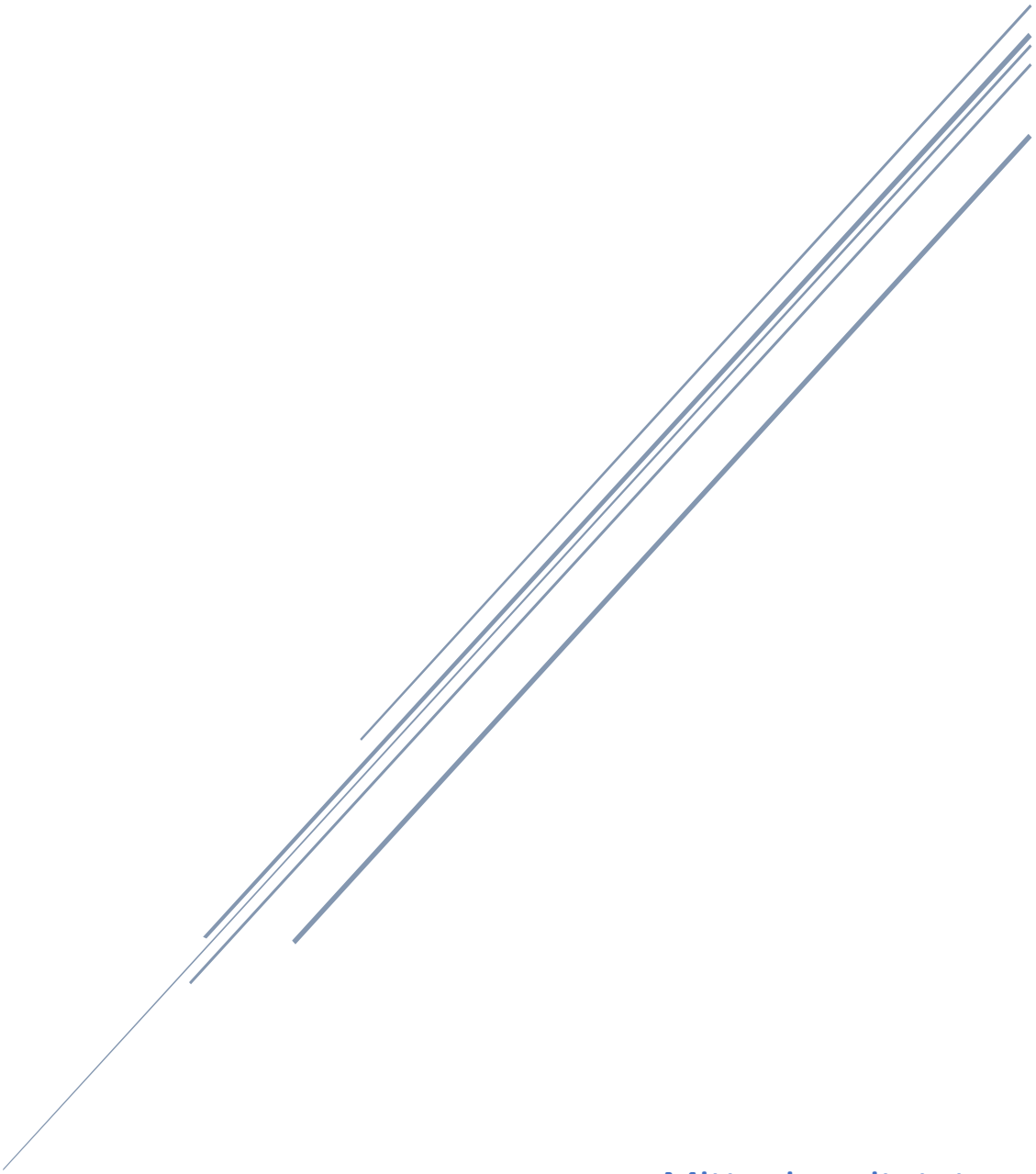
DTD



Mittuniversitetet
XML

Innehållsförteckning

1 Dokumenttypsdefinition (DTD)

Sammanfattning: I de två första lektionerna tittade vi på hur välutformade XML-dokument skapades. Det vill säga XML-dokument som följer de syntaxregler som finns i XML-standarden. Vi ska nu börja skapa giltiga XML-dokument. Detta innebär att dokumentet förutom att vara välutformad även följer de regler som finns i ett schema. I denna lektion tittar vi på hur en DTD fungerar och hur den skapas. I nästa lektion kommer vi in på XML Schema.

1.1 Inledning

I denna och nästa lektion ska vi titta närmare på validering av XML-dokument. Validering innebär att ett XML-dokument kontrolleras mot en fastställd struktur som beskrivs i ett *schema*. Det finns tre vanliga sätt att beskriva denna struktur, nämligen med Document Type Definition (DTD), XML Schema (XSD) och Relax NG (RNG). I resten av denna lektion tittar vi på DTD för att i nästa lektion gå in på XSD.

DTD står för Document Type Definition (svensk översättning blir dokumenttypsdefinition) och skrivs i SGML till skillnad mot XML Schema som skrivs i XML. Detta gör att en DTD ser väldigt olikt ut i förhållande till XML och XML Schema, men det är ett enkelt språk som det är lätt att sätta sig in i och förstå.

När ett XML-dokument skapas är det lät hänt att man skriver fel på till exempel elementnamnen. Tänk dig ett XML-dokument över ett personregister och där nya personer registreras fortlöpande. Ett element har vi valt att kalla för `telefon` och som används för att registrera telefonnummer med. Här kan det vara lätt hänt att vi för enstaka poster skriver `tlf` i stället för `telefon`, framför allt om flera olika personer jobbar med XML-dokumentet:

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

<personregister>
 <person>
  <namn>Karin Nordin</namn>
  <telefon>11111111</telefon>
  <civilstatus>Gift</civilstatus>
  <titel>Chefsredaktör</titel>
 </person>
 <person>
  <namn>Lisa Olsson</namn>
  <tlf>22332233</tlf>
  <civilstatus>Sambo</civilstatus>
  <titel>Sjuksköterska</titel>
 </person>
</personregister>
```

1.2

XML-dokumentet ovan kommer att vara välutformat, men när till exempel en applikation läser och processar innehållet kommer elementen inte att registreras som samma typ av information, något som vi ursprungligen önskade. För att underlätta arbetet med att registrera nya personer och för att undvika fel kan vi använda en DTD. I en DTD deklarerar vi de element och attribut som är giltiga att använda i XML-dokumentet.

Om det finns flera olika personer som arbetar med samma XML-dokument är det många gånger bra att en DTD skapas först. Då får alla personer en översikt över hur elementen ska vara strukturerat i XML-dokumentet.

Med en DTD sägs inget om vilka datatyper de olika elementen kan innehålla utan bara hur själva strukturen ska vara. Detta kan vara information om vilka element som kan användas och vilka element som måste ha attribut. I en DTD kan även andra typer av deklarationer användas, men det kommer vi tillbaka till längre fram i lektionen.

1.2.1 Grundläggande villkor för ett giltigt XML-dokument

Ett XML-dokument måste vara välutformat (se lektion 1) för att kunna definieras som ett XML-dokument. Ett välutformat XML-dokument som dessutom ska vara giltigt måste uppfylla ett av följande alternativ:

- XML-dokumentets prolog inkluderar en dokumenttypsdefinition som definierar innehållet och strukturen i XML-dokumentet.
- XML-dokumentet överensstämmer med innehåll och struktur som definieras i ett XML Schema. Schemat finns i en separat fil.

1.3 Type Definition

En DTD definierar element och innehåller information om elementen och/eller om dokumentet generellt. En DTD kan inkluderas i ett XML-dokument eller så kan den lagras i en separat fil.

Vi kommer nu att skapa en DTD för exemplet ovan med personregistret (som rättar till ”felen” med elementet `telefon`):

```
01 <?xml version="1.0" encoding="UTF-8"?>
02
03   <!DOCTYPE personregister [
04   <!ELEMENT personregister (person+)>
05   <!ELEMENT person (namn, telefon, civilstatus, titel)>
06   <!ELEMENT namn (#PCDATA)>
07   <!ELEMENT telefon (#PCDATA)>
08   <!ELEMENT civilstatus (#PCDATA)>
09   <!ELEMENT titel (#PCDATA)>
10   ]>
11
12      <personregister>
13      <person>
14      <namn>Karin Nordin</namn>
15      <telefon>11111111</telefon>
16      <civilstatus>Gift</civilstatus>
17      <titel>Chefsredaktör</titel>
18      </person>
19      <person>
20      <namn>Lisa Olsson</namn>
21      <telefon>22332233</telefon>
22      <civilstatus>Sambo</civilstatus>
23      <titel>Sjuksköterska</titel>
```

```
24      </person>
25      </personregister>
```

På rad 1 och raderna 12-25 ser du det vanliga XML-dokumentet. Vi ska nu koncentrera oss på själva dokumenttypsdefinitionen på rad 3-10. Vi går igenom denna steg för steg nedan.

Den startar med en `DOCTYPE`-deklaration på rad 3. Denna kopplar samman vår DTD med XML-dokumentet. `DOCTYPE`-deklarationen måste stå i XML-dokumentets prolog, med andra ord före rotelementet. Deklarationen startar med ett utropstecken (`!`) följt av ordet `DOCTYPE`. Därefter kommer namnet på rotelementet och innanför hakparenteserna `[]`.finns hela dokumenttypsdefinitionen nästlad. Deklarationen ser ut som följer:

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

I en DTD deklareras som sagt element och attribut, men i exemplet ovan finns endast deklarationer av element. En deklaration av ett element görs enligt:

```
<!ELEMENT namnPåElementet (innehåll i elementet)>
```

Lägg märke till utropstecknet före `ELEMENT`. Den första deklarationen av element i exemplet ovan är av rotelementet (rad 4):

```
<!ELEMENT personregister (person+)>
```

Här kan du se att innehållet för detta element är `person+`. Detta betyder att innanför elementet `personregister` kan vi nästla en eller flera `person`-element. Hade vi inte tagit med tecknet `+`, hade det varit möjligt att ta med ett och endast ett `person`-element innanför elementet `personregister`.

Nästa element som deklareras är elementet `person` (rad 5):

```
<!ELEMENT person (namn, telefon, civilstatus, titel)>
```

Detta betyder att elementet `person` ska ha följande barnelement och att de måste förekomma i exakt denna ordning och exakt en gång: `namn, telefon, civilstatus, titel`. I kapitel 1.4 ska vi titta på andra alternativ för att ange vilka och hur många barnelement ett element kan ha.

För barnelementen till `person` har vi angett att innehållet ska vara av typen `PCDATA` (parsed character data):

```
<!ELEMENT namn (#PCDATA)>
<!ELEMENT telefon (#PCDATA)>
<!ELEMENT civilstatus (#PCDATA)>
<!ELEMENT titel (#PCDATA)>
```

Alla dessa element kan med andra ord endast innehålla text och inga barnelement. Lägg märke till att vi använder `PCDATA` oavsett vilken datatyp elementets innehåll ska ha. Det görs ingen skillnad på om innehållet ska vara text, tal, datum eller liknande.

1.3.1 Extern DTD

I exemplet ovan hade vi vad som kallas för en intern DTD. Det betyder att hela dokumenttypsdefinitionen står direkt i själva XML-dokumentet. En DTD kan även skrivas i en separat fil. Vi kan ta vår DTD från exemplet ovan och spara i en fil med till exempel namnet `personregister.dtd`:

```
<!ELEMENT personer (person+)>
<!ELEMENT person (namn, telefon, civilstatus, titel)>
<!ELEMENT namn (#PCDATA)>
<!ELEMENT telefon (#PCDATA)>
<!ELEMENT civilstatus (#PCDATA)>
<!ELEMENT titel (#PCDATA)>
```

I vårt XML-dokument måste vi nu ändra på `DOCTYPE`-deklarationen. Vi använder nyckelordet `SYSTEM` och därefter namnet på filen (vi utgår från att `personregister.dtd` ligger i samma katalog som XML-dokumentet):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE personregister SYSTEM "personregister.dtd">

<personregister>
 <person>
  <namn>Karin Nordin</namn>
  <telefon>11111111</telefon>
  <civilstatus>Gift</civilstatus>
  <titel>Chefsredaktör</titel>
 </person>
 <person>
  <namn>Lisa Olsson</namn>
  <telefon>22332233</telefon>
  <civilstatus>Sambo</civilstatus>
  <titel>Sjuksköterska</titel>
 </person>
</personregister>
```

Om du väljer att lagra din DTD i en annan mapp än XML-dokumentet måste sökvägen anges. Vi kan tänka oss att vi lagrar `personregister.dtd` i katalogen `dtd` under samma katalog som XML-dokumentet ligger i. I `DOCTYPE`-deklarationen måste vi då skriva:

```
<!DOCTYPE personregister SYSTEM "dtd/personregister.dtd">
```

Vi kan även använda en URL som pekar ut vår DTD (denna URL existerar inte):

```
<!DOCTYPE personregister SYSTEM "http://miun.se/xml/personregister.dtd">
```

Ett alternativ till `SYSTEM` är att använda `PUBLIC`. Den används vid officiella namn på resurser. Vi använde den i förra lektionen för att knyta XHTML-dokumentet med den officiella standarden för XHTML.

Fördelen med att använda en extern DTD är kanske uppenbar. Vi kan nu använda en och samma DTD för många XML-dokument.

Om du använder några särsvenska tecken i en extern DTD måste XML-deklarationen, där vilken teckenkodning som ska användas, finnas med:

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

1.4 Andra deklarationer i en DTD

Fram till nu har vi tittat på hur element deklareras i en DTD. Utöver denna typ av deklaration finns det tre till. Nedan listas de fyra olika typerna tillsammans med exempel:

1. `ELEMENT` för att deklarera element:

   ```
   <!ELEMENT pris (#PCDATA)>
   ```

2. `ATTLIST` för att deklarera attribut för ett visst element:

   ```
   <!ATTLIST pris valuta CDATA #REQUIRED>
   ```

3. `ENTITY` för att deklarera entiteter (som vi tittade på i lektion 2):

   ```
   <!ENTITY MIUNOSD "Mittuniversitetet i Östersund">
   ```

4. `NOTATION` för att deklarera externt innehåll som inte kommer att tolkas (detta kommer vi inte att titta på i kursen, men listas här för översiktens skull):

   ```
   <!NOTATION jpg SYSTEM "jpgviewer.exe">
   ```

De tre första deklarationerna ska vi nu ta en närmare titt på.

1.4.1 ELEMENT

Ett element kan ha olika typer av innehåll. I exemplet ovan var innehållet antingen andra element eller ren `PCDATA`. Nedan följer en lista över de olika typer av innehåll ett element kan ha och hur detta deklareras:

- `EMPTY` – ett tomt element, elementet kan varken ha barnelement eller `PCDATA`:

  ```
  <! ELEMENT bild EMPTY>
  ```

  Här har vi ett `bild`-element som inte får ha något innehåll, men det kan ha flera attribut. Attributen måste i så fall deklareras. Elementet i ett XML-dokument kan se ut så här:

  ```
  <bild />
  ```

- elementnamn – detta är alltså ett element som bara har barnelement. `person`-elementet nedan är ett exempel på detta:

  ```
  <!ELEMENT person (namn, telefon, civilstatus, titel)>
  ```

- `PCDATA` – elementet har bara `PCDATA` som innehåll. `namn`-elementet nedan är ett exempel på detta:

```
<! ELEMENT namn (#PCDATA)>
```

- blandat – innehållet är en blandning av barnelement och `PCDATA`. Ett exempel på detta i ett XML-dokument kan vara:

```
<namn>Namn:
   <förnamn>Karin</förnamn>
   <efternamn>Nordin</efternamn>
</namn>
```

Här ser du att vi har lagt in `Namn:` som innehåll i elementet `namn` tillsammans med de andra barnelementen. Detta kallas för blandat innehåll. Deklarerat i en DTD ser det ut så här:

```
<! ELEMENT namn (#PCDATA | förnamn | efternamn)* >
```

Här måste vi starta den så kallade innehållsmodellen med `#PCDATA` (denna måste stå först när blandat innehåll används) innan vi listar vilka barnelement som kan användas. Lägg märke till att vi har använt `|` mellan de olika typerna av innehåll. Detta betyder *eller*. Innehållet kan alltså vara `#PCDATA`, `förnamn` eller `efternamn`. Tecknet `*` efter parentesen betyder ingen, ett eller många. Vi kan med andra ord ha så många av de olika innehållstyperna som vi vill. Detta kommer vi tillbaka till snart.

- `ANY` – lämnar det öppet vad innehållet i elementet kan vara. Här kan vi alltså ha vad som helst som innehåll så länge som det är deklarerat.

```
<!ELEMENT allt_möjligt ANY>
```

I listan ovan såg du att vi kan använda tecknet `,` eller `|` för att skilja mellan olika innehåll. När vi använder tecknet `,` specificerar vi ordningsföljden, men om tecknet `|` används för att skilja innehållet kan endast ett av elementen användas. För att illustrera detta kan vi använda exemplet från ovan:

```
<!ELEMENT person (namn, telefon, civilstatus, titel)>
```

Detta betyder att elementet `person` kan ha `namn, telefon, civilstatus` och `titel` som innehåll i denna bestämda ordningsföljd. Nu kommer det till exempel inte gå att använda `titel` före c`ivilstatus` i XML-dokumentet.
Om vi i stället använder `|` för att skilja dessa element kommer det att innebära något annat:

```
<!ELEMENT person (namn | telefon | civilstatus | titel)>
```

Nu kan endast ett av elementen i listan användas som barnelement till `person`. Vilket element är helt valfritt. Om vi vill kunna använda flera element i vilken ordning som helst skriver vi:

```
<!ELEMENT person (namn | telefon | civilstatus | titel)* >
```

Här la vi till tecknet `*` i slutet. Då kan vi använda så många av de olika elementen som helst, i vilken ordning som helst.

För att komplicera det hela ytterligare kan vi skriva följande:

```
<!ELEMENT person (namn, telefon, (civilstatus | titel)) >
```

Innehållet i elementet `person` kan nu vara barnelementen `namn` och `telefon` följt av antingen `civilstatus` eller `titel`, i denna ordningsföljd.

Operatorer

Genom att använda olika operatorer kan vi bestämma antalet av de olika elementen:

? – innebär valbart. Antingen använder man elementet eller så använder man inte elementet. Om man väljer att använda elementet kan man bara använda det en gång.

* - innebär att man kan använda elementet ingen, en eller flera gånger.

+ - innebär att elementet måste användas minst en gång, men kan användas många gånger.

Om vi vill att man ska kunna registrera flera telefonnummer kan vi skriva enligt följande:

```
<!ELEMENT person (namn, telefon+, civilstatus, titel)>
```

Nu kan vi registrera så många telefonnummer som vi vill. Alla `telefon`-element måste dock komma efter elementet `namn` och före elementet `civilstatus`. Ett giltigt XML-dokument ser ut så här:

```
<person>
 <namn>Karin Nordin</namn>
 <telefon>11111111</telefon>
 <telefon>55225522</telefon>
 <telefon>90909090</telefon>
 <civilstatus>Gift</civilstatus>
 <titel>Chefsredaktör</titel>
</person>
```

Följande XML-dokument kommer däremot inte att vara giltigt då elementen `telefon` inte är samlat på rätt ställe:

```
<person>
 <namn>Karin Nordin</namn>
 <telefon>11111111</telefon>
 <civilstatus>Gift</civilstatus>
 <titel>Chefsredaktör</titel>
 <telefon>55225522</telefon>
 <telefon>90909090</telefon>
</person>
```

1.4.2 ATTLIST

I kapitel 1.4 i listan över de olika typerna av deklarationer visades följande exempel på en deklaration av attribut:

```
<!ATTLIST pris valuta CDATA #REQUIRED>
```

Genom denna attributdeklaration kan vi ha följande element:

```
<pris valuta="kr">150</pris>
```

I en attributdeklaration anger vi namnet på elementet som ska ha attributet, namnet på attributet, typ av attribut och ett eventuellt ingångsvärde:

```
<!ATTLIST namnPåElement namnPåAttribut typ ingångsvärde>
```

De olika ingångsvärden ett attribut kan ha är:

1. `#REQUIRED` – detta attribut måste användas i det specificerade elementet. Ett exempel på detta såg du ovan.
2. `#IMPLIED` – det är valfritt om attributet ska användas eller inte.

   ```
   <!ATTLIST person personnr CDATA #IMPLIED>
   ```

3. `#FIXED` – används tillsammans med ett värde och detta värde måste då användas:

   ```
   <!ATTLIST pris valuta CDATA #FIXED "kr">
   ```

   Här anges att det värde attributet `valuta` måste ha är `kr`.

4. Bara ett värde – sätter ett standardvärde som används om attributet i det specificerade elementet inte används.

   ```
   <!ATTLIST pris valuta CDATA "kr">
   ```

En annan sak du bör lägga märke till är att vi i exemplen ovan använder `CDATA` i stället för `PCDATA`. Detta innebär att attributvärden är data som inte ska tolkas.

Attribut med bestämda värden

Vid några tillfällen kan det vara bra att kunna ange ett antal värden som ett attribut kan ha. Tänk dig ett `person`-element som har ett attribut med namnet `kön`. Detta attribut ska bara kunna ha värdet `man` eller `kvinna`. I attributdeklarationen skriver vi då:

```
<!ATTLIST person kön (man | kvinna)>
```

Det brukar vara vanligt att även ett standardvärde anges i deklarationen när en lista används.

1.4.3 ENTITY

Som vi gick igenom i lektion 2 kan vi deklarera entiteter i en DTD. Som nämndes då kan det vara lämpligt vid tillfällen när vi har en längre text som kommer att användas flera gånger i XML-dokumentet. Vi har tidigare sett ett exempel på detta med `MIUNOSD`:

```
<!ENTITY MIUNOSD "Mittuniversitetet i Östersund">
```

Nu kan vi i XML-dokumentet skriva följande för att använda entiteten:

```
&MIUNOSD;
```

1.5 Nackdelar med DTD

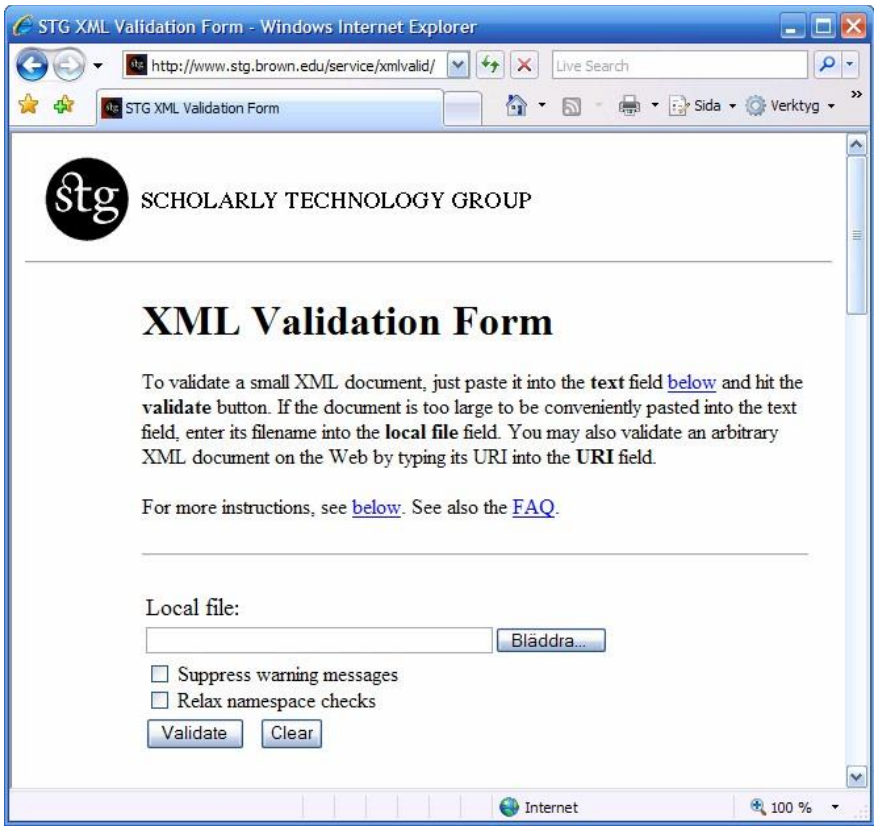
Som du kanske har upptäckt allt eftersom i lektionen så finns det en del nackdelar med DTD. Några av dem är:

- DTD har en annan syntax än XML-dokument och kan därför inte tolkas av en XML-tolk.
- DTD är inte utbyggbart.
- Det finns ingen möjlighet att specificera datatyp för innehållet i elementen.
- Det finns inga regler uppsatta för att en DTD kan ärva från en annan DTD.
- Det kan vara svårt att skriva en bra DTD.

1.6 Validera ett XML-dokument mot en DTD

När du öppnar ett XML-dokument som använder en DTD i din webbläsare så kommer XMLdokumentet inte att valideras. XML-dokumentet kommer enbart kontrolleras om det är välutformat. För att göra en validering måste vi använda en validerande XML-tolk. Vi kan själva programmera en sådan, men det kommer vi inte göra i den här kursen. Man kan direkt i NetBeans validera ett XML-dokument mot en DTD. Ett annat sätt är att använda en validerande XML-tolk på Internet.

Här kan vi antingen ladda upp en fil under `Local file` och sen trycka på `Validate`. Alternativt kan du klistra in hela innehållet i XML-dokumentet med en intern DTD i textrutan längst ner på sidan. Trycker du på `Validate` kommer det upp meddelanden om XMLdokumentet är välutformat och giltigt eller vad som eventuellt är fel. Den är väldigt enkel att använda, men fungerar bara med XML-dokument som använder en intern DTD.



1.6.1 Ett exempel

För att ge exempel på det vi gått igenom i denna lektion ska vi till sist visa ett exempel. Vi kommer att använda samma exempel i nästa lektion om XML Schema. Det blir då enkelt för dig att jämföra DTD mot XML Schema. Nedan visas XML-dokumentet personregister.xml med en intern DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File Name: personregister.xml -->

<!DOCTYPE personregister [
      <!ELEMENT personregister (person)+>
      <!ELEMENT person (namn, adress+, telefon+, civilstatus, titel)>
      <!ATTLIST person personnr CDATA #REQUIRED>

      <!ELEMENT namn (förnamn, efternamn)>
      <!ELEMENT förnamn (#PCDATA)>
      <!ELEMENT efternamn (#PCDATA)>

      <!ELEMENT adress (gata, postnr, postort)>
      <!ELEMENT gata (#PCDATA)>
      <!ELEMENT postnr (#PCDATA)>
      <!ELEMENT postort (#PCDATA)>

      <!ELEMENT telefon (#PCDATA)>
      <!ELEMENT civilstatus (#PCDATA)>
      <!ELEMENT titel (#PCDATA)>
]>
```

```
<personregister>
      <person personnr="1234123412">
            <namn>
                  <förnamn>Karin</förnamn>
                  <efternamn>Nordin</efternamn>
            </namn>
            <adress>
                  <gata>Nordmansvägen 5</gata>
                  <postnr>11111</postnr>
                  <postort>Östersund</postort>
            </adress>
            <telefon>11111111</telefon>
            <civilstatus>Gift</civilstatus>
            <titel>Chefsredaktör</titel>
      </person>
</personregister>
```

Vi har angett att attributet `personnr` måste användas och att flera adresser och telefonnummer ska kunna registreras. Använd lite tid för att gå igenom detta exempel och prova att lägga till, ta bort och/eller ändra ordning på elementen för att se hur XML-dokumentet valideras.