DTD – Document Type Definition:

En række tilfælde er det nødvendigt eller ønskeligt at et XML dokument overholder et bestemt skema. Til dette formål anvendes DTD'er eller **dokument type definitioner**. Der findes en lang række standard DTD'er som er bredt anerkendte – f. eks. for XHTML, SVG eller DocBook. Vi skal vende tilbage til disse. Mange af disse DTD'er er på 1000 eller 5000 linjer!

DTD er et ikke XML baseret sprog! Et DTD skema er ikke velformet XML – selv om det indeholder tags og mærker – altså 'markup'! DTD er et sprog som er en applicering af **SGML** – Structured Generalized Markup Language som er 20 år ældre end XML og langt mere kompliceret. HTML er også en SGML applikation.

Et HTML dokument tilhører altså f. eks. dokument typen **'html'** og det har en **rod** som hedder 'html'!

Et XML dokument kan have **maximalt** een dokument type definition som angiver navnet på **roden**. En DTD kan have et **internt** subset (defineret i XML dokumentet internt) eller et **eksternt** subset (en DTD fil).

Følgende er et eksempel på et **eksternt** DTD gemt i 'telefonliste.dtd':

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!ELEMENT telefonliste (person+)>
<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >
```

DTD anvender en syntaks som kan minde om XML med tags, men DTD er egentligt ikke XML. **Alligevel** kan man bruge en XML erklæring – som vist her – til at sikre sig brug af f.eks. danske bogstaver!

I DTD erklæres elementer og attributter. DTD har et antal typer som kan anvendes på (kun) attributter. DTD opererer også med nogle modeller.

Sekvens:

Modellen her er en sekvens: Et gyldigt dokument skal have et rod element telefonliste, som så kan rumme et antal person elementer!

Indholdet af elementet telefonliste angives i parentesen. I DTD anvendes disse symboloer for kardinaliteten alstå hvor mange gange et element må forekomme:

1. (person+) betyder at der skal være mindst et og evt. mange person elementer
2. (person?) betyder at der kan være 0 eller 1 person element
3. (person*) betyder at der kan være 0, et eller mange person elementer

DTD definerer altså en model og et skema for formen i XML dokumentet. DTD siger ikke meget om indholdet af XML dokumentet – det såkaldte instans dokument.

Det kan ses at et person element kun må indeholde de elementer som står i parentesen. Hver person skal have tre sub elementer og netop i denne rækkefølge (en sekvens).

Et elements type kan være #PCDATA eller EMPTY eller sub elementer – der findes ikke i DTD muligheder for at type et elements indhold. Man kan altså ikke i DTD sige at et element i sin tekst node skal have en dato! I XSD – XML skemaer – kan et element types på denne måde men ikke i DTD.

#PCDATA er tekst som er OK i en XML fil – f. eks. må den ikke indeholde tegn som < eller &! Disse skal escapes med < og &

Hvis XML dokumentet skal være gyldigt skal det overholde dette skema. Det må ikke indeholde tags som ikke er definerede i DTD'en og elementerne skal anføres i XML dokumentet i den angivne rækkefølge og med de angivne kardinaliteter (hvor mange gange kan elementet forekomme).

Vi skriver nu en XML fil der skal valideres mod dette skema:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE telefonliste SYSTEM "telefonliste.dtd">

<telefonliste>

<person>
<fornavn>Erik</fornavn>
<fornavn>Emil</fornavn>
<efternavn>Jensen</efternavn>
<telefon>67677889</telefon>
</person>
<person>
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>
<person>
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>

</telefonliste>
```

Som det ses har vi faktisk indført en 'fejl' i forhold til skemaet idet den 1. person har fået to fornavne – NB XML dokumentet er stadig væk velformet og kan vises i en browser f. eks.! Vi ønsker nu at validere eller kontrollere XML dokumentet.

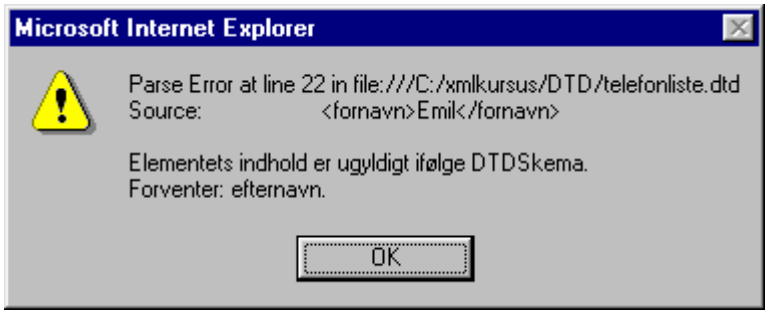
Validering:

Et XML dokument kan valideres på mange forskellige måder hvoraf vi efterhånden skal se på nogle af dem.

Man kan fra adressen http://msdn.microsoft.com downloade en fil IEXMLTLS eller Viewer Tools til Internet Explorer som består af to inf filer. Når man har downloadet og udpakket filen kan man højreklikke på inf filerne og vælge Installer.

Resultatet er at Internet Explorer derefter kan validere et XML dokument med et DTD (eller med et XDR skema som er Microsofts eget skema sprog!) – højreklik på XML dokumentet i IE og vælg valider!

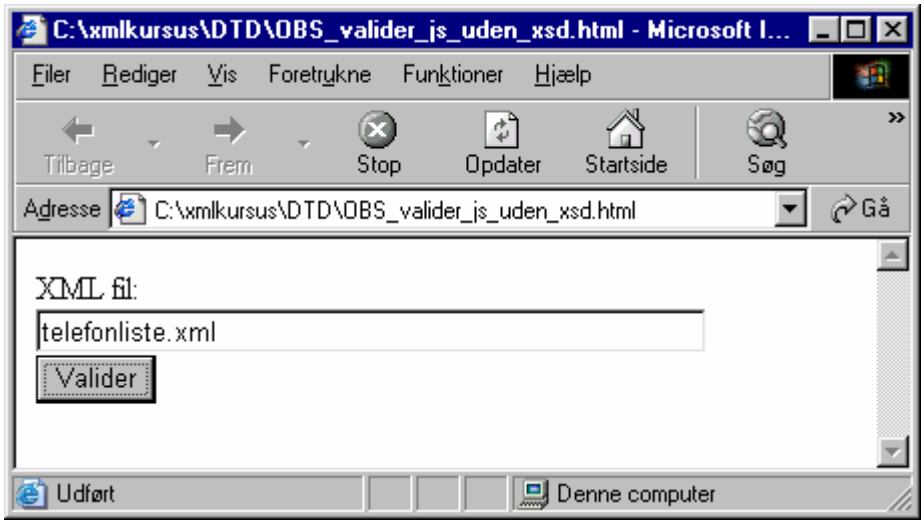
Hvis vi loader personliste.xml – med fejlen – og højre klikker og vælger valider får vi følgende besked i Internet Explorer:

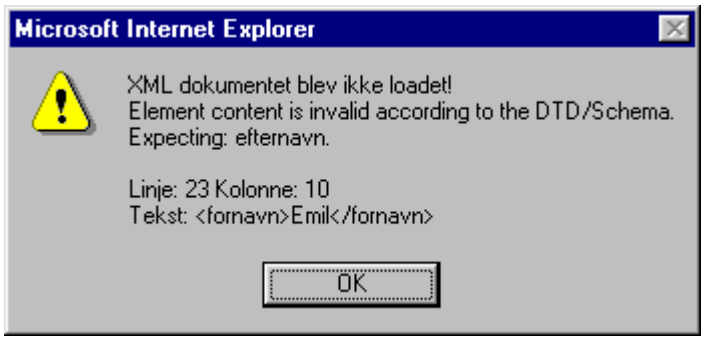


En anden løsning er at skrive et **script** som udfører valideringen således:

```
<html>
<body>
<form name="form" id="form">
XML fil:
<br>
<input type="text" size="50" name="xml" id="xml">
<br>

<input type="button" value="Valider" name="b" id="b" onclick="valider()">
</form>

<script language="javascript">

function valider(){

var doc=new ActiveXObject("MSXML2.DOMDocument.4.0");
doc.async=true;
doc.validateOnParse = true;
doc.load (document.form.xml.value);

if(doc.parseError.errorCode!=0){
alert("XML dokumentet blev ikke loadet!\n"+doc.parseError.reason+"\nLinje: "+doc.parseError.line +" Kolonne:
"+doc.parseError.linepos+" \nTekst: "+doc.parseError.srcText);
}
else {
alert("XML dokumentet er loadet OK!");
//alert(doc.xml);
}
}

</script>
</body>
</html>
```

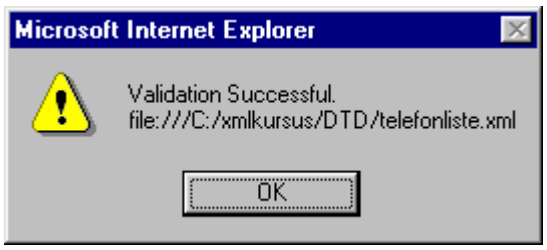
Hvis vi starter dette script program gemt som 'valider_dtd.html' kan vi validere vores telefonliste.xml:



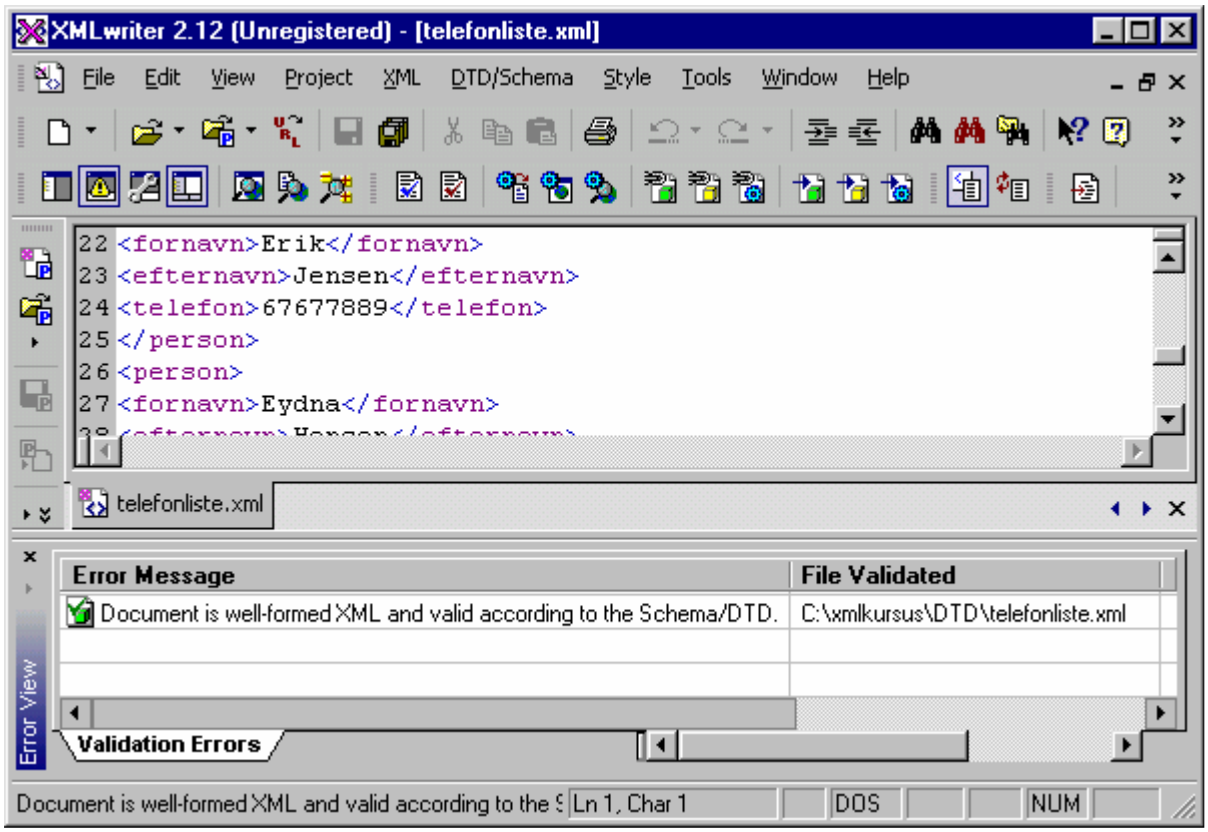


Vi kan se at XML dokumentet er ugyldigt i linje 23 og at skema **processoren** forventer elementet efternavn!

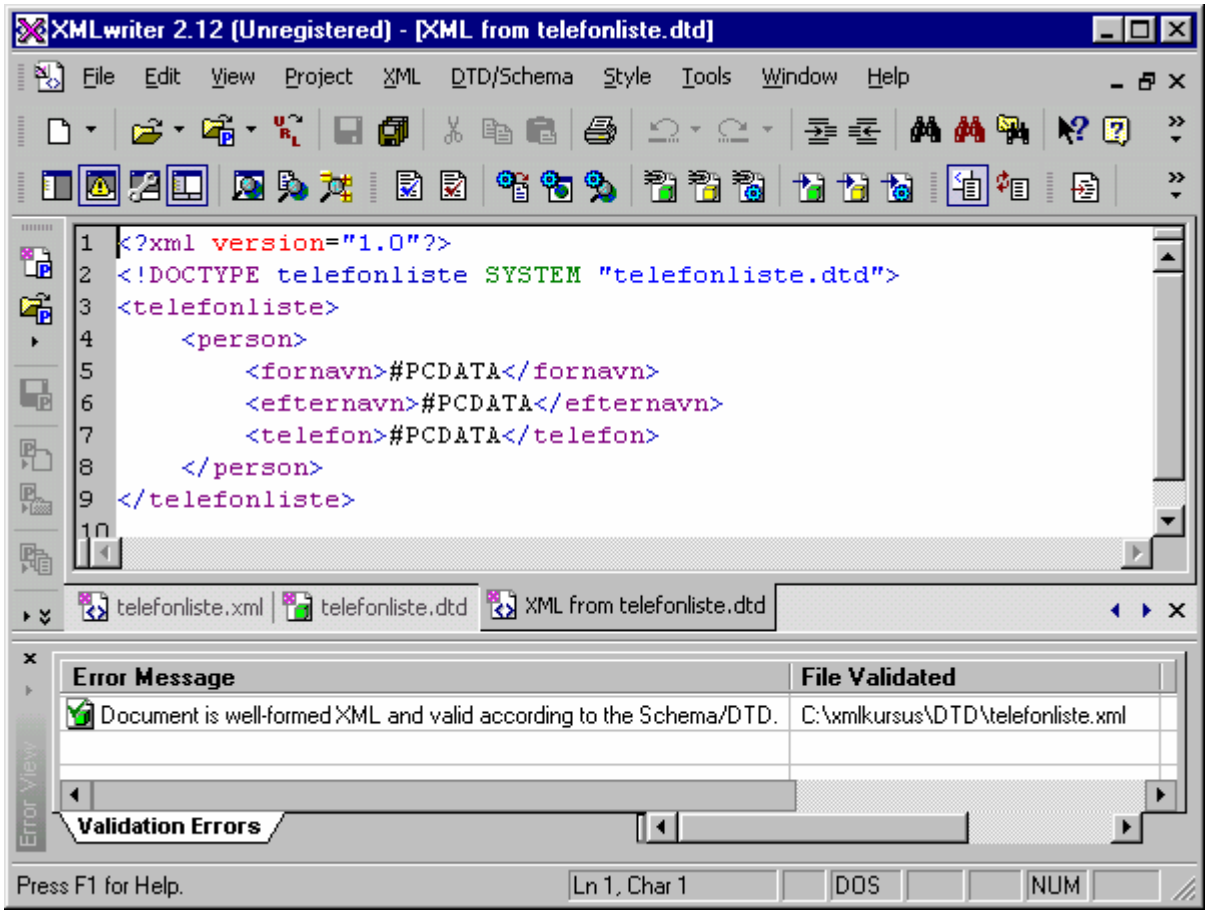
Hvis vi retter fejlen får vi en OK meddelelse:



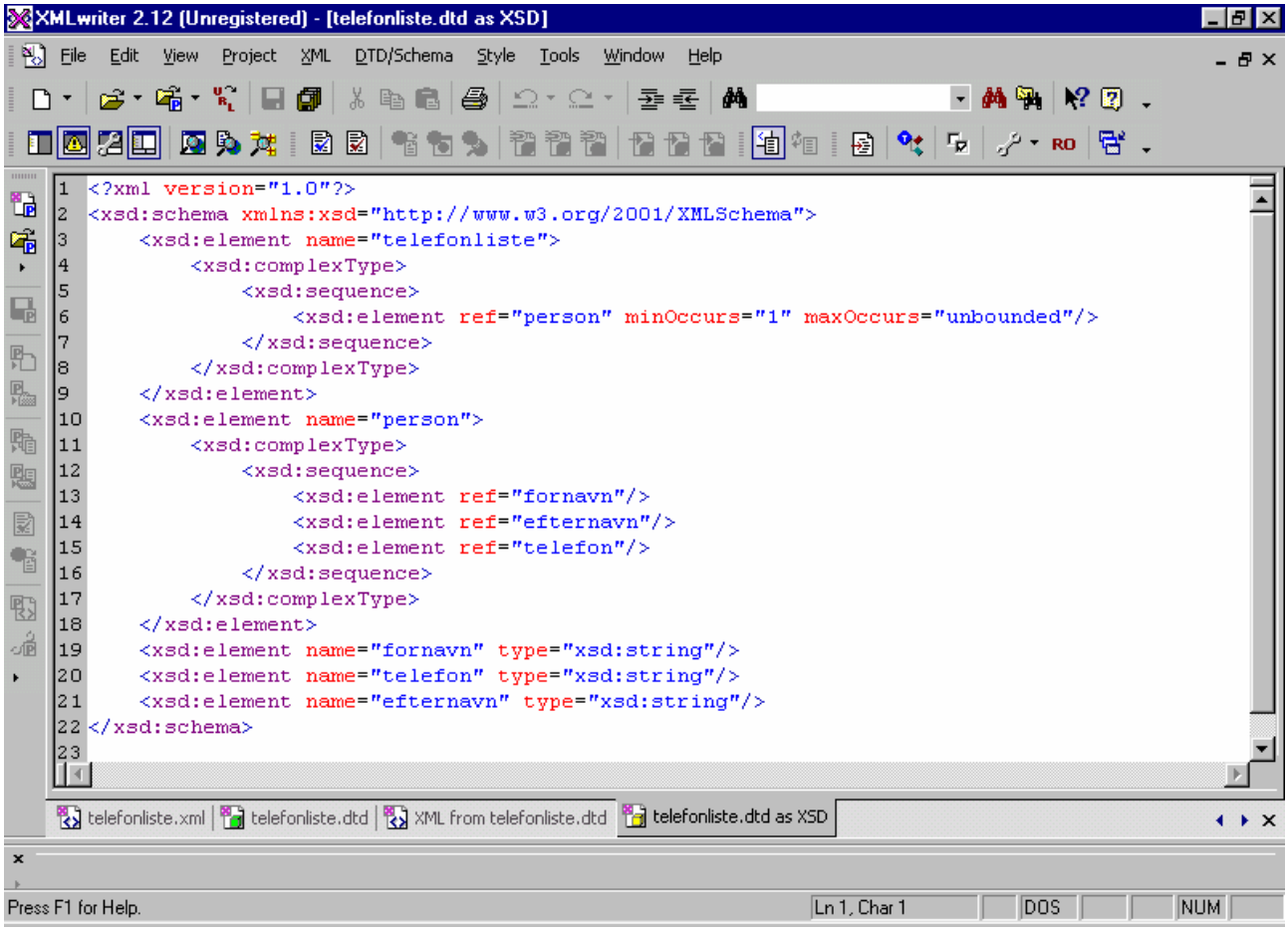
Man kan hente mange forskellige **validator** programmer på **Internettet** hvor der også findes on-line validator programmer. Forskellige værktøjer som **XMLSpy** eller **XMLWriter** - som kan downloades i prøve versioner - kan også validere eller kontrollere XML dokumenter:



XMLWriter kan også **generere** et XML dokument ud fra en DTD definition således:



På samme måde kan **XMLWriter** konvertere et DTD skema til et **XSD** skema. Vi skal senere se på XSD (eller XML) skemaer:



Som det ses er XSD skemaer **væsentligt** mere **komplicerede** end DTD skemaer! De har imidlertid mange **fordele**.

Skemaet **XMLSchema**.dtd kan downloades fra Internettet fra mange steder. Det er en DTD som **definerer** XSD skemaer, deres syntaks og grammatik! Alle XSD skemaer bliver altså **valideret** op imod et DTD!

Attributter i DTD:

Vi kan nu definere nogle attributter til vores telefon liste – telefonliste.dtd:

<?xml version="1.0" encoding="iso-8859-1"?>
<!ELEMENT telefonliste (person+)>
<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >

<!ATTLIST person type (arbejde|familie|venner) "arbejde">
<!ATTLIST telefonliste opdateret_af (kn|on|rh) #REQUIRED>
<!ATTLIST telefonliste opdateret NMTOKEN #REQUIRED>

En attribut defineres altid i formatet: elementets navn – attributtems navn – attributtens type – dens use og evt default værdier.

Den attribut der hedder opdateret_af er en choice fordi den kun kan antage disse tre værdier! Den er #REQUIRED dvs. den skal anføres – hvis den er #IMPLIED kan den evt. anføres (men kan godt have en default værdi).

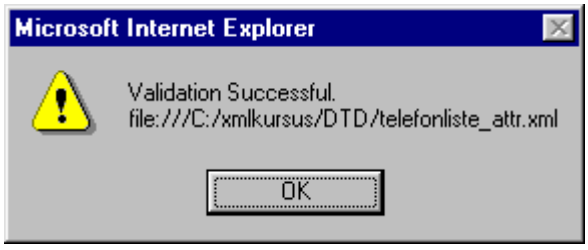
I DTD kan elementer ikke 'types' men der findes typer for attributter:

1. CDATA: er tekst
2. NMTOKEN: er et XML navn som er uden mellemrum
3. ID: dvs. at der kun må forekomme en id med samme værdi i XML dokumentet
4. IDREF: som skal referere til et ID

Vi kan nu skrive en ny XML fil der skal valideres ud fra dette skema:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE telefonliste SYSTEM "telefonliste_attr.dtd">

<telefonliste opdateret_af="kn" opdateret="2003-12-24">
<person type="arbejde">
<fornavn>Erik</fornavn>
<efternavn>Jensen</efternavn>
<telefon>67677889</telefon>
</person>
<person type="familie">
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>
<person>
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>

</telefonliste>
```

Hvis vi validerer telefonliste_attr.xml mod den nye telefonliste_attr.ded fås dette resultat:



Hvis vi angiver en 'forkert' eller 'ulovlig' værdi for attributten type i elementet person således:

```
<person  type="privat">
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>
```

Kan dokumentet ikke længere valideres:

In-line DTD skemaer:

Vi kan også indlægge DTD skemaet direkte i XML filen på denne måde:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE telefonliste [
<!ELEMENT telefonliste (person+)>
<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >

<!ATTLIST person type (arbejde|familie|venner) "arbejde">
<!ATTLIST telefonliste opdateret_af (kn|on|rh) #REQUIRED>
<!ATTLIST telefonliste opdateret NMTOKEN #REQUIRED>
]>

<telefonliste opdateret_af="kn" opdateret="2003-12-24">
```

```
<person type="arbejde">
<fornavn>Erik</fornavn>
<efternavn>Jensen</efternavn>
<telefon>67677889</telefon>
</person>
<person type="familie">
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>
<person  type="venner">
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>

</telefonliste>
```

Entiteter:

En entitet i DTD er et navn som i XML filen **erstattes** med en tekst – vi vil nu se på nogle eksempler herpå.

Vi har allerede tidligere set entiteter som & og ' som kan indsættes i XML dokumenter for at skrive '&' og ''' tegn som ellers ikke er OK i et XML dokument!

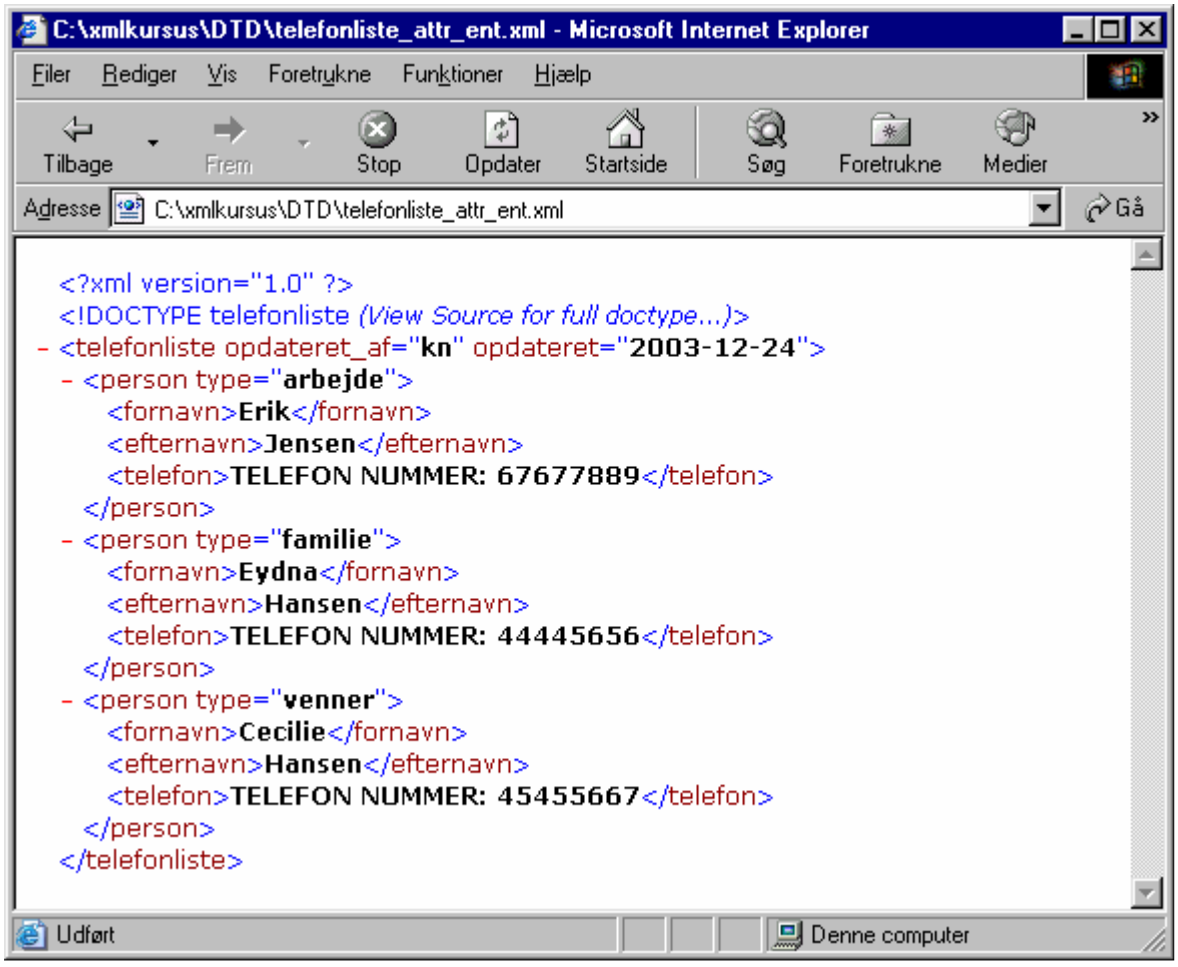
In-line skemaer er især anvendelige hvis der er tale om forholdsvis kortfattede skemaer! Vores sidste skema kan suppleres på denne måde:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE telefonliste [
<!ELEMENT telefonliste (person+)>
<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >

<!ATTLIST person type (arbejde|familie|venner) "arbejde">
<!ATTLIST telefonliste opdateret_af (kn|on|rh) #REQUIRED>
<!ATTLIST telefonliste opdateret NMTOKEN #REQUIRED>

<!ENTITY telefon " TELEFON NUMMER: ">
]>

<telefonliste opdateret_af="kn" opdateret="2003-12-24">
<person type="arbejde">
<fornavn>Erik</fornavn>
```

<efternavn>Jensen</efternavn>
<telefon>&telefon;67677889</telefon>
</person>
<person type="familie">
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>&telefon;44445656</telefon>
</person>
<person type="venner">
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>&telefon;45455667</telefon>
</person>

</telefonliste>

Vi har her erklæret en **entitet** 'telefon' som vi så kan indsætte som &**telefon**; i selve værdierne i **XML** dokumentet!



Som det ses **opløser** Internet Explorer entiteten og **indsætter** den korrekte værdi! En entitet er altså en slags konstant som er velegnet hvis vi bruger den samme tekst mange gange. Det er meget

almindeligt at skrive selvstændige DTD filer som består af mange definitioner på entiteter! Disse ekstra DTD filer kan så inkluderes i et DTD skema med en såkaldt parameter entitet!

Som et eksempel kan vi skrive følgende og gemme dette i **entiteter**.dtd:

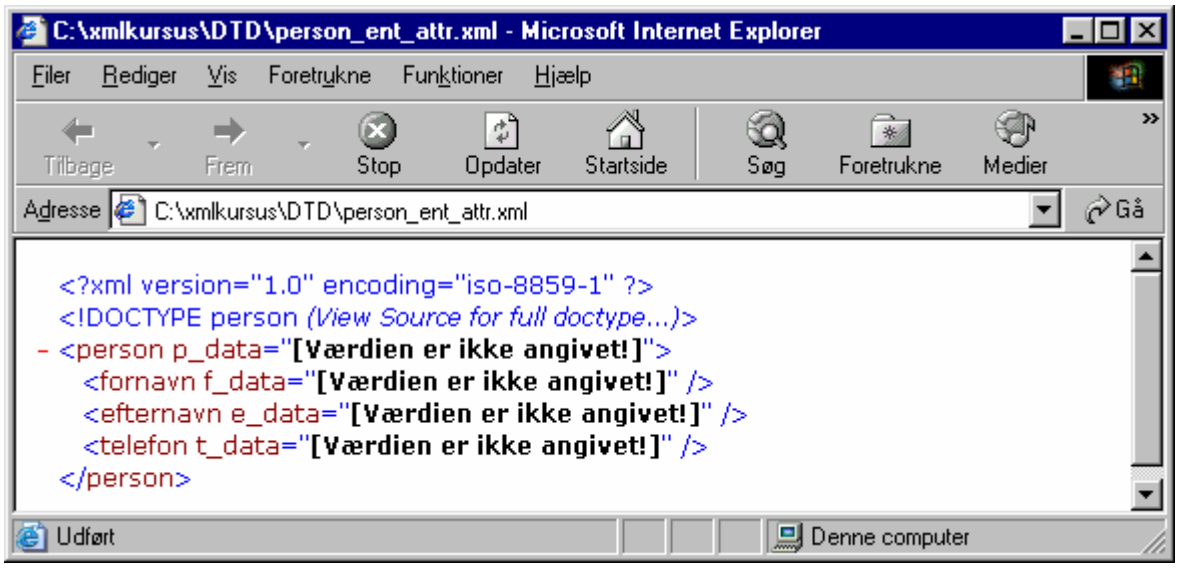
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!ENTITY fodnote 'Dette er en lille fodnote til listen.'>
<!ENTITY fil SYSTEM "information.txt">
```

Den sidste entitet anvender **SYSTEM** – det bevirker at **filen** information.txt læses og dens indhold indsættes som **erstatnings** tekst for entiteten!

Entiteter som værdier i attributter:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >

<!ENTITY x "[Værdien er ikke angivet!]">

<!ATTLIST person p_data CDATA "&x;">
<!ATTLIST fornavn f_data CDATA "&x;">
<!ATTLIST efternavn e_data CDATA "&x;">
<!ATTLIST telefon t_data CDATA "&x;">
```

Dette skema indsætter default værdier for en række attributter i skemaet! I alle tilfælde bruges en entitet som standard værdi. Dette er et nyttigt system når attributter skal gives en default værdi som så måske ikke 'over-rides' i instans dokumentet!



DTD skemaer kun med entiteter:

Man kan udmærket skrive en DTD som **kun** indeholder definitioner på entiteter. En DOCTYPE **behøver** altså ikke at definere et helt dokument med rod, sub elementer osv!

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book [
<!ENTITY tekst "Dette er en tekst! ">
<!ENTITY tekst1 "Dette er en fortsættelse af en tekst! ">
]>
<book>
<data>&tekst;&tekst1;</data>
</book>
```

In-line skema med inkludering af moduler:

Der findes to slags entiteter: almindelige entiteter som vi netop har set to eksempler på og **parameter** entiteter som bruges meget i DTD. Vi skal nu se et større DTD eksempel som illustrerer den række forhold omkring DTD skemaer. Vi bruger her til dels et in-line skema som så inkluderer nogle hjælpe skemaer:

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

<!DOCTYPE telefonliste [

<!ELEMENT telefonliste (person*|note*|adresse*)*>
<!ELEMENT note (#PCDATA) >
<!ELEMENT adresse (#PCDATA) >
```

```
<!ATTLIST adresse id ID #REQUIRED>

<!ENTITY % entiteter SYSTEM "entiteter.dtd">
<!ENTITY % person SYSTEM "person.dtd">
<!ENTITY % modul1 SYSTEM "modul1.dtd">

%modul1;
%entiteter;
%person;

]>

<telefonliste opdateret="140203" opdateret_af="kn" >

<person adresse="a1">
<fornavn>Jens</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44442323</telefon>
</person>
<person adresse="a2" type="familie">
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>
<person adresse="a2" type="venner">
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>
<adresse id="a1">Sortemosevej 22 2730 Herlev</adresse>
<adresse id="a2">Lillevej 2 2000 Frederiksberg</adresse>

<note>
&fodnote;
</note>
<note>
&fil;
</note>

</telefonliste>
```

Vi har her indført nogle nye elementer nemlig adresse og note:

```
<!ELEMENT telefonliste (person*|note*|adresse*)*>
<!ELEMENT note (#PCDATA) >
<!ELEMENT adresse (#PCDATA) >
<!ATTLIST adresse id ID #REQUIRED>
```

Skemaet bestemmer at der kan komme mange personer, mange noter og mange adresser i XML dokumentet. Vi har oprettet en primær **nøgle** id i elementet adresse. Hensigten er at hver **person** refererer til en adresse med en bestemt id. Derfor har hver person en attribut af typen IDREF! Nøglen i elementet person kan kaldes en fremmed nøgle – med begreber fra relationelle databaser.

Det mest epokegørende er at vi har flyttet dele af skemaet over i moduler. Vi inkluderer disse parameter entiteter med denne formel:

```
<!ENTITY % entiteter SYSTEM "entiteter.dtd">
<!ENTITY % person SYSTEM "person.dtd">
<!ENTITY % modul1 SYSTEM "modul1.dtd">

%modul1;
%entiteter;
%person;
```

NB En parameter entitet skal først erklæres og derefter evt. indsættes! Derfor bruger vi disse to faser!

Parameter entiteter bruges til indsætte data i selve **skemaet** – **almindelige** entiteter bruges især til at indsætte data i **XML** dokumentet (og også i skemaet)!

Almindelige entiteter indsættes med &...; - **parameter** entiteter indsættes med formlen %...;

I **XSD** skemaer anvendes en include til at inkludere andre skema filer – i DTD anvendes altså en parameter entitet!

Parameter entiteter er meget brugt i XHTML således at XHTML kan inkluderere flere eller færre moduler.

I værdien efter SYSTEM kunne vi også angive en Internet adresse eller en adresse på localhost f. eks. således:

```
<!ENTITY % entiteter SYSTEM "http://localhost/entiteter.dtd">
```

Inkluderede DTD skemaer med parameter entiteter:

Vi skal nu se på vores hjælpe DTD skemaer:

Indholdet af person.dtd er dette:

```
<!ELEMENT person (fornavn, efternavn, telefon)>
<!ELEMENT fornavn (#PCDATA) >
<!ELEMENT efternavn (#PCDATA) >
<!ELEMENT telefon (#PCDATA) >

<!ATTLIST person adresse IDREF #REQUIRED>
```

Dette svarer stort set til hvad vi allerede har defineret.

Indholdet af **entiteter**.dtd er:

```
<!ENTITY fodnote 'Dette er en lille fodnote til listen.'>
<!ENTITY fil SYSTEM "information.txt">
```

Denne fil indeholder blot almindelige entiteter. Værdien af en entitet kunne også være en længere velformet XML tekst! Vores information.txt er en intetsigende eksempel tekst!

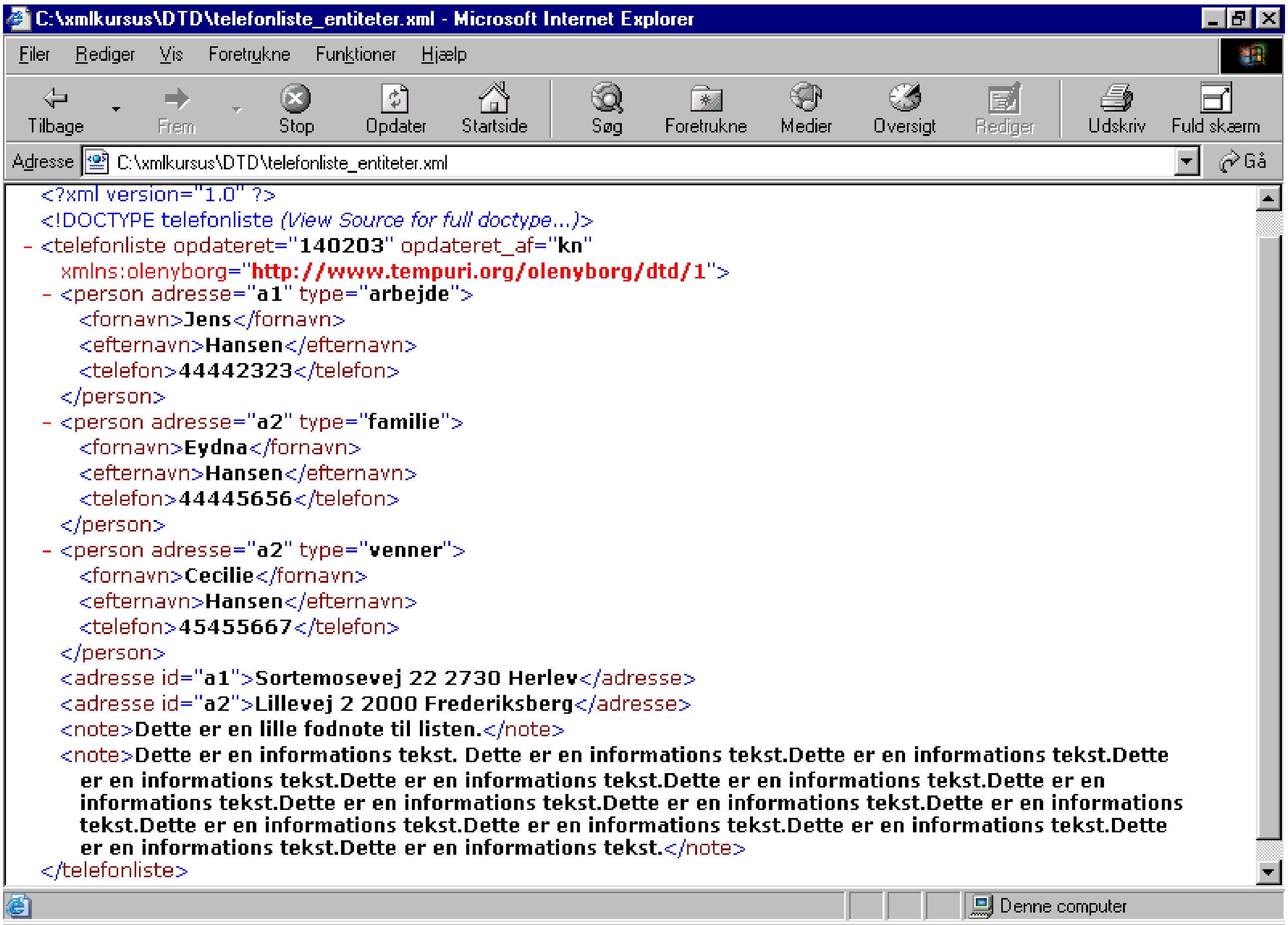
Indholdet af modul1.dtd er:

```
<!ATTLIST telefonliste opdateret NMTOKEN #REQUIRED >
<!ATTLIST telefonliste xmlns:olenyborg CDATA #FIXED
"http://www.tempuri.org/olenyborg/dtd/1" >
<!ATTLIST telefonliste opdateret_af (kn|on|rh) #IMPLIED>
<!ATTLIST person type (arbejde|familie|venner) "arbejde">
```

Det eneste nye er at elementet telefonliste har fået en ny attribut som er et namespace der automatisk indsættes – fordi er #FIXED – dvs. det kan ikke ændres! At anvende faste attributter er meget nyttigt i XML dokumenter!

XML dokumentet personliste:

Vi kan nu se vores nye XML dokument i Internet Explorer:



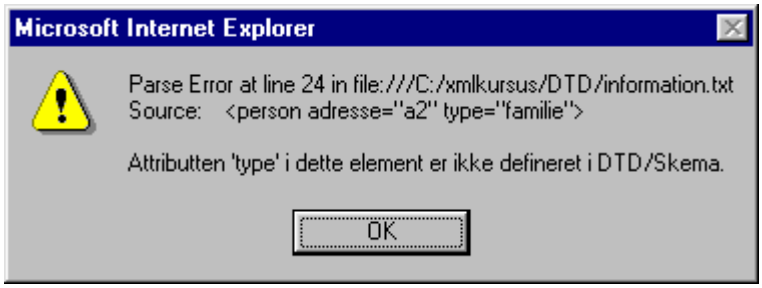

```
  <?xml version="1.0" ?>
  <!DOCTYPE telefonliste (View Source for full doctype...)>
- <telefonliste opdateret="140203" opdateret_af="kn"
    xmlns:olenyborg="http://www.tempuri.org/olenyborg/dtd/1">
  - <person adresse="a1" type="arbejde">
      <fornavn>Jens</fornavn>
      <efternavn>Hansen</efternavn>
      <telefon>44442323</telefon>
    </person>
  - <person adresse="a2" type="familie">
      <fornavn>Eydna</fornavn>
      <efternavn>Hansen</efternavn>
      <telefon>44445656</telefon>
    </person>
  - <person adresse="a2" type="venner">
      <fornavn>Cecilie</fornavn>
      <efternavn>Hansen</efternavn>
      <telefon>45455667</telefon>
    </person>
    <adresse id="a1">Sortemosevej 22 2730 Herlev</adresse>
    <adresse id="a2">Lillevej 2 2000 Frederiksberg</adresse>
    <note>Dette er en lille fodnote til listen.</note>
    <note>Dette er en informations tekst. Dette er en informations tekst.Dette er en informations tekst.Dette
      er en informations tekst.Dette er en informations tekst.Dette er en informations tekst.Dette er en
      informations tekst.Dette er en informations tekst.Dette er en informations tekst.Dette er en informations
      tekst.Dette er en informations tekst.Dette er en informations tekst.Dette er en informations tekst.Dette
      er en informations tekst.Dette er en informations tekst.</note>
  </telefonliste>
```

Vi kan se at vores information.txt – ret intetsigende – indsættes korrekt. Vi kan se at hver person har et adresse id som refererer til en bestemt adresse! Vores **namespace** node bliver også indsat i elementet telefonliste.

XML dokumentet kan derfor valideres:

Kontrol af de eksterne moduler:

Hvis vi bevidst vil indføre en fejl kan vi f. eks. slette attributten type i person elementet som er defineret i et eksternt modul1.dtd! Vi kan nu kontrollere om systemet virker:



Vi kan se at det rent faktisk kontrolleres hvad der står i den eksterne DTD! Attributten type er nu ikke længere defineret og må så ikke bruges i elementet person! Hvad der ikke udtrykkeligt er tilladt er i DTD forbudt. At noget er tilladt betyder i DTD at det er påkrævet!

Modeller i DTD:

sekvens:

I DTD skemaer findes – som i XSD skemaer – forskellige modeller. Vi har set eksempler på modellen sekvens som definerer at de og de elementer skal indgå i en bestemt rækkefølge.

choice:

En anden model kan kaldes en **choice** som kan illustreres med følgende DTD:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE book [
<!ELEMENT bog ((titel, forfatter, tekst, register)|(titel, forfatter))>
<!ELEMENT book (bog)*>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT forfatter (#PCDATA)>
<!ELEMENT tekst (#PCDATA)>
<!ELEMENT register (#PCDATA)>

]>

<book>
<bog>
<titel></titel>
<forfatter></forfatter>
</bog>
<bog>
<titel></titel>
<forfatter></forfatter>
<tekst></tekst>
<!--
```

<register></register>
-->
</bog>

</book>

Vi kan se at der er to muligheder for at oprette en bog – med lidt flere eller færre detaljer! Det tegn '|' som anvendes betyder 'or' (eller) og d.v.s. at man enten kan vælge det som står til venstre eller det som står til højre for or eller '|'. En choice kan indeholde mange indlejrede or sætninger og på den måde blive meget kompliceret!

Hvis vi prøver at validere dette XML dokument fås selvfølgeligt en fejl som vi bevidst har lavet ved at ud kommentere et element:

Vi kan også illustrere modellen choice med et XML dokument som opretter en række figurer:

<?xml version="1.0"?>
<!DOCTYPE figurer SYSTEM "figurer.dtd">
<figurer>
<rektangel>
<_x></_x>
<_y></_y>
<_hen></_hen>
<_ned></_ned>
</rektangel>

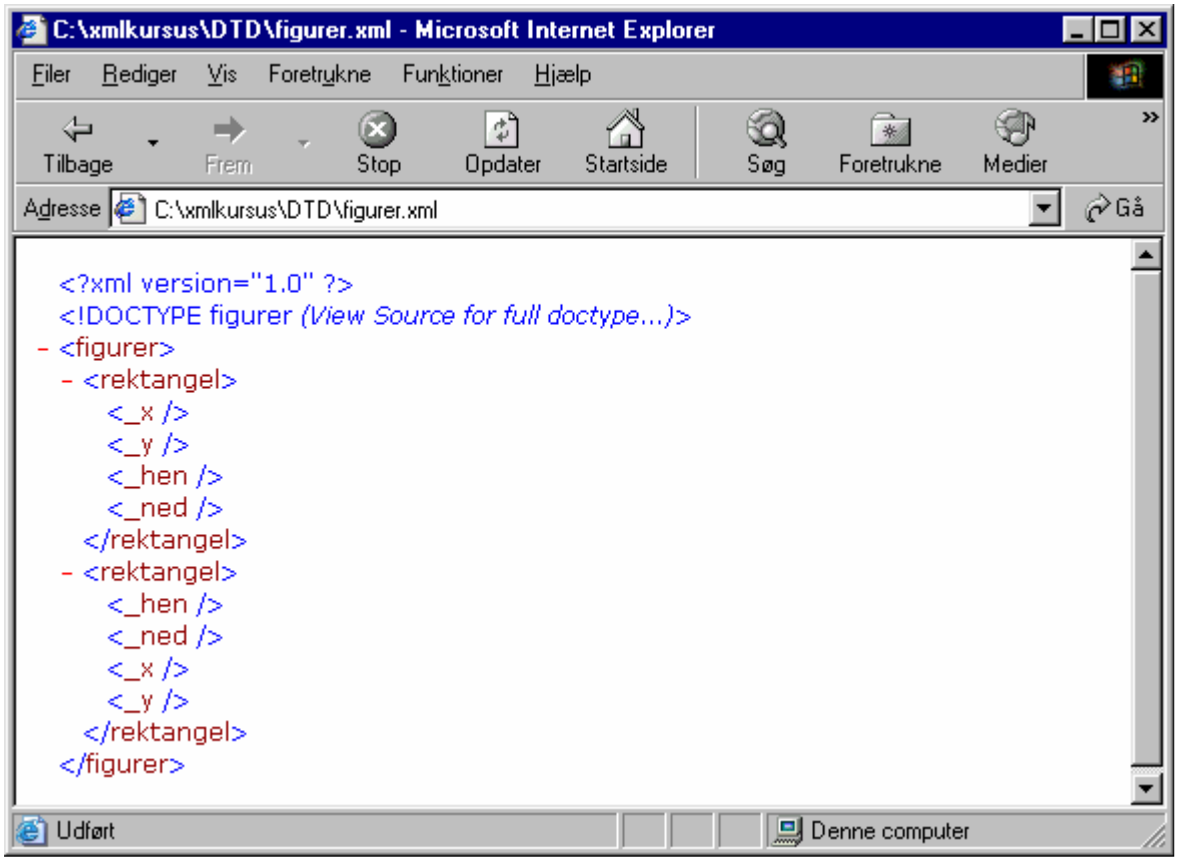
<rektangel>
<_hen></_hen>
<_ned></_ned>
<_x></_x>
<_y></_y>
</rektangel>
</figurer>

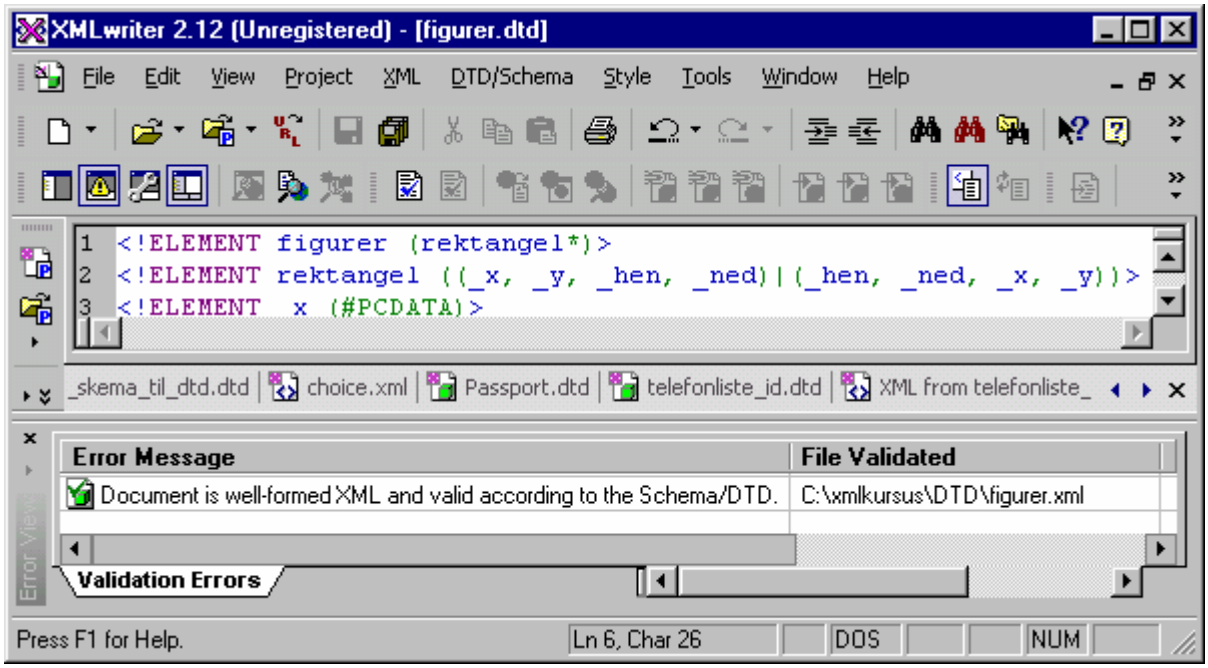
Et rektangel skal have 4 værdier men vi vil gerne kunne vælge i hvilken rækkefølge de skal komme! Hvis vi skal give to muligheder kræver det en OR eller '|' i DTD skemaet således:

<!ELEMENT figurer (rektangel)*>
<!ELEMENT rektangel ((_x, _y, _hen, _ned)|(_hen, _ned, _x, _y))>
<!ELEMENT _x (#PCDATA)>

<!ELEMENT _y (#PCDATA)>
<!ELEMENT _hen (#PCDATA)>
<!ELEMENT _ned (#PCDATA)>

At angive alle valg mulighederne ville være en besværlig proces i DTD – dette er meget nemmere i XSD skemaer – som vi skal se.





mixed:

Et element kan have en **model** der er **mixed** ved at den både indeholder tekst - #PCDATA - og child elementer. Dette er meget almindeligt i **narrative** dokumenter. Et eksempel kunne være:

<tekst>Dette er en lille tekst som er skrevet i <dato>1999</dato>.</tekst>

Et mixed element i DTD skal erklæres således idet #PCDATA **skal** stå først!:

<!ELEMENT tekst (#PCDATA|dato)*>

empty:

<!ELEMENT data EMPTY>

definerer et element som kun kan have attributter – intet tekst indhold!

<data /> eller <data type=”1” />

any:

Den sidste model som anvendes i DTD er any som definerer af et hvilket som helst kan være sub element hvis blot det er defineret i DTD skemaet!

<!ELEMENT data ANY>
<!ELEMENT x>

<!ELEMENT y>
<!ELEMENT z>

Elementet data kan nu indeholde alle elementer som er definerede i skenaet x, y, z og sig selv data! Også denne model kan anvendes til narrative, fortællende XML dokumenter.

Narrative, ikke data centriske dokumenter:

Vi har mest set på **data** centriske XML dokumenter og for deres vedkommende er det også vigtigt at der foreligger et skema. På samme måde som data i en database skal defineres i et bestemt format for at kunne bruges effektivt. Et data centrisk XML dokument ligner en **tabel** med rækker og kolonner, poster og felter.

HTML (og dets fader SGML) blev imidlertid skrevet til primært **narrative** eller fortællende dokumenter (rapporter, artikler) som også har elementer med et mixed indhold. De tags og mærker som anvendes i HTML kommer i høj grad i tilfældig orden. Et afsnit kan indeholde noget med fed skrift eller en tabel eller et billede! Narrative dokumenter er ikke nær så strukturerede som data centriske dokumenter! Rækkefølgen af elementerne er meget mere tilfældig eller uberegnelig!

Vi kan f. eks. skrive en narrativ XML tekst på denne måde:

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

<beretning>
<rubrik>Noget om <sted>DK</sted> i aaret <tid>2003</tid>.</rubrik>
I <sted>Danmark</sted> er der mange steder f.eks. <sted>Jylland</sted> og mange
<begreb>kirker</begreb>.
Tiden er nu i <tid>2003</tid>.
I dette land bor <person>Jens Jensen</person> og <person>Rosalina Hansen</person>.
<rubrik>Noget om <person>Elizabeth</person> og <sted>England</sted> i aaret
<tid>2003</tid>.</rubrik>
I <sted>Storbritannien</sted> er der mange byer og mange <begreb>pubber</begreb>.
Vi er nu i <tid>2003</tid>.
<note><rubrik>Dette er en rubrik i en note</rubrik>Dette er en note om landet England. Dette er en
note om landet England.
Dette er en note om landet England. Dette er en note om landet England.
Dette er en note om landet England. </note>
I dette land bor <person>John</person> og <person>Elizabeth Knight</person>,
som begge arbejder inden for <begreb>turist industrien</begreb>.
</beretning>
```

Dette dokument er vel formet og kan vises i en browser f. eks. Roden beretning indeholder elementer som rubrik, sted eller person som er opmærkninger af en bestemt slags data! Denne slags mærker gør at søgemaskiner kan søge langt mere effektivt i sådanne dokumenter. Man kan f. eks. let finde alle personer eller alle steder. Dette format er langt mere struktureret end gammeldags HTML!

Vi kan sætte et stylesheet på dokumentet for at vise strukturen:

Vi kan se at alle data af en vis type er formatteret med en bestemt farve! Alle begreber er f. eks. markeret med en gul baggrundsfarve.

Pointen ved fortællende XML dokumenter er at vi skal skrive et skema som kan rumme mange forskellige kombinationer af elementer – men vi skal også gøre os klart om alle tænkelige kombinationer er OK! I dette eksempel er det f. eks. muligt at lave en rubrik (overskrift) inden i en note! Dette kunne vi have 'forbudt' – det er disse overvejelser der ligger bag ved skemaer i DTD. Et eksempel fra HTML: Man kan ikke lovligt have en <title> inden i en <body> i HTML. Dette er illegalt fordi DTD'en for HTML har defineret det sådan. En <title> skal ligge inden i en <head>!

Vi kan f. eks. skrive dette DTD skema til eksempel dokumentet:

```
<!ENTITY % elementer "#PCDATA|person|sted|begreb|tid">
<!ELEMENT beretning (%elementer;|rubrik|note)*>
<!ELEMENT person (#PCDATA) >
<!ELEMENT sted (#PCDATA) >
<!ELEMENT begreb (#PCDATA) >
<!ELEMENT tid (#PCDATA) >
<!ELEMENT rubrik (%elementer;)* >
<!ELEMENT note (%elementer;|rubrik)* >
```

Vi kan her se at vi anvender parametre for at gøre skemaet mere overskueligt og nemmere at revidere!

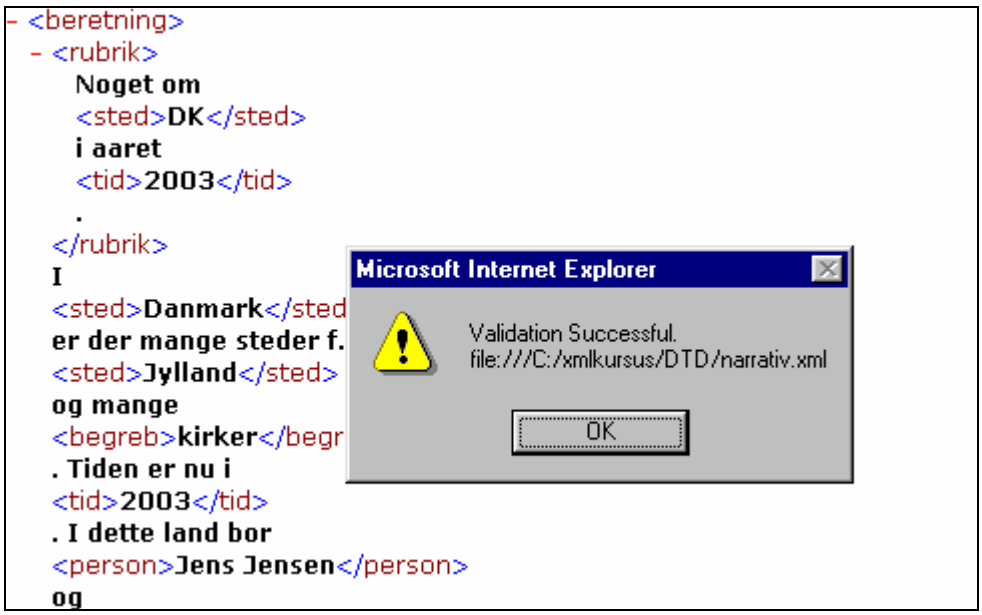
<!ENTITY % elementer "#PCDATA|person|sted|begreb|tid">
<!ELEMENT beretning (%elementer;|rubrik|note)*>

Dette betyder at roden beretning er en choice model som kan indeholde en vilkårlig række af enten rubrik eller note eller de elementer som er defineret i elementer entiteten!

Ved at indføre en variabel eller konstant %elementer; kan vi meget nemmere skrive skemaet! Vi skal blot huske at skema processoren hele vejen erstatter %elementer; med den tekst streng som står i 1. linje!

Både en beretning, en rubrik og en note kan først og fremmest indeholde #PCDATA d.v.s. almindelig tekst. Disse elementer har altså et mixed indhold – de kan bestå af ren tekst blandet med visse elementer i en eller anden rækkefølge!

NB Hvis vi prøver at validere et XML dokument som vises i en browser med et stylesheet får vi at vide at det ikke kan lade sig gøre fordi dokumentet ikke er et XML dokument! Men hvis vi fjerner vores stylesheet fra XML dokumentet kan det valideres!



Officielle entitets definitioner:

Sammen med mange skemaer findes **mængder** af skemaer som definerer bestemte entiteter. Det følgende (som anvendes af skemaet **DocBook,** som også oprindeligt var en SGML applikation) er et eksempel herpå – her defineres en række specielle **tegn**:

<!ENTITY aacute "á"> <!-- LATIN SMALL LETTER A WITH ACUTE -->
<!ENTITY Aacute "Á"> <!-- LATIN CAPITAL LETTER A WITH ACUTE -->

```
<!ENTITY acirc          "&#x00E2;"> <!-- LATIN SMALL LETTER A WITH CIRCUMFLEX
-->
<!ENTITY Acirc          "&#x00C2;"> <!-- LATIN CAPITAL LETTER A WITH
CIRCUMFLEX -->
<!ENTITY agrave         "&#x00E0;"> <!-- LATIN SMALL LETTER A WITH GRAVE -->
<!ENTITY Agrave         "&#x00C0;"> <!-- LATIN CAPITAL LETTER A WITH GRAVE -->
<!ENTITY aring          "&#x00E5;"> <!-- LATIN SMALL LETTER A WITH RING ABOVE -
->
<!ENTITY Aring          "&#x00C5;"> <!-- LATIN CAPITAL LETTER A WITH RING
ABOVE -->
<!ENTITY atilde         "&#x00E3;"> <!-- LATIN SMALL LETTER A WITH TILDE -->
<!ENTITY Atilde         "&#x00C3;"> <!-- LATIN CAPITAL LETTER A WITH TILDE -->
<!ENTITY auml           "&#x00E4;"> <!-- LATIN SMALL LETTER A WITH DIAERESIS -->
<!ENTITY Auml           "&#x00C4;"> <!-- LATIN CAPITAL LETTER A WITH DIAERESIS --
>
<!ENTITY aelig          "&#x00E6;"> <!-- LATIN SMALL LETTER AE -->
<!ENTITY AElig          "&#x00C6;"> <!-- LATIN CAPITAL LETTER AE -->
<!ENTITY ccedil         "&#x00E7;"> <!-- LATIN SMALL LETTER C WITH CEDILLA -->
<!ENTITY Ccedil         "&#x00C7;"> <!-- LATIN CAPITAL LETTER C WITH CEDILLA -->
```

Et eksempel: Et stort C med en cedille kan så skrives som &**Ccedil**; Sådanne entiteter er almindelige i både HTML og XML. I forbindelse med **MathML** – et XML sprog til matematik – er der skrevet et hav af entiteter for at kunne skrive matematiske udtryk som tekst i XML dokumenter!

Anerkendte DTD skemaer – XML applikationer:

Eksempel: DocBook:

Man kan på adressen http://www.docbook.org hente materiale om skemaet DocBook. DocBook var oprindeligt en SGML applikation men findes nu i en XML version. Formålet med dette skema er at udgive og få anerkendt et skema som alle tekstskrivere og forfattere kan bruge. Fordelene ved at et format blev anerkendt er enorme fordi en bog eller tekst derved meget let kan transformeres fra et format til et andet! Linux dokumentationen er f. eks. skrevet i DocBook lige som mange bøger efterhånden er det.

Med DocBook kan man producere 'portable' dokumenter der kan læses af alle programmer på alle styresystemer – Windows, Unix, Linux eller Mac! DocBook ejes ikke af et privat firma og hele projektet er freeware. DocBook har også en udførlig dokumentation som forklarer meningen med skemaet.

Fordelen ved DocBook er også at der findes **værktøjer** til sproget – f. eks. tekstbehandleren Emacs - og færdige **stylesheets** til formatering og visning af teksten! Det er ikke meningen at læserne skal præsenteres for den rene DocBook markup!

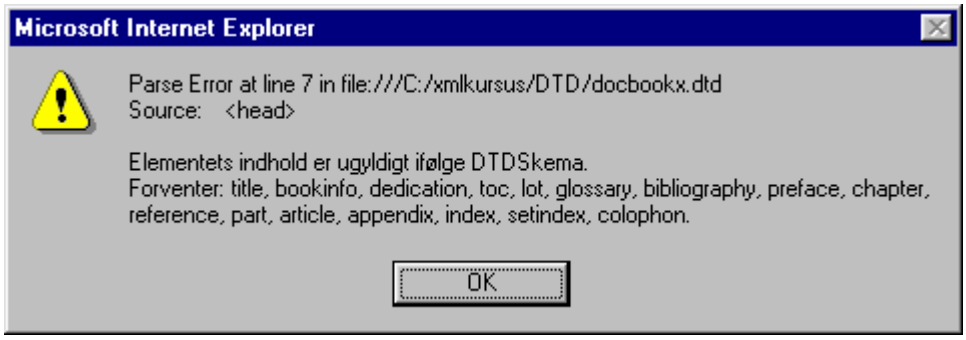
Man kan på den nævnte adresse hente XML skemaet **docbookx**.dtd. Hvis vi har gemt filen docbookx.dtd lokalt – og det er mest praktisk! ellers skal der kaldes op til Internettet hver gang der skal valideres! - kan vi bruge den til at validere denne XML fil:

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

<!DOCTYPE book [
<!ENTITY % modul1 SYSTEM "docbookx.dtd">
%modul1;
]>

<book>
<title>Dette er bogens titel.</title>
<bookinfo>
<author>
<firstname>John</firstname>
<surname>Johnson</surname>
</author>
</bookinfo>

<chapter>
<title>Dette er et kapitel.</title>
<para>Dette er et afsnit.</para>
</chapter>

</book>
```

Roden i DocBook dokumenter **skal** være <book>!

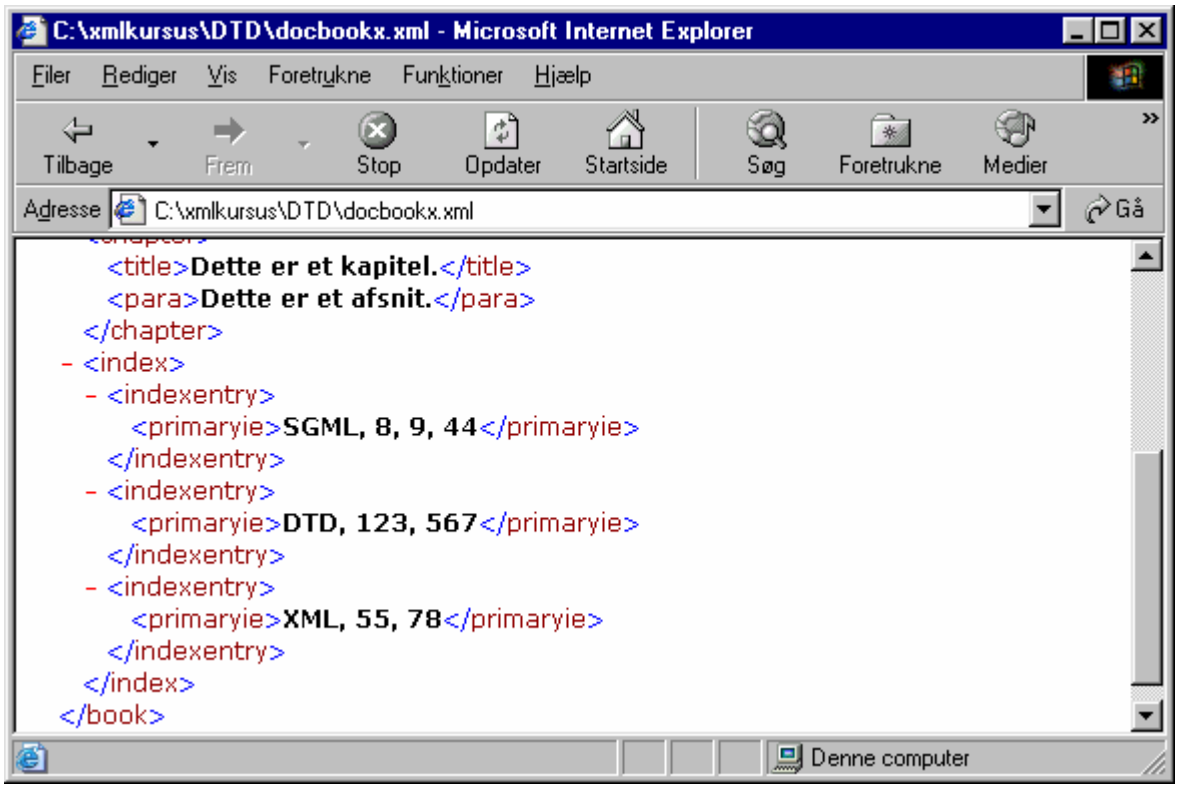
DocBook består af et væld af elementer og attributter. Fordelen er at det er et anerkendt format som alle kan referere til! Det er ikke bundet til nogen producent, firma, software eller styresystem!

Hvis vi indfører en fejl i dokumentet og opretter en <head> uden om <title> kan dokumentet ikke valideres – dette er ikke OK i DocBook:



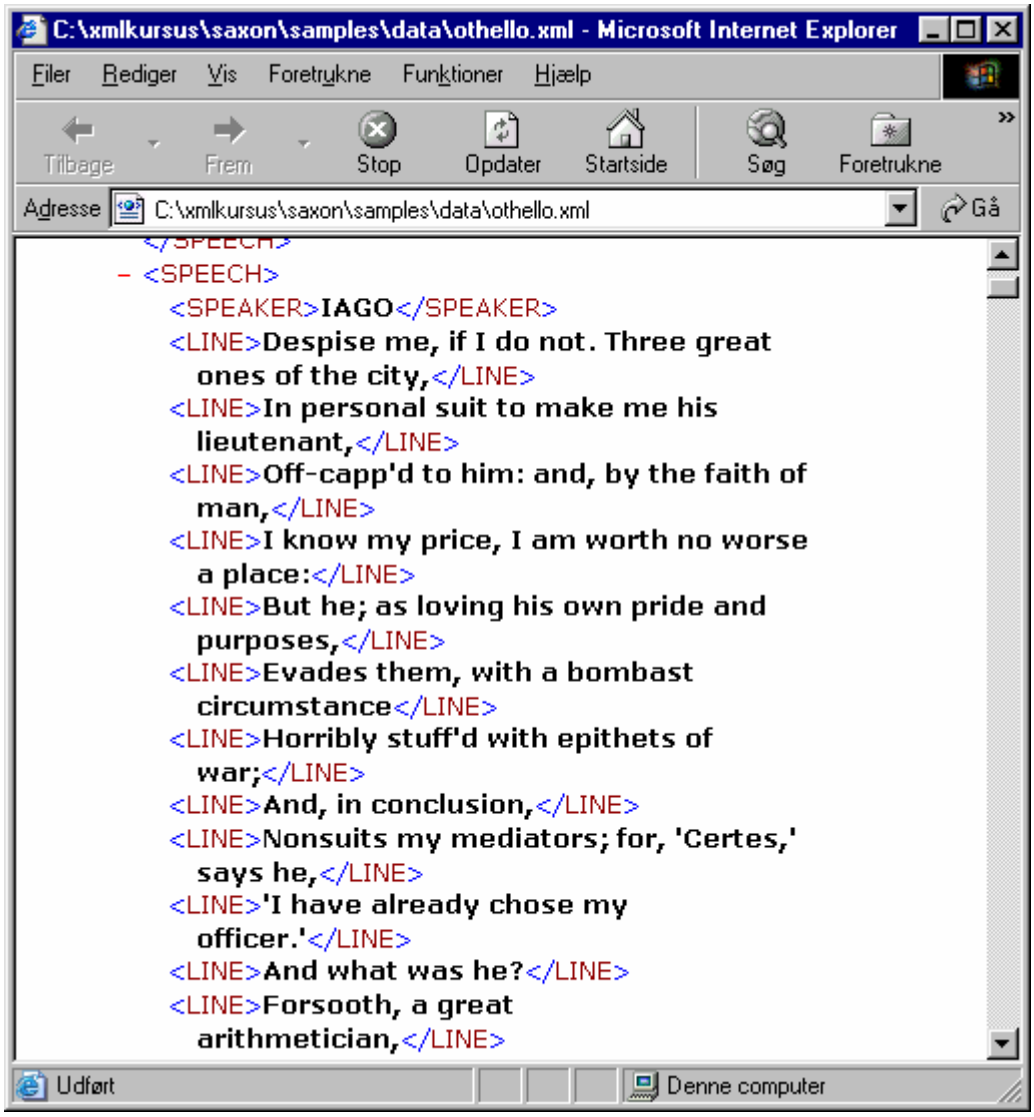
Meget nyttigt får vi også at vide hvilket sub elementer der overhovedet er gyldige i denne kontekst!

I DocBook kan man bl. a. også definere et index eller et register:



Et andet eksempel på et dokument skema er **TEI** – Text Encoding Initiative. Dette skema er også skrevet med henblik på artikler og bøger – især klassisk eller videnskabelig litteratur. Shakespeare er f. eks. udgivet med TEI skemaet.

TEI gør det muligt præcist at opdele de klassiske tekster i små dele og komponenter og derved bliver det muligt at bearbejde dem systematisk! Adressen er http://www.tei-c.org.



Dette eksempel er **Shakespeares Othello** i XML kodning.

XHTML – HTML i XML format:

XHTML er nok den bedst kendte og anerkendte XML applikation. XHTML er et XML sprog som har **omdefineret** og opstrammet HTML som længe har været brugt på web sider.

HTML er en SGML applikation – altså egentligt slet ikke en XML applikation. XHTML er derfor – principielt – et helt nyt sprog! Forskellene mellem HTML og XHTML er dog ikke så store.

Nogle af ulemperne ved HTML har været at de forskellige browsere ikke har fortolket HTML på samme måde. Desuden har det været **accepteret** at skrive 'sjusket' HTML kode som de forskellige browsere så har forsøgt at leve med. I mange tilfælde har browseren måtte **fortolke** hvad sidens forfatter egentligt har ment!

Der findes i dag mange værktøjer som kan transformere et HTML dokument til et XHTML dokument. På http://www.w3.org kan downloades sådanne værktøjer. De to vigtigste regler er at alle elementer skal lukkes og at alle attributters værdier skal sættes i anførsels tegn!

Dette er altså **forbudt** f. eks.:

<p>Dette er et afsnit .....

<img src=etbillede.jpg>

Desuden skal et XHTML dokument have en **DOCTYPE** og et **namespace** i roden. XHTML kan også have en processing instruction!

XHTML er defineret af W3C Konsortiet i flere DTD skemaer. De kan downloades fra http://www.w3.org.

Vi skal her se på den **variant** som hedder XHTML version 1.0 **Strict** – den tilsvarende skema fil hedder xhtml1-strict.dtd. Denne udgave har vist sig at være den mest levedygtige.

Vi kan skrive et XHTML dokument sådan:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"      "xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Dette er XHTML Strict.</title>
 <meta name="author" content="http://www.on.dk" />
</head>
<body>
<h3>Dette er XHTML Strict.</h3>
<li>Et punkt i en liste!</li>
</body>
</html>
```

Linjerne:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"      "xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

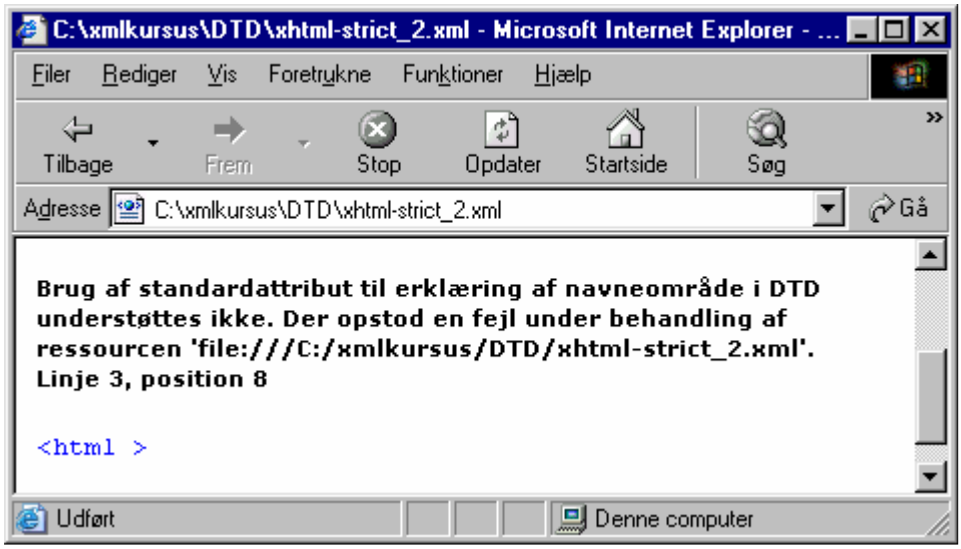
skal indgå i enhvert XHTML **Strict** dokument.

Hvis vi ønsker det **kan** vi dog evt. skrive en DOCTYPE med en tom public ID således:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "" "xhtml1-strict.dtd">
```

Roden **skal** være html og den **skal** have dette namespace.

Hvis vi blot skriver <html> uden et xml navnerum får vi denne besked i Internet Explorer:



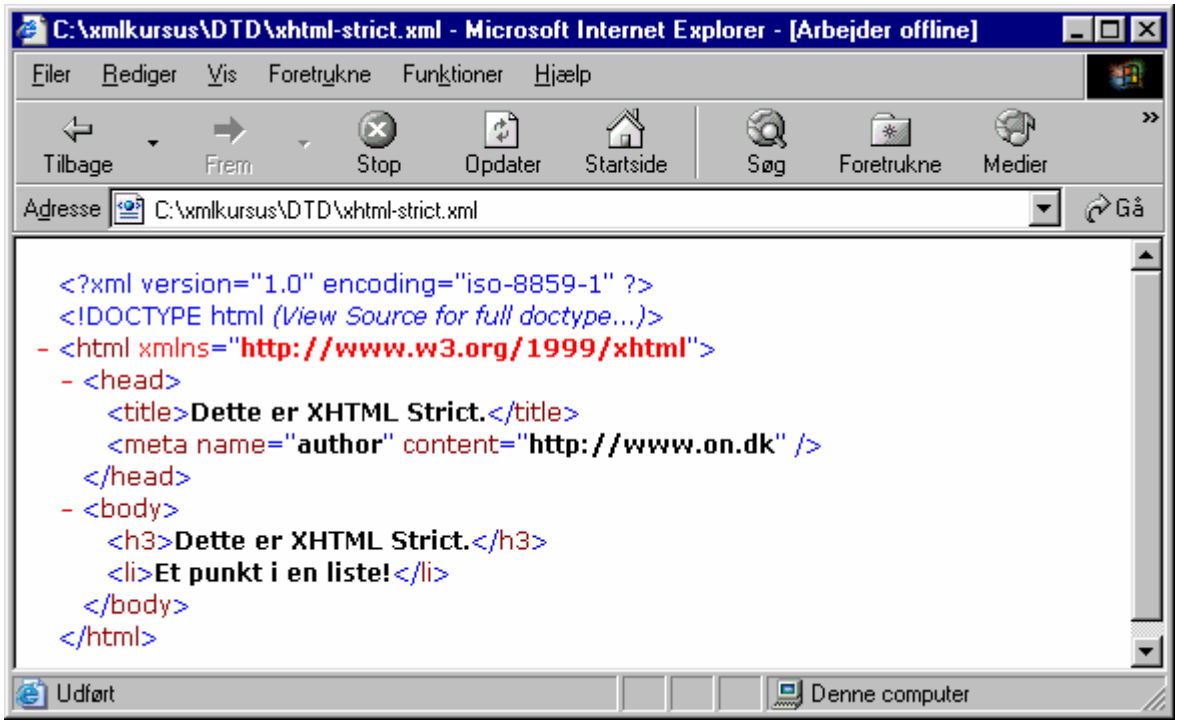
Navneområdet eller XHTML namespacet **skal** altså erklæres!

Her har vi forudsat at DTD'en er **downloadet** og ligger **lokalt**. Ellers skulle der henvises til DTD skemaets beliggenhed på siden http://www.w3.org (ellers kan XML dokumentet **ikke** valideres):

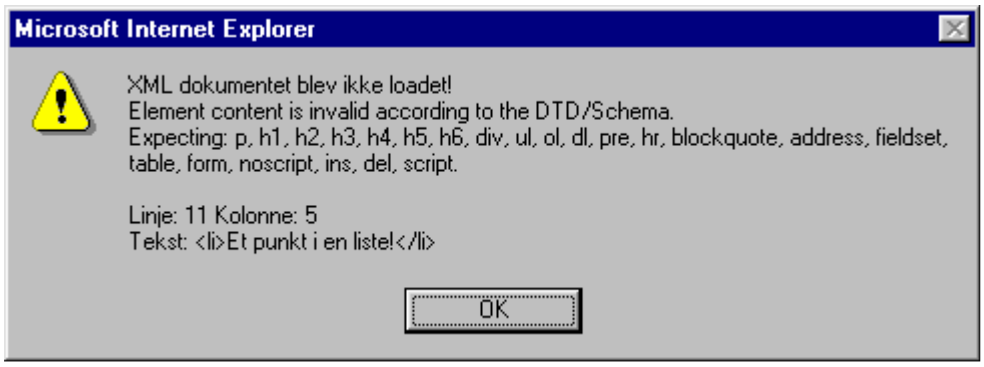
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html .....
```

Ovenstående DOCTYPE har både en public ID og en fil placering (to strenge efter hinanden). Dette er **nødvendigt** med XHTML dokumenter.

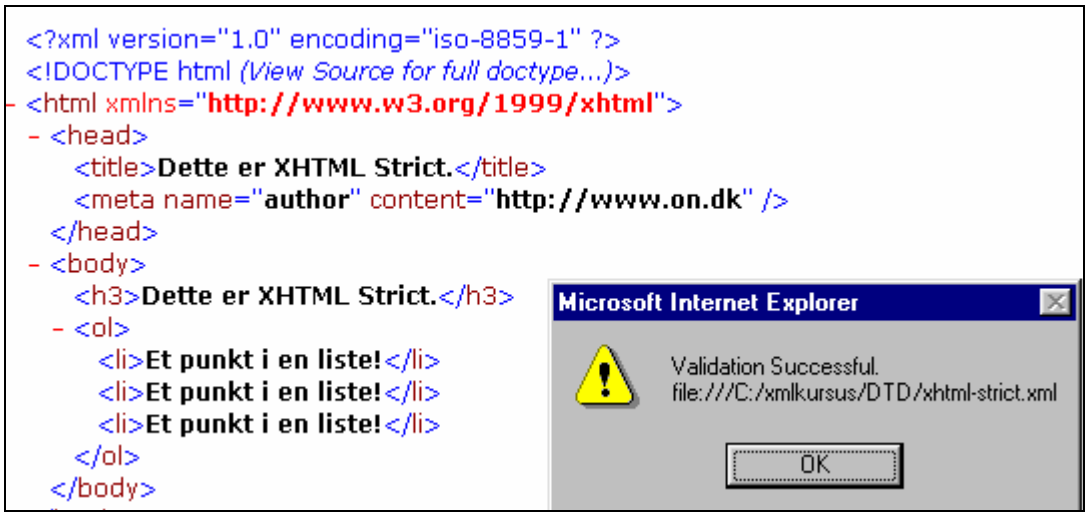
Dokumentet vises OK i en browser:



Hvis vi prøver at validere dokumentet med vores eget script fås:



Vi kan se at valideringen rent faktisk fungerer! XHTML kan ikke have en **<li>** (et punkt i en **liste**) uden at det er **indlejret** i et <ol> eller <ul> element! Hvis vi retter denne fejl bliver dokumentet valideret.



Et almindeligt fænomen i HTML er ikke at skrive **slut** mærker til elementer. F. eks. at skrive en liste således:

Et punkt
Et andet punkt

<table>
...

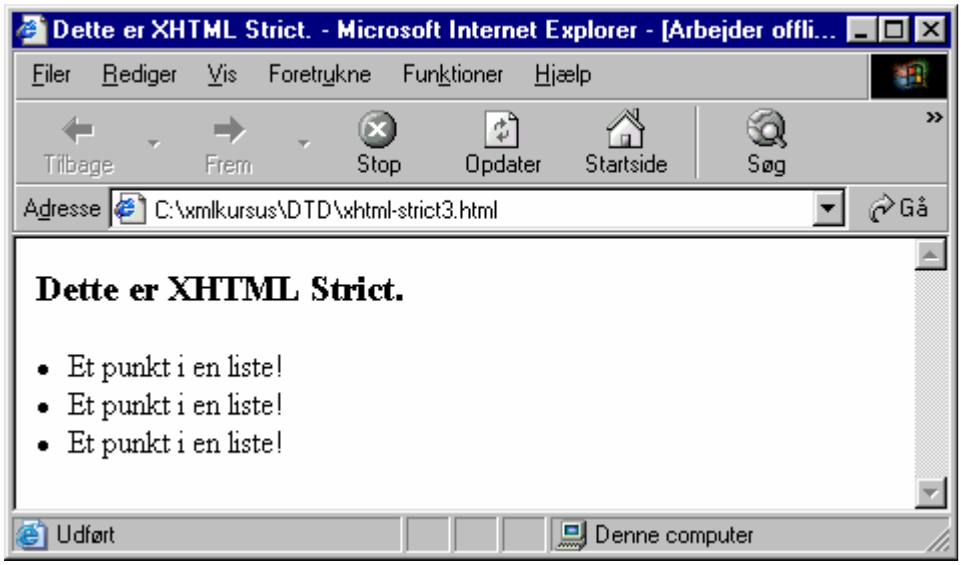
Her er der flere mærker som ikke bliver **lukkede** – men det bliver oftest tolereret i HTML. I XHTML kan dette ikke lade sig gøre – fordi XHTML dokumentet **skal** være gyldig XML! De samme krav gælder!

Vi kan f. eks. skrive dette dokument og gemme det som 'ikke_xhtml.html':

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "" "xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<title>Dette er XHTML Strict.</title>
<body>
<h3>Dette er XHTML Strict.</h3>
Et punkt i en liste!
Et punkt i en liste!
Et punkt i en liste!

Vi kan se mange **'fejl'** her. Der er ingen **<head>** med alligevel en **<title>**. bliver ikke lukkede. <body> og <html> bliver ikke **lukkede**! Alligevel vises dette dokument OK i en browser:



Dette skyldes kun at det er **gemt** som HTML – også selv om det på en vis måde er en XML fil! Hvis teksten gemmes som et **XML** dokument f. eks. som 'xhtml-strict3.xml' kan det **ikke** valideres – **ikke** engang vises!

Vi skal senere vende tilbage til brug af XHTML som findes i flere versioner.

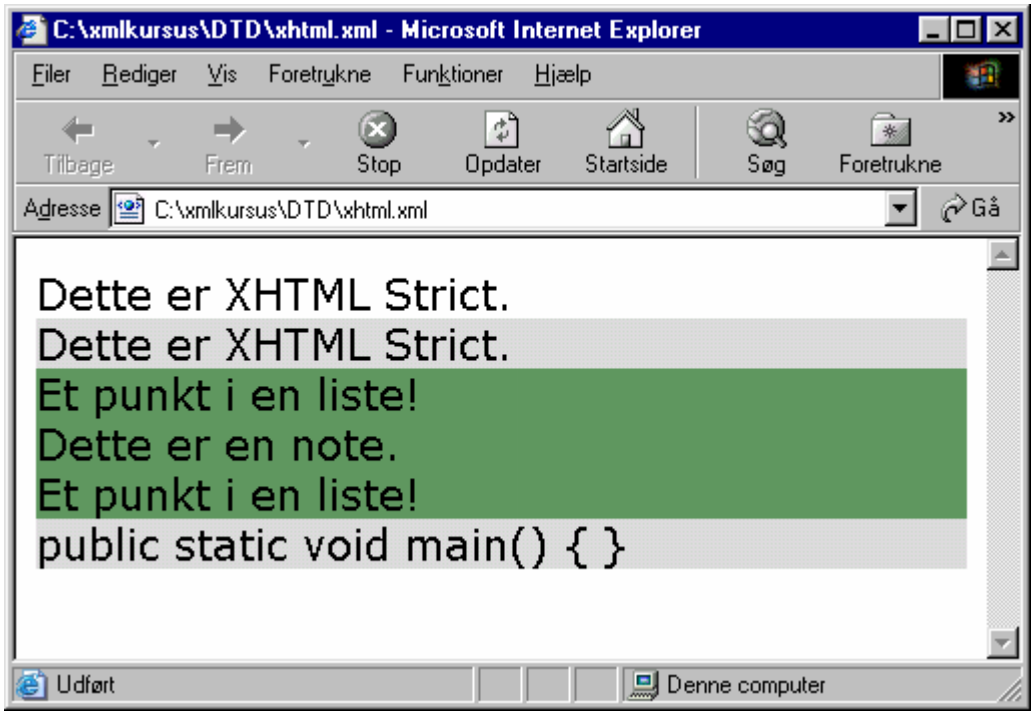
Den sidste version af XHTML version 1.1 bygger på en **modularisering** således man kun inddrager de **moduler** som er nødvendige. Dette gennemføres netop med de **parameter** entiteter som vi har set eksempler på!

I XHTML kan anvendes alle de komponenter som kan anvendes i XML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="strict.css" type="text/css"?>
<!DOCTYPE html PUBLIC "" "xhtml1-strict.dtd"
[
<!ENTITY note "Dette er en note.">
]
>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
 <title>Dette er XHTML Strict.</title>
</head>
<body>
<h3>Dette er XHTML Strict.</h3>
<ul>
<li>Et punkt i en liste!</li>
<li>&note;</li>
<li>Et punkt i en liste!</li>
</ul>
<![CDATA[
public static void main() {

}
]]>
</body>
</html>
```

Her er brugt en række **elementer** som **aldrig** ville forekomme i en HTML sammenhæng! Men dette er et fuldt gyldigt XHTML dokument (selv om **Microsoft** parseren går 'død' i vores CDATA sektion!):



SAXON automatisk produktion af DTD:

Hvis man installerer Java og SAXON – som beskrevet – tidligere får man også et færdigt program med i pakken som kan generere DTD skemaer ud fra et hvilket som helst XML dokument! Java klassen hedder DTDGenerator og kan kaldes på denne måde – her er det forudsat at saxon.jar biblioteket ligger i samme mappe:

java -classpath .;saxon.jar DTDGenerator telefonliste.xml > telefonliste_DTD.txt

Programmet genererer så et DTD skema for telefonliste.xml! Dette gøres typisk som det første skrift. Derefter kan man se på resultatet og finpudse skemaet i hånden!