ook bestaan uit elementen uit één of meer namespaces die met een prefix worden aangeduid.

De prefix xsd wordt vaak voor de namespace van het schema gebruikt, maar uiteraard is een andere prefix toegestaan.

Het schemabestand is zelf ook een XML document en kan gevalideerd worden.

Groothandel.xml | Groothandel.xsd

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3               xmlns ="http://www.groothandel.nl"
4               targetNamespace="http://www.groothandel.nl"
5               elementFormDefault="qualified">
6
7     <xsd:simpleType name="contactpersonen">
8       <xsd:restriction base="xsd:string">
9         <xsd:enumeration value="Piet Paulusma"/>
10        <xsd:enumeration value="Helga van Leur"/>
11      </xsd:restriction>
12    </xsd:simpleType>
13
14    <xsd:element name="order">
15      <xsd:complexType>
16        <xsd:sequence>
17          <xsd:element name="contactpersoon" type="contactpersonen" minOccurs="1" maxOccurs="1"/>
18          <xsd:element name="product" minOccurs="1" maxOccurs="unbounded">
19            <xsd:complexType>
20              <xsd:sequence>
21                <xsd:element name="productid" type="xsd:string"/>
22                <xsd:element name="aantal" type="xsd:int"/>
23              </xsd:sequence>
24            </xsd:complexType>
25          </xsd:element>
26          <xsd:element name="datum" type="xsd:date" minOccurs="1" maxOccurs="1"/>
27          <xsd:element name="klantid" type="xsd:int" minOccurs="1" maxOccurs="1"/>
28        </xsd:sequence>
29      </xsd:complexType>
30    </xsd:element>
31  </xsd:schema>
32
```

4.5.2 Het element <element>

Net als bij een DTD leggen we in een schema de structuur van een XML document vast. We benoemen alle elementen die in het document kunnen of moeten voorkomen.
Als we het document *groothandel.xsd* beter bekijken zien we dat hierin een groot aantal elementen is opgenomen. Deze elementen zijn te herkennen aan het element `<xsd:element>`. Omdat dit een element uit de namespaces XMLSchema-instance van W3Cis, wordt dit vooraf gegaan door de prefix van deze namespace .
Het element `<element>` heeft een aantal attributen. Het belangrijkste attribuut is *name*. Hiermee specificeren we de naam van het element in het XML document.

```
<xsd:element name="cdlijst">
```

Het attribuut *name* is verplicht. Een ander veel gebruikt attribuut is het attribuut *type*. Met *type* kunnen we aangeven wat voor soort data een element kan bevatten.

```
<xsd:element name="jaar" type="xsd:int"/>
```

De bovenstaande code dwingt af dat het element `<jaar>` alleen integer waarden kan bevatten. Een jaar element met de waarde 2008.4 zal bij het valideren een foutmelding opleveren.
Hieronder volgen de belangrijkste datatypen:

datatype	beschrijving
string	een tekst
int	een geheel getal tussen -2147483648 en 2147483647
float / decimal	een getal met decimalen
boolean	Elementen van het type boolean kunnen de waarde true of false hebben. Ook de nummerieke waarden 1 en 0 worden als true en false herkend.
date	een datum in het format YYYY-MM-DD.
time	een datum in het format hh-mm-ss.
id	Een uniek ID voor een element in het document.

Behalve de attributen *name* en *type* noemen we nog een aantal veel gebruikte attributen:

attribuut	beschrijving
name	Hiermee definiëren we de naam van het element
type	Met type leggen we het datatype van een element vast
minOccurs	Met het attribuut minOccurs wordt aangegeven hoe vaak het element minimaal opgenomen moet worden. Hiervoor kan een 0 (het element is optioneel) of een positieve integer worden opgegeven. De default waarde is 1. Als een hogere waarde dan 1 wordt gebruikt zal ook maxOccurs als attribuut moeten worden opgenomen.

attribuut	beschrijving
maxOccurs	Met het attribuut maxOccurs wordt aangegeven hoe vaak het element maximaal opgenomen kan worden. Als waarde kan een positieve integer worden opgegeven. Ook kan de tekst unbounded (oneindig veel) als waarde worden opgegeven. De default waarde is 1.
fixed	Om af te dwingen dat er een vaste waarde wordt opgegeven.
default	Wordt gebruikt om een standaard waarde op te geven.
id	Definieert een uniek ID voor een element.
ref	Om een verwijzing naar een ander (bovenliggend) element te maken. De verwijzing kan niet gebruikt worden naar het element `<schema>`. (Dit is vergelijkbaar met IDREF bij DTD's)

4.5.3 Simpletypes

Vaak voldoen de bovenstaande datatypen niet aan de eisen van de ontwikkelaar. Er zijn extra voorwaarden nodig. Bijvoorbeeld:

- de waarde van het getal moet tussen de 10 en de 50 liggen
- de tekst mag niet langer zijn dan 30 karakters
- er moet een keuze worden gemaakt uit een aantal voorgedefinieerde waarden

We zullen zelf extra typen moeten definiëren waarin we deze voorwaarden vastleggen. Hiervoor gebruiken we `<simpleType>`. In een *simpleType* kunnen we de voorwaarden vastleggen met behulp van nieuwe elementen. We hebben de keuze uit `<annotation>`, `<restriction>` , `<list>` en `<union>`

`<restriction>`	Het woord zegt het al, er kunnen restricties oftewel voorwaarden worden opgegeven.
`<list>`	Met list kunnen we aangeven dat de inhoud een lijst bevat. De items in de lijst worden gescheiden door spaties.
`<union>`	Union kunnen we gebruiken om aan te geven dat de inhoud uit één of meerdere *simpleTypes* kan bestaan, als de inhoud bijvoorbeeld aan de ene of aan de andere voorwaarde moet voldoen.
`<annotation>`	Wordt gebruikt om in-line opmerkingen weer te geven.

Om een simpleType te gebruiken moet de naam worden ingevuld bij het attribuuttype van een element.

Voorbeeld:

```
<xsd:simpleType name="getal">
       <xsd:restriction base="xsd:decimal">
            <xsd:maxInclusive="60"/>
       </xsd:restriction>
</xsd:simpleType>

<xsd:element name="nummer" type="getal"/>
```

De naam van het simpleType *getal* wordt gebruikt als type in het element *nummer*.

```
<xsd:simpleType name="meerkeuzevraag">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="A"/>
      <xsd:enumeration value="B"/>
      <xsd:enumeration value="C"/>
      <xsd:enumeration value="D"/>
   </xsd:restriction>
</xsd:simpleType>
```

Als eerste zien we het element *xsd:simpleType*. Dit element heeft een attribuut *name*. Hier wordt de naam van het type vastgelegd. Naar deze naam moeten we verwijzen als we elementen van dit type willen toevoegen.

Binnen het element *simpleType* leggen we de voorwaarden van dit type vast.
In dit geval wordt er met *xsd:restriction* aangeven dat de inhoud aan een voorwaarde moet voldoen. Het element *restriction* heeft een attribuut base. Hier geven we het basistype van dit element aan, zoals bijvoorbeeld *xsd:string* of *xsd:int*.
In het voorbeeld wordt bij *restriction* afgedwongen dat een keuze gemaakt moet worden uit lijst met gegevens. Dit wordt gedaan met behulp van het element *xsd:enumeration*.
Het attribuut *value* bevat de waarde die gekozen mag worden. Elke keuze moet apart vastgelegd worden met een nieuw *enumeration* element.
Behalve *enumeration* kent *restriction* nog een aantal elementen om een voorwaarde voor een element vast te leggen. Hieronder zijn de mogelijkheden onder elkaar gezet:

<enumeration>
Met dit element definiëren we een lijst met toegestane waarden.
Voorbeeld:
```
<xsd:enumeration value="A"/>
<xsd:enumeration value="B"/>
<xsd:enumeration value="C"/>
<xsd:enumeration value="D"/>
```
Bij de inhoud moet gekozen uit de waarden A, B, C of D.

<maxLength>
Dit element wordt gebruikt om een maximale lengte voor een string op te geven.
Voorbeeld:
```
<xsd:maxLength value="10"/>
```
De inhoud mag niet langer dan 10 karakters zijn.

<minLength>
Om een minimale lengte voor een string op te geven.
Voorbeeld:
```
<xsd:minLength value="3"/>
```
De inhoud moet minimaal 3 karakters lang zijn.

<length>
Om af te dwingen dat een string exact het opgegeven aantal karakters bevat.
Voorbeeld:
```
<xsd:length value="5"/>
```
De inhoud moet uit precies 5 karakters bestaan

<pattern>
Om een patroon voor de inhoud vast te leggen. Dit patroon wordt opgeven door middel van een reguliere expressie
Voorbeeld:
```
<xsd:pattern value="[0-9]{4}\s[A-Z]{2}"/>
```
de inhoud moet voldoen aan het opgegeven patroon.
In dit geval gaat het om een postcode die moet voldoen aan de structuur 4 cijfers, een spatie en 2 hoofdletters.

`<maxExclusive>` Om af te dwingen dat een getal kleiner moet zijn dan de opgegeven waarde (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:maxExclusive value="10"/>
```
De inhoud moet een getal kleiner dan 10 bevatten.

`<maxInclusive>` Om af te dwingen dat een getal kleiner of gelijk moet zijn dan de opgegeven waarde (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:maxInclusive value="10.5"/>
```
De inhoud moet een kleiner of gelijk aan 10.5 bevatten.

`<minExclusive>` Om af te dwingen dat een getal groter moet zijn dan de opgegeven waarde (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:minExclusive value="0"/>
```
De inhoud moet een getal groter dan 0 bevatten.

`<minInclusive>` Om af te dwingen dat een getal groter of gelijk moet zijn dan de opgegeven waarde (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:minInclusive value="-10"/>
```
De inhoud moet een getal groter of gelijk aan -10 bevatten.

`<whiteSpace>` Met wihteSpace kunnen we aangeven wat er gedaan moet worden met witruimte(spaties, tabs ed). Bij *value* kan gekozen worden uit 3 waarden: *collapse*, *preserve*, of *replace*. De optie *preserve* doet niets met extra spaties en laat ze gewoon staan *replace* zal alle tabs en regeleinde harde returns naar spaties omzetten. De optie *collapse* doet het zelfde als *replace* maar zal de spaties aan het begin en eind verwijderen en alle dubbele spaties tot een enkele spatie samenvoegen.
Voorbeeld:
```
<xsd:whiteSpace value="collapse"/>
```
Alle overbodige spaties in de inhoud worden verwijderd.

`<totalDigits>` Hiermee kunnen we de maximale lengte (aantal cijfers) van een getal opgeven (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:totalDigits value="5"/>
```
De inhoud mag uit niet meer 5 cijfers bestaan.

`<fractionDigits>` Hiermee kunnen we het maximale aantal decimalen opgeven (kan alleen gebruikt worden bij getallen).
Voorbeeld:
```
<xsd:fractionDigits value="3"/>
```
De inhoud mag niet meer dan 3 decimalen bevatten.

4.5.3.1 Reguliere expressies

Met behulp van het element *<pattern>* kan een voorwaarde voor de inhoud worden vastgelegd. Dit wordt gedaan met behulp van reguliere expressies. De term reguliere expressie is binnen veel programmeertalen bekend. Een reguliere expressie beschrijft een patroon. We kunnen teksten zoeken die aan dit patroon voldoen. Of we kunnen testen (valideren) of een tekst wel volgens dit patroon is opgebouwd. Dit laatste wordt toegepast in

een schema met een *<pattern>* element. De eenvoudigste vorm van een reguliere expressie is de tekst opgeven waar de inhoud gelijk aan moet zijn.

```
<xsd:pattern value="test"/>
```

Bij dit patroon kan de inhoud alleen het de tekst *test* bevatten om valid te zijn. Met karakter klassen kunnen we het patroon veel uitgebreider maken. Zo kunnen we bijvoorbeeld aangeven dat er op een bepaalde positie een hoofdletter moet staan met [A-Z] kleine letters met [a-z]. Als we alleen x y of z willen gebruiken kan dat aangegeven worden met [x-z]. voor getallen kunnen we gebruik maken van de klasse [0-9]. Willen we een patroon van 4 cijfers en 2 hoofdletters dan kunnen we dat als volgt opbouwen: [0-9][0-9][0-9][0-9][A-Z][A-Z]. Dat is behoorlijk wat telwerk, vooral als de teksten wat langer zijn. Gelukkig is daar ook een oplossing voor. Het zelfde patroon kunnen we ook als volgt geven: [0-9]{4}[A-Z]{2}.

Naast deze patronen kennen we nog een heel aantal patronen gebruiken. Hieronder volgt een opsomming van de belangrijkste:

expressie	beschrijving	voorbeeld
[ ]	lijst van karakters	[Aa] of [XYZ1:_]
[..-..]	een streepje in de karakterlijst geeft een interval aan	[a-z], [A-Z], [0-9], [A-z] of [2-5]
[^]	een karakterlijst met karakters die niet mogen voorkomen.	[^Aa] , [^XYZ1:_] of [^A-CZ]
( )	een groep van karakters	(test)
.	één willekeurig karakter	test.
		(begint met test en dan nog 1 karakter er achter, bijvoorbeeld test1 of testa)
?	het voorgaande karakter of de voorgaande groep mag 0 of 1 keer voorkomen	1t? of (test)?
*	het voorgaande karakter of de voorgaande reeks mag 0 , 1 of meer keer voorkomen	1t* of (test)*
+	het voorgaande karakter of de voorgaande groep moet 1 of meer keer voorkomen	1t+ of (test)+
{n}	het voorgaande karakter of de voorgaande groep moet precies n keer voorkomen	1t{3} of (test){2}
{n,m}	het voorgaande karakter of de voorgaande groep moet tussen n en m keer voorkomen	1t{0,5} of (test){1,2}
{n,}	het voorgaande karakter of de voorgaande reeks moet n of meer keer voorkomen	1t{4,} of (test){2,}

expressie	beschrijving	voorbeeld
|	de ene karaktergroep of de andere karaktergroep	dhr.|mvr. of hoofd(weg|straat)
\s	witruimte	[0-9]{4}\s[A-Z]
\S	geen witruimte	\S[a-z]
\d	numeriek karakter	\d{3}
\D	geen numeriek karakter	\D{3}
\w	alfanumerieke karakters	\w{3}
\W	geen alfanumerieke karakters	\W{3}

4.5.4 ComplexTypes

Met een simpleType kunnen we heel wat voorwaarden stellen. Maar het is niet genoeg. Met een simpleType kunnen we bijvoorbeeld niet vastleggen dat een element uit child-elements bestaat. Ook kunnen we met een simpleType geen attributen definiëren. Dit kunnen we wel voor elkaar krijgen met een *complexType*.
Een complexType hebben we al in een aantal documenten gezien. Eigenlijk bevat elk schema wel een of meerdere complexTypes. Want het root element heeft eigenlijk altijd child-elements en dit kan in een schema niet anders dan met een complexType gedefinieerd worden.

Een belangrijk element binnen een complexType is <sequence>. Met een sequence kunnen we een lijst van elementen opgeven.

Voorbeeld:

```
<xsd:element name="adres">
    <xsd:complexType>
       <xsd:sequence>
            <xsd:element name="straat" type="straat"/>
            <xsd:element name="huisnummer" type="getal"/>
            <xsd:element name="postcode" type="pc"/>
            <xsd:element name="plaats" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Op deze manier geven we aan dat het element adres 4 child-elements heeft. De elementen *straat*, *huisnummer*, *postcode* en *plaats*. Deze elementen moeten in de opgegeven volgorde worden opgenomen

De belangrijkste keuzen binnen een complextype zijn:

`<sequence>` Wordt gebruikt om een volgorde van elementen te definiëren.

`<choice>` Wordt gebruikt om een keuze uit één van de opgegeven elementen te maken. Wordt het attribuut *maxOccurs* gebruikt, dan kunnen we aangeven dat er meer dan 1 gekozen mag worden.
De volgorde is niet van belang. Ook mag een element vaker dan 1 keer voor komen.

`<all>` Geeft aan dat alle elementen precies één keer voor moeten komen. De volgorde maakt niet uit.

`<group>` Wordt gebruikt om een groep van elementen te definiëren. Later kan naar deze groep worden verwezen

We kunnen een complexType ook buiten het root element definiëren. Het voordeel is dat we dit type dan op verschillende momenten kunnen gebruiken.

4.5.5 Het element <attribute>

Met een complexType kunnen we ook attributen voor een element definiëren. Hiervoor wordt het element <attribute> gebruikt. Het *attribute* element is alleen beschikbaar binnen een complexType.

```
<xsd:element name="prijs">
    <xsd:complexType>
       <xsd:attribute name="valuta" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
```

Op deze manier kennen we aan het element *prijs* het attribuut *valuta* toe. Het attribuut *valuta* is van het type *string*. Om een attribuut te definiëren moeten we de naam van het attribuut opgeven. Ook moeten we het type van het attribuut definiëren. Net als bij de inhoud van een element kunnen we naar een type verwijzen. Let op dat de inhoud enkel uit karakterdata bestaat. Dit betekent dat het alleen een standaard type of een gedefinieerd simpleType kan zijn.

In het document zal het element *prijs* er bijvoorbeeld als volgt uit zien:

```
<prijs valuta="€"></prijs>
```

Als we het element *prijs* vullen met een waarde levert dit een probleem op. Het onderstaaande voorbeeld zal bij het valideren een foutmelding geven.

```
<prijs valuta="€">42.35</prijs>
```

Als we attributen op deze manier definiëren, kunnen we het element niet meer vullen met andere gegevens. Meestal is dit niet de bedoeling. Ook elementen met attributen moeten een inhoud kunnen krijgen. Daarom moeten we de definitie van het type uitbreiden.

We kunnen nu kiezen uit twee varianten:

<simpleContent> Wordt gebruikt om een eenvoudige inhoud te definiëren (de inhoud bestaat uit tekst en attributen)

<complexContent> Wordt gebruikt om een complexere inhoud te definiëren (de inhoud mag bestaan uit elementen en attributen)

We moeten aangeven uit wat voor soort inhoud het element bestaat. Dit kan met het element `<extension>`. Het attribuut *base* van dit element is verplicht op te geven. Dit attribuut is te vergelijken met het attribuut *type* van `<element>`.

```
<xsd:element name="prijs">
    <xsd:complexType>
       <xsd:simpleContent>
            <xsd:extension base="xsd:float">
                <xsd:attribute name="valuta" type="xsd:string"/>
            </xsd:extension>
       </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
```

Als we nu een XML document met de volgende inhoud voor *prijs* valideren zal dit geen probleem meer opleveren:

```
<prijs valuta="€">42.35</prijs>
```

Het element *attribute* heeft zelf ook een aantal attributen. Twee van deze attributen hebben we al gezien: *name* en *type*. Hieronder volgt een lijst met belangrijke attributen. De attributen *name* en *type* zijn verplicht, de andere attributen zijn allemaal optioneel.

Attribuut	**beschrijving**
default	Hiermee kan een standaard waarde worden gedefinieerd. Dit attribuut kan niet in combinatie met het attribuut *fixed* worden gebruikt.
fixed	Hiermee kan een vaste waarde worden gedefinieerd. Dit attribuut kan niet in combinatie met het attribuut *default* worden gebruikt.
id	Met dit attribuut kan een uniek *id* aan het attribuut worden gekoppeld.
name	Met *name* kunnen we het attribuut een naam geven. Het attribuut *name* kan niet in combinatie met *ref* worden gebruikt. Wel is het verplicht om één van deze twee op te nemen.
ref	Met *ref* kunnen we verwijzen naar een eerder vastgelegd attribuut. Het attribuut *ref* kan niet in combinatie met name worden gebruikt. Wel is het verplicht om één van deze twee op te nemen.
type	Hiermee dwingen we het type van de inhoud af; dit kan een vooringesteld datatype zijn of een type vastgelegd in een simpleType.
use	Hiermee dwingen we af of het verplicht is om het attribuut op te nemen. We kunnen kiezen uit 3 waarden • optional • prohibited • required Als *use* niet wordt gebruikt wordt standaard de optie optional gekozen. Het attribuut is dus niet verplicht.

4.5.6 De elementen simpleContent en complexContent

Bij het definiëren van attributen kunnen we gebruik maken van de elementen simpleContent en complexContent. Als een complexType alleen karakterdata en attributen bevat gebruiken we simpleContent en als het complexType zelf ook weer elementen en attributen bevat wordt het complexContent element gebruikt. In beide gevallen kunnen we dan met behulp van het element `<restriction>` of `<extension>` een type meegeven.

Het element <restriction> bepaalt hoe de inhoud van een type er uit mag zien. Dit element zijn we ook al bij simpleType tegengekomen. Bij het attribuut *base* van dit element kunnen we een verwijzing naar een ander type opgeven.

Voorbeeld:

```
<xsd:complexType name="klant">
    <xsd:sequence>
       <xsd:element name="voornaam" type="xsd:string"/>
       <xsd:element name="achternaam" type="xsd:string"/>
       <xsd:element name="land" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="nlklant">
    <xsd:complexContent>
       <xsd:restriction base="klant">
           <xsd:sequence>
              <xsd:element name="voornaam" type="xsd:string"/>
              <xsd:element name="achternaam" type="xsd:string"/>
              <xsd:element name="land" type="xsd:string" fixed="Nederland"/>
           </xsd:sequence>
       </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
```

Hier wordt eerst een algemene klant gedefinieerd. Het type *nlklant* is gebaseerd op het type *klant* door het element <xsd:restriction base="klant">. Alle voorwaarden (restricties) van *klant* gelden nu ook voor *nlklant*. Een *nlklant* moet net als een *klant* een *voornaam* en een *achternaam* hebben. Ook moet er een element *land* zijn. Alleen wordt dit element bij *nlklant* vast op Nederland gezet.

Met het element van `<restriction>` leggen we extra voorwaarden op aan een basistype. Als we een basistype juist willen uitbreiden kunnen we het element `<extension>` gebruiken.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.groothandel.nl" targetNamespace="http://www.groothandel.nl"
elementFormDefault="qualified">

<xsd:complexType name="persoon">
  <xsd:sequence>
    <xsd:element name="voornaam" type="xsd:string"/>
    <xsd:element name="achternaam" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="persoonsgegevens">
  <xsd:complexContent>
    <xsd:extension base="persoon">
      <xsd:sequence>
        <xsd:element name="adres" type="xsd:string"/>
        <xsd:element name="huisnummer" type="xsd:string"/>
        <xsd:element name="postcode" type="xsd:string"/>
        <xsd:element name="plaats" type="xsd:string"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="werknemers">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="werknemer" type="persoonsgegevens"
                                    minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Eerst wordt het type *persoon* gedefinieerd. Als we in het type *persoonsgegevens* van een persoon meer gegevens willen vastleggen, kunnen we nu gebruiken maken van het element `<xsd:extension base="persoon">`. Hierdoor worden de elementen *voornaam* en *achternaam* van persoon aan het element *persoonsgegevens* toegevoegd. Verder voegen we daar dan nog een aantal elementen aan toe.

4.5.7 De elementen group en attributeGroup

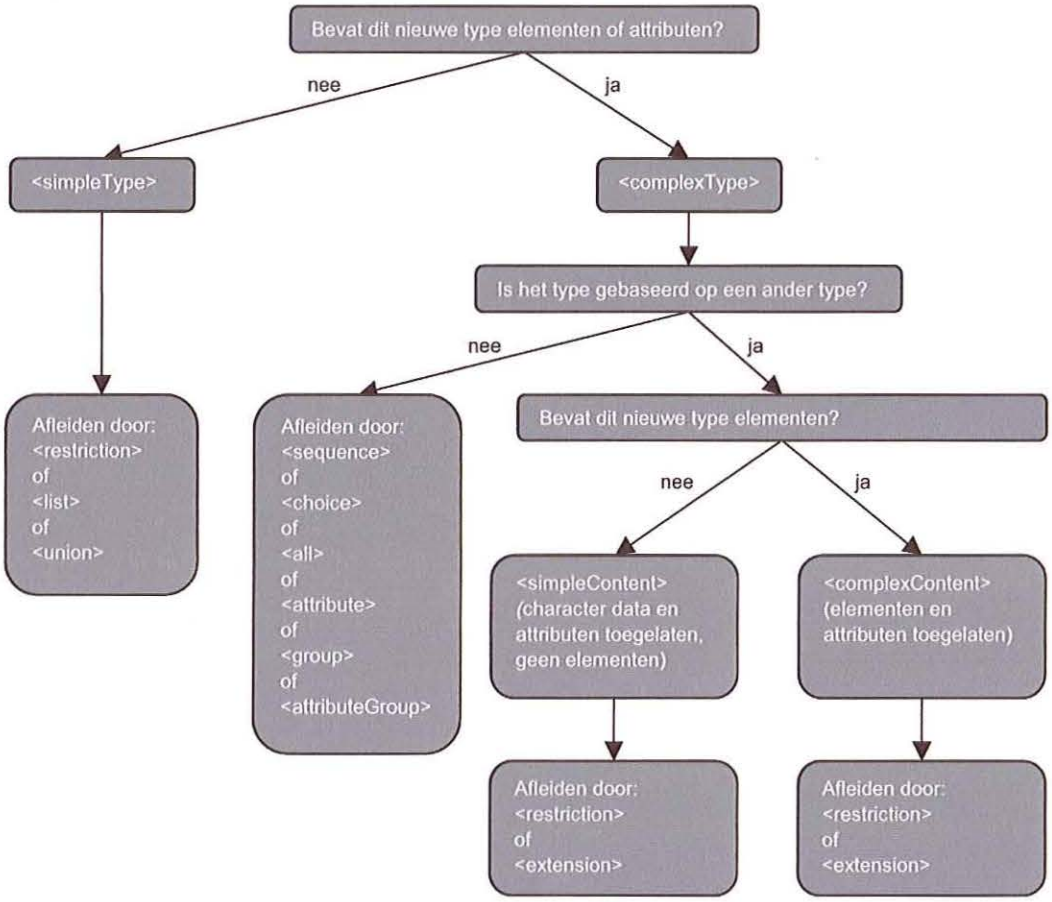
Soms willen we een groep van elementen of attributen op verschillende momenten gebruiken. Dan kunnen we deze elementen samen nemen. Hiervoor wordt het element `<group>` gebruikt. In een groep kunnen we weer de elementen *sequence, choice* of *all* gebruiken om de elementen binnen *group* te definiëren. Binnen een *complexType* kunnen we dan een element `<group>` op nemen. Bij het attribuut *ref* moeten we dan de naam van de *group* waar naar we verwijzen opnemen. Dit zelfde kan ook voor een groep van attributen .

4.5.8 Het element annotation

Binnen elk schema element hebben we de mogelijkheid om het element `<annotation>` te gebruiken om extra commentaar op te nemen. Binnen het *annotation* element kunnen we twee nieuwe elementen gebruiken: `<documentation>` en `<appInfo>`. Het element <documentation> wordt gebruikt om documentatie bij het schema op te nemen. Dit element heeft twee attributen: *source*, om een applicatienaam op te geven en *xml:lang*, om een taal te definiëren. Het element `<appInfo>` wordt gebruikt om specifieke informatie te geven voor de applicatie die de documenten moet verwerken. Ook *appInfo* heeft een attribuut *source*.

4.6 Een schema ontwerpen

Het ontwerpen van een schema is niet altijd eenvoudig. De vele mogelijkheden maken het lastig om te bepalen hoe een type gedefinieerd moet worden. In het onderstaande schema vindt u een aantal aanwijzingen om de juiste keuzen te maken.



OVERZICHT ALLE ELEMENTEN IN XSD:

WWW.W3SCHOOLS.COM/SCHEMA/SCHEMA_ELEMENTS_REF.AS

SCHEMA W3C:

WWW.W3.ORG/2001/XMLSCHEMA.XSD

5 Transformaties en opmaak

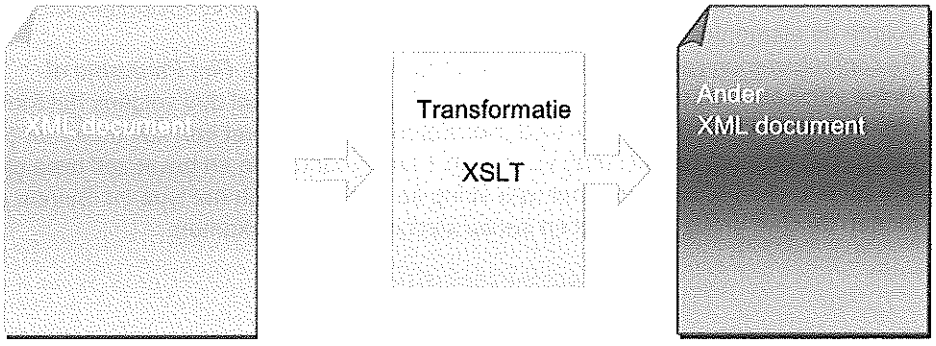
5.1 Inleiding

In hoofdstuk 3 hebben we al even kennis kunnen maken met een aantal manieren waarop XML verwerkt kan worden. Bij het verwerken van XML documenten is het meestal van belang dat het document is opgebouwd volgens een vastgestelde structuur. Pas als de structuur aan bepaalde voorwaarden voldoet, kan het document verwerkt worden. In het vorige hoofdstuk hebben we ons gericht op het vastleggen van die structuur. In dit hoofdstuk gaan we vooral kijken hoe een document verwerkt wordt.
We leren hoe een XML document kan worden omgezet naar een HTML document of naar ander XML document. Hierbij zullen we XSLT gebruiken. Daarnaast zullen we kijken hoe we een XML document op een nette manier kunnen weergeven.

5.2 XSLT Stylesheets

Vaak wordt XML gebruikt om verschillende systemen of organisaties met elkaar te laten communiceren. Het probleem is dat het ene systeem vaak een andere XML verwacht dan het andere systeem. De ene organisatie heeft gegevens nodig waar de andere organisatie niets aan heeft. Om te kunnen communiceren vindt er vaak een vertaalslag plaats. Een voorbeeld hiervan hebben we al in hoofdstuk 3.2 gezien:
Een webwinkel ontvangt een bestelling. De gegevens van de bestelling worden in een XML document verzameld. De webwinkel verwerkt dit document voor de eigen administratie. Bij de afhandeling moet de webwinkel de producten bij een groothandel bestellen. Voor de communicatie met de groothandel wordt het XML document weer gebruikt. Toch kan het XML document niet rechtstreeks naar de groothandel gestuurd worden. Het systeem van de groothandel kan dit XML document niet verwerken.
Er zal een vertaalslag moeten plaatsvinden. Het XML document van de webwinkel moet naar een voor de groothandel bruikbaar XML document vertaald worden. De groothandel stelt een aantal eisen aan het te ontvangen XML document. In een schema kan de groothandel vastleggen hoe het XML document er uit moet zien. De webwinkel zal het eigen XML document moeten vertalen naar een structuur die voldoet aan het schema van de groothandel. Hiervoor kan een XSLT stylesheet worden gebruikt.

In een XSLT stylesheet is een serie regels vastgelegd. Op basis van deze regels wordt er een nieuw document gegenereerd.



Het originele document wordt getransformeerd in een bruikbaar document. Het stylesheet bouwt een nieuw document met een nieuwe boomstructuur.

De gegevens uit het origineel worden in deze nieuwe structuur op de juiste plaats toegevoegd. De elementen die niet in de nieuwe structuur nodig zijn worden weggelaten en gegevens die niet in het origineel zijn te vinden worden toegevoegd. Zo ontstaat een nieuw document in de gewenste structuur.

5.3 Een XML document omzetten in een ander XML document

We gaan dit voorbeeld van de webwinkel nog eens beter bekijken.

Omdat het XML document van de webwinkel niet voldoet aan de structuur die de groothandel verwacht zal er een vertaalslag moeten plaats vinden. Het XML document van de webwinkel moet naar een ander XML document vertaald worden zodat de groothandel de gegevens kan verwerken. De groothandel stelt een aantal eisen aan het te ontvangen XML document. Deze eisen zijn vastgelegd in het schema *groothandel.xsd*. Met behulp van een transformatie zal de webwinkel het eigen XML document om gaan zetten naar een voor de groothandel bruikbaar XML document.

De webwinkel zal nu een XML document moeten sturen dat met dit schema gevalideerd kan worden. Het document *webbestelling.xml* moet getransformeerd worden naar een voor de groothandel leesbaar document. Hiervoor wordt een XSLT stylesheet gebruikt. Dit is een bestand met de extensie xsl. Om een xsl bestand te maken moeten we als eerste goed weten hoe het originele bestand in elkaar zit. Nog belangrijker is hoe het document er na de transformatie uit moet gaan zien.

Het root element van een webbestelling is het element *bestelling*. Binnen *bestelling* komen we de volgende child elementen tegen: *datum*, *klantnaam* en *product*. De elementen *datum* en *klantnaam* komen altijd maar 1 keer voor. Het element *product* is altijd aanwezig maar kan vaker dan 1 keer voor komen. Hier zullen we dus rekening mee moeten houden.
Het element *datum* heeft zelf weer drie children. De elementen *dag*, *maand* en *jaar*. Deze elementen bevatten allemaal een getal en mogen slechts één keer voorkomen.
Ook het element *product* heeft drie children: *productid*, *omschrijving* en *aantal*. Per *product* mogen deze elementen slechts één keer voorkomen. Het element *aantal* is een getal, de andere elementen bevatten een tekstwaarde.

Na deze analyse kijken we hoe het resultaat er uit moet zien. Hiervoor gebruiken we de voorschriften van de groothandel die zijn vastgelegd in het schema *groothandel.xsd*.

Een XML voor de groothandel heeft een rootelement `<order>`. Binnen *order* wordt eerst de contactpersoon van de groothandel genoemd met het element `<contactpersoon>`. Daarna volgen de elementen `<product>`, `<datum>` en `<klantid>`.
Bij *contactpersoon* kan slechts gekozen worden uit personen: *Piet Paulusma* of *Helga van Leur*. Het element *datum* is van het type date en zal dus volgens het formaat jaar-maand-dag worden opgebouwd. Het element *klantid* bevat een klantnummer (een getal). Het element *product* mag vaker dan 1 keer voorkomen en bevat zelf twee child elementen: `<productid>` van het type string en `<aantal>` van het type *int*. Binnen *product* mogen de elementen *productid* en *aantal* slechts 1 keer voorkomen.

5 Transformaties en opmaak

Bij het vertalen zullen we op de volgende onderdelen letten:

- Het rootelement verandert van `<bestelling>` in `<order>`.
- De contactpersoon moet worden toegevoegd.
- Alle bestelde producten moeten worden doorlopen en als *product* in het nieuwe document terugkomen. Er hoeft alleen een *productid* en aantal opgenomen te worden.
- De drie afzonderlijke child elementen van datum moeten omgezet worden naar een waarde in het formaat: jaar-maand-dag.
- Er moet een *klantid* worden toegevoegd: dit zal het klantnummer van de webwinkel zijn.

5.3.1 Een XSLT stylesheet maken

Om de vertaling van het ene document naar het andere document te maken gebruiken we XSLT stylesheets. XSLT is een taal die vastgelegd is door het w3c. Dit is te zien aan de namespace van XSLT: http://www.w3.org/1999/XSL/Transform. Als we in XMLBlueprint een nieuw XSLT document maken wordt het root element <stylesheet> al automatisch aangemaakt. Deze heeft twee attributen, waarmee de namespace en de XSLT versie worden aangeduid.

Binnen het root element wordt ook een <template> element aangemaakt.

[untitled-1].xsl

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3     <xsl:template match="/">
4
5     </xsl:template>
6 </xsl:stylesheet>
```

Het element xsl:template element heeft een match attribuut met de waarde "/". Het template element bepaalt wat er moet gebeuren, zodra in het te bewerken XML document een element wordt gevonden die voldoet aan het match attribuut. De waarde "/" verwijst naar de root, waarvan het root element het enige child element is. We zullen hier verderop in dit hoofdstuk dieper op ingaan. Voor dit moment is het genoeg te weten dat met match="/" het hoogste niveau van het XML document gevonden wordt.

Binnen het element <template> kunnen we de structuur van het nieuwe XML document opgeven.

5.4 XPath

Een XML document bestaat uit een boomstructuur van elementen (ook wel nodes genoemd). Om gegevens uit een XML document te halen zal de boomstructuur doorlopen moeten worden. Hiervoor gebruiken we XPath. Met XPath kunnen we opgeven welk element uit de boomstructuur we willen hebben. XPath beschrijft de weg door de boomstructuur. We zullen de basisprincipes van XPath bespreken.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:web="http://www.webwinkel.nl">
   <xsl:template match="/">
      <order xmlns="http://www.groothandel.nl"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.groothandel.nl /XMLbestanden/Groothandel.xsd">
         <contactpersoon>Piet Paulusma</contactpersoon>
         <product>
            <productid>

            </productid>
            <aantal>

            </aantal>
         </product>
         <datum>

         </datum>
         <klantid>23451</klantid>
      </order>
   </xsl:template>
</xsl:stylesheet>
```

In XMLBlueprint kunnen we kijken wat een XPath expressie oplevert. Hiervoor gebruiken we het tabblad XPath in de explorer.

Een XPath expressie begint vaak met een slash (/). Zoals we eerder hebben behandeld verwijst deze slash naar de root van het XML document (en *niet* het root element: dit is het child element van de root). Voor elke vertakking in de boom gebruiken opnieuw een slash (/). Naast de slash is er nog een aantal tekens die een speciale betekenis hebben in een XPath expressie:

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.

5.4.1 XPath expressies

Soms willen we niet alle child nodes selecteren, maar de eerste of de laatste of bijvoorbeeld de tweede node. Met XPath expressies kunnen we dit oplossen. Ook kunnen we met een pad expressie een voorwaarde of een berekening opgeven. Een pad expressie wordt altijd tussen [ ] gegeven.

Voorbeeld pad expressie	Resultaat
/bestelling/product[1]	Selecteert het eerste *product* dat een child node is van *bestelling*.
(een ander getal mag ook)	**Let op:** IE5 en hoger gebruiken een andere standaard: De eerste node is [0] Volgens de W3C standaard zou dit [1] moeten zijn!!
/bestelling/product [last()]	Selecteert het laatste *product* dat een child node is van *bestelling*.
/bestelling/product [last()-1]	Selecteert het op een na laatste *product* dat een child node is van *bestelling*.
/bestelling/product [position()<3]	Selecteert de eerste twee producten (child nodes) van *bestelling*.
//product[2]/@kleur	Selecteer de waarde van het attribuut *kleur*, van het tweede *product*.
//product[@type]	Selecteert alle nodes *product* met een attribuut *type*.
//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

5.4.2 XPath met namespaces

We hebben gezien hoe we met XPath de boomstructuur van een document kunnen doorlopen. We hebben dit gezien in een document zonder namespaces. Dit is een eenvoudige situatie. Als we werken met namespaces moeten we dit ook in het XPath aangeven.

Dit kunnen we doen door de prefix voor elk element te noemen:

```
<?xml version="1.0" encoding="utf-8"?>
<bestelling
   xmlns:test2="http//www.test2.nl">

   <test2:product kleur="chroom" kleurbkleding="zwart/antraciet">
      <test2:productID>99EK23</test2:productID>
      <test2:omschrijving>eetkamer stoel thom</test2:omschrijving>
      <test2:prijs valuta ="€">74.95</test2:prijs>
      <test2:aantal>6</test2:aantal>
   </test2:product>
```

Als er binnen een XML document een default namespace wordt gebruikt moet er binnen XMLBlueprint ook een prefix gebruikt worden voor het evalueren van een XPath.

Omdat een default namespace geen prefix heeft kunnen we hiervoor zelf een prefix bedenken. Standaard wordt "default" als prefix gebruikt.

Er zijn veel tools om een XPath te evalueren. Meestal hoeft er geen prefix voor de default namespace te worden gebruikt. XMLBlueprint is hierin afwijkend.

5.5 XPath in een stylesheet

Nu we hebben gezien hoe een XPath werkt kunnen we dit toepassen in een XSLT document. We kunnen een XPath opnemen op het moment dat we de inhoud van een elementen of een attribuut nodig hebben in het XSLT document.

We hebben de stylesheet *webbestelling.xsl* uitgebreid, met behulp van XPath expressies. Het element <for-each> wordt gebruikt om een serie elementen uit het bron bestand te doorlopen. Het element <value-of> wordt gebruikt om een waarde uit een element op te halen. Beide elementen hebben een attribuut *select*. Dit attribuut zal de XPath expressie bevatten om het juiste element uit het bronbestand te selecteren. De code van *webbestelling.xsl* ziet er als volgt uit:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:web="http://www.webwinkel.nl">
   <xsl:template match="/">
      <order xmlns="http://www.groothandel.nl"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.groothandel.nl /XMLbestanden/Groothandel.xsd">
         <contactpersoon>Piet Paulusma</contactpersoon>
         <product>
            <productid>
               <xsl:value-of select="//web:productid"/>
            </productid>
            <aantal>
               <xsl:value-of select="//web:aantal"/>
            </aantal>
         </product>
         <datum>
            <xsl:value-of select="concat(//web:jaar,'-',//web:maand,'-',//web:dag)"/>
         </datum>
         <klantid>23451</klantid>
      </order>
   </xsl:template>
</xsl:stylesheet>
```

In de bovenstaande code hebben we er voor gezorgd dat het *productid* en het *aantal* van het bron document worden overgenomen. Alleen bevatte het bron document drie producten terwijl het resultaat alleen het eerste product toont. De hele lijst van producten moet doorlopen worden. Hiervoor gebruiken we het element `<for-each>`. Met *for-each* doorlopen we alle elementen van de opgeven node. Ook hier wordt het attribuut *select* met een XPath expressie gebruikt om de juiste gegevens uit het bron document te selecteren.

```
<contactpersoon>Piet Paulusma</contactpersoon>
<xsl:for-each select="/web:bestelling/web:product">
   <product>
      <productid>
         <xsl:value-of select="//web:productid"/>
      </productid>
      <aantal>
         <xsl:value-of select="//web:aantal"/>
      </aantal>

   </product>
</xsl:for-each>
<datum>
   <xsl:value-of select="concat(//web:jaar,'-',//web:maand,'-',//web:dag)"/>
</datum>
<klantid>23451</klantid>
```

Voor elk product wordt de boomstructuur opnieuw doorlopen en steeds wordt het eerste *productid* en het eerste *aantal* geselecteerd. Dit is natuurlijk niet de bedoeling. Het is ook niet nodig: elke XPath expressie wordt geëvalueerd ten opzichte van de huidige node. Bij <xsl:for-each select="/web:bestelling/web:product"> is de huidige node "web:bestelling". Met xsl:value-of kunnen we daarbinnen direct verwijzen naar de child nodes web:productid en web:aantal.

In het venster Setup XSLT Transformation is het mogelijk om het resultaat gelijk als document weg te schrijven.

5.5.1 XSLT elementen

Behalve de in dit hoofdstuk genoemde XSLT elementen zijn er nog meer. We geven hieronder een opsomming van de meeste gebruikte XSLT elementen.

XSLT element	beschrijving
<xsl:template>	Wordt gebruikt om een sjabloon te definiëren. Het attribuut *match* wordt gebruikt voor de koppeling met een xml element (match="/" geeft een koppeling met het hele document weer).
<xsl:apply-templates>	Wordt binnen een xsl:template element gebruikt. Koppelt een template aan het huidige element of aan de child nodes van het huidige element.
<xsl:output>	Geeft aan wat voor type output moet worden gegenereerd, zoals xml of html.
<xsl:value-of>	Haalt de waarde van het opgegeven element uit het XML document. Het attribuut *select* bevat een XPath expressie.
<xsl:for-each>	Kan worden gebruikt om een bepaalde node-set van een opgeven XML element te doorlopen.
<xsl:if>	Wordt gebruikt om afhankelijk van een voorwaarde iets wel of niet te doen. Aan dit element moet altijd het attribuut *test* worden toegevoegd. De *test* is een expressie die waar of niet waar oplevert.
<xsl:choose>	Zorgt voor een combinatie van verschillende tests. Bevat altijd één of meerdere `<xsl:when>` elementen. Aan elk *when* element wordt altijd een attribuut *test* toegevoegd. De *test* is een expressie die waar of niet waar oplevert. Eventueel kan een element `<xsl:otherwise>` worden gebruikt om aan te geven wat er moet gebeuren als aan geen van de opgegeven voorwaarden wordt voldaan.
<xsl:sort>	Sorteert de opgegeven lijst op basis van het element dat is aangegeven bij het *select* element. Een *sort* wordt altijd gebruikt binnen een `<xsl:for-each>` element. Dit element heeft een aantal attributen om de sorteervolgorde mee te sturen. De belangrijkste attributen zijn order (ascending of descending) en data-type (text of number).

Bij de test van `<xsl:if>` en `<xsl:when>` zullen we er altijd voor moeten zorgen dat de expressie bij test een vergelijking is die waar of onwaar oplevert.
We kunnen hier de volgende operatoren voor gebruiken:

operator	effect
> of >	groter dan
<	kleiner dan
>= of >=	groter dan of gelijk aan
<=	kleiner dan of gelijk aan
=	is gelijk aan
!=	ongelijk aan
not( )	logische negatie
and	logische en
or	logische or
|	samen voeging van twee resultaten

5.6 XML naar HTML

We kunnen een XSLT stylesheet ook gebruiken om een XML document om te zetten naar een HTML document. Zo kunnen we de gegevens in een XML document op een nette manier tonen. We kunnen dan alle HTML elementen gebruiken om het document te tonen. De werkwijze is gelijk aan het omzetten van XML naar een ander XML document. De HTML tags worden als elementen toegevoegd. Hiervoor is wel enige kennis van HTML noodzakelijk.

In het volgende voorbeeld gaan we een XML document van een webbestelling omzetten naar een HTML document zodat de bestelling in een nette lay-out verschijnt.

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

<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:web="http://www.webwinkel.nl" exclude-result-prefixes="web">
    <xsl:output method="html" version="4.01" encoding="UTF-8" indent="yes"/>

    <xsl:template match="/">
        <html>
            <header>
                <title>Bestelling</title>
            </header>

            <body>
                <h1>Bestelling van
                    <xsl:value-of select="//web:klantnaam"/>
                </h1>

                <table>
                    <tr>
                        <th>omschrijving</th>
                        <th>productID</th>
                        <th>aantal</th>
                    </tr>

                </table>

                <br/>
                datum: <xsl:value-of select="concat(//web:jaar,'-',//web:maand,'-',//web:dag)"/>


            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Om de bestelling hieraan toe te voegen hebben we het XSLT element *for-each* nodig. Omdat we de bestelling in een tabel weergeven gebruiken we de HTML tags `<table>`, `<tr>`, `<th>` en `<td>` in onze code.

```
<xsl:for-each select="/web:bestelling/web:product">
    <tr>
        <td><xsl:value-of select="./web:omschrijving"/></td>
        <td><xsl:value-of select="./web:productid"/></td>
        <td><xsl:value-of select="./web:aantal"/></td>
    </tr>
</xsl:for-each>
```

5.6.1 Variabelen

Bij het werken met XSLT stylesheet kan het handig zijn om gebruik te maken van variabelen. In een variabele kunnen we een waarde vastleggen die we later gaan gebruiken. Om een variabele te gebruiken hebben we het XSLT element `<variable>` nodig. De naam van de variabele leggen we vast in het attribuut *name* van dit element. De waarde van de variable kan als inhoud tussen de begin- en eind-tag van variable worden opgenomen. Deze waarde kan niet worden gewijzigd. In de volgende oefening gaan we een variabele kleur definiëren. Met deze variabele gaan we de even en de oneven rijen van de tabel een verschillende kleur geven. Om te bepalen of een product in een even of een oneven rij zit gebruiken we de XPath expressie *position() mod 2 = 0*. De functie position() geeft de relatieve positie van een element binnen het parent element aan, dus in dit geval: het zoveelste product element binnen bestelling. De test *mod 2 = 0* geeft *waar* terug als dit een even positie is.

```
<xsl:for-each select="/web:bestelling/web:product">

   <xsl:variable name="kleur">
      <xsl:choose>
         <xsl:when test="position() mod 2=0">#00dddd</xsl:when>
         <xsl:otherwise>#00aaaa</xsl:otherwise>
      </xsl:choose>
   </xsl:variable>

   <tr>
      <td><xsl:value-of select="./web:omschrijving"/></td>
      <td><xsl:value-of select="./web:productid"/></td>
      <td><xsl:value-of select="./web:aantal"/></td>
   </tr>
</xsl:for-each>
```

5.6.2 Automatische transformatie

In het voorgaande voorbeeld hebben we steeds XMLBlueprint gebruikt om de transformatie uit te voeren. Met behulp van een processing instruction in het XML document kan de transformatie bij het openen in de browser worden uitgevoerd. In de processing instruction geven we het type stylesheet aan en een verwijzing naar het XSLT document.

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

Bij het openen van het document wordt de processing instruction gelezen, de stylesheet wordt opgezocht en toegepast om vervolgens het resultaat in een webpagina te tonen.

5.7 Opmaak met CSS

In hoofdstuk 3 hebben we gezien dat we met behulp van *Cascading Style Sheets* een XML document kunnen opmaken. In dat geval hebben we geen transformaties nodig. Hoewel er aan CSS wel beperkingen zitten is het in verschillende situaties goed te gebruiken. We zullen daarom een aantal aspecten van CSS bespreken.

Met behulp van CSS kunnen we aan de verschillende elementen een opmaak geven. Hiervoor wordt in een CSS de elementnaam opgenomen met daarachter tussen accolades de opmaakkenmerken.

```
Titel{background-color: red;
      font-family:  Arial;
      font-size : 15pt;
      color: #ffffff; }
```

De verschillende opmaakkenmerken worden tussen de accolades genoemd volgens de structuur `kenmerk:waarde;`. In het voorbeeld wordt eerst de achtergrond de kleur rood gegeven, het lettertype wordt op arial gezet en de tekengrootte op 15 punten. Tenslotte krijgt tekst nog de kleur #ffffff (dit is de code voor wit). De volgorde waarin deze kenmerken worden genoemd is niet van belang.

5.7.1 Eigenschappen

We zullen straks in een oefening een eenvoudige CSS maken. Daarvoor hebben we kennis nodig van een aantal eigenschappen die we binnen CSS kunnen gebruiken. In deze cursus hebben we niet de tijd en de ruimte om alle aspecten van CSS bespreken. Daarvoor verwijzen we u naar de cursus Cascading Style Sheets.
Wel geven we een opsomming van een aantal belangrijke opmaakkenmerken:

5.7.1.1 Tekst eigenschappen

eigenschap	beschrijving
font-family	Zet de inhoud van een element in het opgegeven lettertype.
font-size	Zet de inhoud van een element in de opgegeven tekengrootte of het opgegeven percentage.
font-style	Zet de inhoud van een element in de opgegeven tekststijl. U kunt kiezen uit de waarden: *normal* (default), *italic, oblique.*
font-weight	Geeft de tekstdikte aan. U kunt kiezen uit de waarden: *none, bold, bolder, lighter.*
color	Bepaalt de kleur van de tekst. De kleur kan met de naam van één van de standaardkleuren worden aangeven, bijvoorbeeld red, of met een hexadecimale notatie, bijvoorbeeld #FF9A12.
text-indent	Laat de tekst inspringen met de opgegeven lengte of het opgeven percentage.
text-align	Bepaalt de uitlijning van de tekst. U kunt kiezen uit de waarden: *left, right, center* of *justify.*
text-decoration	Wordt gebruikt om extra opmaak als onderstrepen aan een tekst toet tevoegen. U kunt kiezen uit de waarden: *underline, overline, line-through* en *blink.*
line-height	Geeft de regelafstand met de opgeven waarde. Dit kan een lengte zijn maar ook een aantal regels of een percentage.
text-transform	Geeft aan of een tekst al dan niet in hoofdletters moet worden gegeven. U kunt kiezen uit de volgende waarden: *none, capitalize, uppercase* of *lowercase.*

Een lengte kan worden opgeven in centimeters (cm), inches (in) punten (pt) of pixels (px). Ook wordt een lengte nog wel eens in em spaces (em) gegeven. Een em space is de ruimte die een hoofdletter M in neemt in het opgegeven lettertype en de opgeven grootte.

5.7.1.2 Eigenschappen voor gebieden

eigenschap	beschrijving
background-color	Zet de achtergrond kleur van een element op de opgeven waarde. De waarde kan een standaardkleur of een hexadecimale waarde zijn.
background-image	Zet een afbeelding op de achtergrond. De waarde is de URL naar de afbeelding.
border-color	Zet de kleur van de rand op de opgeven waarde. De waarde kan een standaardkleur of een hexadecimale waarde zijn.
border-style	Geeft de lijnstijl van de randen weer. Er kan gekozen worden uit de volgende waarden: *none, hidden, dotted, dashed, solid, double, groove, ridge, inset* en *outset.*
border-width	Geeft de dikte van de randen. Als waarde wordt de dikte meegegeven. Ook kunnen de volgende waarden worden gebruikt: *thin, medium* en *thick.*
margin	Geeft de grootte van de marge. Als waarde wordt de lengte of een percentage opgegeven.
padding	Geeft de grootte van de witruimte tussen de rand en de tekst. Als waarde wordt de lengte of een percentage opgegeven.

In plaats van border-color mag ook border-top-color, border-left-color, border-right-color of border-bottom-color worden gebruikt om 1 specifieke rand een kleur te geven. (Dit geldt ook voor border-style, border-width, margin en padding).

5.7.1.3 De eigenschap "display"

Als laatste element bespreken we nog de eigenschap *display*. Met *display* hebben we verschillende mogelijkheden om de weergave te bepalen. We zullen de belangrijkste hier noemen:

waarde	beschrijving
none	Het element wordt niet getoond.
block	Het element wordt getoond als een block. Het volgende element wordt op een nieuwe regel gezet.
inline	De elementen worden achter elkaar getoond zonder nieuwe regels.
list-item	De elementen van dit type worden als een lijst getoond.
table	De elementen worden als een tabel getoond zoals met het HTML element <table>.
table-row	De elementen worden als een rij getoond zoals met het HTML element <tr>.
table-column	De elementen worden als een kolom getoond zoals

waarde	beschrijving
	met het HTML element <col>.
table-cell	De elementen worden als een cel getoond zoals met het HTML element <td> en <th>.

Met deze elementen kunnen we sturen in de opmaak van een XML document. Daarnaast kunnen we met CSS nog veel meer eigenschappen gebruiken maar het voert voor deze cursus te ver om ze allemaal te bespreken.

5.7.2 Een CSS maken

Om een CSS te maken moeten we goed weten welke elementen we willen tonen en hoe we ze willen tonen. In de volgende oefening willen we het document met een cdlijst op een nette manier tonen. We willen per cd een lijst met tracks met een aantal kleuren en netjes uitgelijnd.

```
cd{display:block;
   margin-bottom :20pt;
   margin-right: 15 cm;
   font-family: arial;
   background-color: #ffffcc;
   }
```

Met display: block wordt aangegeven dat een cd als een afzonderlijk blokje van gegevens moet worden getoond. Met margin geven we extra ruimte aan in dit geval ruimte tussen de verschillende blokken(cd's) en ruimte aan de rechter kant. Beide zijn gegeven in een percentage van het hele blok. Verder wordt een lettertype aangegeven en een achtergrond leur.

```
cdtitel{font-size : 20pt;
        color:darkblue;
       }
```

Voor het element cdtitel wordt de tekengrootte bepaald en de tekenkleur.

```
tracklist{text-indent: 1 cm;
          background-color:#ffff99;
          display: block;
          margin-top:0.3 cm;
          margin-left:5%;
          margin-right: 10%;
         }
```

De tracklist wordt ook als een apart blokje gezien. De eigenschappen van het bovengelegen blok gelden hier ook nog. De tekst wordt 1 cm ingesprongen Merk op dat het blok zowel links als recht wordt ingesprongen. De inspringing rechts komt boven op de 15 cm van het eerste blok waar dit blok een onderdeel van is. De bovenmarge is absoluut en wordt gegeven in cm.

De tracks zullen willen we als een genummerde lijst zien. Hiervoor wordt de volgende code gebruikt:

```
track{display : list-item;
      list-style : decimal;
      list-style-type: decimal;
      }
```

De elementen nummer en jaar hoeven niet getoond te worden dit wordt al volgt aangegeven:

```
nummer{display:none;}

jaar{display :none;}
```

5.7.3 Een CSS koppelen

We moeten er nu alleen nog voor zorgen dat de CSS gebruikt wordt. Hiervoor voegen we een *Processing Instruction* aan het XML document toe.

```
<?xml-stylesheet type="text/css" href="bestandsnaam.css"?>
```

5.8 Opmaak met XSL-FO

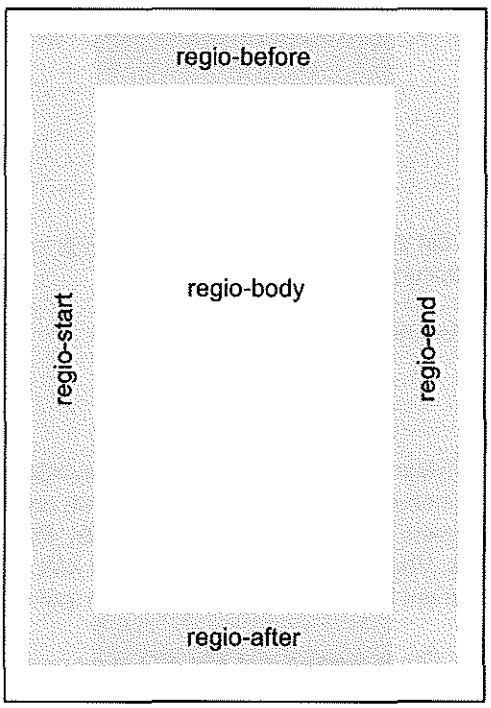
In de vorige paragrafen hebben we XSL stylesheets besproken. We hebben XSL gebruikt om extra transformaties uit te voeren met behulp van XSLT. We kunnen XSL ook gebruiken om een XML document een opmaak te geven. Deze vorm van XSL heet XSL-FO. De afkorting FO staat voor *Formating Objects*. Voor de opmaak zetten we een XML document om in een *.fo* document. Met het .fo document kunnen we bijvoorbeeld een .pdf document maken.
Ook XSL-FO is door het W3C vastgelegd. Om te werken met XSL-FO moet naar de XSL-FO namespace worden verwezen:

```
xmlns:fo="http://www.w3.org/1999/XSL/Format"
```

Meestal wordt hier de prefix *fo* gebruikt. Met XSL-FO hebben we de mogelijkheid om een volledige pagina lay-out te maken. We zullen hier onder de belangrijkste onderdelen bespreken.

5.8.1 XSL-FO elementen

Een .fo document heeft altijd een element `<fo:root>`. Vervolgens wordt de basis opmaak van de pagina gedefinieerd in `<fo:layout-master-set>`. In een *layout-master-set* kan een aantal pagina typen gedefinieerd worden met het element *simple-page-master*. Dit element kan diverse attributen bevatten. In elk geval moet het attribuut *master-name* gebruikt worden om het pagina type een naam te geven. Verder kunnen er attributen voor marges, randen, lettertype, kleuren et cetera. worden gegeven.
Het element *simple-page-master* kan zelf ook weer child elementen bevatten, te weten de elementen *regio-before, regio-after regio-start, regio-end* en *regio-body*. De eerste vierkomen op elke pagina als een soort kop en voetteksten. De *regio-body* bevat de inhoud van de pagina.



Als het nodig is kunnen er meerdere pagina lay-outs worden gedefinieerd door opnieuw een *simple-page-master* element op te nemen.

Na de pagina-indeling gemaakt te hebben kunnen we aangeven dat we een pagina willen maken. Dit doen we met het element <fo:page-sequence>. Het attribuut *master-reference* van dit element wordt gebruikt om naar een *simple-page-master* te verwijzen. Zo bepalen we welke pagina lay-out we gebruiken voor deze pagina.
Om een pagina te vullen met gegevens gebruiken we de elementen *static-content* en *flow*. Binnen deze elementen wordt vaak gebruik gemaakt van het element *block*. Per *block* kan een tekst opmaak worden gegeven.

Er zijn nog veel meer XSL-FO elementen en attributen maar in deze cursus kunnen we die niet allemaal bespreken. In het volgende voorbeeld komen er nog een aantal aanbod.

5.8.2 Een opgemaakt document tonen

Ook het XSL-FO document is nog geen leesbare uitvoer. Om het document in een leesbaar formaat te tonen hebben we en XSL-FO processor nodig. Deze processor kan het fo document omzetten in een ander formaat zoals PDF, RTF of PostScript. De XSL-FO processor is geen onderdeel van XMLBlueprint. We zullen daarom Apache FOP gebruiken.

We starten het programma Fop van af de MS_DOS Prompt. We zullen daarbij het pad en de naam van het fo bestand moeten opgeven en het pad en de naam van het uitvoerbestand.

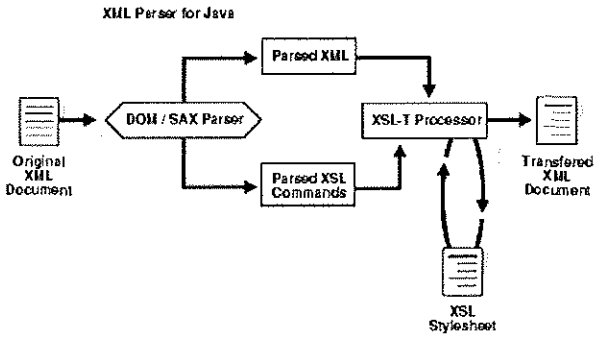
5.8.3 Een XSL-FO document maken

We hebben gezien dat we een XSL-FO document kunnen omzetten naar een ander formaat. Ook hebben we kort besproken hoe een XSL-FO document is opgebouwd. In een XSL-FO document komt veel herhaalde code voor, vaak afhankelijk van de inhoud van het XML document. Het ligt daarom voor de hand om het XSL FO document te genereren uit het XML document met behulp van een XSLT stylesheet. Hiervoor moeten we de structuur van een XSL-FO document kennen. Dit hebben we eerder al gezien aan het begin van deze paragraaf. Vervolgens moeten we op de juiste plek de gegevens uit het XML document invoegen.

6 XML documenten verwerken

6.1 Inleiding

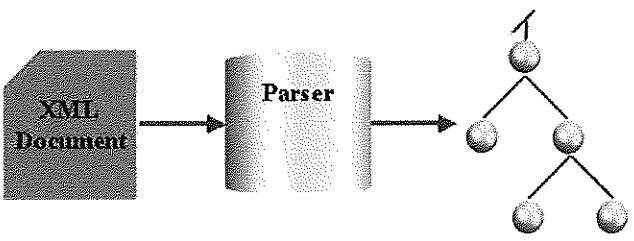
Tot nu toe hebben we op verschillende manieren XML documenten gemaakt en ontworpen. In dit hoofdstuk behandelen we hoe XML documenten verwerkt kunnen worden. Het verwerken wordt ook wel parsing genoemd. In hoofdstuk 3 hebben we hier al even kennis mee gemaakt. Met een parser worden XML bestanden ingelezen om te kunnen gebruiken en bewerken. In dit hoofdstuk gaan we dit wat uitgebreider bekijken. Bij het verwerken van een XML document door een applicatie wordt gebruikt gemaakt van een API (Aplication Programming Interface). Met een API wordt een verzameling definities vastgelegd. Op basis van deze definities kan een applicatie communiceren met een ander programma. Zo kunnen programmeertalen als Java, JavaScript, C++, Visual Basic of .NET allemaal gebruik maken van deze definities om te kunnen communiceren.



We zullen de twee belangrijkste manieren van verwerken (DOM en SAX) bespreken. Bij het ontwikkelen van XML is het van belang om te weten hoe een document verwerkt wordt en wat voor parser we daar dan het beste voor kunnen gebruiken. We zullen daarom ook kijken wat de verschillen zijn en wanneer welke methode handig is.

6.2 DOM

Veel XML documenten worden verwerkt volgens het *Document Object Model (DOM)*. Kenmerkend voor de DOM methode is dat de hele boomstructuur van een XML document eerst in het geheugen van de computer wordt geladen. Pas als het document geladen is kan met de verwerking worden begonnen. Die verwerking kan bestaan uit het inlezen van de gegevens. De DOM API is ook geschikt de structuur of inhoud van het document aan te passen. De DOM API kan ook worden gebruikt om een geheel nieuw document op te bouwen, bijvoorbeeld op basis van een tekstdocument.



DOM is een standaard van het W3C die gebruik maakt van de boomstructuur van XML. Met DOM kun je door de elementen en attributen heen lopen, en kun je bepaalde elementen selecteren. Hiervoor kan bijvoorbeeld een *XPath* expressie worden gebruikt. Een nadeel van het gebruik van de DOM API, is dat het gehele document in het geheugen moet zijn geladen, voordat het document verwerkt kan worden. Dit kan bijvoorbeeld performance problemen geven bij het verwerken van grote documenten.

6.2.1 Een document verwerken met DOM

Voor het verwerken van een XML document met DOM wordt de DOM API gebruikt. In het volgende voorbeeld zullen we een XML document met DOM benaderen. In een HTML document kunnen we een stukje JavaScript opnemen. Deze code vraagt om een XML document en zal het dan gaan verwerken.

We bekijken een voorbeeld van een HTML pagina waarin eerst om een document naam gevraagd wordt en daarna op basis van een tweetal knoppen de inhoud van dit bestand op twee manieren kan tonen. Het document bevat een drietal javascript functies, die we hier kort bespreken.

```
function LaadDocument()
{
    xmlDoc = new ActiveXObject("MSXML2.DOMDocument");
    xmlDoc.load(window.prompt("geef de naam van het XML document","cdlijst.xml"));
    root = xmlDoc.documentElement;
}
```

De functie *LaadDocument* wordt gestart als de pagina wordt geladen. Eerst wordt de variabele *xmlDoc* aangemaakt, in dit geval als een ActiveXObject. Vanaf dit moment kan de variabele *xmlDoc* een documentinhoud bevatten. Vervolgens wordt de inhoud van het opgegeven XML document aan deze variabele gekoppeld, met behulp van de methode *load*. De code voor aanmaken van de variabele en het laden van het document is taalspecifek, en maakt geen deel uit van de DOM API. Tenslotte wordt de root van het document bepaald met behulp van het attribuut *documentElement* van *Document*.

De DOM API is onafhankelijk van de gebruikte programmeertaal. Op www.w3.org kunnen we de documentatie bij deze API terug vinden. Voor documentElement staat daar bijvoorbeeld:

> *"`documentElement` of type Element, `readonly`*
> *This is a convenience attribute that allows direct access to the child node that is the document element of the document."*

De variabelen *xmlDoc* en *root* uit ons voorbeeld kunnen we nu gebruiken in de rest van het document.

Zoals gezegd bevat het document een tweetal knoppen, onder de eerste knop hangt de functie *KnopLijst* die wordt gestart wanneer er op de knop Toon CDlijst wordt geklikt.

```
function KnopLijst()
{
    objNodeList1 = xmlDoc.getElementsByTagName("artiest");
    objNodeList2 = xmlDoc.getElementsByTagName("cdtitel");
    myString = "CD lijst<br><br>";
    for(i = 0; i < objNodeList1.length; i++)
    {
       myString = myString + (i+1)+". "+ objNodeList1.item(i).text+" : " +
       objNodeList2.item(i).text + "<br>";
    }
    Div1.innerHTML = myString
}
```

In deze functie worden eerst alle elementen van een het type artiest en alle elementen van het type cdtitel in een variabele gezet met `xmlDoc.getElementsByTagName`. Vervolgens worden alle items één voor één getoond met behulp van een loop.

De functie *KnopInhoud* wordt gestart als op de tweede knop Toon CD wordt geklikt.

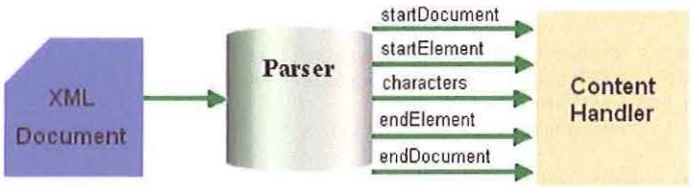
```
function KnopInhoud()
{
    objNodeList = xmlDoc.getElementsByTagName("cdtitel");
    nodenr=window.prompt("geef het nummer op",1)-1;

    commentaar = xmlDoc.createComment("CD "+ (nodenr+1));
    root.childNodes.item(nodenr).appendChild(commentaar );

    window.alert(root.childNodes.item(nodenr).xml);
}
```

Eerst worden alle elementen van een het type cdtitel weer in een variabele gezet. Vervolgens wordt gevraagd een nummer op te geven. Deze waarde wordt aan de variabele *nodenr* gekoppeld. In de laatste regel wordt dit nummer gebruikt om van deze cd alle gegevens te tonen. Hiervoor is *root.childNodes.item(nodenr).xml* in de code opgenomen. Het middenstuk zorgt ervoor dat er nog een element met commentaar wordt toegevoegd.

Voor DOM is het kenmerkend dat het hele document wordt geladen voordat er verder iets kan worden uitgevoerd. In de code zien we dit duidelijk terug. Bij het laden van de pagina wordt het opgegeven document eerst aan de variabele *xmlDoc* gekoppeld en in het geheugen geladen. In de rest van de code wordt deze variabele een aantal malen gebruikt om gegevens te tonen.

6.3 SAX

Een heel andere manier van verwerken is *Simple API for XML* (SAX). Met *DOM* vragen we elementen op uit het document dat in het geheugen staat. Bij SAX laden we niks in het geheugen, maar wordt de documentstructuur doorlopen. Elke keer als de parser een stukje XML herkent, bijvoorbeeld de start van een nieuw element, dan wordt een methode aangeroepen die specifiek is voor dit stukje code. De programmeur kan vervolgens zelf bepalen wat er binnen deze methode moet gebeuren.
Met SAX wordt het XML document dus beschouwd als een reeks gebeurtenissen. Het inlezen van delen van het document wordt wel een parsing event genoemd. De applicatie bepaalt vervolgens hoe het parsing event verwerkt moet worden. De applicatie kan besluiten iets met het gelezen element te doen, of om het te negeren en te wachten op het volgende parsing event.



SAX houdt geen rekening met de hiërarchie van elementen in een XML document. Het leest enkel elementen in. Deze aanpak heeft een aantal belangrijke voordelen. Er is minder geheugenruimte nodig. Alleen delen van elementen die worden ingelezen nemen geheugenruimte in.
Een voordeel van SAX is dus het efficiënte geheugengebruik. SAX is vooral handig om een beperkt deel van een XML Document in te lezen, en in dezelfde volgorde te verwerken. SAX is minder geschikt om verschillende delen van een document te combineren: omdat we altijd maar met een klein stukje XML tegelijk werken kunnen we geen gebruik maken van XPath expressies om te verwijzen naar andere delen van het document.
We hebben in de vorige paragraaf Javascript gebruikt om een document te verwerken. Voor het verwerken van een document met SAX zullen we nu een Java programma gebruiken. Ook hier geldt weer dat u de code niet hoeft te begrijpen. Wel zullen we bij de essentiële onderdelen kort bespreken wat er gebeurt.

In Java zorgt de main methode er voor dat het programa kan worden uitgevoerd. Daarna volgen er meestal Daarna volgen er meestal één of meerdere *Event handlers*. Hierin wordt bijvoorbeeld beschreven wat er gedaan moet worden bij de verschillende parse events. Zo kan er bijvoorbeeld een melding op het scherm getoond worden bij het begin en bij het einde van een document. Een voorbeeld hiervan is hieronder beschreven in de methoden *startDocument* en *endDocument.*

```
public void startDocument ()
{
    System.out.println("Begin document \n");
}

public void endDocument ()
{
    System.out.println("Eind document \n");
}
```

Met de methode *startElement* wordt aangegeven wat er moet gebeuren als er een nieuw element wordt ingelezen. In onderstaand voorbeeld wordt bij het aanroepen van de methode de naam van het element dat wordt ingelezen gekoppeld aan de parameter *elementNaam*

```
public void startElement (String uri, String naam,
                          String elementNaam, Attributes atts)
{
    if (elementNaam.equals ("cdtitel")|| elementNaam.equals ("artiest"))
       {System.out.print(elementNaam +":    ");
       toon = true  ;}
    if (elementNaam.equals ("titel"))
       {System.out.print(nr +":    ");
       toon = true  ;}

    if (elementNaam.equals ("cd"))
    {System.out.println("Begin " + elementNaam);}
}
```

Met behulp van de if regels wordt de code alleen uitgevoerd voor elementen met een opgegeven naam. Ook nu wordt er een tekst naar het scherm geschreven.
De methode *endElement* werkt op dezelfde wijze, alleen wordt deze methode uitgevoerd als de parser het einde van een element tegenkomt.

```
public void endElement (String uri, String naam, String elementNaam)
{
    if (elementNaam.equals ("cd"))
       {System.out.println("Eind " + elementNaam+ "\n");
       nr=1; }
    if (elementNaam.equals ("titel"))
       {nr++;}
}
```

Tenslotte wordt de methode *characters* gebruikt om de inhoud van het huidige element te tonen.

```
public void characters (char ch[], int start, int length)
{
    String karakters = new String(ch,start,length);
    if (toon) System.out.print(new String(ch, start, length)+"\n");
       {toon = false ;}
}
```

De tekstinhoud van een element kan dus alleen worden bereikt binnen de methode characters: zodra de methode *endElement* wordt uitgevoerd is de inhoud van het element niet langer beschikbaar.

6.4 Verschillen tussen DOM en SAX

Het verschil tussen DOM en SAX is de manier van verwerken. Bij DOM wordt altijd eerst de hele boomstructuur in gelezen, bij SAX onderdeel voor onderdeel. Het voordeel van DOM is dat je de boomstructuur van elementen kunt blijven gebruiken. Met een XPath expressie kan te allen tijde een element in de hiërarchie geselecteerd worden omdat elk element in het geheugen is opgeslagen. Bij lange documenten kan dit problemen opleveren. Omdat er te veel gegevens moeten worden ingelezen en opgeslagen wordt de verwerking traag.

SAX heeft natuurlijk ook nadelen. Ten eerste betekent de manier van werken dat er geen mogelijkheid is voor zogenaamde Random Access. Dat wil zeggen dat het niet mogelijk is om met een XPath expressie de inhoud van een bepaald element op te halen. Om de juiste informatie op te halen, moet SAX het hele document opnieuw doorzoeken. In dit soort gevallen is DOM veel handiger, omdat DOM het hele document in het geheugen houdt. Gerelateerd aan dit probleem is dat je geen complexe zoekacties kunt uitvoeren op een document. Je kunt dus moeilijk zeggen "geef me alle elementen X", omdat ook daarmee het hele document doorlopen moet worden. DOM implementaties zijn juist op dit soort acties berekend, en gaan hier dus efficiënt mee om.

6.5 XMLReader

Naast Dom en SAX is XMLReader ook een veel gebruikte parser. SAX en XMLReader komen erg overeen, maar gebruiken een omgekeerde aanpak. Met SAX bepaalt de parser wanneer iets gebeurd. De parser roept de applicatie aan als hij weer wat te verwerken heeft. Met XMLReader is dit juist andersom. De applicatie vraagt telkens om het volgende element. Deze aanpak geeft ontwikkelaars iets meer controle over het verwerken. Zo kunnen we de parser vertellen onder welke voorwaarden we het volgende element willen hebben. Zo kan dus eenvoudig een gedeelte van een document overgeslagen worden, zonder dat de code uit moet zoeken of het element wel relevant is. Uiteindelijk betekent dit dat de verwerking wat sneller is dan met SAX. Een ander voordeel is dat je met XMLReader meerdere documenten tegelijk kunt lezen, iets wat met SAX niet kan, omdat parsing events aan één document gekoppeld zijn.

Appendix A Opdrachten

Opdrachten hoofdstuk 2

Opgave 1.

Open het document *InterieurBestelling.xml.* Dit document is niet well-formed.
Pas het document aan zodat het document well-formed is.

Opgave 2.

Maak een nieuw XML document.
Het document moet de volgende opbouw krijgen:

Het root element moet *cdLijst* heten.
Van een cd moeten de volgende zaken geregistreerd worden:

- titel van de cd
- de artiest
- de totale speelduur als attribuut van cd
- een tracklist met de afzonderlijke tracks van de cd
- Van elke track moet het volgende geregistreerd worden:
    - titel van de track
    - jaartal
    - speelduur als attribuut van track

Voeg minimaal 3 albums toe met elk minimaal 3 tracks.
Zorg er voor dat het document well-formed is.
Sla het document op als cdlijst.xml.

A Opdrachten

A Opdrachten

Opdrachten hoofdstuk 4

Opgave 1.

Maak een DTD *CDLijst.dtd* waarmee het document *CDlijst1.xml* gevalideerd kan worden.
Zorg dat er minimaal 1 cd in het document moet staan en dat er geen maximum aan het aantal cd's wordt gesteld.
De titel van een cd moet verplicht worden ingevuld.
Ook de artiest moet verplicht worden opgegeven.
Van de tracks moet de titel en het jaartal verplicht worden ingevuld.
Zorg ervoor dat het document *CDlijst1.xml* met dit DTD gevalideerd kan worden.

Opgave 2.

Bij de oefenbestanden vindt u het schema *CDLijst.xsd*

In dit schema wordt verwezen naar het SimpleType *jaar* en het SimpleType *tijd*.
Deze typen ontbreken nog in het schema. Defineer zelf deze 2 typen.
Zorg ervoor dat:

- Het jaartal altijd als een getal met 4 cijfers wordt ingevuld.
- Speelduur bestaat uit een aantal minuten en seconden in de vorm van mm:ss.
  Er moeten 2 cijfers voor de seconden gegeven zijn en voor de minuten maximaal 2 en minimaal 1cijfer.

Zorg ervoor dat het document met *CDLijst2.xml* met het schema gevalideerd kan worden.

(Optioneel)
Zorg er ook voor dat het bestand *CDlijst.xml* dat in hoofdstuk 2 is gemaakt met dit schema gevalideerd kan worden.

Op de volgende pagina staat nog een opdracht.

A Opdrachten

Opgave 3. (Optioneel)

Maak een nieuw schema *klanten.xsd.*
Defineer een type persoon.

Het is de bedoeling dat met dit schema een klantenlijst gevalideerd kan worden.

- De klantenlijst kan bestaan uit 1 of meerdere klanten.
- Voor elke klant moet een attribuut klantid worden toegevoegd.
- Een klant kan een particuliere klant (klantp) of een bedrijf of organisatie (klantb) zijn.
- Voor een particuliere klant worden de persoonsgegevens en adresgegevens geregistreerd.
- Van een bedrijf of organisatie moet de bedrijfsnaam, een adres en een contactpersoon worden geregistreerd.
- Verder moet van elke klant minimaal 1 telefoonnummer worden vastgelegd.
- Ook moet van elke klant minimaal 1 e-mailadres geregistreerd worden.

Definieer een type *persoon* dat gebruikt kan worden bij een particuliere klant en bij een contactpersoon.

Persoon moet de volgende kenmerken hebben:

- Voornamen
- Voorletters
- Tussen voegsel
- Achternaam
- Geslacht

Definieer een type *adres.*
Neem daarin deze gegevens op:

- Straat
- Huisnummer
- Toevoeging (als attribuut van huisnummer)
- Postcode
- Plaats

Maak een type geslacht zodat bij geslacht alleen gekozen kan worden uit:
man, vrouw of onbekend.

Voeg een element kortingspercentage toe
Bij dit element moet gekozen worden uit 4 verschillende percentages:
0, 5, 7.5 of 10.

Zorg ervoor dat bij postcode alleen een combinatie van 4 cijfers een spatie en 2 hoofdletters kan worden ingevoerd. Bovendien mag een postcode niet met een 0 beginnen.

Maak vervolgens een XML document *klanten.xml.*
Voeg hier minimaal 2 klanten aan toe: zowel een particuliere klant als een bedrijf.
Zorg ervoor dat *klanten.xml* met het schema gevalideerd kan worden.

A Opdrachten

Opdrachten hoofdstuk 5

1. Opgave 1.

De groothandel gebruikt een ander productID dan de webwinkel.
Pas het document *webbestelling.xsl* aan zodat de omzetting geautomatiseerd wordt.

B-431 wordt ws-434
B-432 wordt th-111
B-433 wordt rg-212
B-434 wordt wm-212

Als een product niet in de opgeven lijst voor komt moet er ??-??? komen te staan.

Opgave 2.

In de oefenbestanden vindt u een schema *nummers.xsd*
Maak een XSLT stylesheet dat het document *CDlijst1.xml* transformeert naar een nieuw XML document. Dit document moet de naam *nummers1.xml* krijgen.
Het nieuwe document moet gevalideerd kunnen worden met *nummers.xsd.*

Opgave 3. (Optioneel)

De cd's uit *CDLijst1.xml* moet overzichtelijk getoond worden in een HTML pagina.
Het overzicht moet er als volgt uit zien:

- Boven aan de tekst "CD lijst" in een nette opmaak
- Van de cd's worden de volgende gegevens in een tabel getoond:

  - Artiest
  - CDtitel
  - Speelduur
  - Jaartal (van de eerste track van de CD)

- Maak zelf tabelkoppen aan
- De achtergrond van de rijen moeten om en om in twee verschillende tinten groen worden getoond.
- Gebruik Xpath expressies om de juiste gegevens op te vragen.
- Geef zelf eventueel andere passende opmaak.

Voer de transformatie uit en sla het resultaat op als *CDLijst1.html*
Voer de transformatie ook uit voor *CDlijst2.xml* en sla het resultaat op als *CDlijst2.html.*

A Opdrachten

UITWERKINGEN

XML introductie

De uitwerkingen ontvangt u van de coach,
na het afronden van deze cursus.

UITWERKINGEN

XML Introductie

Revisie : 1.3

Opdrachten hoofdstuk 2

Opgave 1.

Open het document *InterieurBestelling.xml*. Dit document is niet well-formed.
Pas het document aan zodat het document well-formed is.

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

<bestelling>
     <product kleur="chroom" kleurbkleding="zwart/antraciet">
       <productID>99EK23</productID>
       <omschrijving>eetkamer stoel thom</omschrijving>
       <prijs valuta ="€">74.95</prijs>
       <aantal>6</aantal>
     </product>
     <product kleur="chroom" kleurblad="antraciet">
       <productID>99EK15</productID>
       <omschrijving>eetkamer tafel thom</omschrijving>
       <prijs valuta ="€">245.50</prijs>
       <aantal>1</aantal>
     </product>
     <product kleur="donkerblauw" type="2 zits">
       <productID>27WK23</productID>
       <omschrijving></omschrijving>
       <prijs valuta ="€">424.50</prijs>
       <aantal>1</aantal>
     </product>
     <product kleur="donkerblauw" type="2 zits">
       <productID>27WK23</productID>
       <omschrijving>bankstel rimboe</omschrijving>
       <prijs valuta ="€">724.50</prijs>
       <aantal>1</aantal>
     </product>
     <product kleur="donkerblauw" type="4 zits hoek">
       <productID>27WK23</productID>
       <omschrijving>bankstel rimboe</omschrijving>
       <prijs valuta ="€">1134.50</prijs>
       <aantal>1</aantal>
     </product>
</bestelling>
```

Opgave 2.

Maak een nieuw XML document.
Het document moet de volgende opbouw krijgen:

Het root element moet *cdLijst* heten.
Van een cd moeten de volgende zaken geregistreerd worden:

- titel van de cd
- de artiest
- de totale speelduur als attribuut van cd
- een tracklist met de afzonderlijke tracks van de cd
- Van elke track moet het volgende geregistreerd worden:
    - titel van de track
    - jaartal
    - speelduur als attribuut van track

Voeg minimaal 3 albums toe met elk minimaal 3 tracks.
Zorg ervoor dat het document well-formed is.
Sla het document op als cdlijst.xml.

Opdrachten hoofdstuk 2

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

<!DOCTYPE cdlijst SYSTEM "file:///H:/XMLbestanden/finished/CDLijst.dtd">

<cdlijst>
     <cd speelduur="51:58">
      <cdtitel>Funhouse</cdtitel>
      <artiest>Pink</artiest>
      <tracklist>
       <track speelduur="3:35">
        <titel>So What</titel>
        <nummer>1</nummer>
        <jaar>2008</jaar>
       </track>
       <track speelduur="4:11">
        <titel>Sober</titel>
        <nummer>2</nummer>
        <jaar>2008</jaar>
       </track>
       <track speelduur="4:36">
        <titel>I Don't Believe You</titel>
        <nummer>3</nummer>
        <jaar>2008</jaar>
       </track>
      </tracklist>
     </cd>
     <cd speelduur="40:00">
      <cdtitel>The Turn Of A friendly Card</cdtitel>
      <artiest>The allan Parsons Project</artiest>
      <tracklist>
       <track speelduur="4:57">
        <titel>May Be A Price To Pay</titel>
        <nummer>1</nummer>
        <jaar>1980</jaar>
       </track>
       <track speelduur="4:21">
        <titel>Games People Play</titel>
        <nummer>2</nummer>
        <jaar>1980</jaar>
       </track>
       <track speelduur="5:02">
        <titel>Time</titel>
        <nummer>3</nummer>
        <jaar>1980</jaar>
       </track>
      </tracklist>
     </cd>
     <cd speelduur="76:00">
      <cdtitel>Not Of This World</cdtitel>
      <artiest>Pendragon</artiest>
      <tracklist>
       <track speelduur="9:24">
        <titel>If I Where the wind</titel>
        <nummer>1</nummer>
        <jaar>2001</jaar>
       </track>
       <track speelduur="11:39">
        <titel>Dance Of the Seven Veils</titel>
        <nummer>2</nummer>
        <jaar>2001</jaar>
       </track>
       <track speelduur="16:24">
        <titel>Not Of This World</titel>
        <nummer>3</nummer>
        <jaar>2001</jaar>
       </track>
      </tracklist>
     </cd>
</cdlijst>
```

Opdrachten hoofdstuk 4

Opgave 1.

Maak een DTD *CDLijst.dtd* waarmee het document *CDlijst1.xml* gevalideerd kan worden.
Zorg dat er minimaal 1 cd in het document moet staan en dat er geen maximum aan het aantal cd's wordt gesteld.
De titel van een cd moet verplicht worden ingevuld.
Ook de artiest moet verplicht worden opgegeven.
Van de tracks moet de titel en het jaartal verplicht worden ingevuld.
Zorg ervoor dat het document *CDlijst1.xml* met dit DTD gevalideerd kan worden.

CDLijst.dtd

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


<!ELEMENT cdlijst (cd+)>
<!ELEMENT cd (cdtitel, artiest, tracklist)>
<!ELEMENT cdtitel (#PCDATA)>
<!ELEMENT artiest (#PCDATA)>
<!ELEMENT tracklist (track+)>
<!ELEMENT track (titel,nummer,jaar)>
<!ATTLIST cd speelduur CDATA #IMPLIED>
<!ATTLIST track speelduur  CDATA #REQUIRED>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT nummer (#PCDATA)>
<!ELEMENT jaar (#PCDATA)>
```

Aan *CDLijst1.xml* moet een regel met een Doctype worden toegevoegd.

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

<!DOCTYPE cdlijst SYSTEM "file:///H:/XMLbestanden/CDLijst.dtd">


<cdlijst xmlns="http://www.cd.nl">
  <cd speelduur="51:58">
    <cdtitel>Funhouse</cdtitel>
    <artiest>Pink</artiest>
    <tracklist>
...
```

Opgave 2.

Bij de oefenbestanden vindt u het schema *CDLijst.xsd*

In dit schema wordt verwezen naar het SimpleType *jaar* en het SimpleType *tijd*.
Deze typen ontbreken nog in het schema. Difineer zelf deze 2 typen.
Zorg ervoor dat:

- Het jaar altijd als een getal met 4 cijfers wordt ingevuld.
- Speelduur bestaat uit een aantal minuten en seconden in de vorm van mm:ss.
  Er moeten 2 cijfers voor de seconden gegeven zijn en voor de minuten maximaal 2 en minimaal 1cijfer.

Zorg ervoor dat het document met *CDLijst2.xml* met het schema gevalideerd kan worden.

(Optioneel)
Zorg er ook voor dat het bestand *CDlijst.xml* dat in hoofdstuk 2 is gemaakt met dit schema gevalideerd kan worden.

Opdrachten hoofdstuk 4

CDLijst.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
     elementFormDefault="qualified"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
     http://www.w3.org/2001/XMLSchema.xsd"
     xmlns="http://www.cd.nl"
     targetNamespace="http://www.cd.nl">


     <xsd:simpleType name="tijd">
       <xsd:restriction base="xsd:string">
         <xsd:pattern value="[0-9]?[0-9]:[0-5][0-9]"/>
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:simpleType name="jaartal">
       <xsd:restriction base="xsd:int">
         <xsd:pattern value="[0-9]{4}"/>
       </xsd:restriction>
     </xsd:simpleType>

     <xsd:complexType name="cdbeschrijving">
       <xsd:sequence>
         <xsd:element name="cdtitel" type="xsd:string"/>
         <xsd:element name="artiest" type="xsd:string"/>
         <xsd:element name="tracklist">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="track" type="tracks" maxOccurs="unbounded" />
             </xsd:sequence>
           </xsd:complexType>
         </xsd:element>
       </xsd:sequence>
     </xsd:complexType>


     <xsd:complexType name="tracks">
       <xsd:complexContent>
         <xsd:extension base="trackbeschrijving">
           <xsd:attribute name="speelduur" type="tijd" use="required"/>
         </xsd:extension>
       </xsd:complexContent>
     </xsd:complexType>

     <xsd:complexType name="trackbeschrijving">
       <xsd:sequence>
         <xsd:element name="titel" type="xsd:string"/>
         <xsd:element name="nummer" type="xsd:int"/>
         <xsd:element name="jaar" type="jaartal"/>
       </xsd:sequence>
     </xsd:complexType>

     <xsd:element name="cdlijst">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="cd" minOccurs="1" maxOccurs="unbounded">
             <xsd:complexType>
               <xsd:complexContent>
                 <xsd:extension base="cdbeschrijving">
                   <xsd:attribute name="speelduur" type="tijd" use="optional"/>
                 </xsd:extension>
               </xsd:complexContent>
             </xsd:complexType>
           </xsd:element>
         </xsd:sequence>
       </xsd:complexType>
     </xsd:element>

</xsd:schema>
```

Aanpassing in *CDLijst2.xml:*
In het element cdlijst moet de schemaverwijzing worden opgenomen.

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


<cdlijst xmlns="http://www.cd.nl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.cd.nl
      file:///H:/XMLbestanden/CDLijst.xsd"
      xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">
```

Dezelfde aanpassing moet ook in *CDLijst.xml* worden opgenomen.

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


<cdlijst xmlns="http://www.cd.nl"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.cd.nl
      file:///H:/XMLbestanden/CDLijst.xsd"
      xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional">
```

Opgave 3. (Optioneel)

Maak een nieuw schema *klanten.xsd.*
Definieer een type persoon.

Het is de bedoeling dat met dit schema een klantenlijst gevalideerd kan worden.

- De klantenlijst kan bestaan uit 1 of meerdere klanten.
- Voor elke klant moet een attribuut klantid worden toegevoegd.
- Een klant kan een particuliere klant (klantp) of een bedrijf of organisatie (klantb) zijn .
- Voor een particuliere klant worden de persoonsgegevens en adresgegevens geregistreerd.
- Van een bedrijf of organisatie moet de bedrijfsnaam, een adres en een contactpersoon worden geregistreerd.
- Verder moet van elke klant minimaal 1 telefoonnummer worden vastgelegd.
- Ook moet van elke klant minimaal 1 e-mailadres geregistreerd worden.

Definieer een type *persoon* dat gebruikt kan worden bij een particuliere klant en bij een contactpersoon. Persoon moet de volgende kenmerken hebben:

- Voornamen
- Voorletters
- Tussenvoegsel
- Achternaam
- Geslacht

Definieer een type *adres.*
Neem daarin deze gegevens op:

- Straat
- Huisnummer
- Toevoeging (als attribuut van huisnummer)
- Postcode
- Plaats

Maak een type geslacht zodat bij geslacht alleen gekozen kan worden uit:
man, vrouw of onbekend.

Opdrachten hoofdstuk 4

Voeg een element kortingspercentage toe.
Bij dit element moet gekozen worden uit 4 verschillende percentages:
0, 5, 7.5 of 10.

Zorg ervoor dat bij postcode alleen een combinatie van 4 cijfers een spatie en 2 hoofdletters kan worden ingevoerd. Bovendien mag een postcode niet met een 0 beginnen.

Maak vervolgens een XML document *klanten.xml*
Voeg hier minimaal 2 klanten aan toe: zowel een particuliere klant als een bedrijf.
Zorg ervoor dat *klanten.xml* met het schema gevalideerd kan worden.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema
elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
         http://www.w3.org/2001/XMLSchema
         http://www.w3.org/2001/XMLSchema.xsd"
xmlns="http://www.klanten.nl"
targetNamespace="http://www.klanten.nl">

<xsd:complexType name="klant">
<xsd:complexContent>
<xsd:extension base="klantElementen">
<xsd:attribute name="klantid" type="xsd:string"/>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="klantElementen">
<xsd:sequence>
<xsd:element name="telefoonnummer" type="xsd:int" minOccurs="1"
maxOccurs="1"/>
<xsd:element name="emailadres" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="klantp">
<xsd:complexContent>
<xsd:extension base="klant">
<xsd:sequence>
<xsd:element name="persoonsgegevens" type="persoon"/>
<xsd:element name="adresgegevens" type="adres"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="klantb">
<xsd:complexContent>
<xsd:extension base="klant">
<xsd:sequence>
<xsd:element name="bedrijfsnaam" type="xsd:string"/>
<xsd:element name="adresgegevens" type="adres"/>
<xsd:element name="contactpersoon" type="persoon"/>
</xsd:sequence>
</xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="persoon">
<xsd:sequence>
<xsd:element name="voornaam" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
<xsd:element name="voorletters" type="xsd:string"/>
<xsd:element name="tussenvoegsel" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="achternaam" type="xsd:string"/>
<xsd:element name="geslacht" type="sekse" minOccurs="0" maxOccurs="1"
default="onbekend"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="sekse">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="man"/>
<xsd:enumeration value="vrouw"/>
<xsd:enumeration value="onbekend"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="adres">
<xsd:sequence>
<xsd:element name="straat" type="xsd:string"/>
<xsd:element name="huisnummer">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="xsd:int">
<xsd:attribute name="toevoeging" type="xsd:string"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="postcode" type="pc"/>
<xsd:element name="plaats" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="pc">
<xsd:restriction base="xsd:string">
<xsd:pattern value="[1-9][0-9]{3}\s[A-Z]{2}"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="korting">
<xsd:restriction base="xsd:float">
<xsd:enumeration value="0"/>
<xsd:enumeration value="5"/>
<xsd:enumeration value="7.5"/>
<xsd:enumeration value="10"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:element name="klantenlijst">
<xsd:complexType>
<xsd:sequence>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element name="particulier" type="klantp"/>
<xsd:element name="bedrijf" type="klantb"/>
</xsd:choice>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

</xsd:schema>
```

Opdrachten hoofdstuk 4

Opdrachten hoofdstuk 5

Opgave 1.

De groothandel gebruikt een ander productID dan de webwinkel.
Pas het document *webbestelling.xsl* aan zodat de omzetting geautomatiseerd wordt.

B-431 wordt ws-434
B-432 wordt th-111
B-433 wordt rg-212
B-434 wordt wm-212

Als een product niet in de opgeven lijst voor komt moet er ??-??? komen te staan.

De code binnen het element *productid* wordt als volgt:

```
<productid>
  <xsl:choose>
    <xsl:when test="web:productid='B-431'">ws-434</xsl:when>
    <xsl:when test="web:productid='B-432'">th-111</xsl:when>
    <xsl:when test="web:productid='B-433'">rg-212</xsl:when>
    <xsl:when test="web:productid='B-434'">wm-212</xsl:when>
    <xsl:otherwise>??-???</xsl:otherwise>
  </xsl:choose>
</productid>
```

Opgave 2.

In de oefenbestanden vindt u een schema *nummers.xsd*
Maak een XSLT stylesheet dat het document *CDlijst1.xml* transformeert naar een nieuw XML document. Dit document moet de naam *nummers1.xml* krijgen.
Het nieuwe document moet gevalideerd kunnen worden met *nummers.xsd.*

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">

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

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

    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">

      <nummers xmlns="http://www.nummers.nl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.nummers.nl file:///H:/XMLbestanden/nummers.xsd">
        <xsl:for-each select="/cd:cdlijst/cd:cd">
          <xsl:for-each select="./cd:tracklist/cd:track">
            <nummer>
              <titel>
                <xsl:value-of select="./cd:titel"></xsl:value-of>
              </titel>
              <artiest>
                <xsl:value-of select="../../cd:artiest"></xsl:value-of>
              </artiest>
              <jaar>
                <xsl:value-of select="./cd:jaar"></xsl:value-of>
              </jaar>
              <duur>
                <xsl:value-of select="./@speelduur"></xsl:value-of>
              </duur>
            </nummer>
          </xsl:for-each>
        </xsl:for-each>
      </nummers>

    </xsl:template>
</xsl:stylesheet>
```

Opdrachten hoofdstuk 5

Aanpassing in *CDLijst1.xml*

Er moet in cdlijst1.xml nog een verwijzing naar de namespace www.cd.nl worden gemaakt.

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

<!DOCTYPE cdlijst SYSTEM
"file:///H:/XMLbestanden/finished/Hoofdstuk5/CDLijst.dtd">
<?xml-stylesheet type="text/css" href="cdlijst.css"?>


<cdlijst xmlns="http://www.cd.nl">
```

Opgave 3.

De cd's uit *CDLijst1.xml* moet overzichtelijk getoond worden in een HTML pagina.
Het overzicht moet er als volgt uit zien:

- Boven aan de tekst "CD lijst" in een nette opmaak.
- Van de cd's worden de volgende gegevens in een tabel getoond:
-
    - Artiest
    - CDtitel
    - Speelduur
    - Jaartal (van de eerste track van de CD)

- Maak zelf tabelkoppen aan
- De achtergrond van de rijen moeten om en om in twee verschillende tinten groen worden getoond.
- Gebruik Xpath expressies om de juiste gegevens op te vragen.
- Geef zelf eventueel andere passende opmaak.

Voer de transformatie uit en sla het resultaat op als *CDLijst1.html*
Voer de transformatie ook uit voor *CDlijst2.xml* en sla het resultaat op als *CDlijst2.html*.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">
    <xsl:output method="html"/>
    <xsl:template match="/">

     <html>
      <head>
       <title>cdlijst</title>
      </head>
      <body style="font-family: arial;">

       <h2 style="color:#003300;">CD overzicht:</h2>

       <table>
        <tr bgcolor="#003300" style="font-size:16pt; color:#99FF00;">
         <th>artiest</th>
         <th>titel</th>
         <th>speelduur</th>
         <th>jaar</th>
        </tr>
        <xsl:for-each select="/cd:cdlijst/cd:cd">
         <xsl:sort select="cd:artiest"/>
         <xsl:variable name="kleur">
          <xsl:choose>
           <xsl:when test="position() mod 2=0">#99FF00</xsl:when>
           <xsl:otherwise>#CCFF66</xsl:otherwise>
          </xsl:choose>
         </xsl:variable>
```

```
            <tr bgcolor="{$kleur}">
              <td><xsl:value-of select="./cd:artiest"/></td>
              <td><xsl:value-of select="./cd:cdtitel"/></td>
              <td align="right"><xsl:value-of select="./@speelduur"/></td>
              <td align="right"><xsl:value-of
select="./cd:tracklist/cd:track[1]/cd:jaar"/></td>
            </tr>

          </xsl:for-each>
        </table>
      </body>
    </html>

  </xsl:template>
</xsl:stylesheet>
```

Opdrachten hoofdstuk 5

INDEX