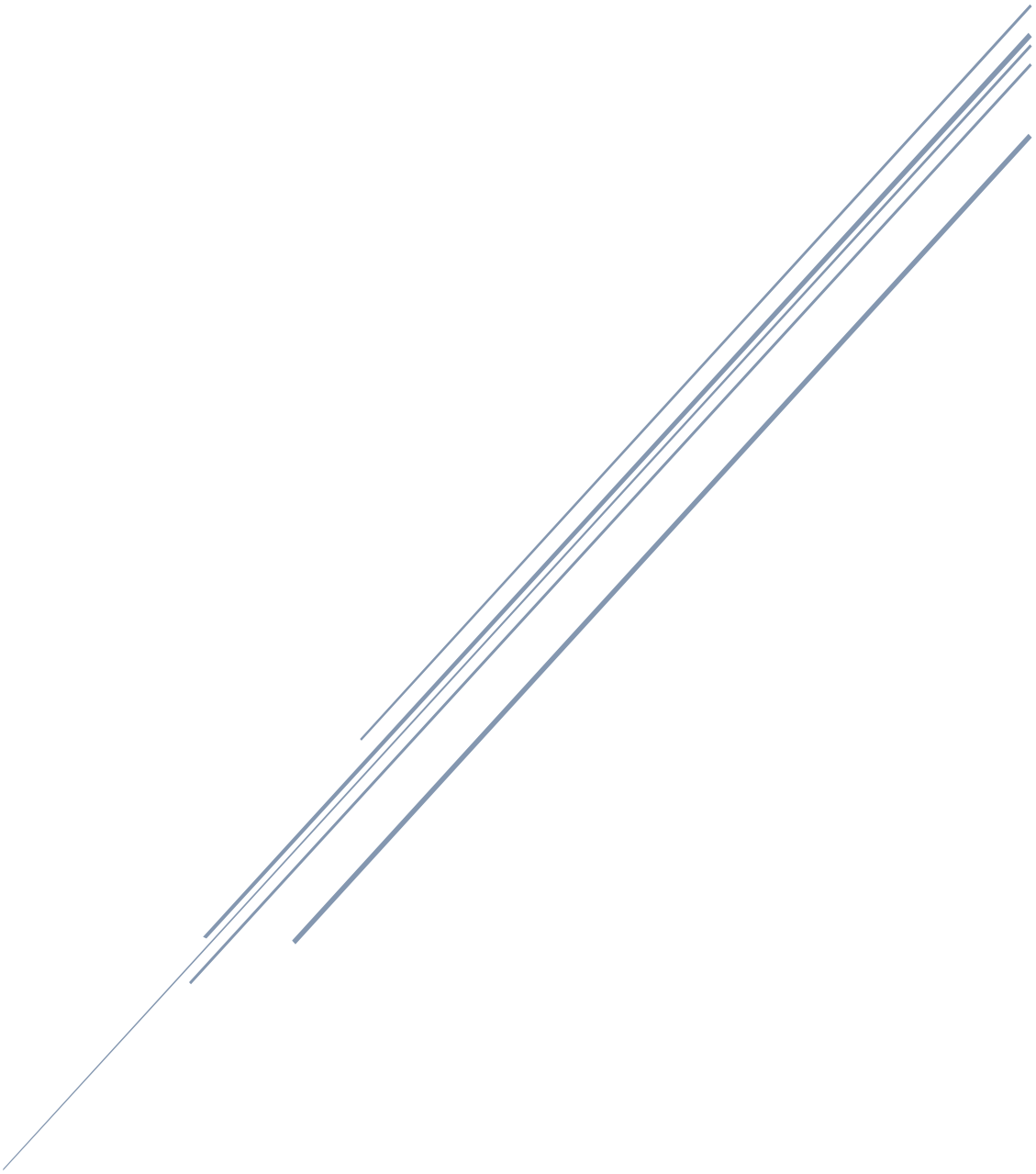
LEKTION 2

Mer om XML



Mittuniversitetet
XML

Innehållsförteckning

1 Mer om skapande av XML-dokument

I denna lektion fortsätter vi att titta på hur XML-dokument skapas. Förra lektionen avslutades med en titt på hur bland annat de särsvenska tecknen hanterades. Detta fortsätter vi med i denna lektion samt hur andra specialtecken hanteras. Utöver detta tittar vi på en del andra viktiga byggklossar som processinstruktioner och namnrymder. I slutet av lektionen tittar vi på modellering och strukturering av XML-dokument.

1.1 Inledning

I förra lektionen såg vi att en del tecken inte kan användas helt utan vidare i ett XML-dokument.
Detta gäller till exempel de särsvenska tecknen. I lektionen gavs exempel på möjliga lösningar. Vi kan använda CDATA-sektioner eller använda en korrekt teckenkodning i XML-deklarationen. I denna lektion fortsätter vi med att titta på hur en del speciella tecken kan hanteras. I kapitel 1.2 ser vi exempel på användning av entiteter. Med entiteter kan vi namn-sätta speciella tecken så dessa namn kan användas i själva XML-dokumentet. Entiteter kan även vara användbara om långa texter ska användas flera gånger i samma dokument. Vi kan då ge texten en förkortning och sen i fortsättningen använda förkortningen i stället för den långa texten.
I kapitel 1.3 kommer vi in på processinstruktioner. Detta används till exempel i samband med stilmallar i XML-dokument. Efter detta tar vi en titt på namnrymder. Detta är en standard som används ofta i samband med XML Schema och XSLT (kommer i senare lektioner).
När du skapar dina egna XML-dokument är det viktigt att du tänker över hur data i dokumentet ska användas innan du bestämmer dokumentets struktur. Av denna anledning kommer de tre sista kapitlen i lektionen att gå igenom hur du bör modellera (strukturera) XML-dokument. Vi kommer även att kort titta på hur en XML-tolk fungerar. Detta kan vara till hjälp när du försöker finna och förstå eventuella fel i ett XML-dokument.

1.2 Entiteter

Det är inte ovanligt att i ett XML-dokument ha text eller annat som återkommer många gånger på olika ställen i dokumentet. Exempel på detta kan vara ett företagsnamn och företagets logo. Tänk dig att vi vill skapa ett XML-dokument för Mittuniversitetet i vilket vi kommer att behöva skriva; *Mittuniversitetet i Östersund* många gånger. Vid sådana tillfällen kan det vara en fördel att skriva namnet på ett ställe och sen referera till detta i resten av XML-dokumentet. Vi kan då till exempel använda förkortningen MIUNÖSD på de ställen i XML-dokumentet där vi vill att texten Mittuniversitetet i Östersund ska stå. Detta kan lösas med hjälp av entiteter. Genom att använda entiteter kan du bland annat koppla ett namn till en längre text. Vi skapar i fallet ovan en entitet med namnet MIUNÖSD, som står för Mittuniversitetet i Östersund. Varje gång MIUNÖSD nu förekommer i XML-dokumentet byts det ut mot Mittuniversitetet i Östersund. Texten kommer att bytas ut varje gång XML-dokumentet läses av XML-tolken (till exempel när dokumentet ska visas i en webbläsare).
Vi har redan sett exempel på användning av entiteter i första lektionen. Där använde vi namnet amp när vi ville använda tecknet & som innehåll i ett element. Detta är en så kallad fördefinierad entitet (som det finns totalt fem stycken av).

En entitet består av två delar; deklarationen (i en DTD) och entitetsreferensen (i XML-dokumentet).
Vidare delas entiteter in i två typer:
- Generella entiteter: används i XML-dokumentet. När vi enbart använder ordet entitet är det denna typ som menas.
- Parameterentiteter: används i schemat (till exempel i en DTD).
Båda typer av entiteter kräver att entiteten deklareras innan den används. I kursen tar vi endast upp generella entiteter. Parameterentiteter ingår inte i kursmaterialet.

1.2.1 Generella entiteter

En generell entitet består alltså av två delar; en **entitetsdeklaration** och en **entitetsreferens**.

Deklaration av entiteter

Nedan följer ett exempel på en intern entitet:

```
<!ENTITY NamnPåEntitet "Detta är ett exempel på en intern entitet. All text
som står innanför citattecknen kommer att användas när vi skriver
entitetsnamnet NamnPåEntitet i XML-dokumentet">
```

!ENTITY används för att visa att här deklareras en entitet. Därefter följer namnet på själva entiteten; NamnPåEntitet. Det är detta namn som ska användas som innehåll i elementen i XML-dokumentet. Sist följer en text innanför " … ". Denna text kommer att användas när namnet NamnPåEntitet används i XML-dokumentet. Detta är en så kallad intern entitet.
Det finns även externa entiteter. Texten finns då i en annan fil än själva XML-dokumentet:

```
<!ENTITY NamnPåExternEntitet SYSTEM
"http://www.miun.se/personal/robert.jonsson/xml/lek04/entitetExt.txt">
```

Namnet på denna entitet är NamnPåExternEntitet. För externa entiteter används ordet SYSTEM. Detta berättar för XML-tolken at det är frågan om en extern entitet och att innehållet
ligger på den URI som kommer direkt efter ordet SYSTEM. URI pekar på, i detta exempel, filen entitetExt.txt. Innhållet i denna fil kan till exempel vara (filen finns inte på riktigt utan används bara som exempel):

```
Detta är ett exempel på en extern entitet. All text i denna fil kommer att
användas i XML-dokument där namnet NamnPåExternEntitet används.
```

Entitetsreferenser

För att använda de entiteter som skapats ovan använder vi samma tillvägagångssätt som vi såg exempel på i kapitel 1.5.1 i första lektion. Det vill säga att vi skriver ett & före namnet på entiteten och ett ; efter namnet:

```
&NamnPåEntitet;
```

Samma gäller vid användning av externa entiteter:

```
&NamnPåExternEntitet;
```

Att använda entiteter i XML-dokument

För att använda entiteter i ett XML-dokument måste entitetsdeklarationen läggas i ett speciellt
element; dokumenttypsdeklarationen:

```
<!DOCTYPE NamnPåRootElementet [ … ]>
```

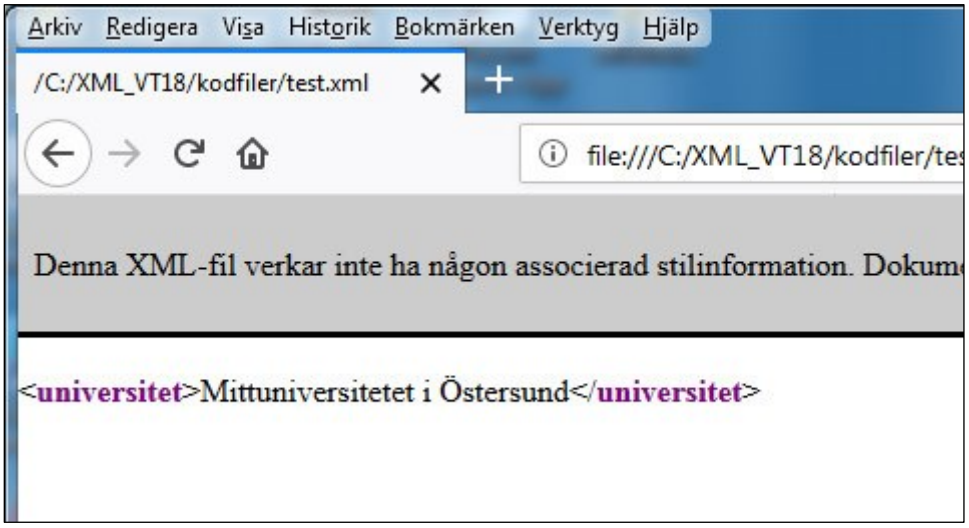
Dokumenttypsdeklarationen används även för att berätta för tolken var schemat finns (DTD).

Mer om just detta kommer vi till i lektions 5. Just nu behöver du bara veta att entitetsdeklarationer måste skrivas i detta element.

Nedan följer nu ett exempel som visar hur vi skapar en entitet som sen används i ett element:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE universitet [
  <!ENTITY MIUNÖSD "Mittuniversitetet i Östersund">
]>
<universitet>&MIUNÖSD;</universitet>
```

XML-dokumentet kan nu öppnas i en webbläsare. I Firefox ser det ut så här:



Texten MIUNÖSD är ersatt med texten Mittuniversitetet i Östersund precis så som det var deklarerat i entiteten.

Svenska tecken som å, ä, ö

I lektion 1 såg vi exempel på hur vi kunde använda olika ”ogiltiga tecken” som innehåll i ett element genom att använda oss av fördefinierade entiteter eller teckenreferenser där ett teckennummer angavs. De fördefinierade entiteterna finns definierade i XML-standarden och förstås av alla tolkar utan att de behöver deklareras. Det är, som sagts ovan, även möjligt att deklarera egna entiteter. Detta kan vara en lösning för att använda våra särsvenska tecken; å, ä och ö eller andra tecken som normalt inte kan skrivas med hjälp av tangentbordet. Vi måste då veta dess teckennummer (Unicode). I tabellen nedan vissas teckennummer för å, ä och ö (inom parantes uttryckt som hexadecimalt):

Tecken	*Teckennummer*	*Tecken*	*Teckennummer*
å	229 (00E5)	Å	197 (00C5)
ä	228 (00E4)	Ä	196 (00C4)
ö	246 (00F6)	Ö	214 (00D6)

Teckennummer kan användas direkt som innehåll i ett element om en teckenreferens används.

Som vi såg i lektion 1 används &# framför teckennumret och ; efter om teckennumret uttrycks decimalt (uttrycks teckennumret hexadecimalt måste vi börja med &#x). Texten:

ä ö Å uttryckt med teckenreferenser blir då:
ä ö Å
Och Mittuniversitetet i Östersund skrivs därmed enligt:
Mittuniversitetet i Östersund
Dessa teckennummer kan du använda som teckenreferenser i innehållet i ett element. Att använda teckenreferenser i texten kan både vara svårt att läsa och inte minst svårt att förstå.
Av den anledningen kan vi skapa entiteter för dessa tecken. Nedan visas ett XML-dokument där de sex tecknen har skapats som entiteter:
Dessa teckennummer kan du använda som teckenreferenser i innehållet i ett element. Att använda teckenreferenser i texten kan både vara svårt att läsa och inte minst svårt att förstå.
Av den anledningen kan vi skapa entiteter för dessa tecken. Nedan visas ett XML-dokument där de sex tecknen har skapats som entiteter:

```
<?xml version="1.0"?>
  <!DOCTYPE test [
  <!ENTITY aa "&#229;">
  <!ENTITY ae "&#228;">
  <!ENTITY oe "&#246;">
  <!ENTITY AA "&#197;">
  <!ENTITY AE "&#196;">
  <!ENTITY OE "&#214;">
]>
```

1.3 Processinstruktioner

Processinstruktioner används för att förmedla instruktioner till applikationen som ska använda
XML-dokumentet. XML-tolken förhåller sig inte till processinstruktionerna utan skickar dem bara vidare. Vad som ska göras med processinstruktionerna är upp till den applikation som får dem. I denna kurs använder vi mest en webbläsare som applikation, men det kan även vara många andra program som använder XML-dokumentet. Processinstruktioner kommer vi att använda i denna kurs för att styra hur XML-dokument ska visas på skärmen.
Fram till nu har vi stött på två olika processinstruktioner (de två vanligaste). Den första är XML-deklarationen i början av prologen:

```
<?xml version="1.0"?>
```

Detta är en speciell form av processinstruktion som visar att dokumentet är ett XML-dokument (den ses oftast inte som en processinstruktion trots att alla processinstruktioner börjar med <? och avslutas med ?>). Den andra processinstruktionen används för att lägga till en stilmall och ska även den anges i prologen:

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

Webbläsaren, eller någon annan applikation, vet då att stilmallen Boksamling.css ska användas när innehållet i XML-dokumentet presenteras.

En processinstruktion skrivs allmänt enligt följande:

```
<?mål datainstruktioner?>
```

<? visar att detta är början av en processinstruktion, mål (target) är den applikation datainstruktionerna ska skickas till och datainstruktion är vad som ska göras. Observera att vi inte kan ha mellanrum eller andra whitespace-tecken mellan ? och mål. Själva namnet på målet ska skrivas med små bokstäver.
Ett exempel på processinstruktion kan vara:

```
<?winamp version="5.33" bitrate="198"?>
```

Målet i processinstruktionen i exemplet ovan är Winamp (ett gammalt hederligt program för att spela upp ljudfiler i Windows). Datainstruktionerna i detta exempel anger att den version av Winamp som ska användas är 5.33 och att den bitrate ljudfilen ska spelas upp i är 198 Kbps.
I denna kurs kommer de mål som används att vara xml och xml-stylesheet (se ovan). I följande exempel används både xml och xml-stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="Boksamling.css"?>
<boksamling>
  <bok>
  <titel>XML Content and Data</titel>
  <författare>Kelly Carey</författare>
  <författare>Stanko Blatnik</författare>
  <sidor>408</sidor>
  <pris>410 kronor</pris>
</bok>
</boksamling>
```

1.4 Namnrymder

Om data kombineras från flera olika källor till ett enda XML-dokument kan det resultera i namnkonflikter för element och attribut. Föreställ dig två olika XML-dokument där ena dokumentet innehåller en boksamling medan det andra dokumentet innehåller en samling över CD-skivor. Nedan följer först boksamlingen:

```
<?xml version="1.0" encoding="UTF-8"?>
<samling>
   <post>
      <titel>The Adventures of Huckleberry Finn</titel>
      <författare>Mark Twain</författare>
      <pris>350kr</pris>
   </post>
  … <!-- Fler poster -->
</samling>
```

och här följer cd-samlingen:

```
<?xml version="1.0" encoding="UTF-8"?>
<samling>
  <post>
      <titel>Violin Concerto in D</titel>
      <kompositör>Beethoven</kompositör>
      <pris>430kr</pris>
   </post>
   … <!-- Fler poster -->
```

</samling>
I båda dessa dokument finns många element med samma namn, till exempel samling och post. Om nu dessa två XML-dokument slås ihop till ett dokument kommer det att uppstå problem med de elementnamn som är lika. Applikationen som läser XML-dokumentet kommer till exempel inte att kunna skilja mellan en bok-post och en cd-post (elementet post). Detta gäller även för elementen titel och pris. Det finns några sätt att lösa detta problem på. Antingen genom att skriva om hela XML-dokumentet eller genom att använda namnrymder (namespace). Namnrymder skiljer mellan olika typer av element med samma namn genom att tilldela dessa element separata namnrymder. Det första som måste göras är att deklarera namnrymden. Detta görs som regel i root-elementet, men kan även göras i andra elements starttagg.

Ovan är ett exempel på en så kallad namnrymdsdeklaration. Här deklareras en namnrymd genom att ange den fasta strängen xmlns: (XML namespace), följt av ett suffix som i detta fall är bok (kan variera). Därefter följer ett namnrymdsnamn i form av en URI, i detta fall används "http://www.mediaOnline.com/books". Det behöver inte finnas något på den plats URI pekar på. En URI används enbart för att den garanterat är unik.
För att sen använda denna namnrymd sätts prefixet bok framför elementnamnet enligt:

<bok:titel>The Adventures of Huckleberry Finn</bok:titel>

Nu kommer detta titel-element att tillhöra namnrymden bok.

I nedanstående exempel har de två separata dokumenten från exemplen ovan slagits samma till ett XML-dokument i vilket namnrymder används:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<samling
   xmlns:bok="http://www.mediaOnline.com/books"
   xmlns:cd="http://www.mediaOnline.com/cds">
<bok:post>
   <bok:titel>The Adventures of Huckleberry Finn</bok:titel>
   <bok:författare>Mark Twain</bok:författare>
   <bok:pris>350kr</bok:pris>
</bok:post>
<cd:post>
   <cd:titel>Violin Concerto in D</cd:titel>
   <cd:kompositör>Beethoven</cd:kompositör>
   <cd:pris>430kr</cd:pris>
</cd:post>
… <!-- Fler poster -->
</samling>
```

Nu kan applikationer som använder dokumentet skilja mellan post-element innehållandes böcker och post-element innehållandes cd-skivor.
Det är även möjligt att ange en namnrymd som ska användas som standard. Vi slipper därmed att använda ett prefix före elementnamnet och alla element utan prefix kommer att tillhöra den så kallade standardnamnrymden. För att deklarera standarnamnrymden används inget suffix i namnrymdsdeklarationen:

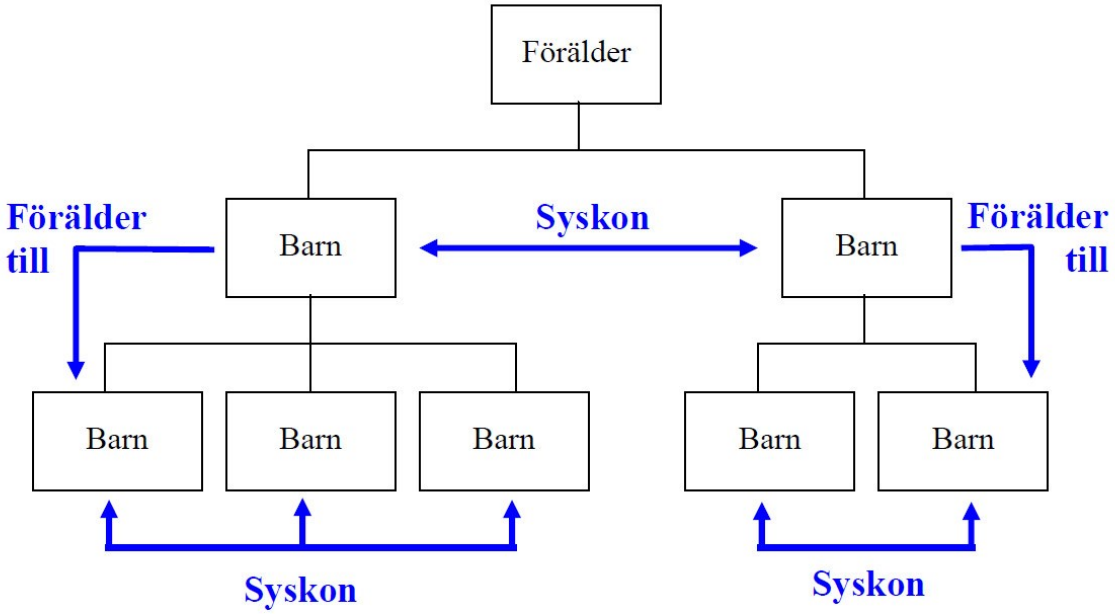
```
<samling
   xmlns="http://www.mediaOnline.com/books"
   xmlns:cd="http://www.mediaOnline.com/cds">
```

I exemplet ovan har vi tagit bort suffixet bok. Nu kommer alla element utan prefix att tillhöra namnrymden som identifieras med namnrymdsnamnet http://www.mediaOnline.com/books.
Namnrymder används i samband med många av de övriga standarder som hör till XML. Exempel på detta är XSLT och XML Schema som vi kommer att titta på i kommande lektioner. I samband med detta används namnrymder för att XML-tolken ska veta hur de olika elementen ska tolkas. För XSLT används en bestämd URI vars betydelse tolken då känner till.

1.5 Den hierarkiska strukturen

Vi ska nu titta på några egenskaper kring den hierarkiska strukturen i ett XML-dokument. Vi börjar med att definiera ordet dokumentträd. Ett dokumentträd används för att beskriva den hierarkiska strukturen i XML-dokument. Ett dokumentträd består av en rotnod överst och därefter följer förgreningar nedåt i överordnade och underordnade element (en upp-och nedvänd trädstruktur med roten högst upp och grenar nedåt). Rotnoden betraktas som dokumentets yttersta nod, vilket innebär att den innehåller hela dokumentet och som kan ha ett oändligt antal noder under sig (barnnod). Varje barnnod kan ha olika antal noder under sig.

Nedan finns en figur som visar sambandet mellan de olika noderna:



Ett dokumentträd kan bland annat användas för att planera, organisera och komma åt data. Genom att skapa ett träd på detta sätt blir det enklare att navigera mellan noderna och att manipulera data som ligger i noderna. Det är denna hierarkiska struktur som DOM (Document Object Model) och XSLT utnyttjar. XSLT kommer vi in på senare.

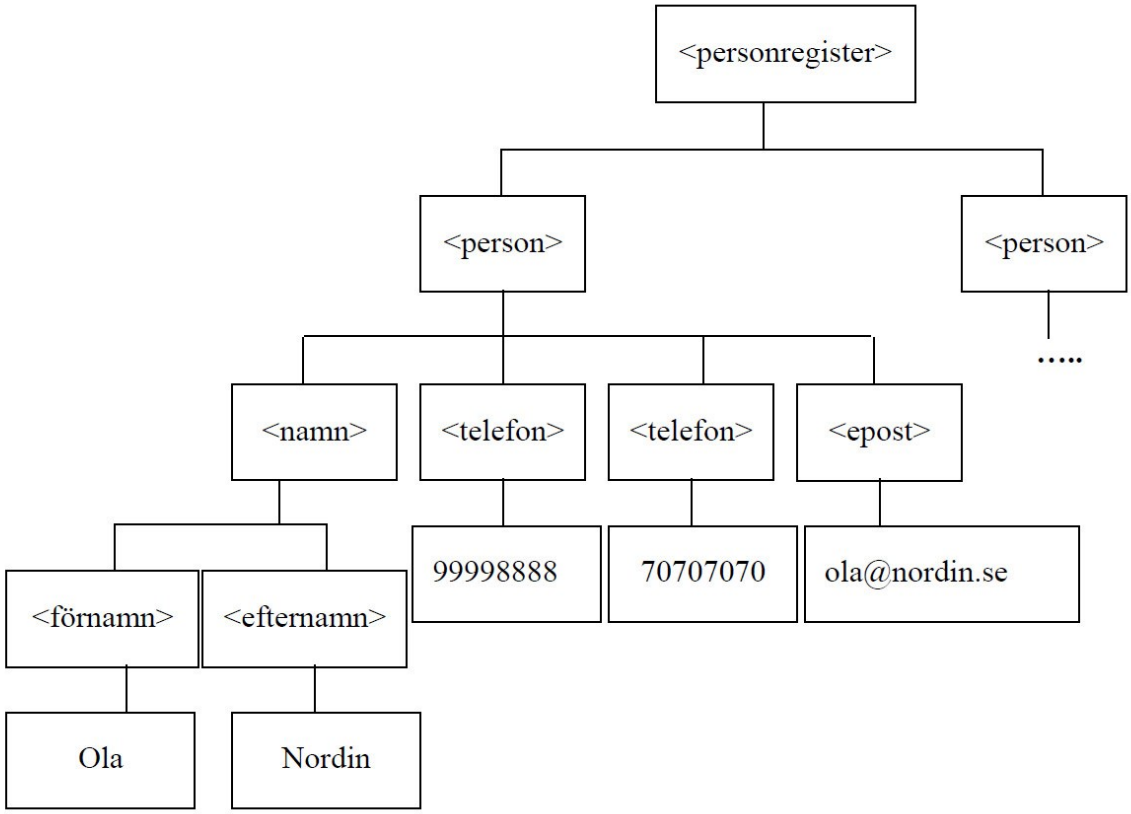
Exempel

Tänk dig ett XML-dokument som lagrar data om flera personer. För varje person har man behov av att lagra olika uppgifter. För Ola har vi kanske tillgång till information om 2 telefonnummer och en e-postadress, medan vi för Karin har ett telefonnummer, ett faxnummer och två e-postadresser. Detta innebär att det krävs olika antal och olika typer av element för att lagra informationen om de två personerna. Se följande exempel:

```
<?xml version="1.0" encoding="UTF-8"?>
<personregister>
  <person>
    <namn>
      <förnamn>Ola</förnamn>
      <efternamn>Nordin</efternamn>
    </namn>
    <telefon>99998888</telefon>
    <telefon>70707070</telefon>
    <epost>ola@nordin.se</epost>
  </person>
  <person>
    <namn>
      <förnamn>Karin</förnamn>
      <mellannamn>Larson</mellannamn>
      <efternamn>Nordin</efternamn>
    </namn>
    <telefon>70707070</telefon>
    <fax>89898989</fax>
    <epost>karin@nordin.se</epost>
    <epost>nordin@online.se</epost>
  </person>
</personregister>
```

Som du finns det för Ola två telefon-element, medan Karin har ett. Karin har istället ett fax-element. Det lagras med andra ord enbart den information som behövs för varje person. Om en uppgift för en person saknas, till exempel ett telefonnummer, måste vi skapa detta element.
Längre fram kommer vi in på DTD och XML Schema som begränsar vilka element som kan användas och hur de kan användas.

Vi kan nu skapa ett dokumentträd av ovanstående XML-dokument (Här tas enbart med nod för en person på grund av platsbrist):



Här ser du att person-elementet är en barnnod till personregister-elementet. Vidare har person-elementet flera underliggande barnnoder. Elementen namn och epost är syskon med varandra.
Är man inte så van att skapa XML-dokument har man en tendens till att skapa väldigt platta dokument. Med detta menas dokument som har få nivåer av element. I exemplet ovan tycker du kanske att det är onödigt att ta med elementet namn och istället skriva alla element som innehåller information direkt under person-elementet. Nedan ser du hur ett sådant XMLdokument
kan se ut:

```
<?xml version="1.0" encoding="UTF-8"?>
<personregister>
  <person>
    <namn>
      <förnamn>Ola</förnamn>
```

```
        <efternamn>Nordin</efternamn>
      </namn>
      <telefon>99998888</telefon>
      <telefon>70707070</telefon>
      <epost>ola@nordin.se</epost>
    </person>
    <person>
      <namn>
        <förnamn>Karin</förnamn>
        <efternamn>Nordin</efternamn>
      </namn>
      <telefon>70707070</telefon>
      <fax>89898989</fax>
      <epost>karin@nordin.se</epost>
    </person>
</person
```

Även om detta är ett välutformat dokument kan det vara mindre effektivt vid datamanipulering.
Just nu kommer det kanske vara svårt att tänka sig varför dett är mindre effektivt, men det kommer att klarna allt eftersom. Det som är viktigt att lägga på minnet är hur användarna ska komma åt och hur de ska använda data. Tänk på hur du strukturerar dokumentet och var inte rädd för att använda några extra element för att få en bättre struktur när du märker upp data (till exempel genom att använda ett extra namn-element ovan).

1.6 XML-tolken

En tolk är ett verktyg som analyserar ett XML-dokument och som kontrollerar om syntaxen är korrekt. Kontrollen är en standardiserad process och beskrivs ingående i specifikationen. För att du ska få en bättre förståelse hur ett XML-dokument läses av ett program ska vi titta lite närmare på hur en XML-tolk fungerar.
Det finns två typer av tolkar:
1. **Icke-validerande tolk**. Gör enbart en kontroll om XML-dokumentet är välutformat.
2. **Validerande tolk**. Utöver kontrollen om XML-dokumentet är välutformat kontrolleras även om dokumentet är giltigt (hur XML-dokumentet överensstämmer mot schemat).
Tolken kontrollerar enbart XML-dokumentets syntax. Den tar inte hänsyn till om den struktur du använt är bra eller dålig och den bryr sig inte heller om själva innehållet i dokumentet (en del tolkar kan göra vissa kontroller på innehållet, till exempel att ett datum-element verkligen innehåller ett datum).
När en tolk kontrollerar ett XML-dokument läser den antingen in hela dokumentet i minnet och bygger en trädliknande struktur av innehållet, eller så tolkas innehållet allt eftersom innehållet läses. Det första sättet använder en trädbaserad tolk (DOM) och det andra sättet använder en händelsebaserad tolk (SAX). Nedan beskrivs kort hur en trädbaserad tolk arbetar (händelsebaserade tolkar går vi inte in på här).

Tolken startar i början av XML-dokumentet för att finna det första elementet. Här kontrolleras om elementet har rätt syntax. Upptäcks ett fel skrivs ett felmeddelande ut och tolkningen avbryts. När användaren rättat till felet kan ett nytt försök att tolka dokumentet göras. När ett element inte innehåller fler fel, lagras informationen om nodtypen i en tabell, för att sen försätta med nästa element. När tolken kontrollerar ett element med barnnoder så kontrolleras alla barnnoder innan tolken går vidare till nästa syskonnod.
När tolken har kommit igenom hela dokumentet och det inte finns fler fel (när XML-dokumentet med andra ord är välutformat), skapas ett tolkat träd. Trädet skickas sen vidare till XML-prosessorn som därmed kan behandla trädet som det önskar.
För validerande tolkar måste det dessutom kontrolleras om elementen i XML-dokumentet överensstämmer med till exempel en DTD. Tolken har då två alternative tillvägagångssätt. Antingen kan en logisk struktur av DTD:n skapas som jämförs med XML-dokumentets struktur. Alternativet är att gå igenom XML-dokumentet ytterligare en gång efter att syntaxen har kontrollerats, för att jämföra element för element med DTD:n.

1.7 Modellering

Målet med detta kapitel är att få dig att tänka över hur du strukturerar dina XML-dokument. Något som är viktigt att tänka på innan du börjar skapa ditt XML-dokument är vad som ska göras med de data som sparas i dokumentet.
Tänk dig att du ska lagra några uppgifter om olika personer. Till en början kan detta låta som en väldigt enkelt uppgift. Det första du måste ta hänsyn till är vilka data som är viktiga att lagra. Oftast bestäms detta utifrån hur XML-dokumentet kommer att användas. Det kommer att vara en stor skillnad på om XML-dokumentet kommer att användas som en gästlista för ett hotell eller om det är frågan om ett straffregister som polisen ska använda. Här kommer olika personuppgifter att prioriteras på olika sätt och hänsyn måste även tas till vilka uppgifter som man får tillgång till och vilka uppgifter som får lagras. Detta påverkar vilka element som ska tas med i XML-dokumentet.
Det är även viktigt hur data lagra i elementen. Vi har nedanstående XML-dokument som innehåller dataprodukter och dess priser:

```
<varulager>
   <produkt>
      <namn>27" IPS skärm</namn>
      <pris>3200 kr</pris>
   </produkt>
   <produkt>
      <namn>Hårddisk 400GB SSD</namn>
      <pris>1100 kr</pris>
   </produkt>
</varulager>
```

Kan en applikationen som läser detta XML-dokumentet räkna ihop det totala priset för alla produkter? Innehållet i elementet pris är en textsträng och det är inte (direkt) möjligt att omvandla texten 3200 kr till ett tal. Samtidig kan vi inte bara ta bort kr eftersom vi då inte vet vilken
valuta priset är angivet i. En lösning är att lägga till ett attribut som visar vilken valuta det är frågan om:

```
<varulager>
  <produkt>
    <namn>27" IPS skärm</namn>
    <pris valuta="kr">3200</pris>
  </produkt>
  <produkt>
    <namn>Hårddisk 400GB SSD</namn>
    <pris valuta="kr">1100</pris>
  </produkt>
</varulager>
```

Ett annat exempel är registrering av flere uppgifter om en person. Vi har följande XMLdokument (dålig modellerat i förhållande till vad vi behöver)

```
<personregister>
  <person>
    <förnamn>Ola</förnamn>
    <efternamn>Nordin</efternamn>
    <telefon>99998888</telefon>
    <telefon>70707070</telefon>
    <telefon>63636363</telefon>
    <epost>ola@nordin.se</epost>
    <epost>olan@jobb.se</epost>
  </person>
</personregister>
```

Som du ser har vi lagt till ytterligare några uppgifter sen tidigare exempel. Vi har lagrat tre telefonnummer och två e-postadresser, men det är inte så lätt att veta var de olika telefonnumren och e-postadresserna gäller. Några av dem är kanske privata och några används i jobbet. Givetvis borde denna typ av uppgifter ha lagrats. Vi kan antingen lägga till attribut i elementen telefon och epost eller skapa helt nya element som skiljer mellan kontaktinformation som rör hemmet och jobbet:

```
<?xml version="1.0" encoding="UTF-8"?>
<personregister>
  <person>
    <förnamn>Ola</förnamn>
    <efternamn>Nordin</efternamn>
    <telefon typ="hem">99998888</telefon>
    <telefon typ="hem">70707070</telefon>
    <telefon typ="jobb">63636363</telefon>
    <epost typ="hem">ola@nordin.se</epost>
    <epost typ="jobb">olan@jobb.se</epost>
  </person>
</personregister>
```

ELLER

```
<?xml version="1.0" encoding="UTF-8"?>
<personregister>
   <person>
      <förnamn>Ola</förnamn>
      <efternamn>Nordin</efternamn>
      <kontakt typ="hem">
         <telefon>99998888</telefon>
         <telefon>70707070</telefon>
         <epost>ola@nordin.se</epost>
      </kontakt>
      <kontakt typ="jobb">
         <telefon>63636363</telefon>
         <epost>olan@jobb.se</epost>
      </kontakt>
   </person>
</personregister>
```

Här ser du exempel på att det finns flera sätt att modellera detta på. Viktigast är att du tänker över vilka program som ska läsa dokumenten och hur data kommer att användas och sen modellerar utifrån det.