Introduktion till XML

Övningar

Mikael Gunnarsson

19 oktober 2018

Följande övningar har anpassats särskilt för utbildningen i digital humaniora, men baseras på tidigare beprövade övningar.

Nytt för i år är att vi drar nytta av OXYGENs funktioner för att generera grammatiker, vilket också gör dem beroende av att du har tillgång till OXYGEN.

I materialet ges emellanåt informationsrutor med en lila-aktig bakgrund. De finns där bl a 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 *grammatik/dokumenttypsdefinition* 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) 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:

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

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

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

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 Innnehåll

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 inte får 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.

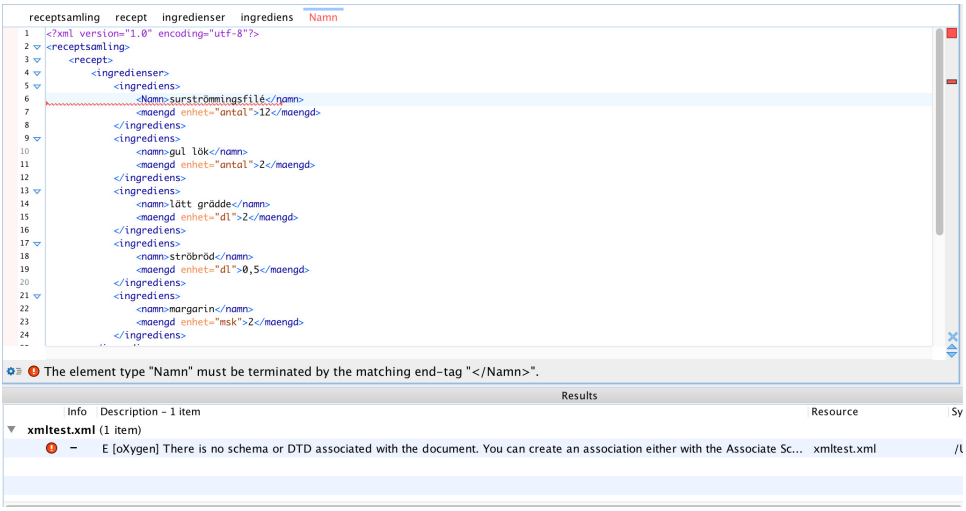
1. **Öppna xml-filen i en webbläsare som kan tolka (“parsa”) ren XML.**
   Starta de webbläsare du har tillgängliga på din arbetsstation och öppna din xml-fil i var och en av dem. Jämför de olika sätt på vilka du får innehållet presenterat.
   Introducera ett fel i din xml-fil. Skriv t ex ett öppnande märke fel, `Namn` i st f `namn`. (Se figur 1 på s. 10) 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

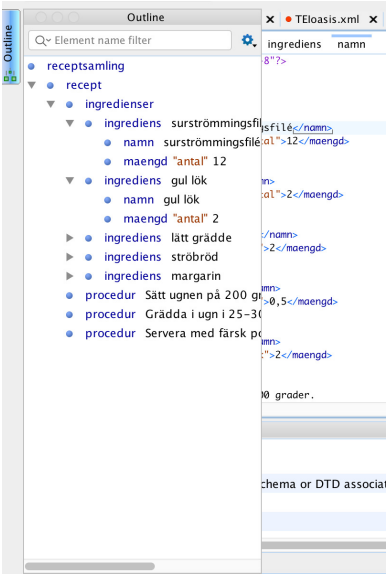
De flesta verktyg för att hantera XML erbjuder olika *vyer* gentemot det innehåll som redigeras. Somliga verktyg kan dölja uppmärkningen helt och hållet och göra det bedrägligt enkelt att redigera xml-kod. Det är vanligen ingenting att rekommendera om det handlar om att lära sig XML. En annan vy som 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, grammatiker — XML scheman

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 troligen hör till den mest använda. Att den är mer uttrycksfull och ger bättre kontroll är dock klart.

I korthet, vad du skall göra när du skriver en grammatik 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.

I det här sammanhanget ser vi inte det som nödvändigt att du lär dig att definiera en grammatik "från scratch", men det är viktigt att förstå vilken roll den spelar och ger också incitament till att närmare granska och värdera de xml-dokument du skapar.

Vi gör det därför enkelt och ska med utgångspunkt från en enkel xml-instans (vår receptsamling) utnyttja OXYGEN för att generera ett XML schema.

2.1 Generera en grammatik med OXYGEN

Se till att du har en version av receptsamlingen öppen i OXYGEN som är identisk med den på sidan 8. Kopiera den annars och gör en ny xml-fil med det innehållet. Har du inte tillgång till OXYGEN får du nöja dig med att läsa och studera, men kan i viss utsträckning kopiera och klistra in från det här materialet och följa stegen i någon annan editor med stöd för XML-schema-validering.

På
https://play.hb.se/media/Generating+XML+Schema+in+oXygen/0_p841cw2a
finns en inspelning av hur du går tillväga i praktiken för att låta OXYGEN generera ett XML schema och sammankoppla den med xml-filen. Följ de anvisningarna så kommer du att få någonting liknande följande grammatik. Det kan möjligen variera i olika OXYGEN-versioner.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="receptsamling">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="recept"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="recept">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="namn"/>
                <xs:element ref="ingredienser"/>
                <xs:element maxOccurs="unbounded"
                ref="procedur"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ingredienser">
        <xs:complexType>
            <xs:sequence>
                <xs:element maxOccurs="unbounded"
                ref="ingrediens"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="ingrediens">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="namn"/>
                <xs:element minOccurs="0" ref="maengd"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="maengd">
        <xs:complexType mixed="true">
            <xs:attribute name="enhet"
            use="required" type="xs:NCName"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="procedur" type="xs:string"/>
    <xs:element name="namn" type="xs:string"/>
</xs:schema>
```

Resultatet är bara en approximering av din tanke och kan behöva justeras.

Om du inte har tillgång till OXYGEN *så kan du kopiera schemat ovan och lagra i en fil* `recept.xsd` *samt addera följande till rotelementets märke i xml-filen.*

```
<receptsamling
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="nyttrecept.xsd">
```

Som du ser har varje element i receptsamlingen ett motsvarande `xs:element` som definierar det. `procedur` och `namn` är enkla element-typer eftersom de endast innehåller text. Dessa har givits namn (värdet till attributet `name`) och en deklaration av vilket innehåll de får ha — `xs:string`. `xs:` är ett namnrymdsprefix som refererar till standarden (namnrymden) *XML Schema* och `string` anger just textinnehåll. `xs:element` är här i dessa fall också noterat som ett *tomt element*.

Definitionen av `recept` är något mer komplex. `xs:complexType` gäller för sådana element som kan ha elementinnehåll — i det här fallet tre element. `namn`, `ingredienser` och `procedur`. Attributet `ref` refererar till dessa element som då definierats på annan plats i grammatiken (på rad 44, 29–36 och 43). `xs:sequence` betyder att ingående element måste förekomma i angiven ordning, och attributet `maxOccurs` med värdet `unbound` att elementet `procedur` kan upprepas (men övriga kan det inte).

Innehållsmodellen för `receptsamling` är likartad, men här förstår du kanske att samlingen endast kan innehålla ett enda recept, vilket troligen inte är vad vi ville. Då hade vi satt `recept` som rotelement i stället. Vi adderar attributet `maxOccurs` med värdet `unbound`.

Värt att notera är för elementet `ingrediens` där `minOccurs` satts till 0 för elementet `maengd`s förekomst, vilket innebär att det är valfritt att använda det. Det måste däremot följa direkt på `namn` och inte föregå det.

```
<xs:choice maxOccurs="unbounded">
    <xs:element ref="namn"/>
    <xs:element ref="maengd"/>
</xs:choice>
```

hade gjort det möjligt att använda båda elementen huller om buller. Notera hur `minOccurs` bestämmer `xs:choice` i stället för något av elementen.

Det är också möjligt att göra elementet ännu mer fritt, genom att gör det till något med blandat innehåll, både text och element. Attributet `mixed` med värdet `true` fixar det:

```
<xs:element name="ingrediens">
    <xs:complexType mixed="true">
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="namn"/>
            <xs:element ref="maengd"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Slutligen, se på definitionen av elementet `maengd` på rad 37–42, s. 13. Elementet innehåller i vårt exempel bara en mängdangivelse, men skillnaden är att det också har ett attribut som behöver definieras och är därmed en ”complexType” och behöver attribut-värde-paret `mixed='true'`. Attributet har en restriktion som säger att det *måste* anges och är av typen `xs:NCName`. `xs:NCName` kan ersättas av `xs:string`.

Ovan beskrivna grammatiska konstruktioner är de vanligaste du kommer att stöta på och sannolikt vad som är tillräckligt i det här sammanhanget. Om du vill utveckla kunskaperna kan W3Schools ’tutorial’ vara en hygglig utgångspunkt.[5]

2.2 Självständigt arbete med annat exempel

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

[5] https://www.w3schools.com/xml/schema_intro.asp

mot. Vanligen finns redan någon färdig tillämpning som kan användas rakt av eller utvidgas efter behov.

Eftersom vi ska utnyttja OXYGEN för att generera scheman och på så vis förkorta vägen till en grammatik så kan du gå tillväga precis som i exemplet i förra avsnittet.

Brevets innehåll kan i korthet beskrivas på följande vis i strukturella termer.

> Brevet innehåller a) ett brevhuvud, b) en hälsningsfras, c) ett antal stycken med radbrytningar, d) ett stycke poesi med strofer, e) en avslutningsfras, f) en signatur, och g) två sidor.
>
> 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 och din uppgift är att börja med att märka upp exempelbrevet på ett genomtänkt vis.

Utgå från analysen ovan och skapa ett nytt xml-dokument för brevets text och finn lämpliga namn för elementen, samt brevet som helhet och även, om du tänker dig flera brev i samma fil, för samlingen av brev.

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

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.