## XML

### Inleiding

XML staat voor e**X**tensible **M**arkup **L**anguage.

Bij XML gaat het om data en beschrijving van die data. Deze beschrijving verloopt via zogenaamde **markup-tags**: een manier om meta-informatie in data te stoppen die beschrijft wat die data is. Aan de data worden **tags** toegevoegd die meer informatie geven over de data, die dus semantiek (betekenis) toevoegen aan de data. Door deze tags toe te voegen creëer je **data-containers**. Elke tag geeft het begin of einde van een nieuwe data-container aan. In XML gaat dat op de volgende manier.

<naam>jopie bakhuis</naam>

Tags worden omgeven door scherpe haakjes (<tag>) net als in b.v HTML.

NB: XML is case-sensitive: <naam> is dus niet hetzelfde als <Naam>!!!

XML is een **meta-taal om markup talen te definiëren**. In XML kun je eigen tags definiëren, zodat je zelf kunt kiezen welke semantiek je toekent aan je data. Deze tags kunnen in principe eender wat zijn in eender welke taal. Door een hele set van tags te definiëren - en de mogelijke relaties tussen de tags - kun je een nieuwe markup taal maken.

XML-tags beschrijven de **inhoud** van een document, niet de vorm. Dit is in tegenstelling tot HTML waar de tags naast de inhoud (vaak) ook de vorm van het document beschrijven.

Een XML-document is gestructureerd als een boom, met de wortel ("root") bovenaan en de takken naar beneden. Bijvoorbeeld:

<nawgegevens>

<naam>Saskia van Tilburg</naam>

<adres>Hof van Hoorn 27</adres>

<plaats>Amsterdam</plaats>

</nawgegevens>

XML-documenten kunnen “**well-formed**” zijn en “**valide**”.

Als het document well-formed is, wil dat zeggen dat het aan de minimale eis voldoet en dat is aan de regels van XML. Dit legt geen eisen op aan de inhoud van de data.

Als het document valide is, worden er ook eisen opgelegd aan de inhoud van het document. Dit komt verderop terug.

Het is mogelijk, maar weinig zinvol, om XML ‘geïsoleerd’ te gebruiken. Het krijgt pas zijn kracht als we voorafgaand aan XML-gebruik eerst een structuurdefinitie vastleggen via XML Schemas of via DTD’s en een check (laten) doen om te controleren of het XML-datadeel wel voldoet aan de via een XML Schema of een DTD vastgelegde grammatica.

Als het XML de check doorstaat, is het XML document valide.

Pas dan kunnen we eigenlijk met vertouwen aan de slag om via een speciale XML-querytaal relevante gegevens op te vragen en/of op een speciale manier te presenteren. Voor het presenteren kan o.a. gebruik gemaakt worden van XSL (‘eXtensible Stylesheet Language’). Dat komt de volgende weken aan bod.

### XML: ‘eXtensible Markup Language’

XML wordt gebruikt om zowel documenten als gegevens te modelleren/structureren. Daarbij moeten we via een ‘open’- en een ‘sluit’-tag aangeven wat onze **XML-elementen** zijn en hoe die zijn genest.

We lezen in de Help-functie van (een oudere versie van) XML-Notepad:

*“An XML document is a tree-like structure. This tree consists of nodes that possess a hierarchical relationship to each other. To promote or demote a node is to raise or lower that node to a different level within that hierarchy.”*

Schematisch zal een XML-model er steeds als volgt uitzien:

<root>

<child>

<subchild>

</subchild>

</child>

</root>

Elk ‘kind’-element kan weer een of meer ‘sub-kind’-elementen bevatten.

Zo zou je in XML de gegevens bij een verenigingslid als volgt kunnen modelleren:

<lid>

<naam>

<voornaam>Jan</voornaam>

<achternaam>Janssen</achternaam>

</naam>

<adres>Annastraat 123</adres>

<plaats>Nijmegen</plaats>

</lid>

Duidelijk is hier te zien, dat een lid-element blijkbaar een ‘naam’-, een ‘adres’- en een ‘plaats’-onderdeel kent, waarvan het ‘naam’-deel kan worden beschouwd als een samengesteld element met een voornaam- en een achternaam-subkind, terwijl ‘adres’ en ‘plaats’ als directe kind–elementen worden beschouwd.

De data die in XML beschreven wordt, hoeft overigens niet persé altijd zo sterk hiërarchisch gestructureerd te zijn, maar kan ook best meer verhalend van karakter zijn. Kijk maar eens naar het onderstaande voorbeeld:

<biography>

<paragraph>

<name><first\_name>Alex</first\_name> <last\_name>Turing</last\_name></name>

was one of the first people to truly deserve the name <emphasize>computer

scientist</emphasize>. Although his contributions to the fields are too

numerous to lst, his best-known are the famous <emphasize>Turing

Test</emphasize> and <emphasize>Turing Machine</emphasize>.

</paragraph>

<definition>The <term>Turing Test</term> is to this day the standard test

for determining whether a computer is truly intelligent. This test yet

has to be passed.</definition>

<definition>A <term>Turing Machine</term> is an abstract finite state

automaton with infinite memory that can be proven equivalent to any other

finite state automaton with arbitrarily large memory. Thus what is true

for one Turing machine is true for all Turing machines no matter how

implemented.</definition>

<paragraph>

<name><last\_name>Turing</name></last\_name> was also an accomplished

<profession>mathematician</profession> and

<profession>cryptographer</profession>. His assistance was crucial in

helping the Allies decode the German Enigma cipher. He committed suicide

on <date><month>June</month> <day>7</day>, <year>1954</year></date>

after being convicted of homosexuality and forced to take female hormone

injections.

</paragraph>

</ biography >

Dit lijkt nauwelijks meer op een echte boomstructuur, maar het is ook geldige XML (mits correct vormgegeven natuurlijk: zie tussenopdracht).

Behalve met een *nesting* van (kind)elementen is het bij XML ook mogelijk om met **attributen** te werken. In het algemeen wordt aanbevolen om de data – XML draait immers om data – in elementen te stoppen, en niet (zozeer) in attributen. Attributen worden vooral gebruikt voor het geven van ‘aanvullende informatie’ bij een bepaald element. Meestal gaat het dan om gegevens die voor een menselijke gebruiker niet zo interessant zijn, maar voor software wel. Ook een vorm van metadata dus …

Deze metadata kan b.v. ook nodig zijn om de gegevens (efficiënt) door een computer te laten verwerken. Denk hierbij aan zaken als identificerende lidnummers, codes, e.d..

Attributen worden als volgt in de openings-tag van het element opgenomen:

<Elementnaam attribuutnaam = “attribuutwaarde”>

Let op de spatie tussen ‘Elementnaam’ en ‘attribuutnaam’ en op de (verplichte!) dubbele aanhalingstekens rond de “attribuutwaarde”.

Attributen moeten uniek zijn binnen een element, maar de volgorde waarin ze opgesomd worden, mag wisselen.

Hieronder is eerst een schermafdruk weergegeven van de freeware XML-Notepad-2007 editor, waarmee we het daarna weergegeven XML-document hebben gemaakt (alle tags zijn gegenereerd!). XML Notepad 2007 Blackboard te downloaden, maar ook op internet.

In dit voorbeeld is ervoor gekozen het ‘lidnummer’ als attribuutwaarde bij ‘LID’ te nemen.



<LEDEN>

<LID lidnummer="12345">

<Naam>

<Voornaam>Jan</Voornaam>

<Achternaam>Janssen</Achternaam>

</Naam>

<Adres>Annastraat 123</Adres>

<Plaats>Nijmegen</Plaats>

</LID>

<LID lidnummer="13579">

<Naam>

<Voornaam>Els</Voornaam>

<Achternaam>Pietersen</Achternaam>

</Naam>

<Adres>Molenweg 33</Adres>

<Plaats>Nijmegen</Plaats>

</LID>

</LEDEN>

Uit dit voorbeeld komt duidelijk het altijd benodigde ‘root’-element naar voren (hier: onder de naam ‘LEDEN’). Let in de treeview-weergave aan de linkerkant van het schermdeel op het gebruik van de verschillende symbolen voor elementen, attributen en hun waardes.

Als we het XML-bestand openen via Internet Explorer, dan toont deze het volgende:



Het ziet er niet erg WWW-appetijtelijk uit, maar dat is ook niet de bedoeling van een kaal XML-bestand (het fatsoenlijk tonen van XML-data komt bij XSLT nog terug). In ieder geval zijn (de huidige versies van) *Internet Explorer en Firefox* blijkbaar in staat de XML-code te interpreteren. Voor elk (sub)element staat een ‘+’ of een ‘-‘ teken om aan te geven dat de waarden en/of dieper geneste inhoud wel of niet aangegeven is. Met een eenvoudige muisklik kun je ‘+’ en ‘ -‘ in elkaar laten overgaan.

Je kunt zien dat als een LID-element dichtgeklapt is, van dat LID alleen de attribuutwaarde getoond wordt. Als zo’n element méér attribuutwaarden heeft, worden attribuutnamen en -waarden achter elkaar in de <LID>openingstag opgenomen; bijvoorbeeld:

<LID lidnummer ="12345" sofinummer ="4334343433">

We kunnen (b.v. met behulp van een eenvoudige editor als NotePad(PlusPlus)) handmatig informatie aan het XML-bestand toevoegen. Stel dat we onderstaande informatie toevoegen, en we doen dit vlak voor de laatste regel in het bestand (</LEDEN>):

<LID kleur = "bruin" >

<Wielen>2</Wielen>

</LID>

En nu laden we de inhoud van het gewijzigde bestand in IE of een andere XML-geschikte Web-browser, dan worden de gegevens van dat vreemde derde ‘lid’ zonder protest getoond. (Onderstaande weergave komt uit FF.)



IE/FF toont de XML, omdat deze well-formed is, maar maakt zich bij het tonen ervan niet druk of er wel ‘valide’ informatie in het XML-bestand staat.

IE/FF beschikt over een zogenaamde parser, die in staat is om de inhoud van het XML-bestand te doorgronden en maakt voor de weergave van het XML-bestand gebruik van een eigen, ‘intern’ stylesheet.

Voor meer informatie over hoe IE omgaat met XML, zie de volgende URL:

<http://www.sgml-ug.nl/HTML/frame_artikel/frame_element/artikelen/software/explorer5.htm>.

Let op dat de informatie al enigszins gedateerd is, maar toch nog steeds wel behoorlijk goed bruikbaar.

We kunnen via XML blijkbaar wel een bestaande structuur van gegevens duidelijk aangeven, maar niet de gegevensstructuur modelleren, noch de te respecteren ‘business rules’ afdwingen.

## Tussenopdracht 1

Maak met NotePad(Pluslus) een XML bestand waarin alle(!) onderstaande gegevens in XML-formaat worden toegevoegd. Maak dus gebruik van elementen en attributen en zorg dat de naam is opgebouwd uit voornaam, familienaam, bijnaam en dat de geboortedatum is opgebouwd uit dag, maand, jaar. Voorlopig géén lege elementen opnemen.

Mijn adresboekske:

Joske Vermeulen: geboren 19/05/1986, tel. 016/589652, e-mail: joske@vermeulen.com

Davy Bosmans, a.k.a. “den Bossie”: tel 016/321954

Bekijk dit XML-bestand vervolgens met IE, met FF én met XML Notepad 2007. Voeg in XML Notepad nog een 3e persoon toe, bewaar je bestand en bekijk met NotePad(PlusPlus) welke XML-code dat oplevert.

NB: Voor NotePadPlusPlus is een plugin “XML Tools” beschikbaar die je een hoop extra features oplevert die je de komende weken kunt gebruiken!

## Tussenopdracht 2

Maak met NotePad(PlusPlus) een XML bestand aan met de naam Turing.XML waarin je de op bladzijde 2/3 genoemde biografie van Turing opneemt door bovenstaande tekst te knippen en te plakken en op te slaan. Laad het XML-bestand in de browser en je zult constateren dat de XML niet well-formed is! Los de problemen op, net zo lang tot je well-formed XML hebt.

## XML Schema

**XML Schema’s** worden gebruikt om de integriteit van een XML document te verifiëren. Voorheen was dit alleen mogelijk m.b.v. een DTD, een Document Type Definition. Helaas hadden DTD’s hun beperkingen, vandaar de overstap naar XML schema’s. DTD’s worden verderop nog behandeld.

Met XML Schema’s kun je hetzelfde als met DTD’s en daarnaast is er nogal wat functionaliteit toegevoegd om onderhoud en uitbreidbaarheid te bevorderen, zoals datatyping en overerving. Bovendien worden XML Schema’s in XML gemaakt, terwijl DTD’s een aparte ‘taal’ gebruikten.

Een Schema is een blauwdruk van een specifiek XML-document. Met een Schema kunnen verschillende regels worden afgedwongen voor een document, zoals:

* Welke types elementen en attributen zijn toegestaan in het document?
* Welke attributen mogen voorkomen in een document?
* Welke elementen zijn kindelementen zijn?
* enz.

Als je eenmaal een XML Schema hebt, kun je vervolgens door zgn. *validators* laten controleren of een XML-bestand voldoet aan de definitie die in het bijbehorende XML Schema is vastgelegd.

Om je XML-document te laten verifiëren m.b.t. het bijbehorende XML schema bestand, staan op Blackboard twee bruikbare tools: zie [Documents][Tools].

* AltovaXML is een offline, commandline tools waar we nog meer mee zullen werken
* De andere is een online validatie-tool van XML Me. Daarbij is het handig dat je je XML en je Schema naar een textbox kunt kopiëren en vervolgens valideren, mar ook dat je wijzigingen kunt aanbrengen en direct kunt hertesten.

Dit zijn overigens maar 2 voorbeelden uit een veel groter aanbod van (al dan niet gratis) XML-tools!

## Functie en gebruik van XML Schema

Om enkele codevoorbeelden van XML Schema’s te verduidelijken gebruiken we het volgende XML bestand:

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

<character>

<name>Luke Skywalker</name>

<species>Human</species>

<language>Basic</language>

<home>Tatooine</home>

</character>

De XML Schema specificatie maakt een onderscheid tussen “simpleType” en “complexType” elementen. Simpele elementen kunnen geen andere elementen bevatten en geen attributen. Complexe elementen kunnen dat wel. Zodra je dus een beetje ‘serieus’ element gebruikt, zul je met complexe elementdefinities te maken krijgen.

In een schema worden deze twee elementtypes vertegenwoordigd door de <xsd:simpleType> en <xsd:complexType> statements.

De gemakkelijkste manier om een simple element in een schema weer te geven is door gebruik te maken van de <xsd:element> declaratie samen met een ingebouwd datatype.

Het volgende simple element

<name>Luke Skywalker</name>

wordt in een schema op de volgende manier weergegeven:

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">;

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

</xsd:schema>

Je ziet in de bovenstaande stukjes code de termen “**xmlns**” en “**xsd**” voorkomen.

Xmlns staat voor “**XML NameSpace**” en een namespace is niet veel anders dan een collectie (space) van namen (names). Namespaces zijn o.a. bedoeld om ‘dingen’ een naam te geven en binnen verschillende collecties dezelfde namen te kunnen gebruiken. Xsd is een zelf te bedenken prefix die je voor de namen zet die je uit een namespace wilt gebruiken, en daarmee kun je dus onderscheid maken tussen evt. zelfde namen uit verschillende namespaces. Overigens is xsd zelf een veel gebruikte prefix.

Ook in programmeertalen als C# en Java komen namespaces voor, ook al heten ze soms anders (bv packages), en de bedoeling ervan is hetzelfde: voorkomen dat twee definities uit verschillende omgevingen, en dus met verschillende betekenissen, voor problemen zorgt. Het begrip ‘bank’ uit de financiële wereld is tenslotte een ander begrip als een ‘bank’ uit de meubelwereld, en als je als makelaar of verhuurder een gebouw wil beschrijven dat dienst kan doen als geldbank, inclusief het meubilair, dan heb je beide begrippen nodig, toch?

Namespaces zullen we verderop nog meer tegenkomen, maar spelen qua diepgang geen al te grote rol in deze module. Wel moet je weten dat de namespace uit het voorbeeld (die officieel te vinden is op "<http://www.w3.org/2001/XMLSchema>") door het W3C-consortium is vastgesteld en alle informatie over o.a. standaard datatypes bevat. Wanneer de datatype naam wordt voorafgegaan door de “xsd” prefix die hierboven is gedefinieerd, dan betekent dit dat het een voorgedefineerd datatype betreft en dus geen user-defined type.

*PS: De gebruikte namespace heeft de vorm van een URL, maar er vindt geen connectie met het internet plaats. Als je hier meer over wilt weten, bekijk dan b.v. een van de documenten die je op Blackboard onder XML/Validatie kunt vinden.*

De XML Schema specificatie heeft ongeveer 40 verschillende ingebouwde datatypen zoals “string”, “integer”, “float”, “boolean”, “time”, enz. Als je meer wilt weten over de ingebouwde datatypes, dan kun je ze opzoeken op

<http://www.w3.org/TR/xmlschema-0/>.

Als we het hele XML-bestand erbij pakken, zien we dat “character” een complex element is en dat complex elementen dus ook weer simple en andere complex elementen kunnen bevatten.

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

<character>

<name>Luke Skywalker</name>

<species>Human</species>

<language>Basic</language>

<home>Tatooine</home>

</character>

Wanneer complexe typen worden gedefinieerd, kan de definitie op twee manieren plaatsvinden. De eerste manier is dat je een **xsd:complexType** declareert en hiervan later een element creëert:

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!−− define a new datatype for a complex element −−>

**<xsd:complexType name="starWarsEntity">**

**<xsd:sequence>**

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

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

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

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

**</xsd:sequence>**

**</xsd:complexType>**

<!−− create an element of this type −−>

**<xsd:element name="character" type="starWarsEntity"/>**

</xsd:schema>

De tweede manier combineert de twee stappen in een stap. De definitie van het complex element is dan ingebed in de **<xsd:element>** declaratie:

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

**<xsd:element name="character">**

<!−− this complex element definition has no name, and is

referred to as an "anonymous" element −−>

**<xsd:complexType>**

<xsd:sequence>

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

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

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

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

</xsd:sequence>

**</xsd:complexType>**

**</xsd:element>**

</xsd:schema>

Het voordeel van de eerste methode is dat deze op meer plaatsen in het schema gebruikt kan worden door simpel naar de naam te refereren.

Het **<xsd:sequence>** statement wordt natuurlijk gebruikt om te verwijzen naar de kind elementen van het complex element.

Het is ook mogelijk om complexe elementen te nesten. Ook dit gaat vrij eenvoudig. Allereerst weer een voorbeeld XML bestand.

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

<gallery>

<character>

<name>Luke Skywalker</name>

<species>Human</species>

<language>Basic</language>

<home>Tatooine</home>

</character>

<character>

<name>Chewbacca</name>

<species>Wookie</species>

<language>Shyriiwook</language>

<home>Kashyyyk</home>

</character>

<character>

<name>Chief Chirpa</name>

<species>Ewok</species>

<language>Ewok</language>

<home>Endor</home>

</character>

</gallery>

Zoals je ziet bevat “gallery” meerdere complexe elementen van het type “character”.

Allereerst zullen we character dus in een definitie omzetten, en vervolgens een referentie naar character opnemen in de definitie van gallery.

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="starWarsEntity">

<xsd:sequence>

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

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

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

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

</xsd:sequence>

</xsd:complexType>

<!−− define the root element and its contents −−>

**<xsd:element name="gallery">**

<xsd:complexType>

**<xsd:sequence>**

**<xsd:element name="character" type="starWarsEntity"**

**maxOccurs="unbounded"/>**

**</xsd:sequence>**

</xsd:complexType>

**</xsd:element>**

</xsd:schema>

Het attribuut **maxOccurs** dient ervoor om het maximum aantal instanties te specificeren van character. De tegenhanger is **minOccurs**. Deze twee attributen staan default op 1, wat wil zeggen dat precies één voorkomen wordt verwacht.

Een mooie functie van schema’s is dat je ook gebruik kunt maken van **overerving**, d.w.z. dat een vorm van OO (Object Oriëntatie) beschikbaar is. Je maakt dan gebruik van het **<xsd:extension>** element. Hier kijken we volgende week verder naar, evenals naar de mogelijkheid om de definitie van elementen in losse bestanden op te nemen, zodat je die makkelijk op meerdere plekken kunt hergebruiken.

## DTD: Document Type Definition

Een **DTD (Document Type Definition)** is een andere, al wat ’oudere’ manier om de definitie van XML-data mee vast te leggen. Je komt het echter nog vaak tegen; denk b.v. aan de DOCTYPEs van (X)HTML-bestanden waarin verwezen wordt naar DTD’s! Daarom behandelen we ze hier wel, zij het kort.

Een DTD bestaat uit een formele set van regels waaraan een XML-document moet voldoen. Het definieert de elementen die gebruikt mogen worden en legt vast waar ze in relatie tot elkaar gebruikt mogen worden.

Je kunt een DTD beschouwen als een soort (tekstueel) conceptueel schema van de datastructuur en de bijbehorende constraints. De modelleringmogelijkheden via DTD zijn echter beperkter dan die van een XML Schema.

Een DTD bevat een aantal declaraties die alle aan de volgende opmaakstructuur: <! . . . . . . > moeten voldoen en waarin [onmiddellijk na het uitroepteken] een van de volgende sleutelwoorden moet staan:

* ELEMENT (voor de element-‘tag’ definities)
* ATTLIST (idem de attribuut definities)
* ENTITY (entiteit definities; zie verderop)
* NOTATION (datatype notation definities; zie verderop)

Deze declaraties worden gegroepeerd en geplaatst binnen een document type declaration die de volgende structuur heeft:

<!DOCTYPE doctypeNAAM [

<!ELEMENT doctypeNAAM (...en hier dan 1e deel van de DTD-definitie...) >

<!...en verder...>

<!...en verder...>

] >

Een XML-document hoeft voor de specificatie van zijn structuur niet persé een bijbehorende DTD te hebben. Toch wordt het opstellen van zo’n DTD vaak aanbevolen – tenzij je gebruik maakt van een XML Schema natuurlijk - om zekerheid af te dwingen over het naleven van de gedefinieerde structuur. Een speciale DTD/XML-parser (een zgn. validator) kan worden gebruikt om eerst de DTD te lezen en daarna te bepalen of het XML-document wel aan dat DTD-sjabloon voldoet. AltovaXML is daar een voorbeeld van.

### DTD-ELEMENT-definities

**ELEMENTs** zijn de primaire eenheden om XML-documenten mee op te bouwen.

De ‘tag’ definities via ELEMENT kunnen ‘nonterminal’ of ‘ terminal’ zijn, dat wil zeggen: er kan mee aangegeven worden, dat een bepaald ELEMENT op zijn beurt verder opgebouwd is uit een of meer andere ELEMENTs, maar ook, dat zo’n verdere opsplitsing niet gebeurt. Dat laatste geval (dus bij ‘terminal’) wordt meestal gebruikt om het gegevenstype ‘**PCDATA’** van dat ELEMENT aan te geven. PCDATA staat voor ‘ parsable character data’, een [getal]string e.d., waarin opmaaktekens als ‘<’,’>’, ‘”’, ‘&’ e.d. als (html -) markup zullen worden geïnterpreteerd.

Laten we even uitgaan van de volgende XML:

<!-- voorbeeld invulling : (dit is een commentaarregel) -->

<BOEK>

<Titel>Over XML</Titel>

<Auteur>Jansen, Jan</Auteur>

<Auteur>Pietersen, Els</Auteur>

<Hoofdstuk>

<Titel>1. Inleiding</Titel>

<Inhoud>

<Paragraaf>Hier staat de inhoud van de eerste paragraaf</Paragraaf>

<Paragraaf>En hier staat de inhoud van de tweede paragraaf</Paragraaf>

</Inhoud>

</Hoofdstuk>

<Hoofdstuk>

<Titel>2. Titel van het tweede hoofdstuk</Titel>

<Inhoud>

<Paragraaf>Inhoud paragraaf 2.1</Paragraaf>

<Paragraaf>Inhoud paragraaf 2.2</Paragraaf>

<Paragraaf>Inhoud paragraaf 2.3</Paragraaf>

</Inhoud>

</Hoofdstuk>

</BOEK>

Voor zo’n voorbeeld, dat een (hele verzameling) boek(en) beschrijft, kan de volgende DTD gegeven worden:

<!DOCTYPE BOEK [

<!ELEMENT BOEK ( Titel, Auteur+, Hoofdstuk+ )>

<!ELEMENT Hoofdstuk ( Titel, Inhoud )>

<!ELEMENT Inhoud ( Paragraaf+ )>

<!ELEMENT Titel (#PCDATA)>

<!ELEMENT Auteur (#PCDATA)>

<!ELEMENT Paragraaf (#PCDATA)>

]>

Een voorbeeld van een ‘nonterminal’ ELEMENT-declaratie hierin is :

<!ELEMENT Hoofdstuk ( Titel, Inhoud )>

Titel en Inhoud worden immers verderop in de DTD verder uitgewerkt als ook weer bestaand uit andere elementen.

De komma’s die in zo’n DTD-regel voorkomen geven aan dat de daarbij gebruikte volgorde van subitems moet worden aangehouden.

Een voorbeeld van een ‘terminal’ ELEMENT-declaratie is:

<!ELEMENT Auteur (#PCDATA)>

PS: Naast #PCDATA komen ook de aanduidingen #CDATA, #EMPTY en #ANY voor.

Via het ‘|’-symbool kunnen we keuzes aangeven. Kijk b.v. naar de volgende regel:

<!ELEMENT Dessert ( ijs | fruit ) >

Dit geeft aan dat Dessert ofwel ijs ofwel fruit is.

Indien bepaalde onderdelen vaker kunnen voorkomen en/of optioneel zijn, kunnen we als ‘*occurrence indicator’* de volgende symbolen gebruiken:

|  |  |
| --- | --- |
| *Symbool* | *Beschrijving* |
| + | Een [verplicht!] element kan een of meer keren voorkomen (>=1) |
| \* | Een [optioneel!] element kan nul of meer keren voorkomen (>=0) |
| ? | Een [optioneel!] element kan hoogstens een keer voorkomen (0, 1) |

### Externe DTD’s

Omdat DTD’s meestal als een ‘standaard’ voor meerdere XML-documenten worden gemaakt, bestaat ook de mogelijkheid zo’n DTD-standaard in een apart bestand op te slaan en er *vanuit relevante (XML-)documenten (!)* als volgt naar te verwijzen (let op het gebruik van het woord *SYSTEM*):

<!DOCTYPE BOEK SYSTEM "C:/IS0/XML/BOEK.DTD" > [

<!... met hier eventuele aanvullende definities voor dit ene document... >

<!....... > ] >

NB1: Bovenstaande met opgenomen worden in het XML-bestand!

NB2: Er moet dan géén ‘<!DOCTYPE BOEK ... ’>-regel meer in dat aparte DTD-bestand staan!

### DTD-ATTLIST-definities

Naast elementen moeten we ook attributen kunnen definiëren. Dit gebeurt m.b.v. een **ATTLIST**. Hiermee kunnen we aangeven dat een ELEMENT een of meer attributen kan hebben, die overigens in een andere volgorde mogen voorkomen als in die ATTLIST staat. Het is niet verplicht dat een ELEMENT attributen heeft.

De syntax voor een ATTLIST-declaratie is:

<!ATTLIST element-naam attribuut-naam attribuut-type ‘default-value’ >

Bij het eerder gegeven voorbeeld <!ELEMENT BOEK (Titel, Auteur+, Hoofdstuk+ )> zou je als attribuutlijst kunnen hebben:

<!ATTLIST BOEK isbnnr CDATA #REQUIRED >

Attributen kunnen *vereist* (#REQUIRED ), *optioneel* (#IMPLIED ) of ‘*vast’* ( #FIXED) zijn (zgn. attribute defaults). Een van deze drie mogelijke values moet ingevuld worden.

Let er op, dat deze attribute defaults wél voorafgegaan worden door een ‘#’, terwijl de eraan voorafgaande type-aanduiding (hier: CDATA) géén ‘#’ heeft.

Let er ook op, dat bij definiëring van een ATTLIST onmiddellijk nà het woord ATTLIST steeds eerst de naam van het XML-element genoemd wordt en pas daarna de naam van het attribuut.

Daarnaast bestaan een viertal z.g. ‘*tokenized attribute types*’: ID, IDREF, ENTITY en NMTOKEN. Hiermee kan de opsteller van de DTD de toegestane waarden voor een bepaald attribuut regelen:

* ID moet een unieke identifier voor het bewuste ELEMENT zijn.
* Attributen van het type IDREF verwijzen naar een ELEMENT met een ID-attribuut; elke IDREF-waarde moet naar een gelijke ID waarde in het XML-document verwijzen.

Door gebruik te maken van ID-attributen en IDREF-attributen kunnen we resp. primary en foreign keys uit de database-wereld simuleren. Via een IDREF-attribuut is een soort foreign key-verwijzing te simuleren, zij het dat zowel de ID- als de ernaar verwijzende IDREF-attribuutwaarde in hetzelfde XML-document moeten voorkomen.

Uit het boek ‘XML, How to program’ van Deitel, Deitel, Nieto, Lin & Sadhu halen we het volgende voorbeeld met betrekking tot het gebruik van ID/IDREF (eerst de DTD en dan het XML-bestand):

<?xml version = "1.0"?>

<!-- Example for ID and IDREF values of attributes -->

<!DOCTYPE bookstore [

<!ELEMENT bookstore ( shipping+, book+ )>

<!ELEMENT shipping ( duration )>

<!ATTLIST shipping shipID ID #REQUIRED>

<!ELEMENT book ( #PCDATA ) >

<!ATTLIST book shippedBy IDREF #IMPLIED>

<!ELEMENT duration ( #PCDATA ) >

] >

<bookstore>

<shipping shipID = "s1">

<duration>2 to 4 days</duration>

</shipping>

<shipping shipID = "s2">

<duration>1 day</duration>

</shipping>

<book shippedBy = "s2">

Java How to Program 3rd edition.

</book>

<book shippedBy = "s2">

C How to Program 3rd edition.

</book>

<book shippedBy = "s1">

C++ How to Program 3rd edition.

</book>

</bookstore>

Merk op dat in bovenstaande XML de DTD-definitie en de bijbehorende XML-data in 1 document zijn opgenomen.

Als je dit complete document opslaat op schijf, bv. als bookstore.XML, en dan door de AltovaXML validator haalt, dan zal deze melden dat de XML valid is.

Verander vervolgens eens bij het laatste book-element de shippedBy-attribuutwaarde in “s5” en valideer nogmaals. Dan verschijnt de melding:

*The XML data is invalid:A reference to undeclared ID 's5' has been found.*

Ook bij het gebruik van attributen is het mogelijk om via zgn. ‘enumerated attribute types’ een lijst van mogelijke waarden (enumeratie) aan te geven die een attribuut mag hebben. Dat kan iets zijn van de vorm:

<!ATTLIST person gender ( M | F ) “F” #REQUIRED>

waarbij “F” als default-waarde is aangegeven.

### DTD-ENTITY-definities

Een ‘entity’ is in XML -terminologie eigenlijk een ‘variabele’ waar een waarde aan toegekend kan worden en waarvan de waarde door referentie via ‘&’+entity-naam+’;’ opgevraagd kan worden.

Ook in HTML hebben we dat al toegepast: omdat in HTML opmaaktekens als ‘>’ niet rechtstreeks kunnen worden afgedrukt, zijn die tekens impliciet in entities/variabelen opgeslagen. Daarom kunnen we via HTML dat ‘>’-teken toch op het beeldscherm krijgen via de referentie &gt; .

Als voorbeeld van het gebruik van een ENTITY-definitie geven we hier:

<!DOCTYPE EntityProbeersel [

<!ELEMENT EntityProbeersel ( #PCDATA ) >

<!ENTITY variabelenaam "haar hond Tarzan" >

]>

<EntityProbeersel>

Els en &variabelenaam; wandelen iedere ochtend door het park!

</EntityProbeersel>

Als we dit bestand in IE of FF laden, dan zien we dat de substitutie heeft plaatsgevonden:



Door gebruik te maken van de ‘SYSTEM’-optie is het ook mogelijk de waarde voor een ENTITY uit een ander bestand te laten ophalen. Dat kan via een constructie als:

<!ENTITY variabelenaam SYSTEM "D:/is0/xml/Varwaarde.txt" >

### Huiswerkopgave voor week 1

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

**Deelopdracht 1:**

1. Schrijf een XML-schema bestand voor het LEDEN-bestand uit tussenopdracht 1. Zorg ervoor dat alle personen uit het adresboekske/LEDEN-bestand een ID-attribuut hebben dat numeriek is (integer).
2. Valideer het XML-bestand t.o.v. dit schema m.b.v. de AltovaXml-tool die je van BB kunt afhalen.
3. Zorg ook (indien je dat niet nog niet hebt gezien ☺) voor fouten in je XML-bestand, zodat je de reactie van de validator ziet bij schendingen tegen het schema. Je moet er bij het nakijken enkele kunnen laten zien en ze kunnen verklaren!

Kijk ook in de slides van week 1 en op BB bij de andere documenten voor aanvullende informatie, want met de wat beknopte info in dit document kom je er niet helemaal!

**Deelopdracht 2:**

Herhaal deelopdracht 1 maar voor het Turing.XML bestand!

**Deelopdracht 3:**

Herhaal deelopdracht 1 nogmaals, maar maak nu een DTD voor het LEDEN-bestand!

Ook deze moet je mbv de AltovaXML tool kunnen valideren, en speel ook hier wat met fouten!

**Algemene criteria:**

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