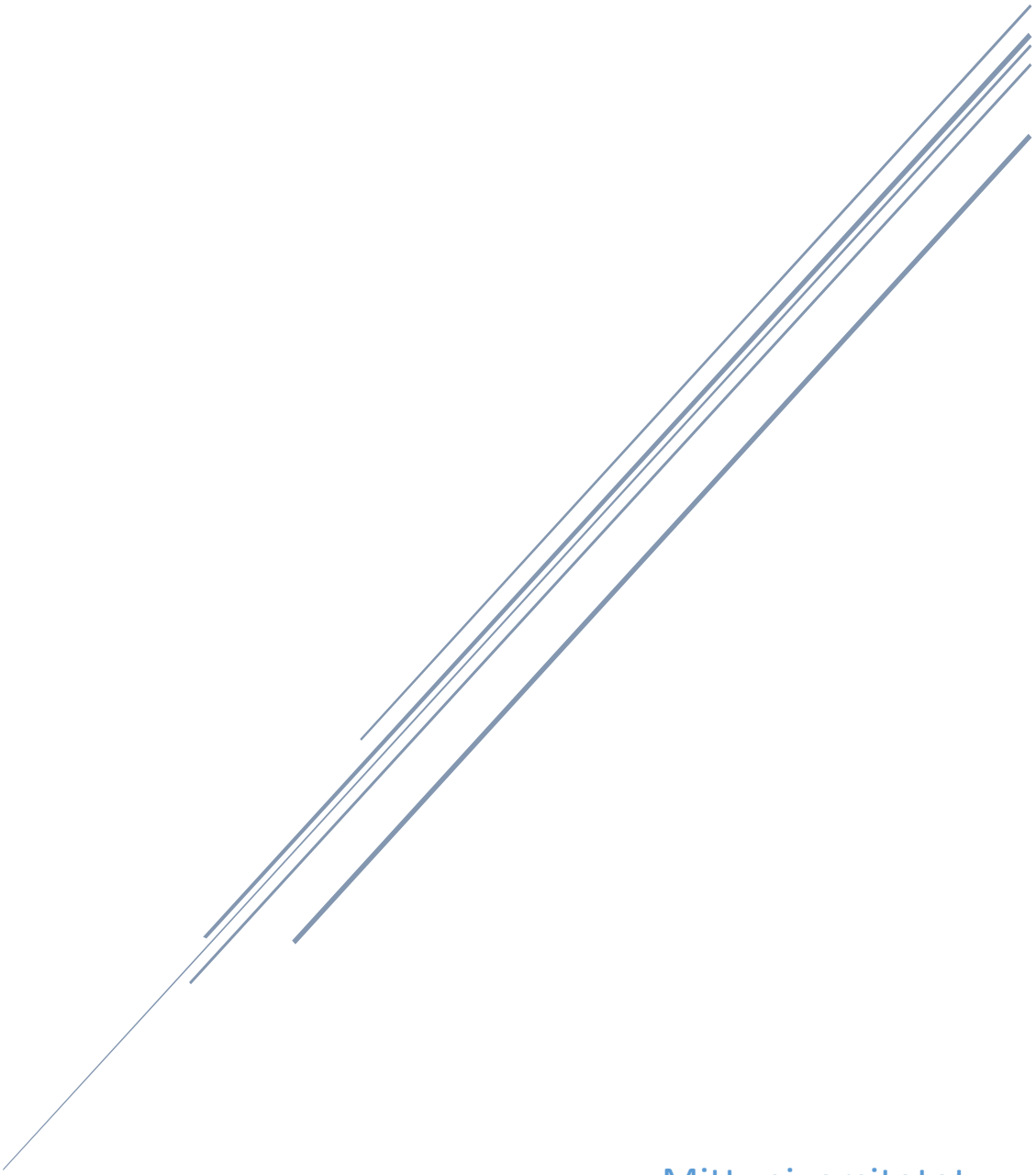
XML-SCHEMA



Mittuniversitetet
XML

1 XML Schema (XSD)

Sammanfattning: I förra lektionen tittade vi på en standard för att göra XML-dokument giltiga, nämligen DTD. I denna lektions ska vi ta oss an XML Schema som är en annan standard för att skapa giltiga XML-dokument. Detta är en betydligt mer omfattande standard i vilken man bland annat kan sätta hårdare krav på strukturen i XML-dokumentet.

1.1 Inledning

Förutom att skapa välutformade XML-dokument började vi i förra lektionen att dessutom skapa giltiga dokument med hjälp av en DTD. I denna lektion ska vi se på en annan standard för att validera XML-dokument, nämligen XML Schema. Vi går igenom hur vi kan upprätta element och attribut i schemat och olika egenskaper som kan sättas i samband med detta. I kapitel 1.7 tittar vi på ett litet större exempel. Använd tid för att gå igenom detta exempel så du förstår det. Prova även att göra förändringar för att se vad som händer.

Det är inte möjligt att validera ett XML-dokument mot ett XML Schema direkt i en webbläsare. För detta måste vi använda speciella program. I kapitel 1.8 går vi igenom ett program som kan användas för validering.

1.2 XML Schema

XML Schema är dokument som beskriver innehållet och strukturen i ett XML-dokument. Här kan attribut som används, och hur elementen ordnas i en hierarkisk struktur i XML-dokumentet, beskrivas. Utöver detta kan begränsningar på innehållet i element eller attribut sättas.

Genom att använda XML schema är det möjligt att till en större grad precisera innehåll och struktur i ett XML-dokument än vad det är möjligt med DTD. Ett schema är betydligt mer komplext uppbyggt än en DTD och dessutom uttrycks XML Schema i XML vilket kan medföra att scheman bli lite svårare att skriva än en DTD.

1.3 Enkla och komplexa datatyper

Innan vi börjar med att skapa egna scheman är det bra att ha klart för sig vad begreppen enkla och komplexa datatyper innebär (simple types och complex types på engelska).

Komplexa datatyper innehåller andra element och/eller attribut som exemplet nedan visar:

```
<person>
   <namn>Ola Nordin</namn>
   <ålder>24</ålder>
</person>
```

Här kan vi se att elementet `person` innehåller två andra element; `namn` och `ålder`. Detta gör elementet `person` till en komplex datatyp. Ett annat exempel på en komplex datatyp är:

```
<telefon typ="mobil">070-98989898</telefon>
```

Elementet `telefon` är en komplex datatyp eftersom den innehåller attributet `typ`.

Element som inte innehåller barnelement eller attribut är enkla datatyper. Låt oss titta på exemplet från ovan igen:

```
<person>
   <namn>Ola Nordin</namn>
   <ålder>24</ålder>
</person>
```

Här är elementen `namn` och `ålder` enkla datatyper då de endast innehåller text.

Komplexa datatyper måste vi alltid deklarera själva, men för enkla datatyper finns det många fördefinierade datatyper vi kan använda. Det är även möjligt att skapa helt egna datatyper som utgår från en av de fördefinierade datatyperna, en så kallad härledd datatyp. Kanske är det så att vi önskar fler begränsningar än att ett element bara ska bestå av ett heltal (vi vill kanske att endast tal mellan 10 och 20 ska användas). Tabellen nedan listar några vanliga fördefinierade datatyper (se Appendix C i kursboken för fler datatyper):

datatyp	*beskrivning*
`string`	en sträng med text
`boolean`	värdet true eller false
`double`	decimaltal
`integer`	heltal
`positiveInteger`	ett heltal större än 0
`date`	ett datum enligt formen yyyy-mm-dd

Alla dessa datatyper kan användas genom att skriva `xsd:datatyp`. Ett exempel visas nedan: `xsd:string`

I kommande kapitel tittar vi närmare på användning av datatyper i olika element.

1.4 Deklarera element

Fram till nu har vi tittat på två olika datatyper, enkla och komplexa. Vi ska nu börja deklarera element som använder dessa datatyper. I XML Schema används då följande element: `xsd:element`

Vi kan använda detta för att skapa ett element med namnet förnamn:

```
<xsd:element name="förnamn" type="xsd:string">
```

Här har vi deklarerat ett element som heter förnamn och där innehållet kan bestå av bokstäver, tecken eller tal (med andra ord en sträng). I ett XML-dokument kan vi därmed skriva:

```
<förnamn>Ola</förnamn>
```

Tänk dig nu följande XML-dokument:

```
<person>
   <namn>Ola Nordin</namn>
   <tlf>34343434</tlf>
</person>
```

Om vi nu ska deklarera elementet person tillsammans med dess barnelement måste vi använda en komplex datatyp. Vidare kommer vi att använda de enkla datatyperna xsd:string på elementet namn och xsd:integer på elementet tlf:

```
1 <xsd:element name="person">
2   <xsd:complexType>
3     <xsd:sequence>
4       <xsd:element name="namn" type="xsd:string" />
5       <xsd:element name="tlf" type="xsd:integer" />
6     </xsd:sequence>
7   </xsd:complexType>
8 </xsd:element>
```

Det första vi gör är att deklarera elementet person. Detta element är en komplex datatyp och hur detta anges ser du på rad 2. På rad 3 anges ordningsföljden för barnelementen. Elementet sequence är en så kallad innehållsmodell och innebär i just detta fall att barnelementen måste komma i exakt den ordningsföljd som anges innanför elementet sequence. I exemplet betyder det att vi i XML-dokumentet först måste skriva namn och därefter tlf. För elementen namn och tlf används enkla datatyper. Namn ska vara en textsträng och tlf ska vara ett tal. Observera att dessa två element är tomma och behöver därför ingen sluttagg, utan starttaggen avslutas i stället med />.

Innehållsmodeller

I exemplet ovan användes elementet sequence som innehållsmodell. Den anger att elementen måste komma i en bestämd ordningsföljd. Det finns ytterligare två innehållsmodeller; choice (välj mellan ett eller flera barnelement) och all (barnelementen kan komma i vilken som helst ordningsföljd, men alla barnelement måste finnas med en gång). Om du i stället vill använda någon av dessa så byter du helt enkelt ut elementet sequence mot någon av de andra.

1.4.1 Skapa egna namngivna komplexa datatyper

I exemplet ovan deklarerar vi en anonym typdefinition. Det betyder att vi nästlat hela typdefinitionen innanför `xsd:element`. Detta är en bra lösning som fungerar så länge vi bara ska använda datatypen en gång. Ska däremot den komplexa datatypen användas flera gånger kan vi definiera elementet globalt och ge det ett namn. Så här kommer det då att se ut (förändringar är markerad med fet stil):

```
1 <xsd:element name="person" type="personType" />
2
3 <xsd:complexType name="personType">
4     <xsd:sequence>
5           <xsd:element name="namn" type="xsd:string" />
6           <xsd:element name="tlf" type="xsd:integer" />
7     </xsd:sequence>
8 </xsd:complexType>
```

I rad 1 deklareras elementet `person` att vara av typen `personType`. Lägg märke till att elementet `person` nu är ett tomt element, resten av koden är inte nästlad inuti detta element. På rad 3 börjar vi att skapa den komplexa datatypen `personType` och det görs på samma sätt som tidigare.

1.4.2 Skapa egna enkla datatyper

Vid vissa tillfällen kan det vara bra att kunna skapa sina egna enkla datatyper. I nästa exempel vill vi skapa en datatyp som representerar ett lottonummer. I lotto används tal från 1 till 35.

```
1 <xsd:simpleType name="lottonummer">
2    <xsd:restriction base="xsd:integer">
3       <xsd:minInclusive value="1" />
4       <xsd:maxInclusive value="35" />
5    </xsd:restriction>
6 </xsd:simpleType>
```

På rad 1 har vi skapat en ny enkel datatyp genom att använda elementet `simpleType`. Ett element i ett XML-dokument som använder denna datatyp får nu inte innehålla några barnelement eller attribut. Som attribut i elementet `simpleType` använder vi `name` med värdet `"lottonummer"`. Detta bestämmer namnet på vår nya datatyp. Vidare ska vi nu ange några begränsningar för vår datatyp. För detta används elementet `restriction` på rad 2. Här används attributet `base` för att ange vilken datatyp vår nya datatyp ska utgå från. Vi ger attributet värdet `xsd:integer` eftersom vi kommer att använda heltal. På rad 3 och 4 kommer själva begränsningarna genom att vi använder elementen `minInclusive` och `maxInclusive`. Med dessa kan vi ange vilket lägsta och högsta värde som är tillåtet att använda.

En variant på dessa två är elementen `minExclusive` och `maxExclusive`. Dessa två alternativ kan vara användbara framför allt när vi använder decimaltal. Nedan ges en förklaring på de fyra elementen vi nyss tittat på:

- `minInclusive`: värdet får minst vara
- `maxInclusive`: värdet får högst vara
- `minExclusive`: värdet måste vara större än angivet värde
- `maxExclusive`: värdet måste vara mindre än angivet värde

Vi kan även skapa en datatyp för bilars registreringsnummer (t.ex. `ABC 123`):

```
<xsd:simpleType name="regnr">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="[A-Z]{3} \d{3}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Här utgår vi från datatypen `xsd:string`. När vi använder `xsd:string` kan både bokstäver, tal och andra tecken användas. För att ange begränsningarna i detta exempel har elementet `pattern` används. Vi kan då använda så kallade reguljära uttryck som kontrollerar om en given sträng följer ett givet mönster. Värdet i attributet `value` anger att tre valfria bokstäver mellan `A` och `Z` ska anges, följt av ett mellanrum, följt av tre valfria siffror. Reguljära uttryckt kan vara väldigt användbara, men kan vara svåra att sätta sig in i hur de fungerar

Ett exempel på där datatypen för registreringsnummer används ser du på nästa sida:

```
<xsd:element name="registreringsnummer" type="regnr" />

<xsd:simpleType name="regnr">
   <xsd:restriction base="xsd:string">
     <xsd:pattern value="[A-Z]{3} \d{3}"/>
   </xsd:restriction>
</xsd:simpleType>
```

Som övning kan du nu prova att själv skapa några enkla datatyper för ett personnummer och en e-postadress. Personnumret ska ha formatet ååmmdd-nnnn (t.ex. 701224-7853) och en e-postadress ska ha formatet förnamn.efternamn@domän.se (t.ex. robert.jonsson@miun.se). Diskutera gärna i forumet olika alternativa lösningar, men försök själv först.

På sidan 348-349 i boken finns en lista över element som kan användas innanför elementet `xsd:restriction` för att ange olika begränsningar värdet datatypen kan ha.

1.4.3 Ange antal gånger ett element kan användas

Det finns ett antal attribut som kan användas för att bestämma hur många gånger ett visst element kan förekomma:

- `minOccurs`: minsta antalet gånger elementet kan användas. Har standardvärdet 1.
- `maxOccurs`: högsta antalet gånger elementet kan användas. Har standardvärdet 1.

För att illustrera användandet säger vi att elementet `tlf` kan användas från 1 till 3 gånger:

```
<xsd:element name="tlf" type="xsd:integer" maxOccurs="3"/>
```

Eftersom standardvärdet för båda dessa attribut är 1 är det inte nödvändigt att använda `minOccurs`, utan vi sätter enbart värdet för `maxOccurs` till 3.

Observera att då standardvärdet är 1 betyder det att om `minOccurs` och `maxOccurs` inte används måste elementet förekomma en enda gång på angiven plats. Det vill säga om elementet endast får förekomma en gång skriver vi:

```
<xsd:element name="tlf" type="xsd:integer"/>
```

Om elementet `tlf` är valfritt, det vill säga att en person inte behöver ha någon telefon, kan detta anges genom att sätta värdet på `minOccurs` till 0:

```
<xsd:element name="tlf" type="xsd:integer" minOccurs="0"/>
```

Om vi inte vet hur många gånger ett element kan användas använder vi värdet `unbounded` för att markera att elementet kan användas oändligt många gånger:

```
<xsd:element name="tlf" type="xsd:integer" maxOccurs="unbounded"/>
```

Ovan betyder att en person måste ha minst en telefon, men kan ha hur många som helst.

Vi kommer nu att utöka exemplet ovan. Det är önskvärt att kunna registrera mellan 0 och 3 personer och för varje person ska det gå att registrera 1 till 3 telefonnummer. Eftersom flera personer ska registreras behöver vi ett nytt rotelement i XML-dokumentet(då rotelementet endast kan användas en gång och vi nu kommer att använda flera `person`-element). Vi använder därför elementet `personregister` som rotelement och får XML-dokumentet på nästa sida:

```
<personregister>
 <person>
       <namn>Ola Nordin</namn>
       <tlf>34343434</tlf>
 </person>
 <person>
       <namn>Karin Larson</namn>
       <tlf>88998899</tlf>
       <tlf>67856756</tlf>
 </person>
</personregister>
```

I schemat använder vi en anonym typdefinition för deklarationen av elementet `personregister`, medan vi använder en namngiven typdeklaration för elementet `person`. Schemat blir enligt:

```
01 <xsd:element name="personregister">
02   <xsd:complexType>
03     <xsd:sequence>
04       <xsd:element name="person" minOccurs="0" maxOccurs="3"
05                    type="personType" />
06     </xsd:sequence>
07   </xsd:complexType>
08 </xsd:element>
09
10 <xsd:complexType name="personType">
11   <xsd:sequence>
12     <xsd:element name="namn" type="xsd:string" />
13     <xsd:element name="tlf" type="xsd:integer" maxOccurs="3" />
14   </xsd:sequence>
15 </xsd:complexType>
```

På rad 1 till 8 återfinns deklarationen av elementet `personer`. Här ser vi att elementet `person` är ett barnelement till `personer`. Det är angett att elementet `person` kan förekomma från ingen till 3 gånger och att den består av typen `personType`. Denna datatyp återfinns på raderna 10 till 15 och är nästan exakt som i de tidigare exemplen.

`minOccurs` och `maxOccurs` kan även användas för innehållsmodellen (elementet) `choice`. Här används de då för att markera hur många av barnelementen som kan väljas i XMLdokumentet. För att visa att ett eller oändligt många barnelement kan väljas skriver vi exempelvi:

```
<xsd:choice minOccurs="1" maxOccurs="unbounded"/>
```

För innehållsmodellen `all` används inte `minOccurs` och `maxOccurs` då alla barnelementen ändå förekommer endast en gång.

1.4.4 Ange värdet för ett element

Det finns två attribut med vilka vi kan ange ett värde på de definierade elementen i schemat:

- `fixed`: anger att fast värde på elementet. Detta värde måste då användas i XMLdokumentet eller vara tomt.
- `default`: sätter ett standardvärde på elementet om inget anges. I XML-dokumentet kan dock ett annat värde anges.

Vi kan till exempel låta elementet `tlf` ha standardvärdet `00000000` om inget annat anges:

```
<xsd:element name="tlf" type="xsd:integer" minOccurs="0"
             maxOccurs="3" default="00000000"/>
```

1.4.5 Ange bestämda elementvärden

Om det är så att ett visst element endast ska kunna innehålla en uppsättning av i förväg bestämda värden kan elementet `enumeration` användas. Vi kan till exempel ange att giltiga värden som innehåll i elementet civilstatus för en person är; `Gift`, `Sambo`, och `Singel`. Vi skapar då en enkel datatyp enligt:

```
<xsd:element name="civilstatus" type="civilstatusType" />

<xsd:simpleType name="civilstatusType">
 <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Gift" />
      <xsd:enumeration value="Sambo" />
      <xsd:enumeration value="Singel" />
 </xsd:restriction>
</xsd:simpleType>
```

Som du ser används elementet `enumeration` för varje giltigt värde. Nu tillåts inga andra värden än dessa i elementet `civilstatus`. Det finns vissa likheter med `choice`. Skillnaden är att `choise` anger vilka element som är giltiga medan `enumeration` anger giltiga värden. Ett exempel på ett giltigt värde i XML-dokumentet är:

```
<civilstatus>Gift</civilstatus>
```

Detta är däremot ett ogiltigt värde:

```
<civilstatus>Ogift</civilstatus>
```

1.5 Deklarera attribut

Att deklarera attribut i XML Schema skiljer sig inte så mycket från att deklarera element. Tänk dig följande XML-dokument:

```
<person personnr="123456-7890">
   <namn>Ola Nordin</namn>
   <tlf>34343434</tlf>
</person>
```

Schemat kommer då att se ut som nedan (deklarationen av attribut har fet stil):

```
01 <xsd:element name="person" minOccurs="0" maxOccurs="3"
02       type="personType" />
03
04 <xsd:complexType name="personType">
05    <xsd:sequence>
06          <xsd:element name="namn" type="xsd:string" />
07          <xsd:element name="tlf" type="xsd:integer"
08             minOccurs="0" maxOccurs="3" default="00000000"/>
09    </xsd:sequence>
10    <xsd:attribute name="personnr" type="xsd:string"/>
11 </xsd:complexType>
```

På rad 10 har vi lagt till ett attribut som hör till elementet som deklareras på rad 1. Observera att deklarationen ligger utanför elementet `sequence`, men innanför elementet `complexType`. Innanför elementet `sequence` ska endast elementdeklarationer förekomma. Attributet som deklareras på rad 10 ges namnet `personnr` och datatypen `string`. I förra lektionen gavs flera exempel på attribut, bland annat detta:

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

Elementet `pris` har ett attribut men namnet `valuta`. Elementet har inget barnelement, men vi måste ändå använda en komplex datatyp när vi deklarerar elementet `pris` på grund av attributet. Om vi vill använda vanligt innehåll (elementet innehåller då inga barnelement utan enbart PCDATA) i ett element av komplex typ, måste vi ange att den komplexa typen ska bestå av blandat innehåll. För detta används attributet `mixed` i elementet `complexType`. Så här ser deklarationen av `pris` ut:

```
<xsd:element name="pris" type="prisType" />

<xsd:complexType name="prisType" mixed="true">
    <xsd:attribute name="valuta" type="xsd:string"/>
</xsd:complexType>
```

Ett problem med denna lösning är att vi inte kan ange vilken datatyp innehållet i elementet `pris` ska ha. Vi kan då i stället använda ett element med namnet `simpleContent`. De gjorda förändringarna visas nedan med fet stil:

```
<xsd:element name="pris" type="prisType" />

<xsd:complexType name="prisType" mixed="true">
  <xsd:simpleContent>
    <xsd:extension base="xsd:integer">
      <xsd:attribute name="valuta" type="xsd:string" />
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Som du ser har vi här lagt in elementet `simpleContent` innanför elementet `complexType`. Vi kan därmed använda elementet `extension` för att ange vilken datatyp elementet `pris` ska ha. Som du ser är det satt till ett heltal. Lägg märke till att alla eventuella attribut läggs innanför elementet `extension`.

1.5.1 Specificera hur attribut ska användas

Till elementet `attribute` finns ett attribut med namnet `use` som kan användas för att ange hur det deklarerade attributet ska användas. Attributet `use` kan ha följande värden:

- `required`: säger att detta attribut måste finnas med.
- `fixed`: attributets värde sätts direkt i schemat. Används tillsammans med attributet `value` där själva värdet sätts.
- `default`: sätter ett standardvärde på attributet om inget annat anges.
- `optional`: anger att detta attribut är valfritt att ta med.

- `prohibited`: detta attribut kan inte användas.

I exemplet ovan kan vi till exempel ange att attributet `valuta` måste vara med enligt:

```
<xsd:attribute name="valuta" type="xsd:string" use="required"/>
```

Vi kan i likhet med element ange en uppsättning av i förväg bestämda värden även för attribut. Även här används då elementet `enumeration`. Som ett exempel kan vi låta element `tlf` ha ett attribut `typ` för att ange vilken typ av telefonnummer det är frågan om. Detta attribut vill vi ska ha något av följande värden; `mobil`, `hem` och `jobb`. Vi skapar då en ny enkel datatyp enligt:

```
<xsd:attribute name="typ" type="telefonType" use="required" />

<xsd:simpleType name="telefonType">
 <xsd:restriction base="xsd:string">
      <xsd:enumeration value="mobil" />
      <xsd:enumeration value="hem" />
      <xsd:enumeration value="jobb" />
 </xsd:restriction> </xsd:simpleType>
```

I exemplet ovan används ett `enumeration`-element för varje värde attributet kan ha.

1.6 Schemats syntax

Fram till nu har vi endast deklarerat de olika elementen och attributen i schemat. Nu är det hög tid att skapa ett komplett schema. Kom ihåg att XML Schema i sig är ett XML-dokument och måste därför börja med en XML-deklaration:

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

I XML Schema används alltid ett och samma rotelement och det är elementet `schema`. Som vi nämnde inledningsvis knyts detta till namnrymdsnamnet:

```
http://www.w3.org/2001/XMLSchema
```

Därmed kommer rotelementet att se ut som nedan:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Alla element- och attributdeklarationer måste nästlas innanför rotelementet.

I schemat kan det vara en god idé att börja med information till de som ska läsa schemat (kan vara personer och/eller applikationer). Vi använder då elementet `annotation`. Innanför detta element används sen elementet `documentation` som består av text/kommentarer till personer och elementet `appInfo` som är information riktad mot applikationer. Vi nöjer oss här med information till användaren och skriver följande:

```
<xsd:annotation>
 <xsd:documentation> Information
      om personer
 </xsd:documentation>
</xsd:annotation>
```

1.7 Ett exempel

Vi ska nu gå igenom att lite större exempel för att sammanfatta det vi lärt oss i denna lektion. Vi använder samma exempel som i lektion 5, så att det blir enklare att jämföra DTD med XML Schema. Vi har följande XML-dokument (personregister.xml):

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

<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>
```

Nedan följer en lista över de begränsningar vi vill ska gälla för detta XML-dokument:

- Det ska vara möjligt att registrera från 1 till 10 personer i XML-dokumentet.
- För varje person ska man kunna registrera från 1 till 4 adresser.
- För varje person ska man kunna registrera fån 0 till 5 telefonnummer.
- Attributet personnr ska vara ett heltal bestående av 10 siffror.
- Telefonnummer ska bestå av 8 siffror.
- Postnummer ska bestå av 5 siffror.

Schemat för detta XML-dokument är (personregister.xsd):

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:annotation>
        <xsd:documentation>
            Information om personer
        </xsd:documentation>
    </xsd:annotation>

    <xsd:element name="personregister">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="person" type="personType"  maxOccurs="10"  />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="personType">
        <xsd:sequence>
            <xsd:element name="namn" type="namnType" />
            <xsd:element name="adress" type="adressType"
                         maxOccurs="4" />
            <xsd:element name="telefon" minOccurs="0"
                         maxOccurs="5" type="telefonType" />
            <xsd:element name="civilstatus" type="xsd:string"/>
            <xsd:element name="titel" type="xsd:string"/>
        </xsd:sequence>

        <xsd:attribute name="personnr" use="required">
            <xsd:simpleType>
                <xsd:restriction base="xsd:positiveInteger">
                    <xsd:pattern value="\d{10}"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>

    <xsd:complexType name="namnType">
        <xsd:sequence>
            <xsd:element name="förnamn" type="xsd:string" />
            <xsd:element name="efternamn" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="adressType">
        <xsd:sequence>
            <xsd:element name="gata" type="xsd:string" />
            <xsd:element name="postnr" type="postnrType" />
            <xsd:element name="postort" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
```

```
54
55      <xsd:simpleType name="telefonType">
56          <xsd:restriction base="xsd:positiveInteger">
57              <xsd:minExclusive value="9999999" />
58              <xsd:maxInclusive value="99999999" />
59          </xsd:restriction>
60      </xsd:simpleType>
61
62      <xsd:simpleType name="postnrType">
63          <xsd:restriction base="xsd:positiveInteger">
64              <xsd:minExclusive value="9999" />
65              <xsd:maxInclusive value="99999" />
66          </xsd:restriction>
67      </xsd:simpleType>
68
69  </xsd:schema>
70
```

På rad 1 börjar en XML-deklaration. På rad 4 finns schemats rotelement. Därefter ges lite information om själva schemat (rad 6 - 10). På rad 12 finner vi deklarationen av XML-dokumentets rotelement som är `personregister`. Som du ser används en anonym typdefinition för detta element. Det hade lika gärna kunna varit namngivet så som vi gjort med flera av de övriga elementen. På rad 12-18 anger vi att elementet `personregister` ska kunna ha 1-10 `person`-element. För elementet `person` har vi skapat en egen typdeklaration vilken kommer på raderna 20-38. Här ser du att elementet `person` kan ha ett antal element och ett attribut. För elementen `namn` och `adress` har vi åter igen skapat egna komplexa datatyper (raderna 40 till 45 och 47 till 53). Vidare har vi angett begränsningar på elementen `telefon` och `postnr` samt attributet `personnr` (raderna 55 till 60 och 62 till 67).

Använd tid för att gå igenom detta exempel. Prova sen att validera enligt exempel nedan.

1.8 Validera XML-dokument med XML Schema

Om du provar att öppna `personregister.xml` i en webbläsare kommer dokumentet inte att valideras mot schemat. Anledningen är att webbläsare endast kontrollerar om ett XML-dokument är välutformat och inte om det är giltigt.

Det finns en hel del validatorer (program som kontrollerar om ett XML-dokument är giltigt i förhållande till ett XML Schema) på Internet som kan användas. Några av dessa program laddas hem och installeras lokalt på datorn, medan andra program tillåter att XML-dokumentet med tillhörande XML Schema laddas upp och kontrolleras online. Men det enklaste är att validera direkt i NetBeans.