Kapitel 2

En teknisk beskrivning av XML och dess syntax

Det här kapitlet inleds med bakgrunden till XML. Sedan introduceras de delar som XML består av: XML, XSL, XLL och DOM. Tyngdpunkten i kapitlet ligger på den tekniska beskrivningen av delarnas uppbyggnad och syntax. Beskrivningen baseras på specifikationer och rapporter utgivna av W3C (se referenslistan i slutet på rapporten)

Bakgrunden till märkordsspråk

Att använda märkord, så kallade ”taggar”, för att beskriva ett dokuments innehåll har många fler fördelar än det faktum att man skapar HTML-dokument på så sätt. Grundidén med att använda märkord är att ge dokument en logisk innebörd både för författare och maskin. Detta genom att använda tecken ur det alfabet som författaren är familjär med, både för att beskriva dokumentets semantik och dess syntax. Detta gör dokumentet portabelt med lång livslängd då det är oberoende av maskin eller applikation för att kunna tolkas.

De första idéerna till att strukturerade dokument kunde utbytas och manipuleras på ett enkelt sätt om de följde ett standardiserat och öppet format har sitt ursprung på 60-talet. Då utvecklades GenCode av en kommitté på Graphic Communications Association, GCA. GenCode var avsedd att beskriva generiska typsättningskoder på ett leverantörs- och maskinoberoende format. Detta för att kunna utbyta dokument mellan olika utrustningar med bibehållet utseende. Ett annat försök gjordes av IBM som tog fram Generalized Markup Language, GML, för att strukturera upp sina interna dokument, vilket var allt från manualer till kontrakt och projektbeskrivningar. GML var även konstruerat för att kunna parallellpublicera dokumenten.

I början av 80-talet bildade representanter från GCA och IBM en kommitté som fick namnet ”Computer Languages for the Processing of Text”. Kommittén var underställd det amerikanska standardiseringsinstitutet ANSI och deras mål var att standardisera sättet att specificera, definiera och använda märkord för att beskriva dokument och dess struktur. Resultatet blev SGML.

Standard Generalized Markup Language, SGML blev en ISO standard 1986. SGML var utvecklat för att definiera och använda portabla dokumentformat. Det var utvecklat till att vara formellt för att kunna valideras, strukturerat för att kunna hantera komplexa dokument och utbyggbart till att kunna hanteras av databaser. Valideringen och beskrivningen av dokumentets struktur görs genom en så kallad Document Type Definition, DTD. En DTD beskriver vilka märkord dokumentet innehåller, hur de förhåller sig till varandra och deras eventuella attribut.

I slutet av 80-talet kom CERN att använda sig av SGML då en sommarvikarierande dataprogrammerare vid namn Tim Berners-Lee använde sig av SGMLs idiom till sin nya hypertextapplikation NeXUS. Berners-Lee lånade ett urval av märkord från en av CERNs SGMLs DTDer och definierade ett typografiskt utseende för varje märkord och lade till hypertextlänkar. HyperText Markup Language, HTML, är en instans av SGML och definieras av en DTD. Det är upp till en applikation att tolka och presentera varje HTML-dokument utifrån DTDn. 1994 slog HTML igenom i och med Internet, World Wide Web, WWW, och applikationer som Lynx, Mosaic (numera Netscape Navigator) och Internet Explorer. HTML har reviderats och utökats många gånger sedan dess och är idag uppe i version 4.0.

XML utvecklades av en arbetsgrupp på World Wide Web Consortium,W3C, kallad XML Working Group som bildades 1996 och lämnade in sin rekommendation till XML version 1.0 i februari 1998. Syftet med XML är att skapa ett generiskt SGML som skulle fungera på WWW som HTML gör idag. XML, som är en delmängd av SGML, utvecklades för att vara enkelt att implementera och vara kompatibelt med både SGML och HTML.

Vad är XML?

XML är en delmängd av SGML till skillnad från HTML som är en instans av SGML. Syftet med XML är att skapa ett generiskt SGML, det vill säga något som fungerar på samma sätt som SGML, som skall fungera på WWW som HTML gör idag. XML utvecklades för att vara enkelt att implementera och vara kompatibelt med både SGML och HTML. En liknelse mellan SGML och XML är att XML har ungefär 20% av SGML komplexitet men 80% av dess funktionalitet.

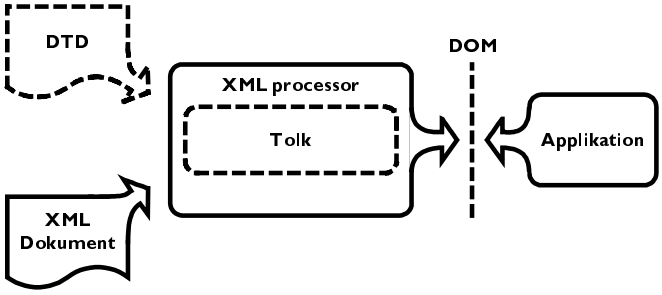
XML är ett ramverk för att definiera dokument. Det betyder att XML-specifikationen i sig inte definierar ett dokuments innehåll utan specificerar hur dokument skall definieras. Varje instans av XML, ett XML-dokument, beskriver sitt egna innehåll. Detta betyder att det är upp till användaren att själv bestämma vad dokumentet skall innehålla och hur det är uppbyggt, exempelvis är det fullt möjligt utifrån XML att beskriva HTML.

Ett XML-dokument måste vara *korrekt formulerat* (well formed) till skillnad från ett SGML och HTML-dokument. Vad som menas med ett korrekt formulerat dokument är att det uppfyller ett antal regler som specifikationen ställer på dess märkord. Att det är syntaktiskt korrekt utifrån dess specifikation. Konkret betyder det att varje öppnade märkord måste ha ett avslutande märkord, som HTMLs paragraf `<P>` som då måste ha ett avslutande paragraf `</P>`. Märkord som inte har ett logiskt slut som radbrytning `<BR>` i HTML skrivs som ett avslutande märkord `<BR/>` med den skillnaden att snedstrecket placeras i slutet av märkordet. Ett märkords eventuella attribut måste inneslutas av citattecken, exempelvis `<IMG SRC='foo'/>`. Ett XML-dokument måste innehålla ett rotelement, ett märkord som innesluter dokumentet alla andra eventuella märkord, som HTMLs `<HTML> </HTML>`.

Ett XML-dokument kan, men behöver inte, *valideras* utifrån en *Document Type Definition*, DTD. En DTD beskriver en klass av XML-dokuments grammatik, vilka märkord dokumentet kan innehålla, märkordens inbördes ordning och dess eventuella attribut. En validering kan vara nödvändig om det ställs krav på dokumentets innehåll, att all ”nödvändig” information finns där. Visserligen kan en validering endast konstatera att ett dokument uppfyller kraven på dess metadata, dess märkord och attribut. Validering utifrån en DTD gör ingen typkontroll, att varje märkord innesluter ”vettig” information.

Figur 1

Schematisk illustration av en XML-processor.

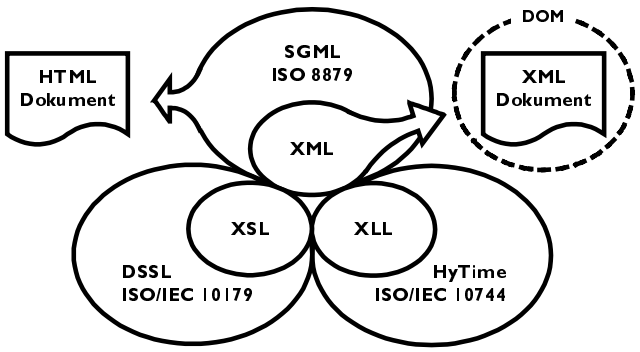


Kraven på korrekt formulerade XML-dokument gör att det inte är nödvändigt att ett dokument valideras och att det är enkelt att konstruera en XML-processor. En XML-processor (se figur 1) är det verktyg som används för att översätta XML-dokument till en format som en dator förstår. En XML-processor använder sig av en tolk (parser) för att avgöra om dokumentet är korrekt formulerat och eventuellt om det är grammatiskt korrekt. XML-processorn översätter sedan, om tolken inte funnit några fel, dokumentet till ett format lämpligt för en applikation att komma åt dess innehåll och struktur. XML-processorn är i sig

bara en typ av container innehållande data från ett XML-dokument. En XML-processor gör det möjligt för en applikation, som en webbläsare, att via ett interface (DOM) ”göra någonting” med dokumentet, som att presentera dess innehåll för en användare.

Figur 2

Illustration på XML och de delar XML består av.



XML (se figur 2) består av, förutom själva XML, två systerrekommendationer, eXtensible Style Language, XSL och eXtensible Link Language, XLL. Dessa två är inte nödvändiga för att få XML att fungera utan är snarare två typer av mekanismer som ger XML mer funktionalitet. En XSL-formatmall är i sig ett XML-dokument genom att det är definierat utifrån XML. Syftet med en XSL-formatmall är att ge ett XML-dokument ett grafiskt utseende. XLL ger ett XML-dokument länkmöjligheter, som HTML hypertextlänkar. XSL och XLL härstammar från två liknande existerande standarder, på samma sätt som XML härstammar från SGML. XSL baseras på Document Style Semantics and Specification Language, DSSSL, och XLL på Hypermedia/Time-based Structuring Language, HyTime. Både DSSSL och HyTime används tillsammans med SGML som XSL och XLL används med XML.

XML-specifikationen

Ett XML-dokument har en *logisk* och en *fysisk* struktur. Dess logiska struktur är ett dokuments innehåll, som element, attribut, text och så vidare. Dess fysiska struktur är själva dokumentet som en lagringsenhet, *entitet*, det vill säga de filer eller andra typer av lagringsenheter som tillsammans bygger upp ett dokumentet. Varje lagringsenhet kan innehålla en del av, eller ett dokuments hela, logiska struktur. Vanligtvis består ett dokument endast av en lagringenhet, en *dokument entitet*.

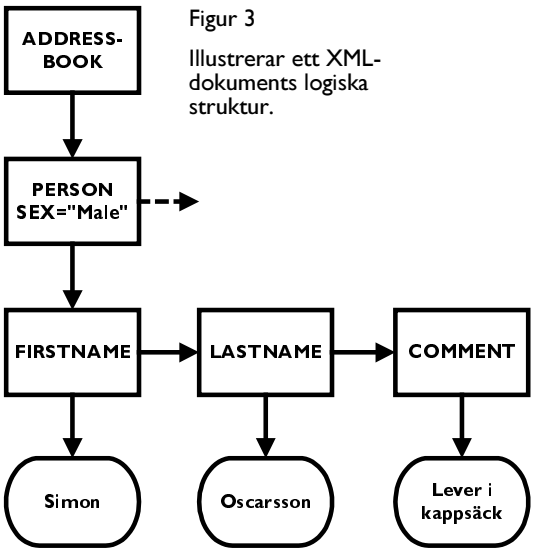
En XML-dokuments logiska struktur (se figur 3) består av ett eller flera element som kan vara öppnande, avslutande eller tomma märkord. Varje *element* är av en typ, identifierad av ett namn (ibland kallad generisk identifierare) och har noll eller flera *attribut* knutna till sig där varje attribut specificeras av ett namn och ett värde. Varje logisk struktur har en hierarkisk ordning, en trädstruktur, med endast en rot.

Ett XML-dokuments fysiska struktur (se figur 4) består av en eller flera entiteter. Varje entitet har ett innehåll, exempelvis en logisk struktur, och identifieras av ett namn. Varje dokument har en entitet som kallas för dokumententitet, vilket är roten för ett dokuments fysiska struktur. Dokumententiteten fungerar som startpunkt för en XML-processor och kan i sin tur innehålla andra entiteter.

En entitet behöver inte innehålla ett dokument utan kan innehålla ett tecken, en del av en text, en bild eller vara av något annat typ. Entiteter delas upp i två olika typer, *tolkningsbara* (parsed) eller *icke tolkningsbara* (unparsed). En tolkningsbar entitets innehåll refererar till en ersättningstext, som ett makro. En icke tolkningsbar entitet är en resurs vars innehåll kan, men behöver inte vara, en text. Om det är en text, behöver det inte nödvändigtvis vara en text i XML-format. Varje icke tolkningsbar entitet associerar till en *notation*, identifierad av ett namn, vilket i sin tur refererar till resursen. En notation kan till exempel vara information om en hjälpapplikation för att visa en viss typ av bilder, där varje bild är en icke tolkningsbara

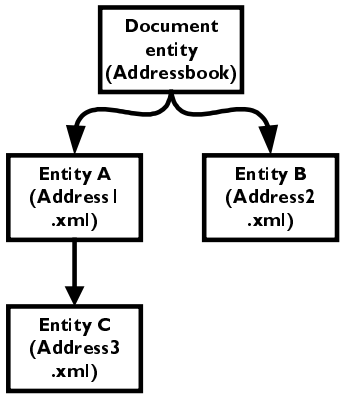
entitet. Tolkningsbara entiteter kan vara av två typer, *generella-* eller *parameterentiteter*. En generell entitet används i dokumentet medan parameterentiteter används i en DTD.

Ett XML-dokuments logiska struktur delas upp i en *prolog*, *rotelement* och en *epilog*. I prologen deklareras *processinstruktioner* som ett dokuments *XML-deklaration* och *dokument typ deklaration*. Rotelementet innefattar själva XML-dokumentet och epilogen kan innehålla processinstruktioner. Ett dokuments logiska och fysiska struktur beskrivs närmare senare i rapporten.



Figur 3

Illustrerar ett XML-dokuments logiska struktur.



Figur 4

Illustrerar ett XML-dokuments fysiska struktur.

```
<?xml version='1.0'?>
<!DOCTYPE ADDRESSBOOK
     SYSTEM 'Addressbook.dtd'>
<!-- Dess logiska struktur -->
<ADDRESSBOOK>
   <PERSON SEX='Male'>
   <FIRSTNAME>Simon</FIRSTNAME>
   <LASTNAME>Oscarsson</LASTNAME>
   <COMMENT>
      Lever i kappsäck
   </COMMENT>
   </PERSON>
</ADDRESSBOOK>
```

```
<?xml verion='1.0'?>
<!DOCTYPE ADDRESSBOOK
   SYSTEM 'Addressbook.dtd'>
<!-- Dess fysika struktur -->
&EntityA;
&EntityB;
```

Restriktioner och specialtecken

Vissa tecken är reserverade av XML-specifikationen och måste refereras till via inbyggda generella entiteter, då de används utöver dess betydelse. Detta sker på samma sätt som i HTML där vissa tecken inte kan användas direkt utan måste användas via dess entiteter, som HTML och-tecken ”&” som skrivs som `&`. De två tecken som inte kan användas till annat än enligt specifikationen är ”&” och ”<”. ”&” används för att inleda en entitet och ”<” för att inleda ett element. Andra tecken som är reserverade i vissa fall, beroende på i vilket sammanhang de används i, är dubbelt citattecken (”), enkelt citattecken (’), större än tecken (>), procent (%) och strängen (]]>).

I XML är det skillnad på gemena och versala bokstäver, XML är så att säga ”case sensitive”. Ett namn som identifierar ett element, attribut eller entitet måste skrivas på samma sätt i XML-dokument som det är deklarerat i dess DTD, det vill säga matcha gemena och versala bokstäver. Detta förutsätter att dokumentet använder en DTD, om inge DTD används har det ingen betydelse hur gemena och versala bokstäver används. Ett namn på ett element, attribut, attributvärde (med vissa undantag) eller entitet får inte innehålla vilka tecken som helst. Ett namns första tecken måste börja med tecken från alfabet eller ideografiska tecken definierade i Unicode och ISO/IEC 10646 eller tecknen ”_” och ”:”. Namnets andra tecken

och framåt får innehålla tecken från alfabet eller ideografiska tecken definierade i Unicode och ISO/IEC 10646 samt siffror och tecknen ”.”, ”-”, ”_” och ”:”. Inga blanktecken (se Blankteckenhantering) får förekomma i ett namn. Ett namn får heller inte börja med bokstavskombinationen ”xml”, som är reserverat för framtida bruk av XML-specifikationen.

I olika sammanhang används enkelt och dubbelt citattecken, (’) och (”). Det kan exempelvis vara vid deklaration av ett attribut eller attribut till en processinstruktion. Det är valfritt att använda enkelt eller dubbelt citattecken i sådana fall så länge de följer ett antal regler. Regel nummer ett är att enkla och dubbla citattecken matchas, det vill säga, börjar en attributdeklaration med enkelt citattecken så måste det avslutas med enkelt citattecken. Regel nummer två är att om ett citattecken förekommer inom ett annat så måste det inre vara enkelt om det yttre är dubbelt och vise versa. Regel nummer tre är att citattecken ej får överlappa varandra.

Special tecken	Entitet	Anmärkning
&	&amp;	Representerar sig själv endast i kommentarer, processinstruktion och CDATA sektion.
<	&lt;	Representerar sig själv endast i kommentarer, processinstruktion och CDATA sektion.
>	&gt;	Representerar sig själv över allt, dock ej där strängen ”]]>” behövs (avslut på en CDATA sektion).
”	&quot;	Kan inte förekomma i ett attribut där dubbelt citattecken (”) används för att avsluta attributet.
’	&apos;	Kan inte förekomma i ett attribut där enkelt citattecken (’) används för att avsluta attributet.
%		Representerar sig själv över allt i ett XML-dokument (därför finns ej någon entitet), dock ej i en DTD där tecknet representerar början på en parameterentitet.

Tabell 1. Illustration på specialtecken och dess entiteter.

Blankteckenhantering

Vissa tecken refereras till som blanktecken eller ”white space”. Sådana tecken är mellanslag, tabulering, radmatning och radbrytning. Nummer 32, 9, 13 respektive 10 i Unicode, ISO/IEC 10646 och ASCII teckenrepresentationer. Dessa tecken kan vara intressanta att användas i presentationssyfte i ett dokument, exempelvis som radbrytning för ett nytt stycke. Vilka av dessa tecken skall bevaras eller ignoreras då XML-processorn lämnar över information till applikationen?. Den generella regeln är att alla tecken bevaras som är innehåll till ett element, ett märkord, deklarerat till att vara av *blandat innehåll* eller bara innehåller element. Blandat innehåll är då ett element innehåller andra element och tolkningsbara tecken, så kallat PCDATA. Det betyder att exempelvis en radmatning inte bevaras av en processor om den förekommer inne i ett märkord, som mellan ett elementnamn och ett attribut. Dock får inga mellanslag får förekommer i ett namn, exempelvis ett elementnamn, då detta genererar ett fel (se Restriktioner och specialtecken). Blanktecken ignoreras då det förekommer i epilogen eller prologen av ett dokument.

Ett exempel som illustrera blankteckenhantering:

```
<?xml
   version='1.0'?>
<ADDRESSBOOK>
   <PERSON
      SEX='Male'>
      <FIRSTNAME>Simon</FIRSTNAME>
      <LASTNAME>Oscarsson</LASTNAME>
   </PERSON>
</ADDRESSBOOK>
```

I exemplet ovan bevaras alla blanktecken i rotelementet, det vill säga mellan <ADDRESSBOOK> och </ADDRESSBOOK> med det undantaget att radmatningen i elementet <PERSON> inte bevaras. Blanktecken i prologen kommer inte att bevaras, det vill säga allt innan rotelementet.

Kommentarer

Ibland kan det vara önskvärt att kommentera sina XML-dokument. Kommentarer kan förekomma valfritt i ett dokument eller i en DTD. Kommentarer kan däremot inte förekomma innanför ett märkord. En kommentar inleds med strängen ”<!--” och avslutas med ”-->”. En kommentar kan innehålla valfria tecken inklusive blanktecken med undantag för strängen ”--”.

Några exempel som illustrerar giltiga och icke giltiga kommentarer:

```
<!-- This is a legal comment -->
<?xml <!--This is an illegal comment--> version='1.0'?>
<!--This is also an illegal comment <!-- because a comment can't wrap
itself around-->-->
```

PCDATA-sektion

En Parsed Character Data, PCDATA-sektion är det ett XML-dokuments eller en DTDs innehåll vanligen är definierad till att vara om det inte är ett märkord, det vill säga text. En sektion som är PCDATA tolkas av XML-processorn för att kontrollera om det exempelvis innehåller entiteter som skall ersättas med dess ersättningstext. Om en PCDATA-sektion innehåller tecken som är reserverade av XML-specifikationen, måste dessa tecken refereras till genom de fördefinierade entiteterna med vissa undantag (se Restriktioner och specialtecken). PCDATA kan förekomma var som helst i ett rotelement.

Ett exempel där texten är av typ PCDATA:

```
<?xml version='1.0'?>
<ADDRESSBOOK>
   <PERSON SEX='Male'>
      <FIRSTNAME>Simon</FIRSTNAME>
      <LASTNAME>Oscarsson</LASTNAME>
      <COMMENT>Här måste specialtecken referers med dess entiteter, som
exempelvis &qout;&amp;&qout; eller &qout;&lt;&qout;</COMMENT>
   </PERSON>
</ADDRESSBOOK>
```

CDATA-sektion

En Character Data, CDATA-sektion kan användas i XML-dokument för att definiera ett område text till att ej tolkas av XML-processorn till skillnad från en PCDATA-sektion. En processor registrerar bara strängen ”]]>” väl inne i en CDATA-sektion, vilket är strängen för att avsluta sektionen. CDATA kan förekomma var helst PCDATA förekommer.

Ett exempel som illustrerar en CDATA-sektion:

```
<?xml version='1.0'?>
<ADDRESSBOOK>
   <PERSON SEX='Male'>
      <FIRSTNAME>Simon</FIRSTNAME>
      <LASTNAME>Oscarsson</LASTNAME>
      <COMMENT><![CDATA[Här kan vilka tecken som helst förekomma, som
exempelvis '&' eller '<' utan att behöva använda dess
entiteter]]></COMMENT>
   </PERSON>
</ADDRESSBOOK>
```

Teckenrepresentation

XML tolkar tecken enligt teckentabeller representerade av ISO/IEC 10646 och Unicode. ISO/IEC 10646 är en standard som bildades 1993 av International Organization for Standardiztion, ISO. Unicode är specificerad av ett konsortium bestående av ledande amerikanska datortillverkare. Både ISO/IEC 10646 och Unicode baseras på teckenrepresentationer enligt Universial Multiple-Octet Coded Character Set, USC. USCs syfte är att täcka in alla skrifttecken som används i världen inklusive matematiska och andra symboler. USC finns i en 16 eller 31 bitars teckenrepresentation (USC-2 och USC-4). Unicode baseras på USC-2 och ISO/IEC 10646 på USC-4. Unicode version 1.1 är kompatibel med ISO/IEC 10464.

Många typer av kommunikationsprotokoll som exempelvis SMTP (används för att skicka e-post) och domännamn på Internet har restriktioner för vilka tecken som får användas. I fallet SMTP och domännamn används en 7 eller 8 bitars representation av tecken. Då UCS används i sådana fall går det inte bara att ta de första 7 eller 8 bitarna av UCS teckenrepresentation. USC Transformation Format, UTF är algoritmer som transformerar USC till en 7, 8 eller 16 bitars representation (UTF-7, UTF-8 och UTF-16). En XML-processor måste kunna tolka tecken enligt UTF-7 och UTF-8.

URI

En Uniform Resource Identifier, URI, är ett sätt att identifiera resurser på Internet som text, video, ljud, bilder eller program. URI är ett samlingsnamn för URL, URN (och URC). Grovt sett identifierar en URL (Uniform Resource Locator) en resurs via en absolut länk medan en URN (Uniform Resource Name) identifierar en resurs via en relativ länk. En URC (Uniform Resource Characteristics) definierar en mängd av attribut som kan användas för att beskriva en resurs. Den vanligast förekommande URIn är URL som används bland annat för att adressera webbsidor på Internet.

En typisk URI (URL) beskriver:

- Mekanismen för att komma åt en resurs.
- Den dator som resursen finns på.
- Namnet på resursen (filnamn) på datorn.

Exempelvis, följande URI:

```
http://www.w3.org/TR/REC-xml
```

Exemplet identifierar en fil som är åtkomlig genom webbprotokollet HTTP som finns på en dator vars namn är ”www.w3.org” i biblioteket ”TR” med filnamnet ”REC-XML”.

XML använder URI för att peka ut externa resurser, som en externt deklarerad DTD. XML-specifikationen kräver att en XML-processorer kan hantera URI enligt UTF-8s teckenrepresentation.

XML-processorn

En XML-processor används för att tolka och översätta ett XML-dokument så att exempelvis en applikation kan komma åt dokumentets innehåll. XML-processorer delas in i två grupper, *validerande* och *icke validerande*. Både en validerande och icke validerande processor måste kunna avgöra om ett dokument är korrekt formulerat enligt de krav som ställs av XML-specifikationen. En validerande processor måste även kunna avgöra och ett dokumentet uppfyller den grammatik som finns deklarerade i en DTD. För att lyckas med det så måste en validerande processor kunna läsa och behandla hela DTDn och alla externa tolkningsbara entiteter som används i dokumentet.

En icke validerande processor behöver bara kontrollera att dokumententiteten är syntaktiskt korrekt och den internt deklarerade DTDn (se DTD). Även om en icke validerande

processor inte behöver validera ett dokument, så måste den behandla alla deklarationer i den interna DTDn som inte är externa entiteter. Alla interna parameterentiteter måste användas till att normalisera attribut, ersätta entiteter med dess ersättningstext och knyta normalvärden till attribut. Att normalisera attribut betyder att alla attribut som använder parameterentiteter som normalvärde, måste normaliseras genom att byta ut entiteten med dess ersättningstexten.

DTD

En Document Type Definition, DTD, beskriver ett XML-dokuments grammatik som är deklarerad i ett dokuments dokumenttyp-deklaration (se Dokument typ deklaration). Eller DTDn är resultatet då en tolk har tolkat det som finns deklarerat i dokumenttyp-deklarations *interna* eller *externa* delmängd. Kort och gott, en DTD beskriver en grammatik och DTD:n finns deklarerad i ett dokuments dokumenttyp-deklaration. Det kan vara lite förvirrande till en början och samtidigt en olyckligt vald akronym, DTD, för både ett dokuments definition som dess deklaration. Vanligast är att man kort och gott kallar ett dokuments definition för ”DTD” och dess deklaration för just ”deklaration”.

Generellt sett beskriver en DTD ett dokuments logiska och fysiska struktur. Det vill säga, vilka element som en klass av dokument kan eller måste innehålla, hur elementen förhåller sig till varandra, varje elements eventuella attribut och ett dokuments eller DTDs entiteter.

Konkret sett beskriver en DTD ett antal element, attribut, entiteter och notationer. Det vill säga *element typ deklaration*, *attribut list deklaration*, *entitet deklaration* och *notation deklaration*. Element och attribut tillhör ett dokuments logisk struktur och entiteter och notationer tillhör dess fysiska struktur.

Elementtyp-deklaration

En elementtyp-deklaration beskriver ett element som ett XML-dokument kan innehålla. Varje element är unikt och kallas för en elementtyp. Deklarationen beskriver också vilka element en elementtyp kan innehålla, dessa elements inbördes ordning och hur många gånger de får förkomma i elementtypen. En elementtyp kan bara deklareras en gång, det vill säga det kan inte finns två element med samma namn. Varje element som en elementtyp är deklarerad till att innehålla måste i sin tur, på samma sätt, deklareras.

Deklarationsspecifikation
`Elementtyp-deklaration ::= '<!ELEMENT' namn innehållsspecifikation '>'`

Tabell 2. Illustrerar en elementdeklaration

En elementtyp identifieras av ett namn, som måste uppfylla kraven på namn enligt XML-specifikationen (se Restriktioner och specialtecken), och består av en innehållsspecifikation, som beskriver dess innehåll. Om innehållsspecifikationen är av typen elementinnehåll eller blandat innehåll består den av en lista, en lista omsluts av parenteser, som består av ett antal delar. Delarna i listan är antingen element, PCDATA eller listor, som i sig kan innehålla element, PCDATA eller listor, det vill säga listor i listor. Varje del separeras i listan med ett kommatecken eller med en ELLER-symbol (|). Kommatecken betyder att delarna måste förekomma i den ordningen i dokumentet som de förekommer i listan. ELLER symbol betyder att antingen den ena eller den andra delen får förkomma i dokumentet. Varje del kan anta en operator som beskriver hur många gånger varje del får förkomma i dokumentet. Dessa tecken är plus (+) för en eller flera gånger, stjärna (*) för noll eller flera gånger samt frågetecken (?) för noll eller en gång. Avsaknaden av operator betyder att delen måste ingå exakt en gång.

En innehållsspecifikation kan vara en av fyra typer:

- Elementinnehåll
- Blandat innehåll
- EMPTY
- ANY

Elementinnehåll

En elementtyp deklarerad till att innehålla andra element, *elementinnehåll*, är förälder till sitt innehåll. De element den innehåller blir då barn till elementtypen, det vill säga de element som finns i innehållsspecifikationen är barn till elementet som identifieras av elementtyp-deklarationens namn. Alla barn till en elementtyp måste inneslutas av dess förälder då det används i XML-dokument. Barnens inbördes ordning och hur många gånger de får förekomma beskrivs av innehållsspecifikationen.

Exempeldeklaration	Betydelse
`<!ELEMENT ADDRESSBOOK (PERSON+)>`	Elementet `ADDRESSBOOK` innehåller en eller fler `PERSON` element.
`<!ELEMENT PERSON (FIRSTNAME,LASTNAME, COMMMENT?)>`	Elementet `PERSON` är innehåller en `FIRSTNAME` följt av en `LASTNAME` följt av en valfri `COMMENT`.
`<!ELEMENT MAIL (FROM,TO+,(TEXT|IMAGE)+>`	Elementet `MAIL` innehåller en `FROM` följt av en eller fler `TO` följt av en eller flera `TEXT` eller `IMAGE`. Resultatet av den sista listan blir att `TEXT` och `IMAGE` kan förekomma samtidigt flera gånger, så länge minst ett av elementen förekommer minst en gång.

Tabell 3. Illustrerar element med elementinnehåll.

Blandat innehåll

En elementtyp deklarerad till att innehålla *blandat innehåll* kan förutom element också innehålla PCDATA. I övrigt gäller samma sak för elementtyper med blandat innehåll som för elementtyper med elementinnehåll.

Exempeldeklaration	Betydelse
`<!ELEMENT ADDRESSBOOK (PERSON|#PCDATA)*>`	Elementet `ADDRESSBOOK` innehåller noll eller fler `PERSON` element eller PCDATA. Ett element som definieras till att ha blandat innehåll måste ha ELLER separerat innehåll med noll eller fler operatorn (*) efter lista. Det vill säga av typer `(A|B|..|#PCDATA)*`.
`<!ELEMENT NAME (#PCDATA)>`	Ett element kan även definieras till att bara innehålla PCDATA. Detta är vanligtvis vad varje löv i ett XML-dokumentet är, det vill säga text.

Tabell 4. Illustrerar element med blandat innehåll.

EMPTY

En tom elementtyp kan inte innehålla någonting alls. Ett exempel på sådant element är HTMLs radbrytning <BR> med den skillnaden att då ett tomt element används i ett XML-dokument måste det avslutas med ett snedstreck, det vill säga <BR/>.

Exempeldeklaration	Betydelse
`<!ELEMENT BR EMPTY)>`	Elementet `BR` innehåller ingenting alls. Ett sådant element används i XML-dokumentet som ett avslutande element, som i det här fallet ` `.

Tabell 5. Illustrerar ett tomt element.

ANY

En ANY elementtyp kan innehålla valfria element och/eller PCDATA i valfritt antal eller ordning eller vara tomt. De eventuella element som används i en ANY elementtyp i XML-dokumentet måste dock själva vara deklarerade i DTDn.

Exempeldeklaration	Betydelse
`<!ELEMENT BODY ANY)>`	Elementet `BODY` får innehålla valfritt antal valfria element inklusive PCDATA. Dock så måste de element som förekommer i ett element av typ ANY också vara deklarerade.

Tabell 6. Illustrerar element med valfritt innehåll.

Attributlist-deklaration

En attributlist-deklaration syftar till att specificera namn, typ och eventuella normalvärden till ett attribut associerat till ett deklarerat element. Ett attribut kan endast specificeras i ett öppnande eller tomt märkord.

Deklarationsspecifikation
`Attributlist-deklaration ::= '<!ATTLIST' elementnamn (attributnamn attributtyp normalvärde?)* '>'`

Tabell 7. Illustrerar en attributdeklaration

En attributlist-deklaration associerar ett attribut till ett deklarerat element. Ett element kan ha ett eller flera attribut associerade till sig. Varje attribut identifieras med ett attributnamn och kan vara av olika attributtyp. Ett attribut kan ha ett normalvärde knutit till sig. Ett attributs normalvärden kan ha en egenskap, som #REQUIRED, #IMPLIED eller #FIXED. #REQUIRED betyder att elementets attribut alltid måste ha ett attributvärde. #IMPLIED betyder att elementets attribut inte behöver, men kan ha, ett attributvärde. Om ett attribut inte är definierat till att vara varken nödvändigt eller valfritt så kan ett element ha ett fast attributvärde #FIXED. Ett normalvärde deklarerat till att vara #FIXED behöver inte specificeras i dokumentet, utan knyts automatiskt till elementet av XML-processor till det värde specificerat i DTDn efter #FIXED. Ett attributnamn och attributvärde, med vissa undantag, måste uppfylla kraven på namn enligt XML-specifikationen (se Restriktioner och specialtecken).

Ett attribut kan vara en av tio möjliga attributtyper.

- Sträng
- Uppräkning
- ID
- IDREF
- IDREFS
- ENTITY
- ENTITIES
- NMTOKEN
- NMTOKENS
- NOTATION

Strängattribut

Ett attribut av typen *strängattribut* kan anta ett attributvärd som får innehålla ett valfritt antal tecken av valfri typ (se CDATA), med undantaget för tecknen ”<” och ”&” som måste refereras till med dess entiteter (`<` och `&`).

Deklarationsspecifikation	
`Strängattribut ::= '<!ATTLIST' elementnamn attributnamn 'CDATA' normalvärde? '>'`	
Exempeldeklaration	**Betydelse**
`<!ATTLIST PERSON SEX CDATA #IMPLIED>`	Elementet `PERSON` har attributet `NAME` som är av typ CDATA vars attributvärde kan, men behöver inte, förekomma.

Tabell 8. Illustrerar ett attribut av typ CDATA.

Uppräkning

Ett attribut av typen uppräkning kan anta ett av de möjliga attributvärdena specificerade i en attributvärdelista. Varje attributvärde måste uppfylla de krav som gäller för giltiga tecken enligt NMTOKEN-attribut (se NMTOKEN/NMTOKENS).

Deklarationsspecifikation	
`Uppräkning_attribut ::= '<!ATTLIST' elementnamn attributnamn '(' attibutvärde1 '|' ... '|' attributvärden ')' normalvärde? '>'`	
Exempeldeklaration	**Betydelse**
`<!ATTLIST ITEM COLOR (RED|GREEN|BLUE) 'RED'>`	Elementet `ITEM` har attributet `COLOR` som är av typ uppräkning. Det attributvärden som finns att välja mellan är `RED`, `GREEN` och `BLUE`, om inget anges är normalvärdet `RED`.

Tabell 9. Illustrerar ett attribut av typ uppräkning.

ID/IDREF/IDREFS

Attributtyperna ID, IDREF och IDREFS är nära relaterade till varandra. Ett element deklarerat till att ha ett ID-attribut måste ha ett attributvärde som är unikt i de dokument det förekommer. Ett ID-attributvärde är så kallad GID (Global IDentifier), det vill säga, ett värde som inte får förekomma mer än en gång i samma rymd. ID, IDREF och IDREFS-

attribut måste vara deklarerat till att vara nödvändig, #REQUIRED, eller valfri, #IMPLIED. Ett ID, IDREF och IDRERS-attributvärde måste uppfylla kraven på namn enligt XML-specifikationen (se Restriktioner och specialtecken).

Ett IDREF-attribut är ett ID-attributs motsvarighet. Ett element deklarerat till att ha ett IDREF-attribut måste ha ett attributvärde som matchar ett ID-attributvärde till ett annat element i samma dokument. Det vill säga ett IDREF måste refererar till ett befintligt ID.

IDREFS-attribut fungerar på samma sätt som IDREF med den skillnaden att ett IDREFS-attributvärden refererar till flera elements ID-attributvärde i samma dokument. Attributvärdena till ett IDREFS-attribut separeras med blanktecken, som mellanslag.

Deklarationsspecifikation
`ID-attribut ::= '<!ATTLIST' elementnamn attributnamn 'ID' normalvärde '>'`
`IDREF-attribut ::= '<!ATTLIST' elementnamn attributnamn 'IDREF' normalvärde? '>'`
`IDREFS-attribut ::= '<!ATTLIST' elementnamn attributnamn 'IDREFS' normalvärde? '>`

Exempeldeklaration	Betydelse
`<!ATTLIST PART PARTNUMB ID #REQUIRED>`	Elementet `PART` har attributet `PARTNUMB` som är av typ ID vars attributvärde måste förekomma.
`<!ATTLIST ITEM TYPE ID #REQURIED PARTS IDREFS #REQUIRED>`	Elementet `ITEM` har två attribut, `TYPE` och `PARTS`. Där `TYPE` är ett ID-attribut och `PARTS` är ett IDREFS-attribut, det vill säga attributet `PARTS` kan ha ett eller fler attributvärden där vardera refererar till ett ID-attributvärde.

Tabell 10. Illustrerar ett attribut av typ ID och IDREFS.

ENTITY/ENTITIES

Ett attribut av typ ENTITY måste ha ett attributvärde som matchar ett namn, som i sin tur identifierar en icke tolkningsbar entitet (se Entitet deklaration) deklarerad i samma DTD. Ett ENTITY-attribut kan användas till att låta ett attributvärde vara av en typ som inte är text, som en bild.

ENTITIES-attribut fungerar på samma sätt som ENTITY med den skillnaden att ett ENTITIES-attributs värden refererar till fler icke tolkningsbara entiteter, som separeras med blanktecken.

Deklarationsspecifikation
`ENTITY-attribut ::= '<!ATTLIST' elementnamn attributnamn 'ENTITY' normalvärde? '>'`
`ENTITIES-attribut ::= '<!ATTLIST' elementnamn attributnamn 'ENTITIES' normalvärde? '>'`

Exempeldeklaration	Betydelse
`<!ATTLIST PERSON IMAGE ENTITY>`	Elementet `PERSON` har attributet `IMAGE` som är av typ ENTITY. Ett attributvärde till ett attribut av typ ENTITTY måste referera till ett deklarerad entitet.

Tabell 11. Illustrerar ett attribut av typ ENTITY.

NMTOKEN/NMTOKENS

Ett NMTOKENs attributvärde har inte samma restriktiva regler vad beträffar de tecken som dess attributvärde får innehålla, till skillnad från kraven på namn enligt XML-specifikationen (se Restriktioner och specialtecken). Ett NMTOKENs attributvärde kan börja med de tecken som normalt gäller för ett namns andra tecken och framåt. Däremot får ett NMTOKENs attributvärde inte börja med teckenkombinationen ”xml” eller innehålla blanktecken.

NMTOKENS attributs värden innehåller flera NMTOKEN separerade med blanktecken, på samma sätt som för IDREFS och ENTITIES.

Deklarationsspecifikation
`NMTOKEN-attribut ::= ’<!ATTLIST’ elementnamn attributnamn ’NMTOKEN’ normalvärde? '>'`
`NMTOKENS-attribut ::= '<!ATTLIST' elementnamn attributnamn 'NMTOKENS' normalvärde? '>'`

Exempeldeklaration	Betydelse
`<!ATTLIST ADRESS POSTALCODE NMTOKEN>`	Elementet `ADRESS` har attributet `POSTALCODE` som är av typ NMTOCKEN.

Tabell 12. Illustrerar ett attribut av typ NMTOKEN.

NOTATION

Ett attribut av typ NOTATION består av en lista med ett eller flera attributvärden, där varje attributvärde måste matcha ett namn som identifierar en notation (se Notation deklaration) deklarerade i DTDn. Ett NOTATION-attribut liknar mycket ett ENTITY-attribut då båda refererar till en extern resurs. Skillnaden är det som är skillnaden mellan en notation och en entitet.

Deklarationsspecifikation
`NOTATION-attribut ::= '<!ATTLIST' elementnamn attributnamn 'NOTATION' '(' attibutvärde1 '|' ... '|' attributvärden ')' normalvärde? '>'`

Exempeldeklaration	Betydelse
`<!ATTLIST DOCUMENT FORMAT NOTATION (PSCRIPT,PDF)>`	Elementet `DOCUMENT` har attributet `FORMAT` som är av typ NOTATION och kan ha attributvärdet `PSCRIPT` eller `PDF`.

Tabell 13. Illustrerar ett attribut av typ NOTATION.

Entitetdeklaration

En entitet är ett typ av makro som ersätter sig själv med det de refererar till då det används i ett dokument. Det en entitet refererar till kallas för en resurs, som ett tecken, ett dokument eller en bild. En entitet kan referera till något som tillhör XML, som en text, då det kallas för en tolkningsbar entitet. Om det refererar till något som inte tillhör XML, som en bild, då kallas det för en icke tolkningsbar entitet.

Varje icke tolkningsbar entitet associerar till en notation vilket i sin tur refererar till resursen (se Notation deklaration). XML bryr sig inte om innehållet av en icke tolkningsbar entitets resurs. Hur en sådan resurs skall hanteras är upp till den applikation som använder dokumentet, som en webbläsare som visar bilderna som är specificerade i ett HTML-dokument.

Tolkningsbara entiteter kan vara av två typer, generella eller parameter entiteter (se Tabell 14). En generell entitet används i ett XML-dokument medan parameterentiteter används i en DTD. Varje tecken i Unicode och ISO/IEC 10646 kan refereras till via en *tecken entitet* med dess hexadecimala eller decimala värde, både i XML-dokumentet och DTDn.

Referensspecifikation

```
Generellentitet-referens ::= '&' entitetsnamn ';'
Parameterentitet-referens ::= '%' entitetsnamn ';'
Teckenentitet-referens ::= '&' ( '#x' hexadecimalt_värde | '#'
  decimalt_värde ) ';'
```

Tabell 14. Illustrerar hur entiteter används i ett dokument.

Vissa tecken är reserverade av XML-specifikationen. Dessa tecken har fördefinierade entiteter deklarerade av XML-processorn. Dessa är tecknen för ”&”, ”<”, ”>”, ”””” och ”’” vars entiteter är `&`, `&It;`, `>`, `&quot;` respektive `&apos;`.

En tolkningsbar entitets innehåll kan antingen vara *intern* eller *extern*. En intern entitet definierar en ersättningstext direkt i deklarationen medan en extern entitet refererar till en annan entitet som, i sin tur definierar en ersättningstext. En extern entitet kan vara då ett dokument refererar till ett annat dokuments innehåll, det vill säga för att bygga upp ett dokuments fysiska struktur.

Deklarationsspecifikation

```
Generellentitet ::= '<!ENTITY' namn ( '"' ersättningstext '"' |
  nyckelord ( '"' identifierare '"' )? '"' URI '"') '>'
Parameterentitet ::= '<!ENTITY' '%' namn ( '"' ersättningstext '"' |
  nyckelord ( '"' identifierare '"' )? '"' URI '"' ) '>'
```

Tabell 15. Illustrerar hur tolkningsbara entiteter deklareras.

En intern och extern entitetsdeklaration (se Tabell 15) innehåller ett namn som identifierar entiteten, vilket är det namn som användas i dokumentet eller DTDn för att referera till entiteten. En intern entitet deklarerar dess ersättningstext mellan citationstecken (”). Då en extern entitet deklareras åtföljs namnet av ett nyckelord som antingen kan vara SYSTEM eller PUBLIC. SYSTEM betyder att entiteten är lokal, det vill säga tillhör systemet, medan PUBLIC betyder att entiteten är tillgänglig för alla. Nyckelordet PUBLIC åtföljs av en publik identifierare som innehåller information om entiteten är en ISO standard, vem som äger den, vilken typ av entitet det är, namnet på den och vilket språk den tillhör. Den publika identifieraren åtföljs av en URI som pekar ut entiteten. Nyckelordet SYSTEM åtföljs bara av en URI. Både den publika identifieraren och URI måste inneslutas av citationstecken (”).

En icke tolkningsbar entitet kan bara vara generell och extern, det vill säga den kan bara användas i ett XML-dokument och referera till en extern resurs. Detta då resursen i sig inte behöver vara XML och då inte heller kan deklareras i XML.

Deklarationsspecifikation

```
Generellentitet ::= '<!ENTITY' namn nyckelord ( '"' identifierare '"'
  )? '"' URI '"' 'NDATA' resursnamn ) '>'
```

Tabell 16. Illustrerar hur icke tolkningsbara entiteter deklareras.

En resurs som en icke tolkningsbar entitet (se Tabell 16) pekar ut måste definieras som NonDATA, NDATA, och måste ha ett resursnamn som matchar ett namn för en deklarerat notation (se Notation deklaration).

Exempeldeklaration	Betydelse
`<!ENTITY SOS 'Simon Oscarsson'>`	Entiteten `SOS` är generell och intern vars ersättningstext är `Simon Oscarsson`. Det vill säga om entiteten `&SOS;` används i ett dokument ersätts den med texten `Simon Oscarsson`.
`<!ENTITY % AttList '(RED|GREEN|BLUE)'>`	Parameterentiteten `AttList` är intern vars ersättningstext är `(RED|GREEN|BLUE)`. Det vill säga om entiteten `%AttList;` används i DTD ersätts den med texten `(RED|GREEN|BLUE)`.
`<!ENTITY SIMON SYSTEM 'simon.gif' NDATA gif>`	Entiteten `SIMON` är extern och icke tolkningsbar vars värde är en bild (`simon.gif`). En sådan entitet kan användas som attributvärde till ett attribut av typ entitet.
`<!ENTITY A SYSTEM 'Address1.xml'`	Entiteten `A` är extern och tolkningsbar vars värde är ett XML-dokument (`Address1.xml`).

Tabell 17. Illustrerar olika entitettyper.

Notationdeklaration

En notationdeklaration är till för att informera en XML-processor eller dess klientapplikation om en extern resurs, som någon typ av hjälpapplikation som exempelvis kan visa bilder. En deklarerad notation kan refereras till av ett attribut eller en entitet.

En notationsdeklaration identifieras och refereras till genom ett namn. I övrigt definieras den externa resursen på samma sätt som för en extern entitet med den skillnaden att en publik identifierare kan, men behöver inte, peka ut resursen med en URI om systemet som, XML-processorn verkar i, kan lokalisera resursen med dess publika identifierare.

Deklarationsspecifikation
`Notation-deklaration ::= '<!NOTATION' namn nyckelord ( '"' identifierare '"' )? '"' URI '"' '>'`

Exempeldeklaration	Betydelse
`<!NOTATION PSCRIPT SYSTEM 'GSview.exe'>`	Notationen `PSCRIPT` refererar till en hjälpapplikation `Gsview.exe`. En sådan notation kan användas som ett attributvärde till ett attribut av typ notation.

Tabell 18. Illustrerar en NOTATION.

Villkorssektion

En *villkorssektion* är en del av en dokumenttyp-deklarations externa delmängd som inkluderas eller ignoreras från den logiska strukturen av en DTD. Det vill säga, en villkorssektion kan låta vissa delar av en DTD tillhöra DTDn och andra inte, utan att för det ta bort de delar som inte skall tillhöra DTDn. En villkorssektion kan innehålla fullständiga deklarationer, kommentarer, processinstruktioner och nästlade villkors sektioner.

Om nyckelordet av villkorssektionen är INCLUDE så tolkas innehållet av sektionen som en del av DTDn. Om nyckelordet av villkorssektionen är IGNORE så är innehållet av sektionen inte en logisk del av DTDn. Ett område som är definierat till att ignoreras måste dock tolkas av processorn för att upptäcka eventuella nästlade villkorssektioner och slutet av

området. Om en villkorssektion med nyckelordet INCLUDE finns inom en villkorssektion med nyckelordet IGNORE kommer hela området att ignoreras.

Deklarationsspecifikation

`Villkorssektion ::= inkludera | ignorera`
`inkludera ::= '<![INCLUDE' '[' extern_delmängd ']]>'`
`ignorera::= '<![IGNORE '[' extern_delmängd ']]>'`

Exempeldeklaration	Betydelse

`<!ENTITY % draft 'INCLUDE'>`
`<!ENTITY % final 'IGNORE'>`
`<![%draft;[`
`<!ELEMENT BOOK`
`(COMMENT*, TITLE, BODY)>`
`]]>`
`<![%final;[`
`<!ELEMENT BOOK (TITLE, BODY)>`
`]]>`
	Först deklareras två parameterentiteter, `draft` och `final`. Genom att använda villkorssektioner med parameterentiteterna kan delar av DTD enkelt inkluderas eller ignoreras.

Tabell 19. Illustrerar hur villkorssektioner används.

XML-dokument

Ett XML-dokument är en tolkningsbar extern entitet som i sig kan referera till andra externa entiteter (XML-dokument) som tillsammans bildar dokumentets totala struktur, dess fysiska struktur. En av entiteterna agerar som rot och kallas då för dokumententitet och fungerar som startpunkt för en XML-processor. Dokumententiteten skiljer sig från de andra entiteterna genom att den inte har ett namn. XML-specifikationen beskriver inte hur en processor skall lokalisera en dokumententitet. Vanligtvis förekommer dokumententiteten som en parameter i processorns indata, som ett filnamn till en fil som innehåller dokumententiteten.

Varje entitet som representerar ett dokument har ett innehåll som består av tre delar, en prolog, ett rotelement och en epilog. Prolog kan bestå av ett antal processinstruktioner och kommentarer. Prologen bör innehålla en XML-deklaration som innehåller information om dokumentets typ. En prolog kan, men behöver inte, innehålla en dokumenttyp-deklaration. Ett dokument som deklarerar en dokumenttyp måste vara validerbar, annars förutsätts det vara korrekt formulerat (se Vad är XML). Ett rotelement innehåller själva dokumentet, dess logiska struktur, och består av ett antal element med eventuella attribut och entiteter. Epilogen kan bestå av ett antal processinstruktioner och kommentarer.

Processinstruktioner

Processinstruktioner ger möjligheten att i ett XML-dokument förmedla instruktioner till en specifik applikation. En processinstruktion påbörjas med strängen ”<?” och avslutas med ”?>” och identifierar en målapplikation och ett kommando. Målapplikationen ”xml” är reserverad av XML-specifikationen. En processinstruktion anses inte vara en del av ett dokuments innehåll, det som återfinns mellan de två frågetecknen ”?”, utan förmedlas vidare till målapplikationen av tolken. En processinstruktion kan förekomma var som helst i ett dokument, det vill säga i prologen, i rotelementet eller i epilogen. Däremot gäller samma sak för processinstruktioner som för kommentarer, de kan inte förekomma innanför ett annat märkord.

Exempeldeklaration	Betydelse
`<?xml version='1.0'?>`	Exempel på en processinstruktion som deklarerar ett dokuments XML-deklaration.

Tabell 20. Illustrerar en processinstruktion

XML-deklaration

En XML-deklaration specificerar vilken typ av dokument det är. Det är inte nödvändigt men brukligt med en XML-deklaration, speciellt då XML-specifikationen antar nya versioner.

En XML-deklaration bör specificera vilken XML-version dokumentet använder sig av genom att ange dess versionsnummer. Det finns möjlighet att ange vilken typ av teckentabell dokumentet använder sig av. En XML-processor måste åtminstone kunna tolka tecken enligt UTF-8 och UTF-16 av 10646 standarden och Unicode (se Teckenrepresentation). Om inget anges antas dokumentets tecken vara av UTF-8 eller UTF-16. Ett dokument kan vara fristående vilket betyder att inga externa entiteter används, om externa entiteter finns deklarerade i dokumentet kommer dessa att ignoreras av XML-processorn. Entiteter som är fördefinierade av XML-specifikationen går att referera till även om dokumentet är specificerat att vara fristående (se Entitet deklaration). Om inget anges så antas XML-dokumentet inte vara fristående.

I övrigt så gäller, enligt XML-specifikationen, att i XML-deklarationen använda gemena bokstäver, det vill säga att målapplikation och attribut skrivs med gemena bokstäver.

Deklarationsspecifikation
`XML-deklaration ::= '<?xml' versionsinformation? teckenhantering? fristående? '>'`
`versionsinformation ::= 'version' '=' ''' versionsnummer '''`
`teckenhantering ::= 'encoding' '=' '"' teckentabell '"'`
`fristående ::= 'standalone' '=' '"'  ( 'yes' | 'no ' ) '"'`

Exempeldeklaration	Betydelse
`<?xml version='1.0' encoding='UTF-8' standalone='yes'?>`	XML-deklarationen deklarerar ett dokument skrivet i version 1.0 av XML som använder en teckenrepresentation enligt UTF-8 samt att dokumentet inte använder externt deklarerade entiteter.

Tabell 21. Illustrerar en XML-deklaration.

Dokumenttyp-deklaration

En dokumenttyp-deklarationen (Document Type Declaration) deklarerar ett XML-dokuments element, attribut, entiteter och notationer. En dokumenttyp-deklaration kan antingen refererar till en intern eller extern delmängden. En intern delmängd (internal subset) betyder att DTDn finns deklarerad internt i XML-dokumentet. En extern delmängd (external subset) betyder att DTDn refereras till en extern entitet, vanligtvis en fil. Ett XML-dokument kan deklarera både en intern och en extern delmängd, då den resulterande DTDn är summan av de båda delmängderna. Vid multipel deklaration måste varje delmängd vara distinkt, det vill säga snittet av det två måste vara tom.

Varje dokumenttyp-deklaration har ett namn (se Tabell 22) som måste vara detsamma som namnet på rotelementet. Namnet åtföljs i en internt deklarerad delmängd av klammerparanteser ([]) som innesluter DTDn. I en externt deklarerad delmängd åtföljs namnet av ett nyckelord som antingen kan vara SYSTEM eller PUBLIC. SYSTEM betyder att DTDn är lokal, det vill säga tillhör systemet, medan PUBLIC betyder att DTDn är tillgänglig för alla. Exempelvis kan ett stort företag som har standardmallar för olika typer av

XML-dokument låta dessa vara publika för alla berörda på företaget. Nyckelordet PUBLIC åtföljs av en publik identifierare som innehåller information om DTDn är en ISO standard, vem som äger den, vilken typ av DTD det är, namnet på den och vilket språk den tillhör. Den publika identifieraren åtföljs av en URI som pekar ut DTDn. Nyckelordet SYSTEM åtföljs bara av en URI. Både den publika identifieraren och URI måste inneslutas av citationstecken (").

Deklarationsspecifikation	
`Dokumenttyp-deklaration ::= '<!DOCTYPE' namn ('[' DTD ']' | nyckelord ( '"'identifierare''' )? '"'URI''' ) '>'`	
Exempeldeklaration	**Betydelse**

`<?xml version='1.0'?>`
`<!-- External subset -->`
`<!DOCTYPE ADDRESSBOOK SYSTEM`
`'Addressbook.dtd'>`
`<ADDRESSBOOK>`
`...`
`</ADDRESSBOOK>`
	Exempel på en dokumenttyp-deklaration som deklarerar en extern delmängd.

`<?xml version='1.0'?>`
`<!-- Internal subset -->`
`<!DOCTYPE ADDRESSBOOK [`
`<ELEMENT ADDRESSBOOK`
`(PERSON)+>`
`<ELEMENT PERSON`
`(FIRSTNAME, LASTNAME)>`
`]>`
`<ADDRESSBOOK>`
`...`
`</ADDRESSBOOK>`
	Exempel på en dokumenttyp-deklaration som deklarerar en intern delmängd.

Tabell 22. Illustrerar en dokumenttyp-deklaration.

Rotelement

Ett XML-dokument få endast innehålla en rot, ett så kallad rotelement. Rotelementet innehåller alla andra element i dokumentet och kan inte vara tomt. Alla element i ett rotelement är barn till rotelementet, som är förälder till sina barn.

Ett exempel som illustrerar ett rotelement:

```
<?xml version='1.0'?>
<!-- Rootelement -->
<ADDRESSBOOK>
   ...
</ADDRESSBOOK>
```

XSL

XSL, eXtensible Stylesheet Language, är ett W3C projekt som är under pågående undersökning. En XSL-formatmall används för att presentera innehållet i ett XML-dokument för en användare i något applikationsberoende format, som exempelvis HTML. XSL är definierat i XML, det vill säga XSL är en instans av XML, ett XML-dokument, som innehåller element, attribut och entiteter deklarerade i en DTD som tillsammans beskriver XSL. XSL-specifikationen består av två delar, den ena delen kallas *trädkonstruktion* och den andra för *formatobjekt*. Den här rapporten kommer att ge en kort introduktion till formatobjekt medan trädkonstruktion beskrivs mer utförligt.

I och med att XSL är under pågående undersökning är denna beskrivningen inte fullständig. Den framtida XSL-specifikationen kommer förmodligen innehålla mer funktionalitet än vad den här rapporten beskriver. Ett exempel på sådan funktionalitet är att inkludera skriptspråk (exempelvis JScript eller Java Script) som exekveras i XSL samt speciella metoder utvecklade