Introduktion till XML
Övningar

Mikael Gunnarsson

24 oktober 2017

Följande övningar har reviderats och ompaketerats under hösten 2017, men baseras på tidigare beprövade övningar som har ca 15 år ”på nacken”.

I materialet ges informationsrutor med en blåaktig bakgrund. De finns där som extra kommentarer på basis av vanliga misstag som vi fått erfara genom åren.

1 Inledning : välformad XML

Den här inledande uppgiften går ut på att du skapar en fil som utgör ett välformat (*well-formed*) xml-dokument. Innehållet som utgör exempel är en receptsamling, vilken som helst, emedan en sådan har en relativt enkel men väl förutsägbar struktur. Det finns dock ingenting som hindrar att du i stället väljer att göra en bibliografi, en lista över dina CD-skivor, de filmer som din favoritregissör ligger bakom, eller en adresslista över dina vänner.

Syftet med denna inledande övning är alltså att du skall bli bekant med vad som är karaktäristiskt för ett välbyggt (eller *välformat*) dokument. Begreppet “välformade dokument” konstituerar tillsammans med begreppet “*giltiga* dokument” kärnan i XML-uppmärkningen som verksamhet.

I övningarna kan då och då ges referenser till HTML, endast för jämförelser, eftersom det brukar vara någon som är bekant med HTML eller XHTML sedan tidigare. Om du inte är det, så strunta i dem.

Efter de inledande övningarna kommer du sedan att skapa en *DTD* (Document Type Definition) som *definierar* ett xml-dokument. Det du åstadkommer i den här första övningen är endast en innehållslig struktur och det står dig egentligen fritt att välja vilken struktur du vill, inom vissa gränser.

Varje välformat xml-dokument kan läsas av de flesta webbläsare, även om innehållet utan en definition av dess presentationsformat är relativt ointressant. Kom ihåg att för det ändamålet bör du företrädesvis arbeta i FIREFOX. Övriga webbläsare kan fungera, men uppföra sig lite oväntat.

Ett xml-dokument (en *dokumentinstans* i XML-termer) kan skapas i vilken enkel textredigerare som helst, förutsatt att den kan spara vad du gör med standardiserad teckenkodning — **UTF-8**.

> *Lämpliga rena texteditorer är t ex* NOTEPAD++ *och* EMACS.
> *Ordbehandlingsprogram (som* MS WORD*) ska undvikas.*

Innan du fortsätter ska du hämta hem lite arbetsmaterial som du behöver, från `https://github.com/Mikael61/DH`. Välj från den gröna knappen `Clone or download` och ladda ner allt innehåll som en zip-fil. Packa därefter upp den lokalt med lämpligt uppackningsprogram. Se till så att du vet var i det lokala filsystemet du sparar och packar upp arkivet.

Börja med att öppna ett nytt fönster[1] i OXYGEN (eller en annan dedikerad XML editor, t ex EDITIX) och spara det i en ny mapp med t ex namnet `xmltest.xml`. Om du får möjlighet att välja filtyp så välj `XML document`. Du kan naturligtvis döpa filen till vad du vill, men den ska ha extensionen `.xml` (Använd inte mellanslag. Använd något av tecknen a–z, A–Z, 0-9 eller understreck).

> *Att ge filer och mappar namn som är robusta nog för att inte skapa problem av olika slag är synnerligen viktigt, inte bara i samband med XML. Bokstäver med diakriter ska undvikas så långt det är möjligt, liksom olika skiljetecken och mellanslag. Om du behöver beskrivande filnamn så använd gärna s k "camel case" (`mittDokument.txt`) eller understreck (`mitt_dokument.txt`).*

Det första du nu ska göra är att ange att det är fråga om ett xml-dokument. Om inte editorn redan lagt till det[2] så gör du det genom att i det än så länge tomma "bladet" skriva:

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

[1] Vanligen `File` -> `New`

[2] Du får aldrig ha flera xml-deklarationer.

Tänk på att XML gör skillnad mellan gemener och versaler, så använd alltid samma kast i märkena och var noggrann när du skriver av instruktioner. Värdet för encoding är dock inte kastkänsligt (*case sensitive*). **Inga tecken får förekomma *före* xml-deklarationen, inte ens mellanslag eller tomma rader.**

Ett xml-dokument måste per definition vara välformat, vilket i praktiken innebär att de program som ska läsa filen (t ex en webbläsare) inte accepterar några felaktigheter i koden. De kontrollerar alltså syntaxen i ditt xml-dokument och om dokumentet inte är välbyggt kan de ge dig indikationer på att något är fel. Titta därför lite extra noga på de eventuella felmeddelanden du i så fall får.

Prova med att öppna filen i FIREFOX genom att först starta den och därefter välja att öppna filen från menyn.

Du har nu angett att dokumentet följer W3Cs regler för XML 1.0 och att teckenkodningen följer den ISO-standard som kallas utf-8. Teckenuppsättningen inkluderar stöd för å, ä och ö. Av tekniska skäl skall du emellertid undvika användningen av å, ä och ö i xml-märkena.

1.1 Innehåll

Nu skall du fylla på med innehållet och dess struktur. Som övningsexempel väljer vi att skapa en receptsamling. Du kan kalla elementen för vad du vill, så länge namnen inte innehåller några mellanslag. Undvik även punkt (.), understreck (_) och siffror först i element-namnet, samt språkspecifika tecken ö h t (t ex å, ä och ö).

I en av de filer du laddade ner tidigare (`recept.txt`) finns ett färdigt recept i ren textform. Öppna, studera det och fundera först över av vilka komponenter det består (små och stora, fristående och sammanhängande med andra).

Det första vi gör är att skapa ett sk *rotelement*. Det är motsvarigheten till det html-element som omger ett HTML-dokuments innehåll, och även här skall startmärket ligga först i dokumentet (men efter xml-deklarationen) och slutmärket ligga sist.

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

Låt oss anta att vi bestämt oss för att varje recept kan bestå av följande element: {ingredienser, procedur, ingrediens}. En möjlig lösning med utgångspunkt från vårt

exempelrecept är som följer. Gör samma sak och sök förstå hur OXYGEN på olika sätt försöker hjälpa dig när du skriver namn på elementen.

Givetvis kan du kopiera och klistra in, men en poäng här är att bli vän med hur OXYGEN fungerar (eller din alternativa xml-editor) så gör inte det.

```
<?xml version="1.0" encoding="utf-8"?>
<receptsamling>
    <recept>
        <namn>Surströmmingslåda</namn>
        <ingredienser>
            <ingrediens>
                12 surströmmingsfiléer
            </ingrediens>
            <ingrediens>
                2 gula lökar
            </ingrediens>
            <ingrediens>
                2 dl lätt grädde
            </ingrediens>
            <ingrediens>
               0,5 dl ströbröd
            </ingrediens>
            <ingrediens>
              2 msk margarin
            </ingrediens>
        </ingredienser>
        <procedur>Sätt ugnen på 200 grader.
            Varva filéerna med hackad lök i
            en smord ugnsform. Strö över ströbrödet,
            klicka ut margarinet och slå på grädden.</procedur>
        <procedur>Grädda i ugn i 25-30 minuter.</procedur>
        <procedur>Servera med färsk potatis,
            tomater, gurka, tunnbröd, smör,
            en god ost, messmör och eventuellt
            hackad gul lök.</procedur>
    </recept>
</receptsamling>
```

Som du ser så omges (de flesta element) i XML med ett start- och ett slutmärke. Startmärket *öppnar* ett element och slutmärket *stänger* det. (Många kallar märken för taggar")

*'Märke' och 'tagg' är alltså synonymer, men 'märke' (eller tagg) och 'element' är **inte** synonymer.*

Fortsätt nu med att lägga till minst ett recept till. Kom ihåg att bevara dokumentets välformadhet i fråga om att ett och endast ett *rotelement* får finnas.

Var noga med att "nästla" (*nest*) start- och slutmärken på ett korrekt sätt när du skriver dina xml-dokument, eftersom de annars inte kommer att accepteras av t ex en webbläsare — dokumentet är då inte längre välformat. Det är egentligen ganska svårt att i OXYGEN göra fel i dessa avseenden, men tvingas du tillfälligt använda en vanlig texteditor är det lätt hänt.

Välj nu ut ett eller ett par element som kan delas upp och specificeras ytterligare, t ex `ingrediens`. Fundera också lite kring vad man kan vinna på att så detaljerat som möjligt beskriva varje del av receptet.

Vi tänker oss nu att vi funnit att vi vill ge en högre granularitet åt ingredienserna. Vi väljer att skilja mellan ingrediensens benämning och den mängd av ingrediensen som fordras av receptet, och samtidigt introducerar vi ett "attribut" som vi kallar `enhet` för ett element `maengd`.

```
<maengd enhet='antal'>12</maengd>
```

*Notera att `enhet` är attributets **namn** och `antal` dess **värde**, samt att attributet och dess värde endast läggs till det öppnande märket. `</maengd>` är tillräckligt för att stänga det element som har egenskapen att ha ett innehåll som är en måttenhet av den typ som anges.*

Tomma element konstrueras ofta utan slutmärke. Tomma element blir aktuella då du vill ha element som kanske ska beteckna en viss bestämd punkt i ett dokument, t ex en radbrytning eller en sidbrytning. Ett annat exempel är när du vill referera till en digital bild som lagras i en extern fil (elementet `img` i HTML är ett kardinalexempel).

`<nyrad></nyrad>` eller `<nyrad/>` är två ekvivalenta sätt att hantera tomma element på, där det senare är det vanligaste.

Observera att just 'nyrad' tycks presentationsorienterat och därför borde anses bryta mot grundtanken bakom XML. Radbrytningar och sidbrytningar är meningslösa i XML-koden (de ska åstadkommas med stilmallar), om de inte används i syfte att beskriva *fysiska objekt, som gamla manuskript, vilket är fallet i samband med t ex TEI.*

Observera också att ett element som är definierat som tomt (EMPTY) inte få ha något som helst innehåll, inte ens ett mellanslag.

Fundera på om du kan komma på ett lämpligt tomt element att infoga i ditt dokument! Om inte, så hoppa över det och ta bara med dig att möjligheten finns.

Det är möjligt att du kommit fram till någonting liknande det följande.

```
<?xml version="1.0" encoding="utf-8"?>
<receptsamling>
    <recept>
        <namn>Surströmmingslåda</namn>
       <ingredienser>
            <ingrediens>
                <namn>surströmmingsfilé</namn>
                <maengd enhet="antal">12</maengd>
            </ingrediens>
            <ingrediens>
                <namn>gul lök</namn>
                <maengd enhet="antal">2</maengd>
            </ingrediens>
            <ingrediens>
                <namn>lätt grädde</namn>
                <maengd enhet="dl">2</maengd>
            </ingrediens>
            <ingrediens>
                <namn>ströbröd</namn>
                <maengd enhet="dl">0,5</maengd>
            </ingrediens>
            <ingrediens>
                <namn>margarin</namn>
                <maengd enhet="msk">2</maengd>
            </ingrediens>
        </ingredienser>
        <procedur>Sätt ugnen på 200 grader.
            Varva filéerna med hackad lök i
            en smord ugnsform. Strö över ströbrödet,
            klicka ut margarinet och slå på grädden.</procedur>
        <procedur>Grädda i ugn i 25-30 minuter.</procedur>
        <procedur>Servera med färsk potatis,
```

```
            tomater, gurka, tunnbröd, smör,
            en god ost, messmör och eventuellt
            hackad gul lök.</procedur>
    </recept>
</receptsamling>
```

1.2 Syntaxkontroll

Så länge du använder en XML-editor med realtidsvalidering[3] så blir slutresultatet troligen korrekt.

Det kan dock hända att du befinner dig på resande fot och inte har tillgång till en sådan men ändå behöver redigera något. Där är då två huvudsakliga metoder som står till buds.

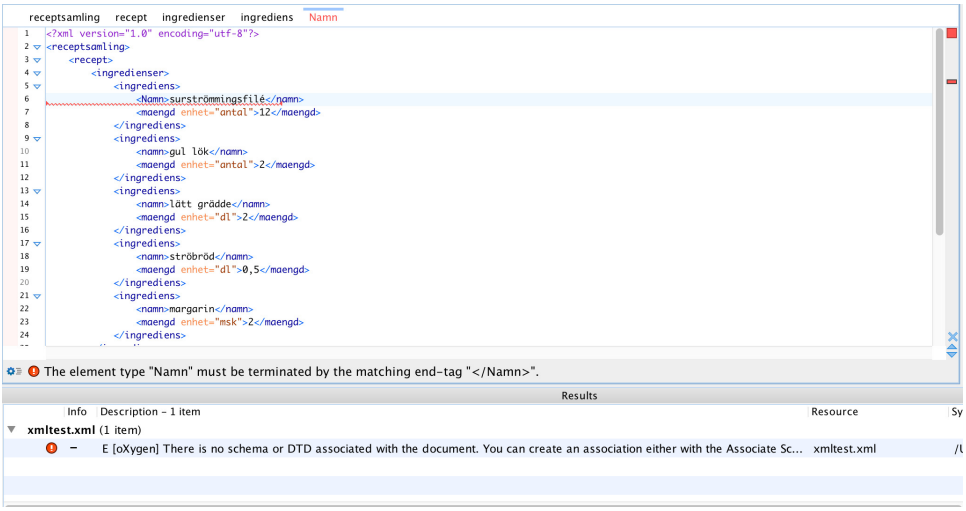
1. **Öppna xml-filen i en webbläsare som kan tolka (“parsa”) ren XML.**
   Starta de webbläsare du har tillgängliga på din arbetsstation och öppna din xml-fil i var och en av dem. Jämför de olika sätt på vilka du får innehållet presenterat.
   Introducera ett fel i din xml-fil. Skriv t ex ett öppnande märke fel, `Namn` i st f `namn`. (Se figur 1) Spara filen och ladda om filen i vardera webbläsaren.

2. **Använd en syntaxkontroll online.**
   Det har funnits och finns många tjänster för ändamålet. De “kommer och går”. `https://www.w3schools.com/xml/xml_validator.asp` är ett exempel

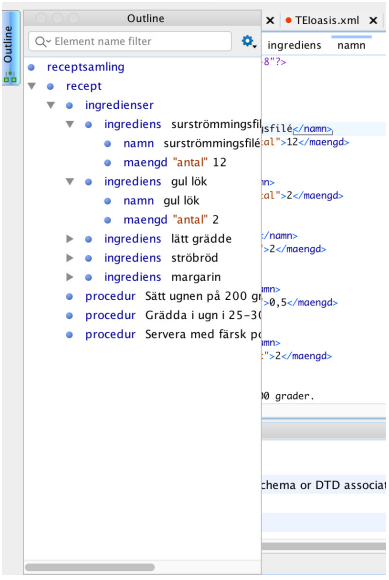
De flesta verktyg för att hantera XML erbjuder olika *vyer* gentemot det innehåll som redigeras. Somliga verktyg kan dölja uppmärkningen helt och hållet och göra det bedrägligt enkelt att redigera xml-kod. Det är vanligen ingenting att rekommendera om det handlar om att lära sig XML. En annan vy som kan vara användbar är att få en vy som tydliggör den trädstruktur som varje xml-dokument utgör. I OXYGEN kan den aktiveras via `Window`-menyn (`Show view` -> `Outline`) i version 18.0 och resultera i vad du ser i figur 2. [4]

[3] Dvs en editor som kontrollerar syntaxen under arbetet med filen och genererar varningar så snart något inte är korrekt. OXYGEN, EDITIX och XML SPEAR är exempel på sådana.

[4] De flesta editorer förändras mer eller mindre när de släpps i nya versioner, så det kan krävas annan hantering i din version.



Figur 1: Syntaxfel i OXYGEN



Figur 2: Strukturerad vy av innehållet som det visas i OXYGEN, med “outline” aktiverad

2 Dokumentmodeller — Document Type Definition

Så här långt har vi endast arbetat med att försöka skapa välformade xml-dokument. Vi ska nu övergå till att studera förutsättningarna för och poängen med *giltiga* (valida) xml-dokument.

Det är egentligen ganska poänglöst att använda XML för att strukturera text om det lämnas helt fritt att använda vilka element och attribut som helst. En huvudpoäng är att *begränsa* frihetsgraderna på ett sådant sätt att det för en viss tillämpning – t ex för beskrivning av en samling digitaliserade brev – är förutsägbart vilka element som kan förekomma och hur de får förekomma. På så vis ökar möjligheterna till samarbete och att skapa kringverktyg som kan behandla innehållet på olika sätt – t ex skapa en webbpresentation utifrån en samling xml-uppmärkta texter.

Det finns olika "formalismer" för ändamålet, varav *XML Schema* i dagsläget hör till den mest använda. XML scheman är dock lite för avancerade att introducera i nuläget och vi nöjer oss med en äldre formalism som är något enklare att hantera: DTDer (dokumenttypsdefinitioner).

I korthet, vad du skall göra när du skriver en DTD, är att *deklarera varje element och attribut* som du avser att använda i dina xml-dokument och bestämma *vad varje element får innehålla och eventuellt vilka värden ett attribut kan anta* — dvs, i xml-terminologi, en innehållsmodell (*content model*) för varje element.

Innan dess gäller det dock att noggrannt fundera över vilken förväntade struktur en samling av texter kan tänkas ha. Vi ska här ta ett exempel som med benäget tillstånd lånats av James Cummings (tidigare på Oxford Text Archive, nu vid Newcastle University). Exemplet finner du i det material du laddade ner från Github och packade upp. Exemplet består av ett brev scannat och återgivet i två bildfiler, `letter1.jpg` och `letter2.jpg`. Innehållet har transkriberats och återfinnes i filen `letter-to-LG.txt`. Vi tänker oss alltså att vi skapar en "grammatik" för beskrivning av en samling brevs innehåll.

Egentligen är det en mycket komplex process att skapa regler för uppmärkning av en viss texttyp. Det är många överväganden som behöver göras kring vilka element och attribut som behövs och hur de ska få användas, samt hur de ska benämnas. Inte minst behövs en överblick över de texters struktur som definitionen ska svara mot. Vanligen finns redan någon färdig tillämpning som kan användas rakt av eller utvidgas efter behov.

I det följande följer ett mycket tentativt förslag till grammatik.

Brevet innehåller a) ett brevhuvud, b) en hälsningsfras, c) ett antal stycken med radbrytningar, d) ett stycke poesi med rader, e) en avslutningsfras, f) en signatur, och g) en sidbrytning.

Vi antar att somliga av dessa delar inte kan förekomma varsomhelst i ett brev, att somliga inte alltid förekommer och att somliga alltid finns där.

Analysen är mycket grov, men vi nöjer oss så här långt.

2.1 Skapa DTDn

Precis som med xml-dokumenten så kan DTDer skapas med en enkel texteditor. OXYGEN ger oss dock lite hjälp.

Välj `New` från `File`-menyn, och därefter typen `Document Type Definition`. Du ges endast en xml-deklaration. Spara filen med extensionen `.dtd`, t ex `xmltest.dtd` i samma mapp som du har för avsikt att spara dina xml-filer.

Vi börjar lämpligen från "roten", själva brevet. Vi använder konstruktionen `<!ELEMENT >` för att deklarera ett element. Direkt efter ELEMENT skriver vi namnet på elementet och därefter en uppgift om vad det får innehålla (dvs innehållsmodellen).

```
<!ELEMENT brev ANY>
```

Nyckelordet `ANY` anger att elementet `brev` kan innehålla både text och andra element.

> *`ANY` är ingen bra innehållsmodell, därför att den inte begränsar innehållet alls. DTDer med endast `ANY` som innehållsmodeller är meningslösa. Använd helst inte alls `ANY`.*

Om ett element får innehålla andra element (dvs längre ner i trädstrukturen) anger man namnet på elementet/elementen inom parentes. Skall de förekomma i en särskild ordning använder man kommatecken mellan elementnamnen, annars särskiljs de med ett lodrätt streck (|). Ett tomt element har nyckelordet `EMPTY`.

Även om DTDer är enklare än XML Scheman så är notationen inte helt lätt att begripa sig på, och innehållsmodellerna kan bli väldigt komplexa och ogenom-

skinliga. Du behöver dock inte fördjupa dig i dem för de ändamål vi har här, men om du ändå vill det så kan W3Schools "tutorial" vara en hygglig startpunkt — https://www.w3schools.com/xml/xml_dtd_intro.asp

Vi har ovan räknat upp sju element som vi alla ska deklarera på motsvarande vis, men inte med ANY.

Vi har att bestämma vad följande element ska innehålla

```
<!ELEMENT brevhuvud      >
<!ELEMENT salutering     >
<!ELEMENT stycke         >
<!ELEMENT poesi          >
<!ELEMENT avslutningsfras    >
<!ELEMENT signatur       >
<!ELEMENT sidstart       >
```

Vi kan bestämma att vissa element endast får och kan innehålla text, på modellen <!ELEMENT salutering (#PCDATA)>, att vissa element kan innehålla både text och andra element, <!ELEMENT stycke (#PCDATA|sidstart|nyrad)*>, att vissa element endast kan innehålla en eller flera versrader, <!ELEMENT poesi (versrad)+>, och att vissa element är tomma, <!ELEMENT sidstart EMPTY>.

Notera dock att alla element måste vara definierade, och i exemplen ovan är inte versrad och nyrad definierad, vilket är nödvändigt.

Försök på egen hand att konstruera en full DTD innan du vänder på bladet och ser ett förslag till lösning.

```
<!ELEMENT brev ANY>
<!ELEMENT brevhuvud (#PCDATA)>
<!ELEMENT salutering (#PCDATA)>
<!ELEMENT stycke (#PCDATA|sidstart|nyrad)*>
<!ELEMENT poesi (versrad)+>
<!ELEMENT avslutningsfras (#PCDATA)>
<!ELEMENT signatur (#PCDATA)>
<!ELEMENT sidstart EMPTY>
<!ELEMENT nyrad EMPTY>
<!ELEMENT versrad (#PCDATA)>
<!ELEMENT brevsamling (brev)+>
```

Som du ser har det också lagts till ett ytterligare element `brevsamling`, eftersom det vore poänglöst att skapa en grammatik för endast ett brev.

Vi har också talat om attribut, och även sådana måste definieras. Exempelvis kan vi vilja säga vilken sida det är som startar vid en viss punkt genom ett attribut `sidnummer`.

```
<!ATTLIST sidstart sidnummer CDATA #REQUIRED>
```

Nyckelordet `CDATA` betyder att värdet måste vara ren text och `#REQUIRED` att det måste anges.

Du får inte deklarera element och attribut mer än en gång i en DTD.

Det finns egentligen mycket mer begränsningar som kan sättas upp av en DTD men vi nöjer oss så här långt. Poängen här är bara att du ska få en uppfattning av vad DTDer och XML Scheman gör, vilket är nästa steg.

2.2 Giltig XML

Skapa nu ett nytt dokument och välj `XML document` om du får den möjligheten. Utgå från följande “template”.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE brevsamling SYSTEM "xmltest.dtd">
<brevsamling>
    <brev>
    </brev>
</brevsamling>
```

Rad 2 innehåller här något du inte sett tidigare, en dokumenttypsdeklaration. Deklarationen knyter samman det här dokumentet med den grammatik du definierat i filen `xmltest.dtd`.

Det är först nu vi kan ställa oss frågan om dokumentet är giltigt eller inte, dvs om det följer de regler vi ställt upp. De flesta editorer känner nu av att det finns en grammatik (i DTD-form) och söker kontrollera innehållet mot vad som är tillåtet. Om det inte sker brukar ett sådant val kunna göras. I OXYGEN görs det via `Document -> Validate` i menyn. Om du provar det och inte får några felmeddelanden så är allt OK och du kan gå vidare.

> *Medan du arbetar, ignorera inte ett felmeddelande utan ta reda på vad det kan bero på. Under hand som du skriver kan felmeddelanden dyka upp som försvinner när du är klar med allt innehåll i ett element — men annars ska där inte finnas felmeddelanden.*

Din uppgift är nu att ta den transkriberade brevtexten och märka upp den utifrån den grammatik du skapat i `xmltest.dtd`. Ett effektivt sätt är att markera den text som utgör ett element, och välja `Surround with tag` från `Document -> Markup`, varvid du får möjligheten att välja märke från en lista på de element som finns i grammatiken. Det finns också kortkommandon så att du slipper använda menyerna — Cmd + e på en Mac t ex.

Delvis är det en rutinmässig sak att göra det (så om du vill kan du hoppa över att märka upp radbrytningarna för att spara tid). Däremot är det inte helt enkelt att avgöra var gränsen mellan stycken går. Det blir en tolkningsfråga.

När du är klar har det kanske blivit lite oordning på texten i källan och trädstrukturen blir svårare att se. Det finns i allmänhet en funktion för att formatera och indentera källkoden. Se om du kan hitta den.

I mappen `spoiler` finns i filen `brev.tex` en möjlig lösning på uppgiften, men låt den ligga tills du själv kommit fram till en lösning.

Om du har tid över, kan du återvända till exemplet med receptsamlingen (eller om du valde att göra något annat) och skapa en DTD för den på motsvarande vis som du gjort för brevet.

3 'XML is not the end' – transformation

> *Observera att det här avsnittet* ***måste*** *du ha* OXYGEN *för att genomföra. Givetvis finns det sätt att göra det på även i* EDITIX *eller* XML SPEAR*, men de är allt för komplicerade att gå igenom här.*
>
> *Du kan dock prova xsl ändå, genom att utgå ifrån* `brev.xml` *och* `brev.xsl` *i materialet från* GITHUB*, och undersöka hur du gör i* EDITIX *och/eller i* XML SPEAR *för att transformera xml-filen.*

Du har säkert undrat över vad vi ska med xml-filen till för "vanligt folk". Ingen vill väl läsa text inbäddad i xml-kod?

Nej, troligen inte. Poängen är förstås att en xml-källa skapar stor flexibilitet och gör att vi kan generera flera olika former av presentation. Den vanligaste tekniken för ändamålet är *XSL* som egentligen innefattar flera xml-tillämpningar.

Med hjälp av XSL kan rutiner skrivas för att omvandla xml-kodens element till presentationer i pdf- eller html-form, för att bara nämna två möjliga former.

OXYGEN är redan förberedd för att hjälpa till med det och distribueras med färdiga `stilmallar` för olika ändamål. De är dock, och måste vara, skrivna för en bestämd grammatik och kan inte användas om inte denna grammatik använts. OXYGEN har bland annat stöd för *TEI* som du antagligen kommer att stifta bekantskap med längre fram under utbildningen.

I mappen `spoiler` finns i filen `letter-to-LG.xml` "vårt" brev i färdigt TEI-kodat skick. Öppna den i OXYGEN och jämför gärna först med din egen tolkning av brevet.

Välj därefter `Document` -> `Transformation` -> `Configure Transformation Scenario(s)` och `TEI P5 XHTML`. Slutligen `Apply Associated`.

Om allt gått väl ska du nu ha fått en html-fil som öppnas i ett nytt webbläsarfönster. Resultaten skiljer sig en del från det förväntade och transformationen behöver egentligen några parametrar innan den körs, men du ser i alla fall ett derivat av din källa, som fortfarande är orörd.