XML generelt:

XML:

XML – **eXtensible** Markup Language – er et abstrakt **metasprog** eller et **format** som er defineret i en standard fra W3C konsortiet (DOM modellen). XML skal bruges til at skabe 'XML applikationer' eller **konkrete** anvendelser af XML.

Det er meget anbefalelsesværdigt at gå ind på adressen http://www.w3.org og checke de officielle dokumenter der beskriver XML! Her finder man en fuldstændig specifikation af alle de elementer som omtales i dette kursus! De fleste af dokumenterne er skrevet i et klart og tydeligt sprog.

XML stammer fra **SGML** – Standard Generalized Markup Language, der blev udviklet i 1970-erne og tildels brugt i det amerikanske militær og statsvæsen. SGML er nu en officiel ISO standard (International Standards Organization). Ulempen ved SGML var at syntaksen var uhyre kompliceret. HTML var en SGML applikation (konkretisering). DTD skemaer er – som vi skal se – en SGML applikation og det er dokument formatet DocBook også (som omtales senere i kurset).

XML blev designet til Internettet og til brug på Web sider som HTML. I det senere forløb er XML imidlertid blevet brugt til meget andet – databaser, interne **messages** i computeren og på netværk, konfiguration af programmer o.s.v. Efterhånden har XML gået sin sejrs march på alle mulige områder!

XML minder om **HTML** – den afgørende forskel er at HTML består af ca 100 forskellige **forud** definerede tags eller mærker (så som <b> eller <table>), mens der egentligt **ikke** findes forud definerede mærker i en XML applikation.

En anden afgørende forskel er at HTML er et **sprog** som alle **browsere** 'forstår' dvs at bestemte HTML mærker altid får browseren til at **formatere** (tegne og male!) web siden på en bestemt måde. Ideen med XML er netop at **adskille** data fra deres formatering (helt **modsat** HTML)! I princippet er der ingen browsere der 'forstår' XML (en XML applikation) – fordi ingen tags er forud definerede!

En tredie forskel er at de fleste browsere er meget 'tolerante' over for fejl i HTML dokumentet – hvorimod INGEN formelle fejl tolereres (af en **parser** eller **processor**) i XML! Vi skal snart se eksempler herpå.

Træ struktur:

Alle XML dokumenter er **grafer** og gyldige **træ** strukturer. Et konkret eksempel herpå kunne være:

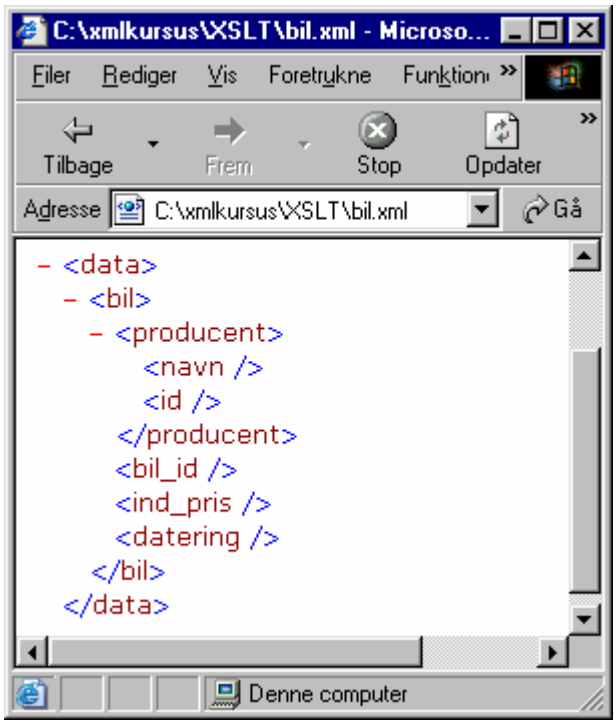
<data>Data.</data>

Vi kan omsætte dette gyldige XML dokument i denne træ struktur:

1. hele dokumentet
    a. data (det eneste og første **barn** af hele dokumentet som er **parent**)
        i. tekst node med værdien 'Data.' (**barn** af data noden)

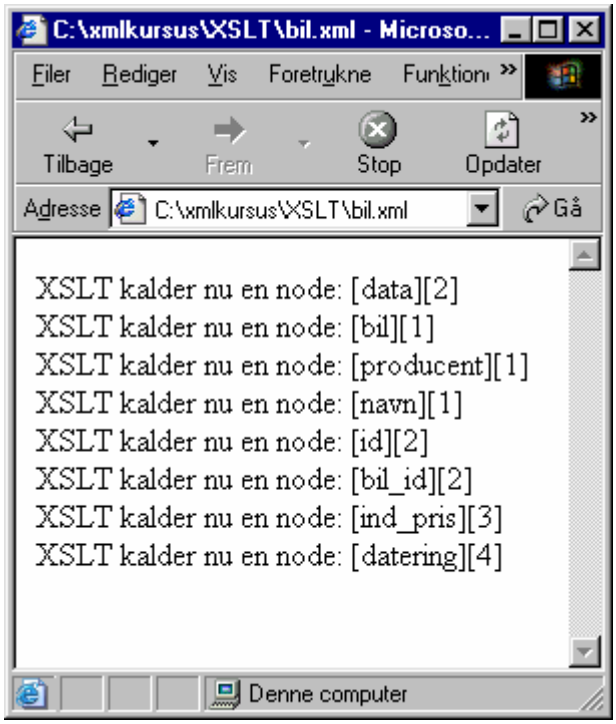
Dette uhyre simple dokument har altså tre **niveauer**!

Et andet eksempel kunne være denne **bil**:



Vi kan se at træet har forskellige **dybder**. I første **geled** kommer data, i næste **geled** kommer bil, i næste **geled** kommer producent, bil_id, ind_pris og datering! Når vi – i XML - taler om **positionen** taler vi altid om elementets nummer (position) i det **pågældende** geled eller i det pågældende niveau! I XML er det vigtigt at holde fast i at vi altid befinder os i en bestemt kontekst – altså et bestemt sted i træet!

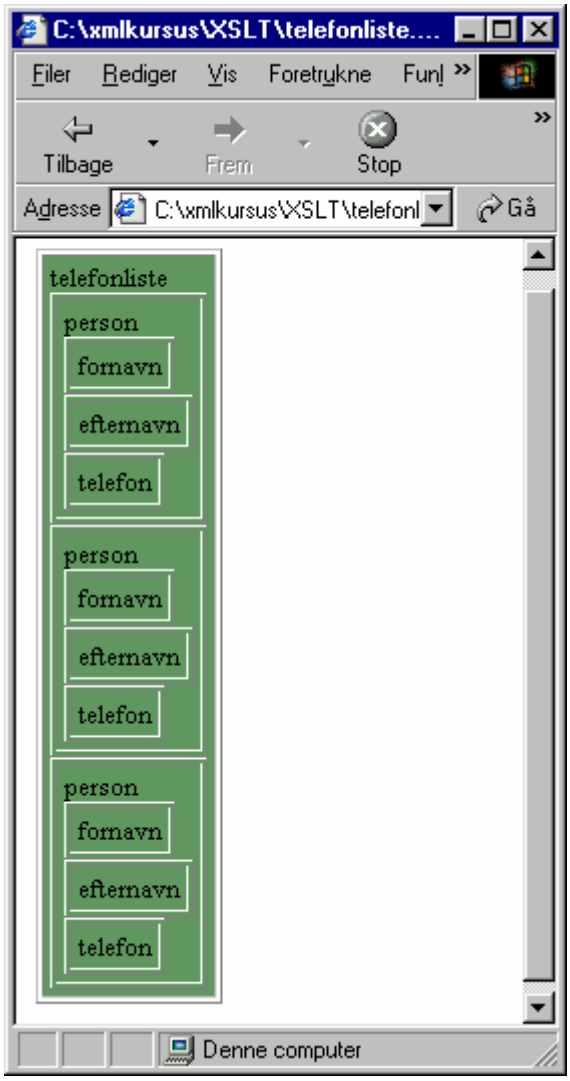
Nedenstående viser de forskellige elementers **dybde** (tallet i de skarpe parenteser) – her produceret af en stylesheet **processor**:



Positionen er i XML 1 baseret – den starter med nr 1. Det kan forvirre at elementet data har nummer 2 men det skyldes at selve xml erklæringen – som vi skal se – er nummer 1! Læg mærke til at både id og bil_id har nummer 2 – men på to forskellige niveauer!

Kinesiske æsker eller russiske dukker:

Et XML dokument består af kinesiske **æsker** eller russiske dukker uden på og inden i hinanden. Elementer der ligger uden på er **wrappers** eller **containers**! Et eksempel på en telefonliste som kinesiske æsker kan vises sådan:



Vi kan se at super containeren er elementet telefonliste og at den indeholder mindre containere af typen person!

Besøgs eller Visiting algoritmer:

Disse algoritmer – metoder - definerer i hvilken **rækkefølge** et XML dokument læses! Hvis vi tager et lidt mere indviklet eksempel end vores første kunne det være:

<data>
<person>Ole</person>
<person>Erik</person>
</data>

Træet ser nu således ud:

1. hele dokumentet
    a. data
        i. person
            1. tekst
        ii. person
            1. tekst

Objekterne i og ii er **børn** af data! i og ii er også **siblings** eller søskende!

En XML **parser** læser dokumentet i en bestemt **rækkefølge** - nemlig (normalt) **'dybde** først':

1. hele dokumentet
2. data
3. person nr 1
4. **teksten** for person nr 1
5. person nr 2
6. **teksten** for person nr 2

Man kan bruge andre algoritmer der f. eks. læser **'bredde** først' men disse bruges normalt ikke i XML.

Princippet i XML er normalt det at hvis man **fjerner** alle mærkerne – skal teksten kunne læses som det var meningen! Denne tekst er XML dokumentets **content** eller indhold eller **value**!

Hoved ideen i XML er at **adskille indhold** og **form** eller format! Det samme dokument kan have forskellige formater selv om indholdet – **content** – er det samme! Formen er blot en ydre form!

I **HTML** er form og indhold blandet sammen fordi mærkerne – som f. eks. <b> der betyder vis teksten i fed skrift – jo ikke siger noget om indholdet men kun om skrift typen! I HTML har man **INGEN** muligheder for at markere at 'dette er et årstal' f. eks.! Det kan man i XML.

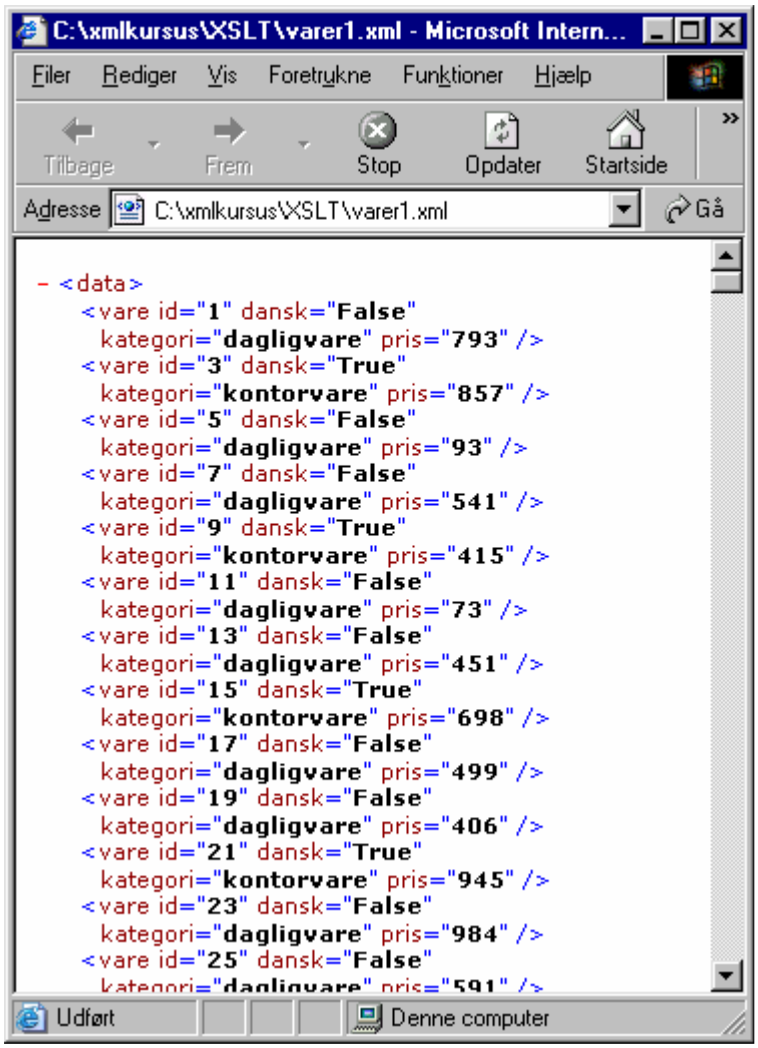
Et lille **eksempel** på forholdet indhold-form her kunne være disse to XML dokumenter:

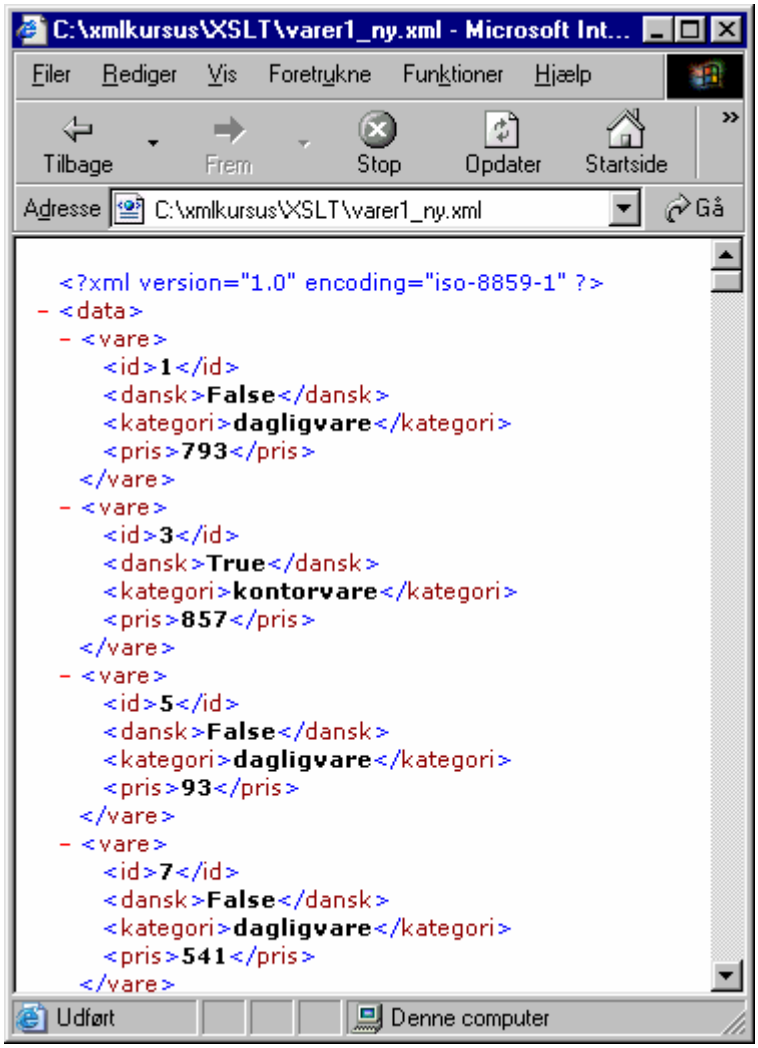
<person fornavn="Rosalina" efternavn="Hansen" />

```
<person>
  <fornavn>Rosalina</fornavn>
  <efternavn>Hansen</efternavn>
</person>
```

Disse to XML dokumenter har fuldstændigt det **samme** indhold eller informations indhold! Det første rummer informationen som værdien af **attributter** – det sidste indeholder under **elementer** som har et **tekst** indhold! Men **content** eller informations indholdet er præcist det samme! Den sidste form kaldes nogle gange for element **normal** formen. I XML kan man let oversætte fra det ene format til det andet.

Et andet eksempel på forskellige former men samme indhold er disse to XML dokumenter:


C:\xmlkursus\XSLT\varer1.xml - Microsoft Intern...
Filer Rediger Vis Foretrukne Funktioner Hjælp
Tilbage Frem Stop Opdater Startside
Adresse C:\xmlkursus\XSLT\varer1.xml
Gå
- <data>
<vare id="1" dansk="False" kategori="dagligvare" pris="793" />
<vare id="3" dansk="True" kategori="kontorvare" pris="857" />
<vare id="5" dansk="False" kategori="dagligvare" pris="93" />
<vare id="7" dansk="False" kategori="dagligvare" pris="541" />
<vare id="9" dansk="True" kategori="kontorvare" pris="415" />
<vare id="11" dansk="False" kategori="dagligvare" pris="73" />
<vare id="13" dansk="False" kategori="dagligvare" pris="451" />
<vare id="15" dansk="True" kategori="kontorvare" pris="698" />
<vare id="17" dansk="False" kategori="dagligvare" pris="499" />
<vare id="19" dansk="False" kategori="dagligvare" pris="406" />
<vare id="21" dansk="True" kategori="kontorvare" pris="945" />
<vare id="23" dansk="False" kategori="dagligvare" pris="984" />
<vare id="25" dansk="False" kategori="dagligvare" pris="591" />
Udført
Denne computer

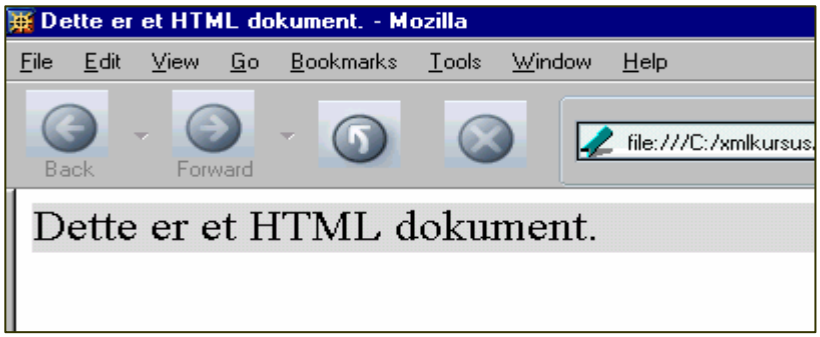


XML og HTML:

Vi vil se på to eksempler: Først et **HTML** dokument:

```
<Html>
        <Head>
                <Title>Dette er et HTML dokument.</Title>
        </Head>
        <Body>
                <div style="font-size:18pt;background-color:#dedede">
                Dette er et HTML dokument.
                </div>
        </Body>
</Html>
```

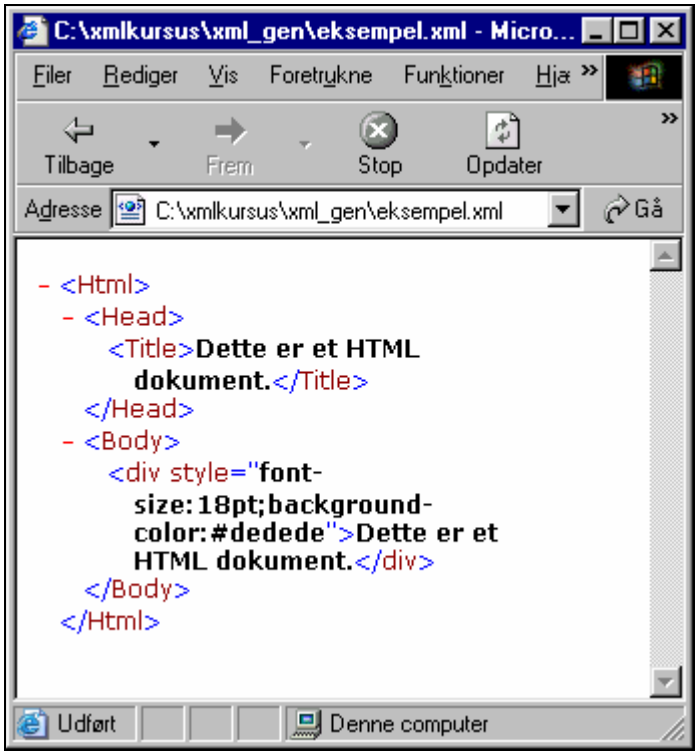
Når vi åbner filen 'eksempel.html' i Mozilla ses dette:



Mozilla **forstår** HTML og sætter en **titel** i titellinjen og en **div**-blok i selve dokumentet. Dette dokument er rent faktisk et gyldigt velformet XML dokument også – der blot tilfældigvis anvender de forud definerede tags eller mærker!

Det er vigtigt at huske at <HTML> og <html> er det **samme** i **HTML** som **ikke** skelner mellem store og små bogstaver! I **XML** er de to elementer **totalt** forskellige!! XML er **case sensitive** d.v.s. der er **altid** forskel på <fornavn> og <Fornavn>!

Hvis vi gemmer teksten som 'eksempel.xml' (altså blot ændrer fil **typen**!), fås denne side i en browser – her er det Internet Explorer:



Hvad er en parser?

En **parser** er et program som **læser** en XML tekst fil på en bestemt måde. Et XML dokument kan selvfølgeligt vises i et almindeligt tekst behandlings program – men Notesblok – f. eks. – kan ikke parse XML dokumentet!

Hvis vi bruger et simpelt eksempel kan vi forklare hvad **parseren** egentligt gør. Parseren læser tekst filen tegn for tegn – **byte** for byte.

<x>44</x>

Parseren læser dette tegn:	**Stadier eller events i læsningen:**
<	parseren opretter et element, et objekt – parseren forstår at node typen er element!
x	
>	elementet gemmes nu under navnet 'x', element slut! Parseren kontrollerer at navnet er gyldigt ifl. XML reglerne – må f. eks. ikke indeholde et mellemrum!
4	
4	
<	parseren kan **nu** se at tekstnoden er '44' – parseren opretter et nyt objekt af typen tekst node
/	parseren 'forstår' at dette er et slut mærke, et slut element
x	
>	parseren kontrollerer at navnet på det sidste objekt 'x' passer med det tidlige objekt navn! Objektet afsluttes og oprettes.

Som vi kan se affyrer parseren visse **events** når den læser et bestemt tegn! Den sætter bestemte ting i gang når den støder på bestemte bytes eller tegn! Dette er den afgørende **forskel** på en almndelig læse proces (som når vi åbner filen i Notesblok) og når vi **parser** dokumentet!

En parser kontrollerer at basale regler i XML overholdes. Den 'forstår' de forskellige node typer som element, attribut eller xml-erklæring! Den kan slå op i en XML regelsamling!

Hele **ideen** med XML dokumenter er at de **skal** kunne parses!

En XML parser er en 'tokenizer' som **opdeler** teksten i XML dokumentet i 'tokens' som har en bestemt betydning, har en bestemt **type**! På den måde kan parseren oprette **objekter** som f. eks. elementer!

Hvis parseren støder på et **anførselstegn** (' eller ") regnes det for en **attribut** – derfor må et anførselstegn **KUN** anvendes på en bestemt måde i XML!

Hvis parseren støder på tegnet '&' opfatter den det ALTID som starten på en **reference** til en entitet (mere herom senere)!

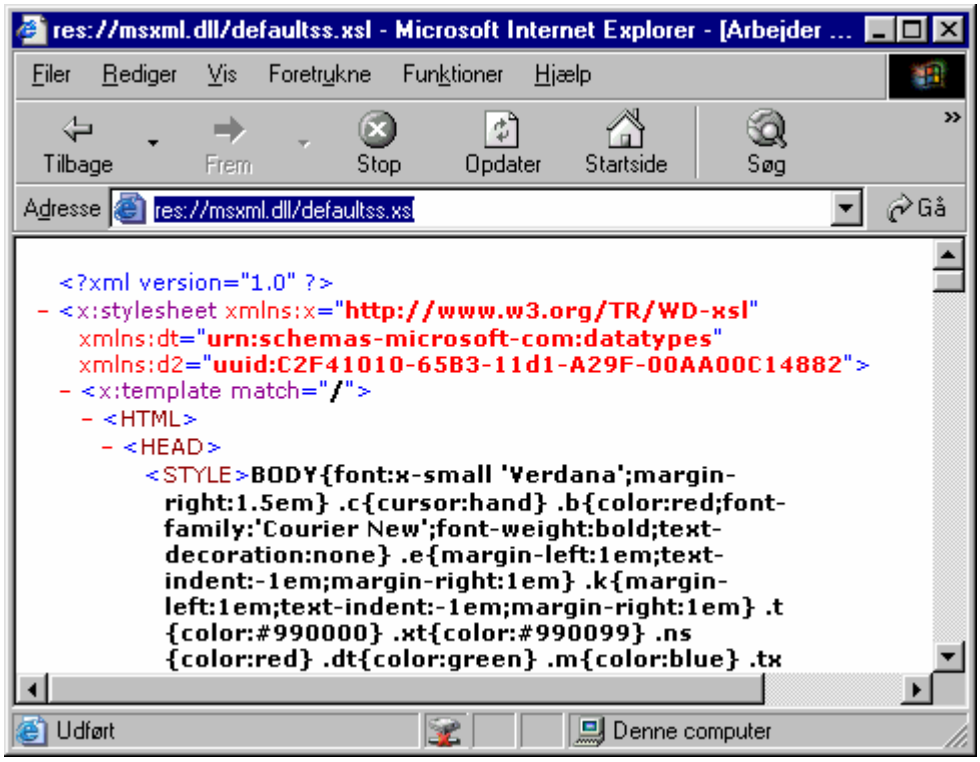
Velformet:

Browsere som Internet Explorer eller Mozilla eller Netscape Navigator har en indbygget XML **parser** eller **processor**. Denne processor kontrollerer om XML dokumentet er 'velformet'! Hvis det er velformet vises XML dokumentet i de fleste browsere med et default stylesheet . I Internet Explorer vises dokumentet med syntaks farver, + og – knapper, indrykninger og dynamisk HTML!

Man kan checke dette **default** stylesheet ved at skrive følgende i **adresse** linjen i Internet Explorer:

res://msxml.dll/defaultss.xsl

Man får så vist XSL typografi arket:



Alle XML dokumenter (modsat HTML dokumenter!) **SKAL** være vel formede – ellers **kan** de simpelt hen **ikke** læses eller parses!

Minimumskravet til **alle** XML dokumenter – det være sig skemaer eller stylesheets eller SOAP dokumenter - er:

1. Dokumentet skal være et gyldigt **træ** med en **rod** (et 'documentElement')
2. Hvis dokumentet har en xml erklæring skal den stå som det allerførste tegn i dokumentet!

3. Et dokument må kun af een DOCTYPE eller dokument type (jvf senere om DTD og dokument typer)
4. Alle elementer skal indledes med et **start** mærke (fx <body>) OG afsluttes med et **slut** mærke (fx </body>)
5. Alle attributters værdier skal sættes i enkelte eller dobbelte **anførselstegn**
6. Det samme element må **ikke** have 2 attributter med samme navn
7. **Kommentarer** må ikke sættes inden i en tag eller mærke
8. Elementer skal være korrekt **'nestede'** – ikke overlappe. Parseren opbygger en stack således at det sidste erklærede element også er det første der poppes fra stacken! Hvis vi skriver en slut tag skal den svare til det start mærke som ligger øverst i stakken!
9. Navne på elementer og attributter skal være et 'XML **Name'** eller rettere i praksis et **'NCName'** (non colon name) dvs overholde det følgende:
    a. navnet må kun starte med et bogstav (fra et hvilket som helst alfabet!) eller en understregning (ikke et tal eller en bindestreg f. eks.)
    b. navnet må **ikke** indeholde tegnene: ", ', $, ; eller % eller : (et kolon må kun anvendes til namespaces)
    c. navnet må **ikke** indeholde noget mellemrum af nogen art (ej heller tabulator eller linjeskift!)
    d. navnet må **ikke** starte med tegnene 'xml' eller 'XML' eller en lignende kombination!

10. Følgende tegn må **ALDRIG** anvendes fordi de kan forveksles med mærker eller XML funktioner:
    a. < (i stedet skrives en **entitet**: '<)
    b. > (i stedet: '>')
    c. ' (i stedet: ''')
    d. " (i stedet: '"')
    e. & (i stedet: '&')

Hvis vi springer tilbage til **eksempel**.xml kan vi se at dette XML dokument overholder disse regler. Man kan chekke om et dokument er velformet på mange forskellige måder – men at loade det i en browser som Mozilla eller Internet Explorer er det simpleste i første omgang.

Vi vil nu ændre dokumentet og skabe nogle 'fejl' – for at illustrere hvad der **IKKE** er en gyldig XML fil:

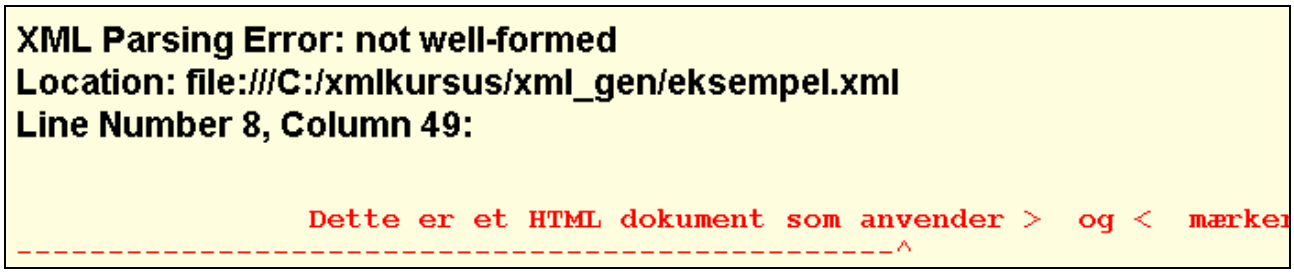
Hvis vi f. eks. retter slut mærket til </Htm> kommer denne fejlmelding:

Dette beviser at IE virkeligt parser dokumentet og kræver at dokumentet skal være velformet!

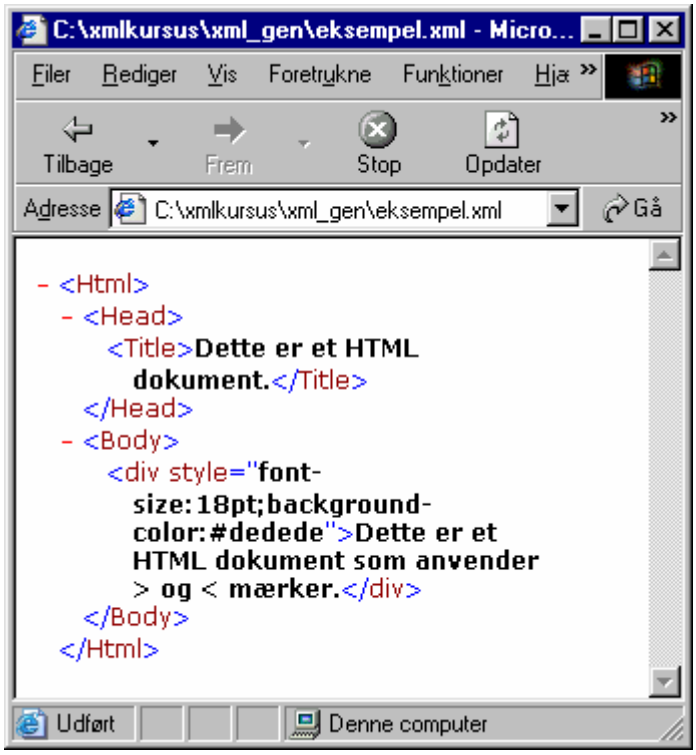
Hvis vi ændrer div teksten til:

```
<div style="font-size:18pt;background-color:#dedede">
Dette er et HTML dokument som anvender > og < mærker.
</div>
```

Fås denne fejl melding i Mozilla:



Vi kan så ’escape’ de to tegn ved at skrive entiteterne: > og < og resultatet bliver:



Nu kan vi helt frit anvende de to symboler, fordi er **'escapede'**. Vi kan selv **definere** 'entiteter' som vi skal se senere.

Hvis vi skriver denne XML fil:

<græsk_tekst>Dette er en græsk tekst!</græsk_tekst>

Får vi også en fejlmelding:

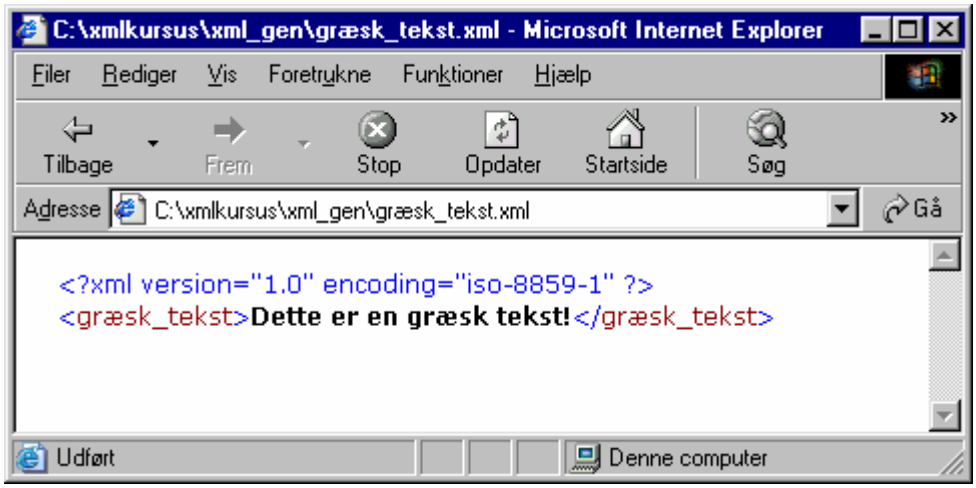
Læg mærke til at parseren (som egentligt en **SAX** parser) læser **forfra** - **tegn** for tegn - og **stopper** første gang den støder på et tegn som er et 'problem' for den! Her er det tegnet 'æ' som kommer efter '<gr'. Alle parsere (SAX parsere) læser på denne måde – i princippet kan hele XML dokumentet faktisk læses hvis en evt. fejl først findes i det sidste tegn!

Her **er** XML dokumentet faktisk meget velformet – men det rummer tegn som IE ikke forstår! Dette kan vi udbedre ved at starte med en XML **erklæring** – som det er sædvane (men ikke obligatorisk!) i XML dokumenter:

<?xml version='1.0' **encoding**='iso-8859-1'?>
<græsk_tekst>Dette er en græsk tekst!</græsk_tekst>

Vi skal senere se på **encoding** som angiver det alfabet eller det tegn sæt (character set) eller den kodning som anvendes i dokumentet! I dette tilfælde: iso-8859-1 som betegner de tegn som anvendes i engelske og europæiske skrift sprog ('Latin-1') – altså også danske bogstaver som æ, ø og å!

Vi får nu:



Vi kan også skrive en XML fil således – blot den er vel formet:

```
<?xml version='1.0' encoding='iso-8859-7'?>
<γρ∫σκ_τεκστ>
Dette er en γρ∫σκ tekst!
</γρ∫σκ_τεκστ>
```

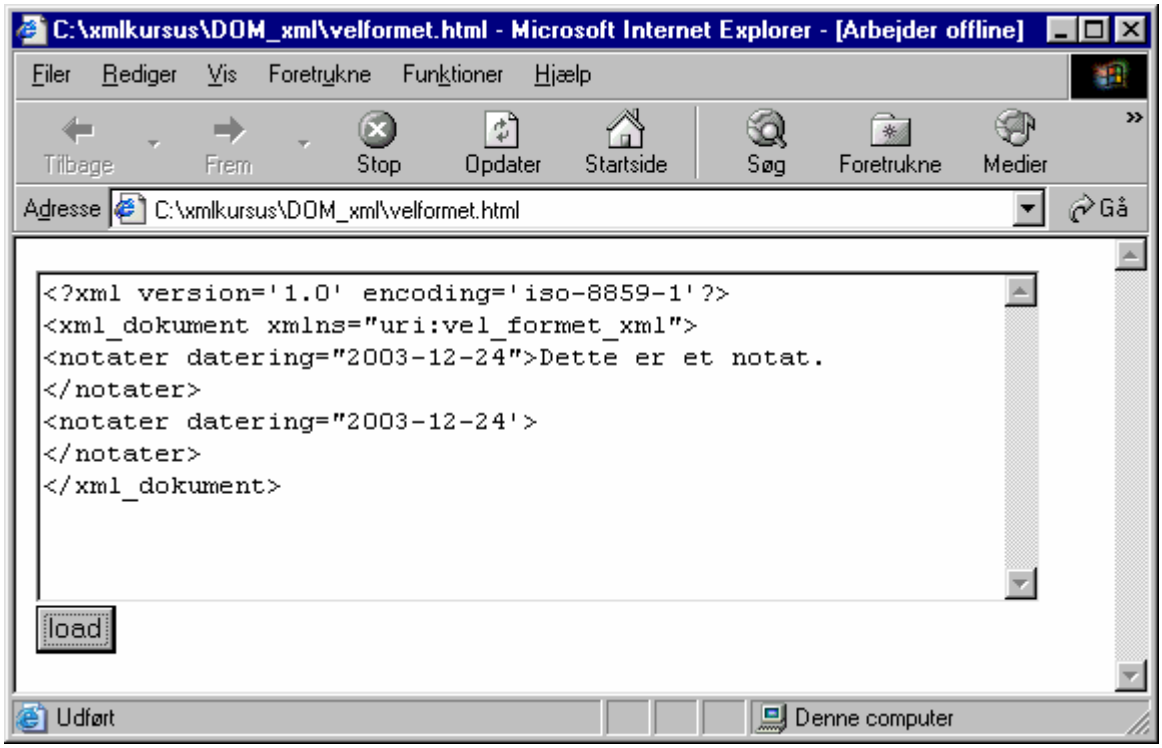
Selv om man ikke er god til græsk kan det ses at start og slutmærket svarer til hinanden! Dokumentet er altså **velformet**! (En browser kan normalt ikke klare denne sammenblanding af flere alfabeter – men dokumentet er stadig væk OK!).

XML kan - fordi det altid anvender **Unicode** tegnsættet (som kan rumme 2^{16} eller 65.536 forskellige tegn!) - håndtere **alle** tegn fra levende og døde sprog - også f. eks. japanske eller kinesiske eller indiske skrifttegn (ideogrammer)!

XML gemmes som standard eller default i **Unicode** med en **encoding=**'UTF-8' hvor tegn gemmes som 1, 2, 3, 4, 5 eller 6 bytes (dvs et tegn fylder fra 8 til 48 bits **afhængigt** af om det er amerikansk 'a' eller det kinesiske skrifttegn for 'stor' som er en mand der spreder ben og arme!).

Et script program der kan teste om et dokument er velformet:

Vi kan skrive et lille script program som ser ud som dette:



Vi kan nu **teste** ved at indtaste forskellige eksempler på XML i tekst boksen og klikke på **load**-knappen. I det viste tilfælde bliver resultatet:

Fejlen er at vi har anvendt to forskellige slags **anførsels** tegn i attributten!

Scriptet kan skrives sådan her – det sætter også et XML eksempel i boksen:

```
<body>
   <form name="form" id="form">
      <textarea rows="10" cols="60" name="xml" id="xml">
      <?xml version='1.0' encoding='iso-8859-1'?>
      <xml_dokument xmlns="uri:vel_formet_xml">
      <notater datering="2003-12-24">Dette er et notat.
      </notater>
      <notater datering="2003-12-24'>
      </notater>
      </xml_dokument>
      </textarea>
      <input type="button" value="load" onClick="load_doc()" >
   </form>
</body>
```

```
<script>

function load_doc(){
   var doc=new ActiveXObject("MSXML2.DOMDocument.4.0");
   doc.loadXML(document.all("xml").value);
   var err=doc.parseError;
   if(err.errorCode==0){
   alert("Dokumentet er velformet:\n\n"+doc.xml);

   }
   else alert ("Fejl: "+err.reason+"\nLinje: "+err.line+"\nKolonne: "+err.linepos);
}

</script>
```

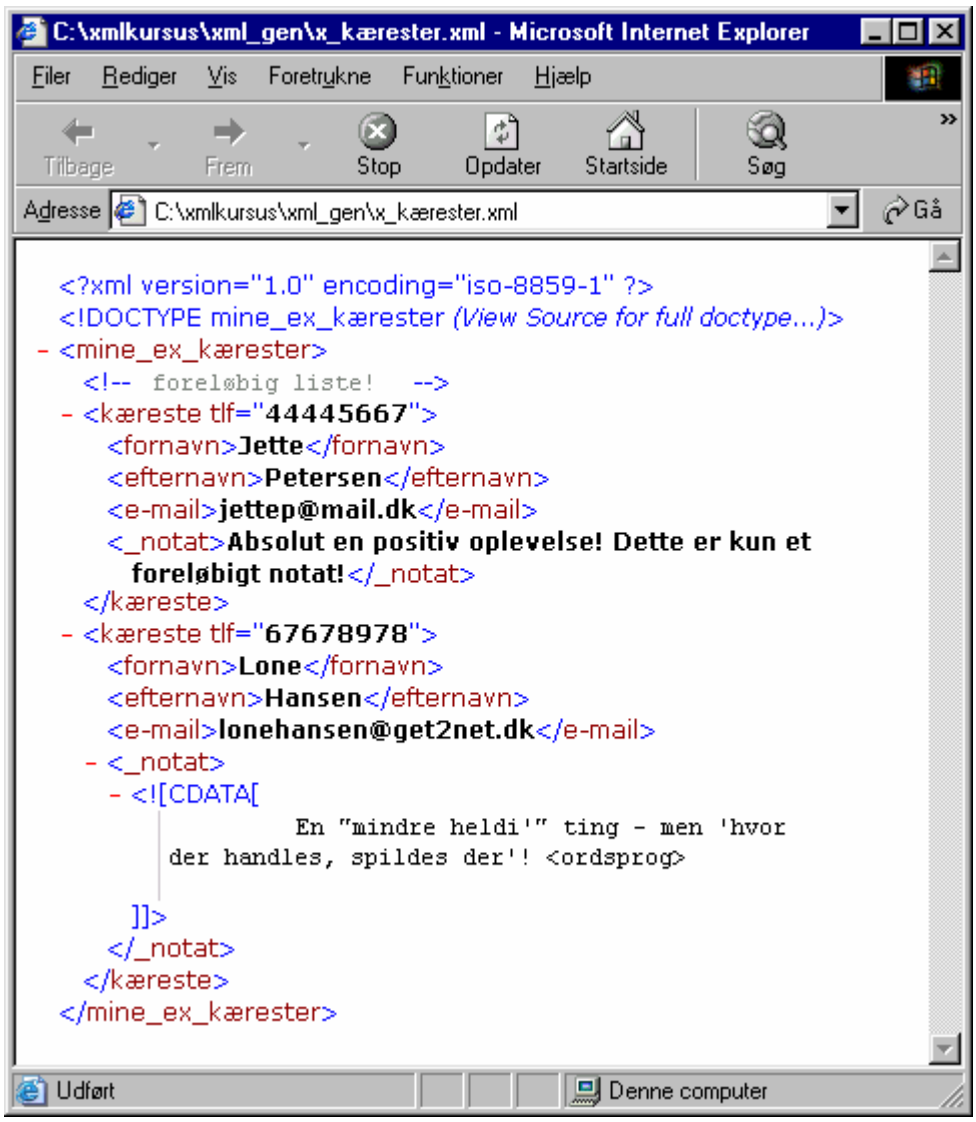
XML eksempel til illustration af generelle elementer:

Vi kan skrive følgende XML fil:

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

<!-- Dette er et såkaldt data centrisk XML dokument -->

<!DOCTYPE mine_ex_kærester [
<!ENTITY notat ”Dette er kun et foreløbigt notat!”>
]>

<mine_ex_kærester>
<!-- foreløbig liste! -->

<kæreste tlf="44445667">
                    <fornavn>
                    Jette

                    </fornavn>
                    <efternavn>
                    Petersen

                    </efternavn>
                    <e-mail>
jettep@mail.dk

                    </e-mail>
                    <_notat>
Absolut en positiv oplevelse! &notat;
                    </_notat>
</kæreste>
<kæreste tlf="67678978">
                    <fornavn>
                    Lone
                    </fornavn>
                    <efternavn>
                    Hansen
                    </efternavn>
```

```
                <e-mail>lonehansen@get2net.dk</e-mail>
                <_notat>
                <![CDATA[
                En ”mindre heldi’” ting – men ’hvor der handles, spildes der’! <ordsprog>
                ]]>
                </_notat>
</kæreste>

</mine_ex_kærester>
```

Når dette dokument vises i Internet Explorer ses:



Kommentarer:

Kommentaren (skrives som <!-- -->) FØR dokumentets rod vises ikke, men den anden kommentar vises OK.

Elementerne FØR roden kaldes oftest for **prologen**. Her består prologen af en XML erklæring med såkaldte pseudo attributter og en **DOCTYPE**, som erklærer at dette dokument tilhører dokument

typen 'mine_ex_kærester'! Man kan sige at vores XML applikation hedder 'mine_ex_kærester' Markup Language! Lige som HTML tilhører DOCTYPE html!

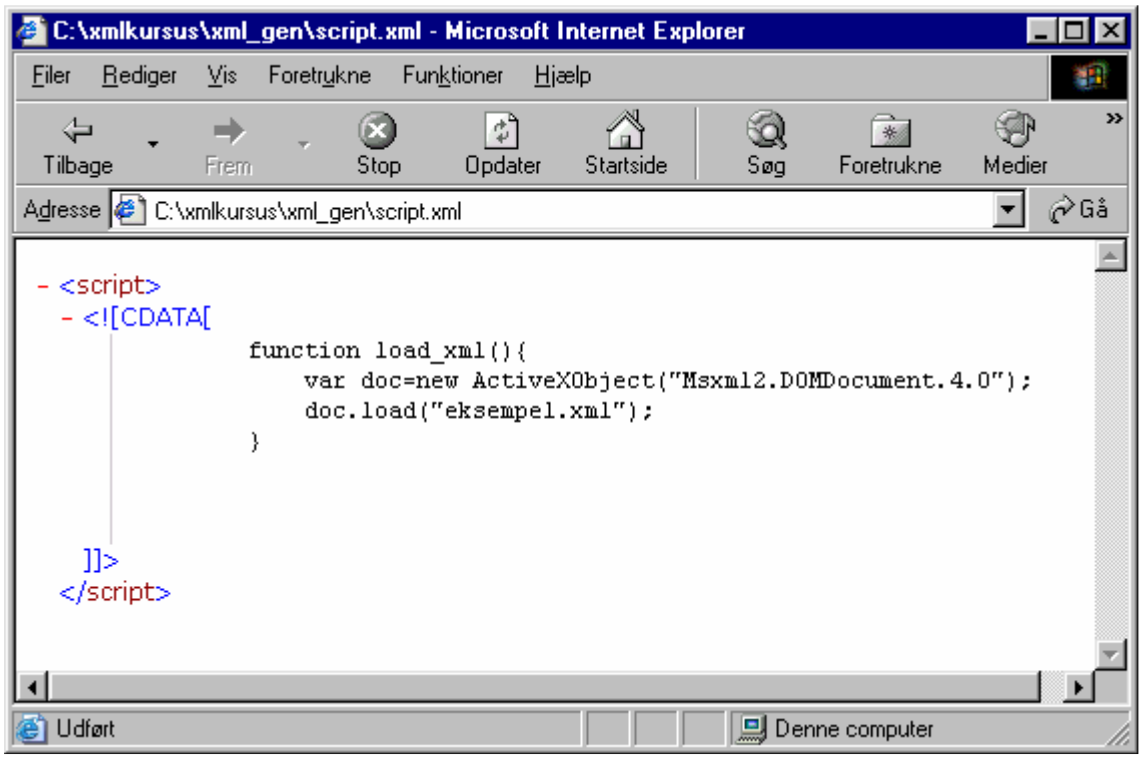
Vores DOCTYPE vises ikke fuldt ud i browseren men den erklærer blot en **entitet** ved at sætte den lig med en tekst! Når vi siden anvender &notat; indsætter vi entiteten! Dette svarer f. eks. til en #define i programmerings sproget C. Entiteter kan også bruges til at indsætte et kinesisk skrift tegn eller indholdet af en fil på 25 MegaByte!

Vi skal vende tilbage til entiteters anvendelse i afsnittet om **DTD** skemaer!

I det sidste _notat har vi omgivet tekstnoden med en såkaldt **CDATA** sektion. CDATA betyder Character Data og oprettelsen af en CDATA sektion betyder at vi frit kan skrive hvad som helst også bruge tegn som < eller ".

Det som står inden i en CDATA sektion bliver **IKKE** parset (ikke checket for velformethed!) af XML processoren! Det modsatte af CDATA er - i XML - **#PCDATA** som bruges om data eller tekst, som skal **parses** af XML processoren!

CDATA bliver blot afleveret som en blok af uparsede, rå tegn! CDATA sektioner kan også anvendes f. eks. til programmerings kode eller indlagte eksempler på HTML eller XML.



Som det ses gemmes og vises data nøjagtigt som de skrives med indrykninger og tomme linjer! I HTML ville dette svare til at vi brugte elementet <pre>!

Mellemrum, tabulator og linjeskift:

Hvis vi laver et mellemrum eller et linjeskift **mellem** to elementer bliver det normalt ignoreret (fjernet) af en XML processor! Dette svarer helt til hvad en **HTML** parser gør! Derfor fjernes alle linjeskift f. eks. i en HTML fil som vises i en browser!

Selve **tekstindholdet** (tekst noden) bliver derimod gemt som det er af parseren! Hvis vi laver et mellemrum i en attribut er mellemrummet en **del** af attribut værdien. Også hvis vi laver 27 mellemrum ved siden af hinanden!

Det samme gælder **elementer**. Det som står mellem start og slut mærket er en sekvens af tegn koder (der egentligt gemmes som tal eller bits) – **også** hvis de består af mellemrum! F. eks. kan vi skrive et element således:

```
<kæreste tlf="67678978">
                <fornavn>Lone</fornavn>
                <efternavn>Hansen</efternavn>
                <e-mail>
                   lonehansen@get2net.dk
                </e-mail>
                <_notat>
                <![CDATA[
                En "mindre heldi'" ting - men 'hvor der handles, spildes der'! <ordsprog>
                ]]>
                </_notat>
</kæreste>
```

Her har vi rent faktisk – måske uden at vi tænker over det! – indført **linjeskift** i elementerne <e-mail> og <_notat>! Vi har **indrykket** en række data efter <kæreste>! Vi har også indført et **linjeskift** efter <kæreste<!

Den producerede tekst fil ser derfor nogenlunde sådan ud:

Vi kan f. eks. se at der før CDATA sektionen reelt er to linjeskift og at CDATA sektionen faktisk er længere ude til venstre end e-mail adressen som er indrykket med 4 mellemrum i forhold til fornavn og efternavn!

XML processoren **bevarer** altså mellemrum o.s.v. **inden** i elementerne. Dette er vigtigt når vi skal transformere XML dokumenter til f. eks. tekst dokumenter! I XSLT er der funktioner som kan **ændre** denne default fremgangsmåde nemlig:

- preserve-space elements="*"
- strip-space elements="*"

Den første bevarer space i alle elementer, den anden fjerner space i alle elementer! Vi skal senere se på disse funktioner.

Producenter og forbrugere af XML dokumenter:

Man kan skelne mellem XML dokumenter der er **produceret** af **mennesker** 'i hånden' og dokumenter som er produceret af et **software** program! Et program kan let producere meget store og meget indviklede XML dokumenter på kort tid!

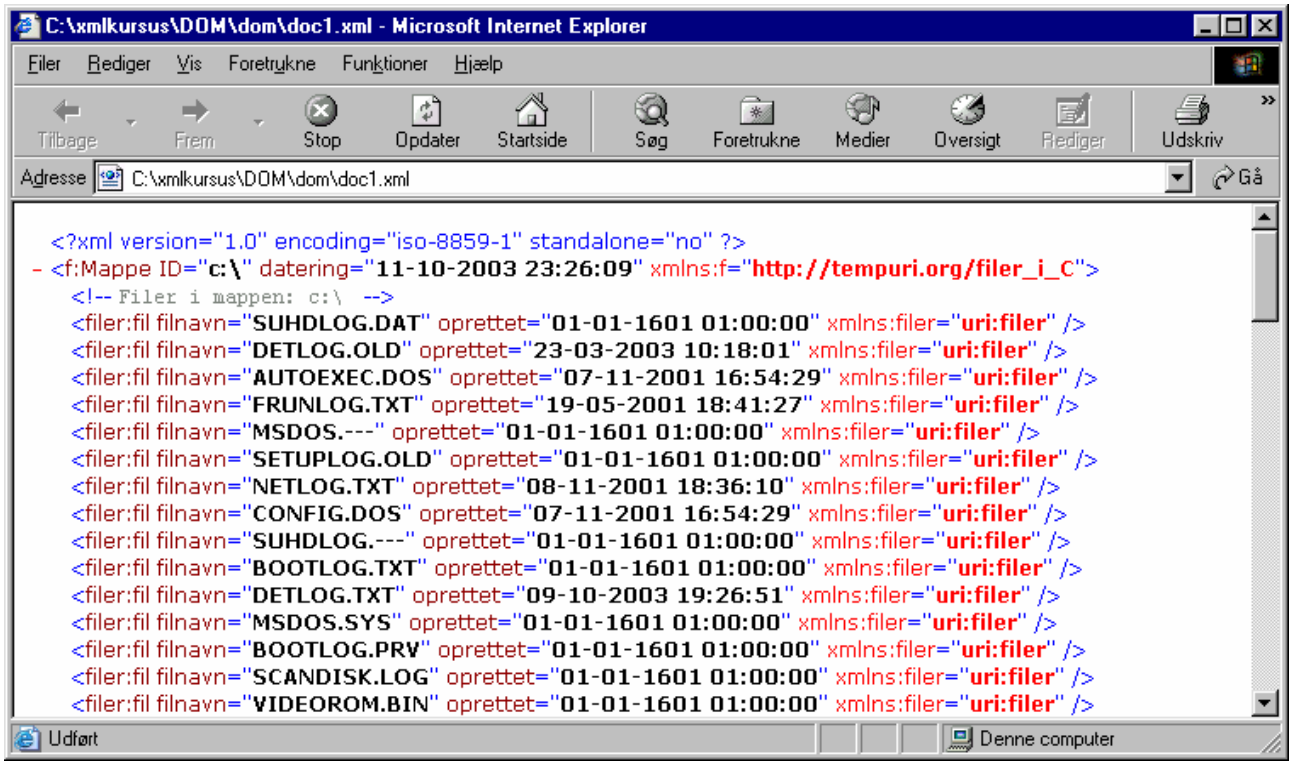
På samme måde er der stor forskel på om **modtageren** eller **forbrugeren** af et dokument er en maskine (et program) eller et levende menneske! XML dokumenter der skal kunne læses af almindelige mennesker skal være mere overskuelige for at være til nogen hjælp. Men stadig væk er den enorme fordel ved XML at disse dokumenter altid **kan** læses af almindelige dødelige i modsætning til andre formater!

Mange XML dokumenter bliver kun brugt som **formidling** mellem software programmer! Disse dokumenter har altså en ganske bestemt **funktion** og er ikke primært beregnet til at give informationer til andre!

XML dokumenter bliver f. eks. produceret af et klient program og sendt til en XML server. Et sådant XML dokument bliver aldrig nogen sinde gemt som en fast fil. Det eksisterer kun virtuelt – i en hukommelse – i RAM. Det har måske en levetid på et halvt sekund!

Gnumeric regneark til Linux systemet producerer hele tiden mængder af XML data for at kunne holde styr på cellerne i regnearket!

Et eksempel på et dokument som er skrevet af et program er følgende fil liste over filer i C drevet! Det er let at producere et sådant – meget langt – dokument med f. eks. C# som det er gjort her! Hvis et almindeligt menneske skulle have skrevet det var det nok blevet skrevet anderledes. Stadig væk viser et sådant dokument styrken ved XML – at det kan gemme detaljerede data meget struktureret:



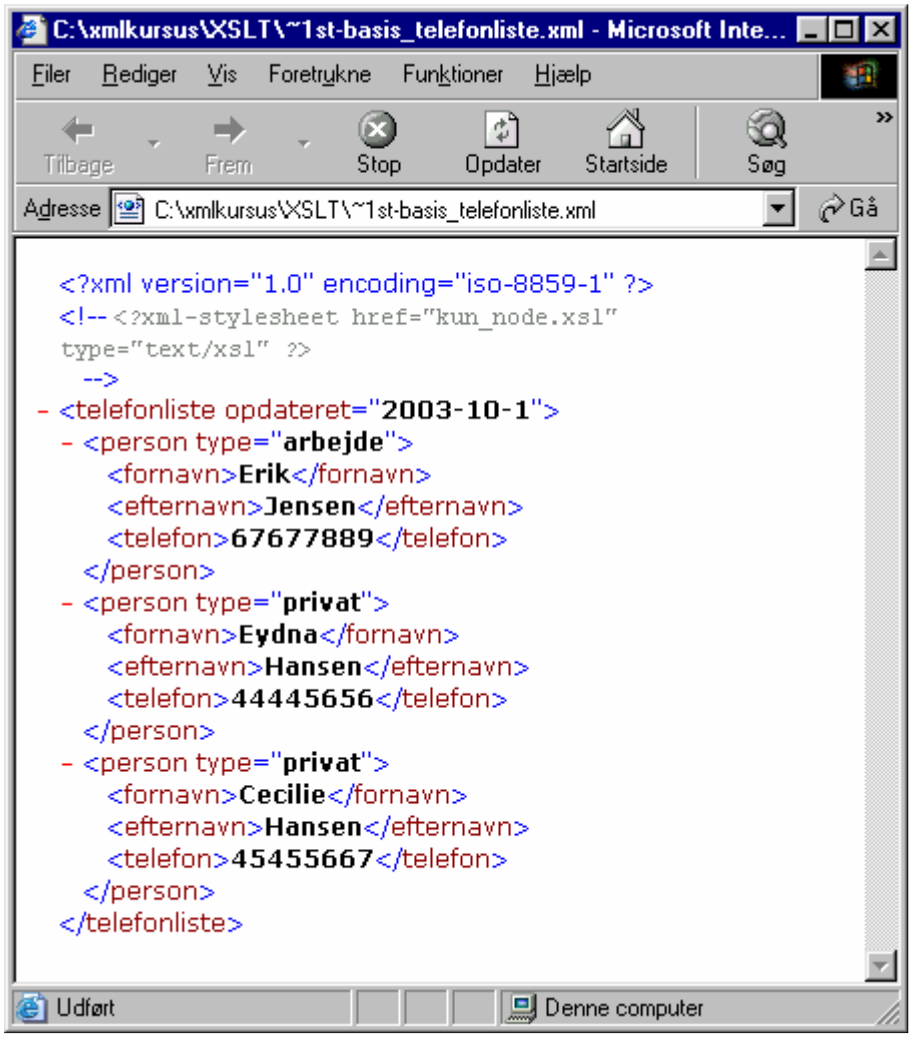
To typer af XML dokumenter:

Der findes især to typer af XML filer nemlig **datacentriske** og **narrative** eller fortællende dokumenter.

Data centriske XML dokumenter:

Eksemplet ovenfor er et datacentrisk dokument lige som f. eks. en lagerliste over bøger hos en boghandel. Disse dokumenter minder om en **database** tabel. Der oprettes måske flere tusind poster med enslydende felter.

Et andet eksempel på et data centrisk XML dokument kunne være denne telefonliste, der kunne omfatte flere hundrede ensartede poster:



XML dokumentet svarer nogenlunde til denne **tabel** - her vises kun en post som eksempel:

type	**fornavn**	**efternavn**	**telefon**
privat	Eydna	Hansen	44445656

Vi kan også sammenligne med de såkaldte **komma separerede** filer som i gamle dage – og faktisk stadig – er brugt til at gemme data i:

Cecilie,Hansen,45455667
Rosaline,Jensen,34344556
Ole,Nyborg,23236778

osv.

Det er klart at disse komma separerede filer har store **begrænsninger**! Links eller nøgler kan ikke skrives. Der er ingen kontrol af at data skrives 'korrekt'. Hvis en database bruger sådanne filer – og det kan alle database systemer – kan der let opstå **fejl**. Her er et dokument format som XML klart overlegent!

Narrative eller fortællende XML dokumenter:

Narrative dokumenter – HTML tilhører stort set denne type – er ikke så **strukturerede**. Narrative XML dokumenter er rapporter, artikler, bøger, alle slags almindelige skriftlige oplæg! SGML blev i sin tid skabt for sådanne 'almindelige' dokumenter. Her kan forskellige elementer blandes frit – **også** selv dokumentet har et skema som det skal valideres imod.

Et eksempel på et **narrativt** eller fortællende dokument kunne være dette:

XML dokumentet er her **stylet** med et typografi ark eller stylesheet sådan at person navne vises med rødt, årstal med blåt og sted navne med grønt o.s.v.

Den afgørende fordel ved at anvende XML i narrative dokumenter er, at man kan **søge** i teksten på en effektiv måde fordi data er **'typede'** – informationer om steder ligger i et element <sted> (helt modsat end i HTML)!

På den måde kan man **finde** alle personer eller alle årstal! XML er et langt bedre format for **søgemaskiner** (robotter) på Internettet end HTML. En HTML tekst med forskellige oplysninger skal 'screen scrapes' dvs. man skal ind og søge i selve teksten f. eks. med en tekst behandling!

Selve XML filen er skrevet således:

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

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

<!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>
```

Vi skal siden vende tilbage til **narrative** dokumenter og til hvordan man kan skrive XML skemaer til dem. Ligesom i HTML er der grundlæggende det 'problem' at de forskellige elementer – her f. eks. <note> eller <person> - kan forekomme i helt forskellig orden og i forskellige kontekster!

XML dokumentets struktur:

Ideen i XML er at det samme **content** eller **indhold** kan kodes på **forskellig** måde – mere eller mindre hensigtsmæssigt! Formen er noget andet en indholdet! Hvis kodningen foretages af software kan den gøres meget kompliceret – hvis den skal gennemføres af rigtige mennesker sætter det selvfølgeligt nogle begrænsninger!

Vi skal se på et lille eksempel på denne problematik her. Den samme information kan kodes på to forskellige måder – her er det eksempel fra SVG Scalable Vector Graphics, et XML sprog:

```
<polygon points="10, 78 33,66 77,88" />

<polygon>
        <point x="10" y="78" />
        <point x="33" y="66" />
        <point x="77" y="88" />
</polygon>
```

Tekst eller informations indholdet er præcist det samme – men ifølge de flestes mening er den sidste løsning bedre fordi den klart viser hvad de forskellige tal er for en type! I det første eksempel er der principielt kun tale om en bunke ustrukturerede tal!

At hente XML fra en database tabel:

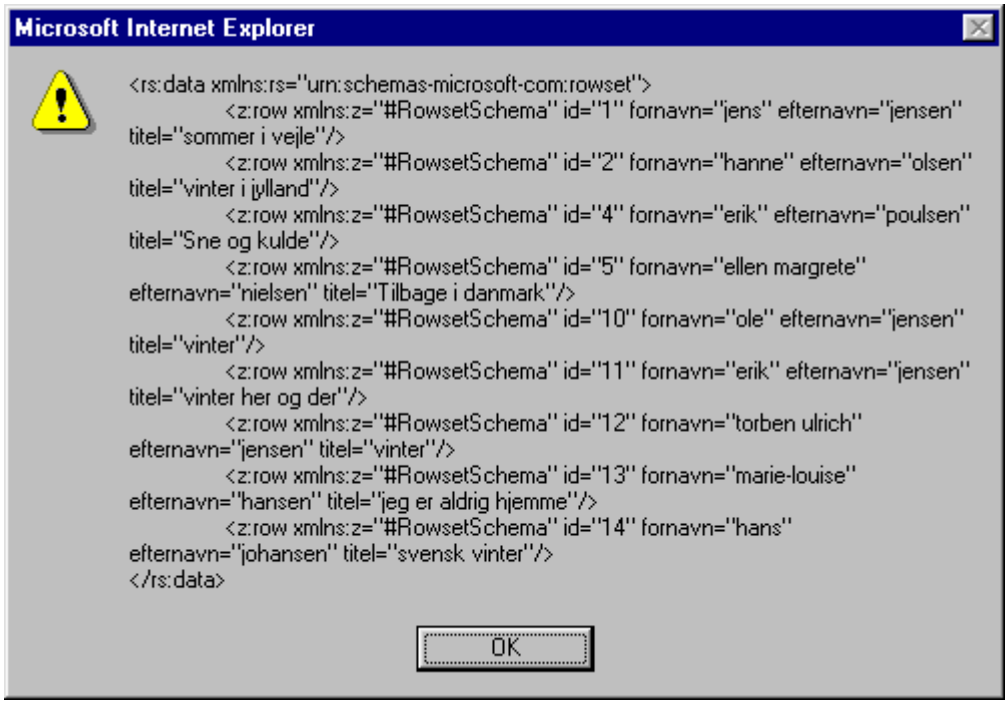
Vi skal også se hvordan man på en ret enkel måde kan hente XML dokumenter fra **database** tabeller. Hvis man har databaser kan man på den måde let få adgang til XML dokumenter som man så kan arbejde med.

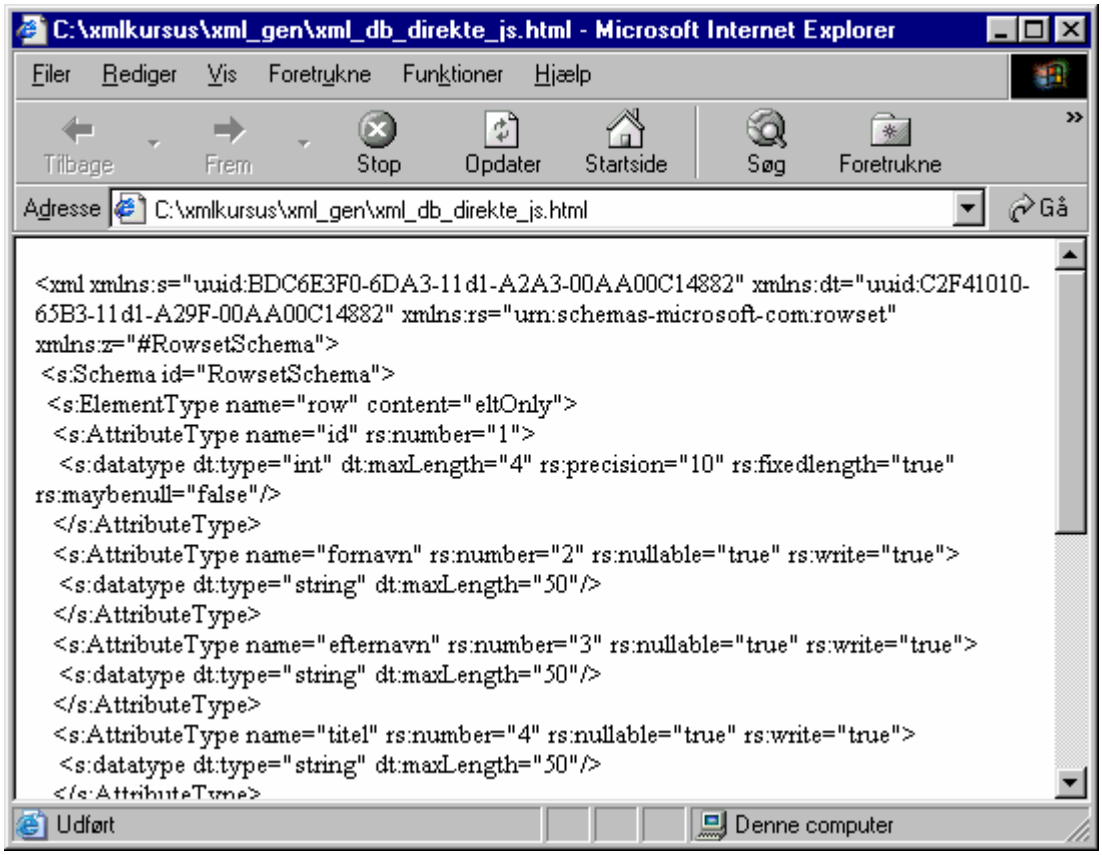
Vores metode bygger på et **<script>** i en HTML fil. En sådant script – her skrevet i Javascript – kan kalde objekter og funktioner og etablere **forbindelse** med en database. Nedenstående eksempel forudsætter at man har installeret en database og **registreret** den som en system DNS. Her er brugt en MS Access database 'boghandel' og tabellen 'bog':

```
<body>
<div id="tekst"></div>
<script>

function db() {
   doc=new ActiveXObject("Msxml2.DOMDocument.4.0");
   connection = new ActiveXObject("ADODB.Connection");
   connection.open("boghandel","","");
   recordset = new ActiveXObject("ADODB.Recordset");
   recordset.open ("select * from bog", connection);
   recordset.Save (doc, 1);
   alert(doc.firstChild.lastChild.xml);
   tekst.innerText=doc.xml;
   recordset.close();
   connection.close();
}

db();
</script>
</body>
```

Funktionen db() kaldes og etablerer en forbindelse til databasen. En database kan kun bruges hvis der 'kaldes op' til den med en 'connection'!

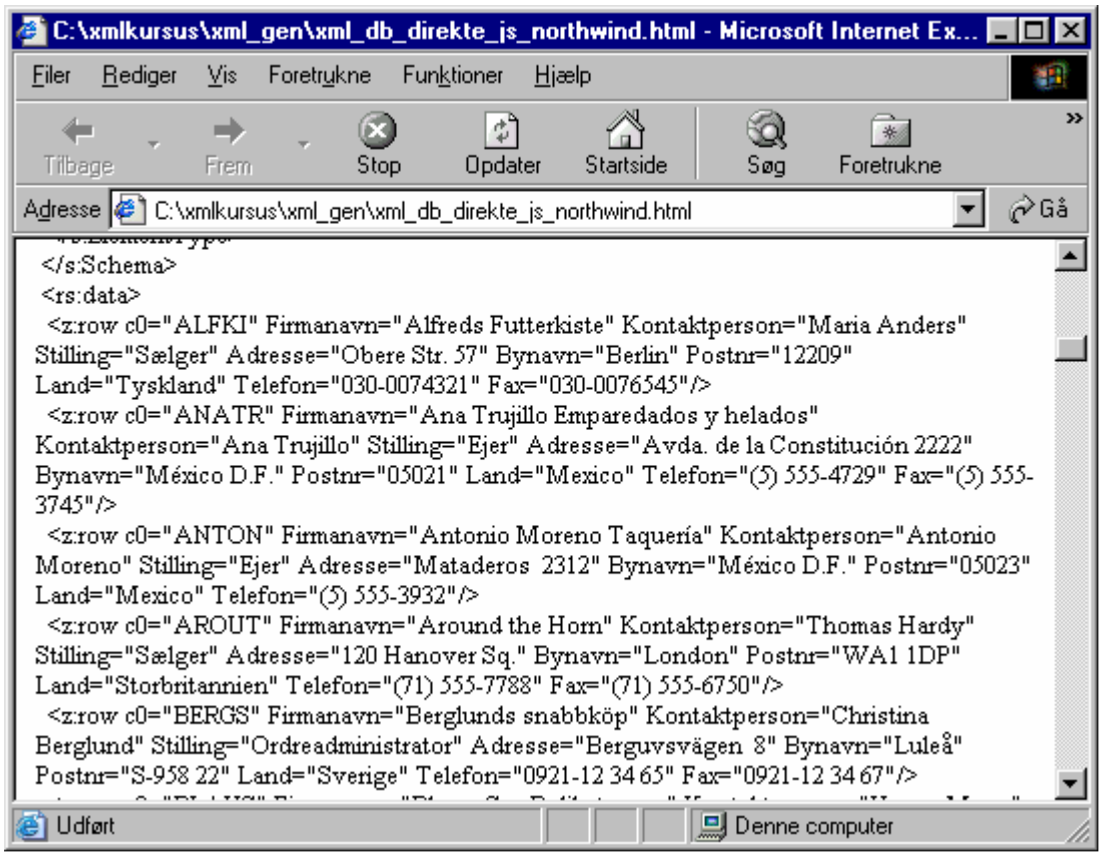
Vi bruger en **SQL** sætning som betyder at vi vil have alle **poster** i tabellen bog. Derefter udfører vi det trick at vi gemmer det hentede **recordset** i et XML DOM dokument! På denne måde får vi **gratis** skabt et XML dokument med data fra tabellen! Parameteren '1' i save metoden betyder 'adPersistXML' altså at vi ønsker at gemme i XML formatet!





Vi kan her se **XDR** skemaet til XML dokumentet. Vi kan også se at en post eller række er kodet som en **row** med **attributter**! Dette sker **automatisk**!

Vi kunne på samme måde koble os op på MS databasen **Northwind** som oftest findes i Windows. Her får vi gratis XML data – som det ses her fra tabellen Kunder:



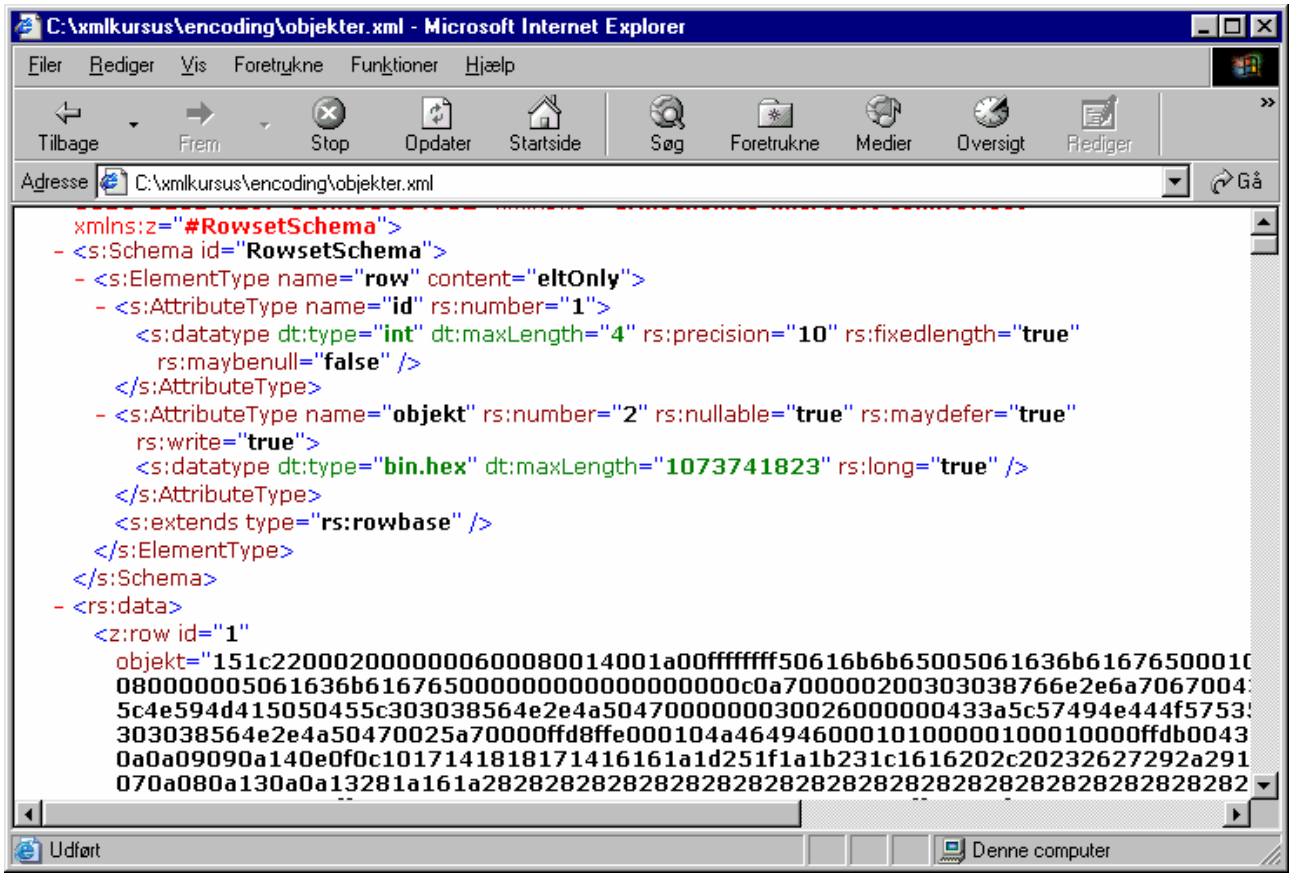
Binære data i XML dokumenter:

Vi kan også med den viste metode få kodet binære data ind i et XML dokument. Eksemplet her bygger på at vi har oprettet en Access database med en tabel med to felter: id og objekt. I feltet objekt gemmer vi så binære objekter – her JPG billeder! Vi kunne også have gemt Word dokumenter!

Vi kan så koble os op på denne database – som også skal registreres på maskinen fordi vi bruger ODBC – få vist og kodet vores binære data som binHex kode. Vi skal siden vende tilbage til eksemplet! Binære filer som JPG billeder kommer til at fylde rigtigt meget hvis de skal sendes inden i et XML dokument! I eksemplet nedenfor er dokumentet gemt som en selvstændig XML fil for bedre at kunne overskue resultatet!:

```
<body>
<div id="tekst"></div>
<script>

function db() {
   doc=new ActiveXObject("Msxml2.DOMDocument.4.0");
   connection = new ActiveXObject("ADODB.Connection");
   connection.open("objekter","","");
   recordset = new ActiveXObject("ADODB.Recordset");
```

```
    recordset.open ("select * from objekter", connection);
    recordset.Save (doc, 1);
    tekst.innerText=doc.xml;
    recordset.close();
    connection.close();
}

db();
</script>
</body>
```



Vi kan se at systemet selv finder ud af at data **typen** for kolonnen objekt er 'bin.hex' – fordi datatypen for feltet i database tabellens kolonne er **binært**! Vi skal i forbindelse med XSD skemaer igen se på datatypen bin.hex. I attributten **maxLength** kan vi se hvor mange bytes disse JPG filer kommer til at fylde! En bin hex kodning vil sige at hver byte i JPG filen oversættes til to bytes i XML dokumentet! De binære **hexadecimale** tal går fra 0 til A, B, C, D, E og F (= 15). I dette tilfælde fylder XML dokumentet 414.000 bytes! Et almindeligt JPG billede 30 eller 50.000 bytes i sig selv!

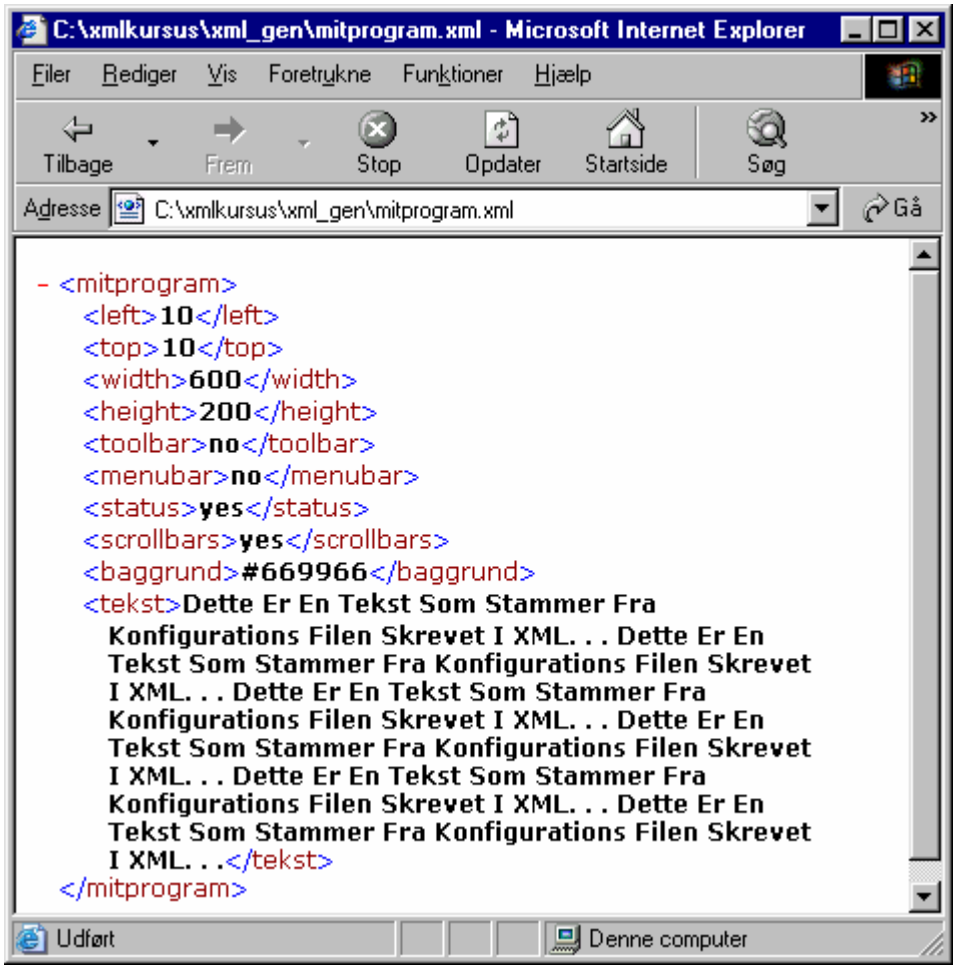
Konfigurations filer – et eksempel på et XML dokument:

Det er meget almindeligt at skrive **konfigurations** filer som definerer en række **parametre** for hvordan et program skal opstarte. I .NET skrives disse filer altid i XML. Hvis programmet hedder

’mitprogram.exe’ hedder den tilhørende konfigurationsfil ’mitprogram.config’! Vi skal dog her se på et eksempel der ikke har noget at gøre med .NET:

Vi kan illustrere brugen af XML til konfigurations filer på denne måde:

Vores mitprogram.xml ser således ud og definerer hvordan et vindue skal se ud:



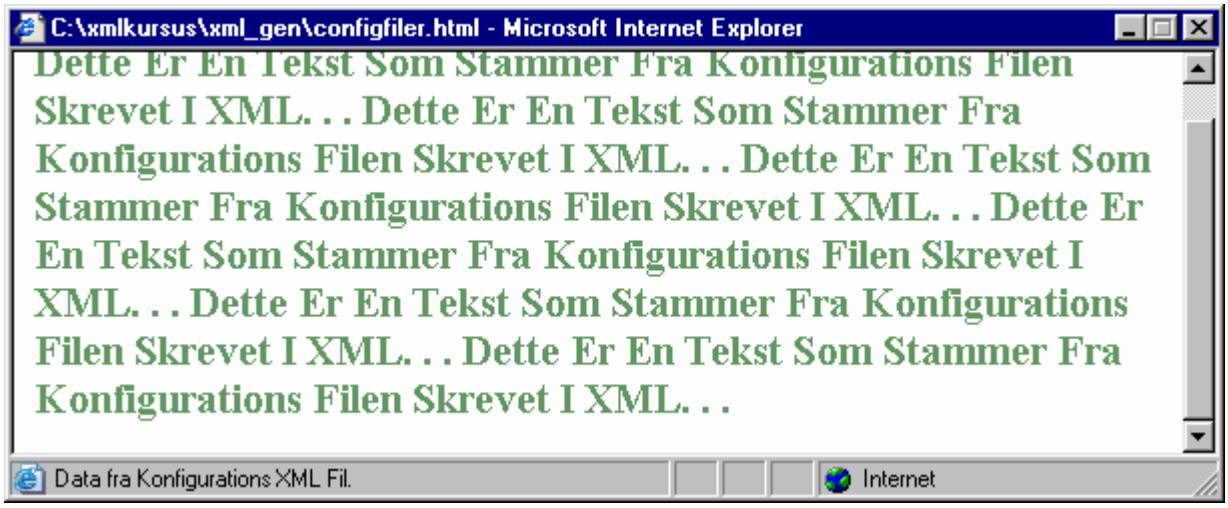
Vi kan nu udnytte den konfig fil i dette script som åbner et vindue:

```
<script>
var doc=new ActiveXObject("Msxml2.DOMDocument.4.0");
doc.load("c:/xmlkursus/xml_gen/mitprogram.xml");

var t=doc.firstChild.lastChild.text;
var l=doc.firstChild.childNodes(0).text;
var tp=doc.firstChild.childNodes(1).text;

var w=doc.firstChild.childNodes(2).text;
var h=doc.firstChild.childNodes(3).text;
//var scr=doc.firstChild.childNodes(7).text;
var b=doc.firstChild.childNodes(8).text;
```

```
var
w=window.open("","","left="+l+",top="+tp+",toolbar=no,menubar=no,scrollbars=yes,status=yes,width="+w+",height=
"+h+"");
w.document.write("<h2><font color="+b+">"+t+"</h2>");
w.status="Data fra Konfigurations XML Fil.";

</script>
```

Dette script henter så alle værdierne fra XML filen og indsætter dem i de parametre som findes til metoden **window.open**()! Læg også mærke til at vi lægger tekst ind i statuslinjen!

Vi skal siden vende tilbage til denne form for scripting! På http://msdn.microsoft.com findes en del materiale om scripting!



Typer af noder i et XML dokument:

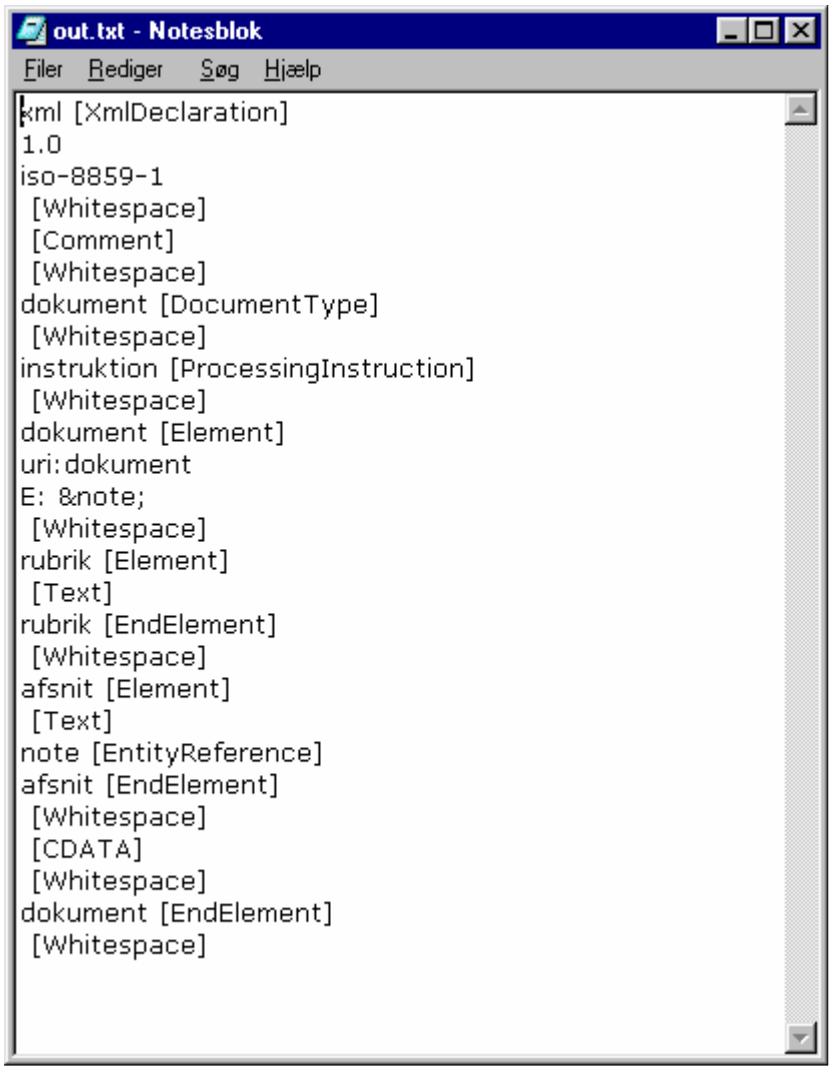
Alle ting eller objekter i et XML dokument er noder! Alt er en node – et element, en attribut en tekst!

Som et eksempel kan vi skrive dette dokument som stort set illustrerer alle XMLs node typer:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<!--
Dette er et demo dokument til at vise node typer i XML.
-->
<!DOCTYPE dokument [
<!ENTITY note "en note.">
]>
<?instruktion dette er en processing instruction?>
<dokument xmlns="uri:dokument">
<rubrik>
Dette er en overskrift.
</rubrik>
<afsnit>Dette er et afsnit. Dette er &note;</afsnit>
<![CDATA[
        public static void Main(){
```

```
        if (x<y){}
        //
        }
]]>
</dokument>
```

Hvis dette dokument læses af en SAX parser (.NET) kan vi få udskrevet alle de **objekter** eller **noder** som parseren støder på punkt for punkt:



Vi kan se at et **namespace** er en selvstændig **node**! En **reference** til en defineret entitet er også en selvstændig node – her f. eks. anvendt inden i en attribut! Mellemrum, linjeskift og tabulator (**whitespace**) er en node! Vi kan også se at parseren skelner mellem et **start** mærke og et slutmærke. Et start mærke er en **selvstændig** node. Fordi **alt** hvad der er anbragt inden i '<' og '>' er en node – der så kan indeholde andre sub noder! **Teksten** til noden er child af elementet eller attributten og en selvstændig node!

XML parsere:

Mange forskellige programmer har i dag en **indbygget** parser. Det vil sige at hvis man f. eks. har installeret en af de almindelige **browsere** som Internet Explorer, Netscape eller Mozilla har man også **samtidigt** installeret en XML parser! På samme måde bruger f. eks. Microsoft Office pakken også en XML parser. Hvis man har downloadet XML editorer som **XMLSpy** eller **XMLWriter** – som kan downloades i en prøve version – har man **også** fået sig en XML parser. Dette gælder også hvis man har downloadet C# editoren (og XML editoren) SharpDevelop.

Alle disse parsere kan så være mere eller mindre tilgængelige for brugeren!

Der eksisterer i dag en mængde forskellige XML parsere - programmer eller klasser som kan læse og 'forstå' XML og som kender reglerne for et gyldigt XML dokument.

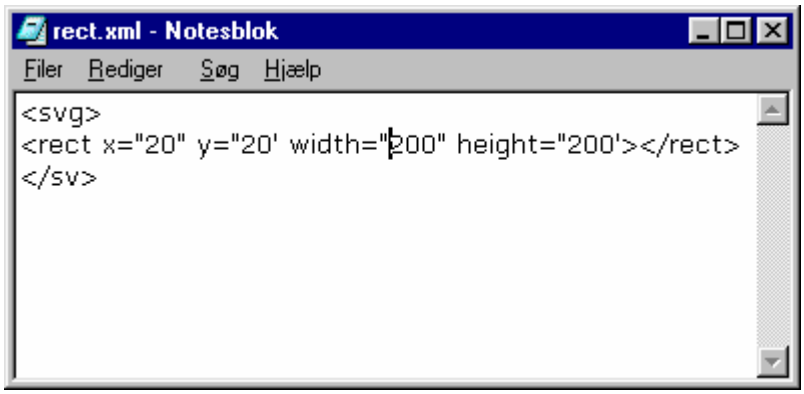
Nogle parsere kan både køre på Windows, Unix og Linux f.eks. Nogle kan kun køre på et bestemt operativ system. Men p.g.a. W3C standarderne er alle parsere nogenlunde ens i strukturen!

Her skal nævnes nogle af de muligheder som kan downloades fra Internettet:

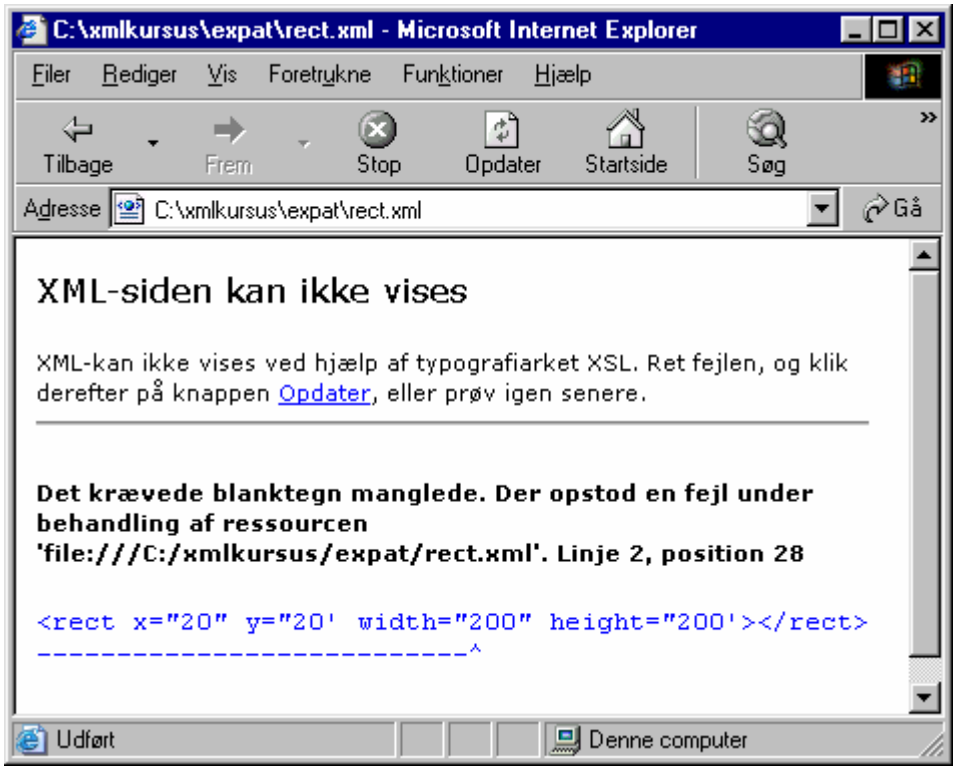
MSXML:

er Microsofts XML parser. De fleste eksempler i dette kursus tager udgangspunkt i MSXML som findes i mange udgaver. Internet Explorer 6.0 har indbygget denne parser – dog kun i den næst sidste version nemlig 3.0!

Et XML dokument med adskillige fejl:



I Internet Explorer vises dette hvis vi søger at loade filen:



SAXON:

SAXON parseren som er skrevet af Michael H. Kay kan downloades fra adressen http://saxon.sourceforge.net. Først og fremmest downloades **saxon**.jar som er et bibliotek af **Java** klasser til XML.

SAXON er primært beregnet til at style eller transformere XML dokumenter.

For at kunne bruge disse klasser skal man altså have installeret en udgave af **Java** (Java Virtual Machine) på maskinen.

Forskellige udgaver af Java kan downloades fra mange steder f. eks. fra http://java.sun.com. Den store **fordel** ved at bruge Java klasser er at de fungerer på **alle** maskiner – Windows, Unix, Linux, Mac! Det er programmer som er platforms uafhængige.

Med i pakken er også en meget detaljeret **dokumentation** om XML, XPATH og XSL, som under alle omstændigheder er meget læseværdig!

Med i materialet er også **bible**.xsl og man kan downloade hele Bibelen til visning med dette stylesheet!

Med SAXON følger også en stribe færdige programmer.

Vi skal her kun se på **hvordan** man helt konkret kan bruge SAXON parseren. Som et konkret eksempel vælger vi XSL transformatoren i SAXON. Vi kan kalde processoren på denne måde på en DOS linje – i en prompt:

java -classpath .;saxon.jar com.icl.saxon.StyleSheet -o output.html othello.xml flexnumber.xsl

Java klasser eller programmer kaldes altid med **java** – som er Java **fortolkeren**! En klasse kan **kun** køre hvis den fortolkes af java.exe – med mindre den er en applet som vi vil se eksempler på.

Vi skal lidt senere se på hvordan man kan anvende Java klasser mere generelt!

Ovenstående **forudsætter** at **saxon**.jar biblioteket ligger i den **samme** mappe! Ellers kan java fortolkeren ikke finde biblioteket.

Vi kalder altså en konkret Java klasse ved navn **StyleSheet** som ligger i det anførte namespace eller **package**! Der kan anføres forskellige **flag** - -o angiver output filen. –a bruges for at angive at stylesheetet er defineret i XML dokumentet – lige som –pi i Microsoft processoren. Parametre kan opregnes til sidst som vist.

Dette kan se besværligt ud men det hele kan gøres nemmere hvis man på **Windows** skriver f. eks. en **bat** fil med dette indhold (kommandoen) gemt som f. eks. style.**bat**! Man kan også anvende SAXON i **scripts** som vi skal se.

Som et meget lille eksempel på hvordan man kan bruge SAXON parseren kan vi se på denne Java klasse:

```
import com.icl.saxon.aelfred.*;

public class Parser {

public static void main(String args[]) {
SAXDriver sax=new SAXDriver();
try {
sax.parse(args[0]);
}
catch (Exception e){System.out.print(e.toString());}
}

}
```

Denne lille klasse kan så parse XML dokumenter og kontrollere om de er vel formede! Den henter et XML dokument fra DOS linjen på denne måde:

java -classpath .;saxon.jar Parser telefonliste.xml

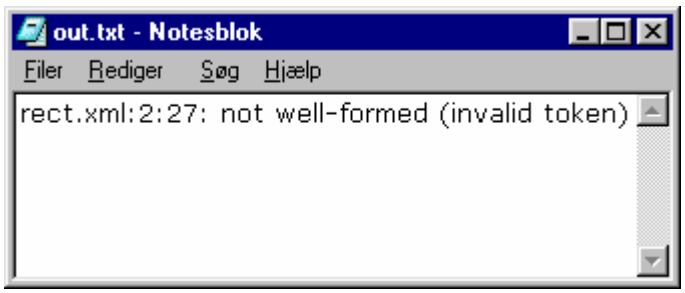
Vi kunne have gjort klassen lidt mere informativ ved at få udskrevet linje nummer og kolonne for eventuelle fejl! Men alligevel fungerer denne lille Java klasse som en OK XML parser!

SAXDriver er en såkaldt SAX2 parser som vi skal se på i afsnittet om SAX.

Expat:

er en XML parser skrevet i programmerings sproget C. Den skal også programmeres i C. Den er en **SAX** parser skrevet af James **Clark**. Den kan altså kun fungere hvis der skrives **event_handlere** til parseren som man gør i SAX.

Den kan frit downloades fra http://www.sourceforge.net. Når filerne er udpakket fås et færdigt program **xmlwf**.exe som kan checke om et XML dokument er velformet – vi bruger her det fejlagtige eksempel fra før:



Der findes en meget detaljeret **vejledning** i hvordan man kan skrive C **kode** til denne parser. Expat kan bruges sammen med f. eks. Borlands C++ Builder eller andre builder programmer til C eller C++.

Expat kan downloades i forskellige udgaver og kan køre på både Windows og **UNIX** og **Linux**!

Xerces:

er nok en af de mest anerkendte XML parsere som primært er skrevet af Java folk til programmering i Java (**Xerces**-j) Xerces parseren kan dog også fås i en C/C++ udgave (**Xerces**-c). Den har lige som Expat den fordel at den kan anvendes til **forskellige** operativ systemer – Windows, Mac, UNIX eller Linux!

Xerces kan downloades fra http://www.apache.org . Der er en meget detaljeret vejledning i hvordan man skriver Java kode til de hundreder af Java klasser som følger med!

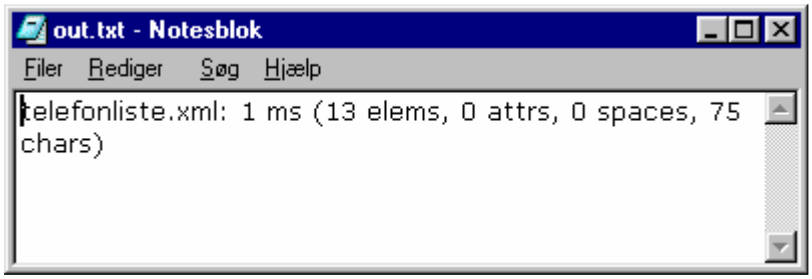
Xerces er en fri, open tool parser som kan klare næsten enhver opgave!

Med Xerces kan man også (i C udgaven til **Windows**) få en række færdige **programmer** som parser XML dokumenter på Windows platformen:

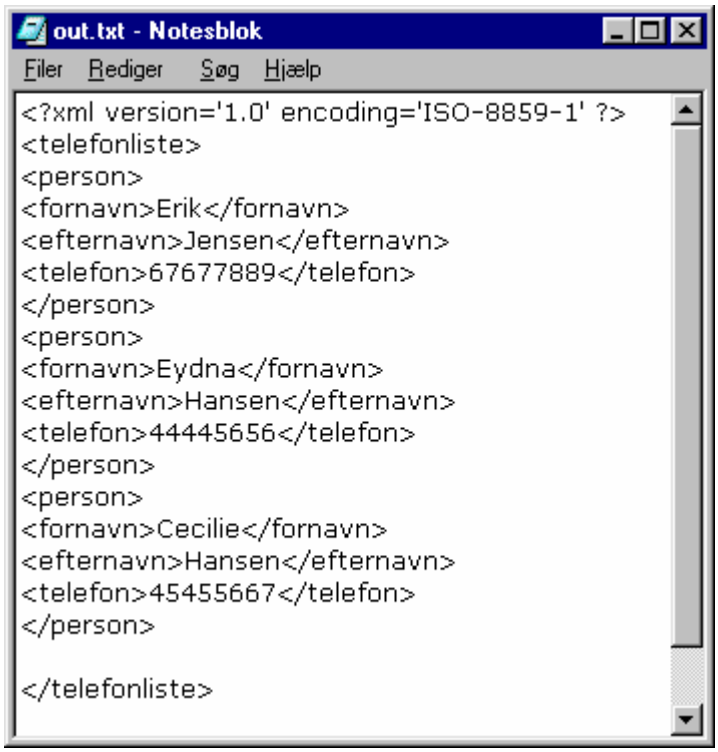
1. **SAXCount**.exe som er en parser
2. **DOMPrint.exe** som parser og udskriver et XML dokument
3. **SAXPrint.exe** som gør noget tilsvarende
4. **Redirect.exe** som kalder et externt DTD og validerer
5. **enumval.exe**
6. **stdinparse.exe**

Med flere! Disse programmer kan også **validere** hvis de kaldes med en option –v. Man kan altså let kontrollere og validere XML dokumenter med disse indbyggede programmer!

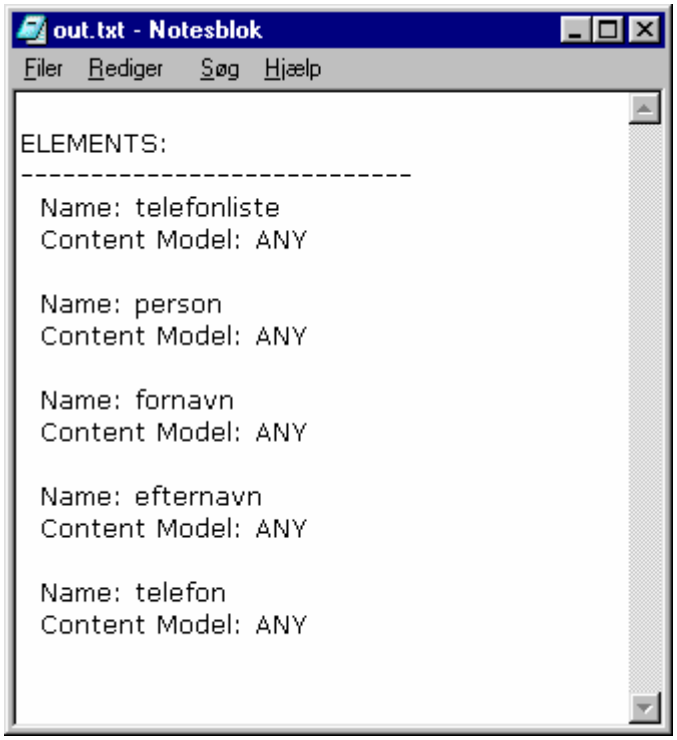
Resultatet af en **SAXCount** kan være:



DOMPrint:



Programmet **enumval**.exe, som opregner XML dokumentets elementer:



Vi skal siden se på nogle eksempler på hvordan man kan bruge de – mange - forskellige **Xerces** parsere.

XML parsere fra Microsoft .NET:

Man kan downloade .NET platformen fra http://msdn.microsoft.com og installere det på en Windows maskine.

.NET fylder – trods alt - kun ca. 20 MegaBytes.

.NET er et ekstra lag (styre system) som lægges oven i Windows styresystemet. I .NET får man adgang til en mængde færdige programmer eller klasser og det er muligt at skrive programmer i en række forskellige sprog bl. a. CSharp, Visual Basic og JScript.

Der findes en række forskellige parsere i .NET. Den - måske - vigtigste af dem er **XmlTextReader** som grundlæggende er en **SAX** parser – blot lidt mere brugervenlig!

Vi skal senere se på XmlTextReader.

XML parser fra Oracle:

Fra http://www.oracle.com kan frit downloades Oracles XML parser, en række supplerende værktøjer og en masse information og vejledning.

Parseren kan programmeres i Java eller i C eller C++.

XML parser fra Chilkat Software:

Denne parser kan også frit downloades fra adressen:

http://www.chilkatsoft.com/

hvor den findes i flere **forskellige** udgaver bl. a. en ActiveX Komponent til ASP og scripts. Parseren er lille, hurtig og kompakt. Den har mange specielle funktioner og er 'brugervenlig'!

Den er i mange henseender et gyldigt alternativ til de store. Ulempen kan være at denne parser i nogle tilfælde bruger sine egne metode navne osv. – Det kan forvirre i starten. Men stort set er properties og metoder de samme i alle XML parsere!

Denne parser har nogle specielle metoder der kan **søge**, **sortere**, **kode** og **komprimere** XML dokumenter! Et eksempel på et script der anvender **Chilkat** parseren i ASP versionen ser sådan ud:

```
<script>
function asp_xml(){
try {

                    //Opret et ASP XML parser objekt:
                    var xml = new ActiveXObject("AspXml.AspXml");

  xml.loadxmlfile("c:/xmlkursus/asp/telefonliste.xml");

  alert(xml.GetXml());
  //document.write(xml.GetXml());

  //tree walking:

  antal = xml.NumChildren;
  var i = 0;
  var s="";
  var node;
  var n;

  while (i < antal){
     node = xml.getChild(i);

     //ekstra attr:
     node.AddAttribute ("id_nummer", i);

     n = node.FirstChild();
     s+="\n"+n.content;
     n = n.NextSibling();
     s+="\n  "+n.content;
     n = n.NextSibling();
     n.AddAttribute ("privat", "true");

     s+="\n  "+n.content;
     i++;
```

```
  }

  alert(xml.GetXml());

  //alert(s);
  //document.write("<pre>"+s+"</pre>");
  }

  catch(e){ alert(e.description) ;}

  }
  xml();
</script>
```

At sortere kan gennemføres meget 'brugervenligt' således:

```
xml.LoadXmlFile("c:/xmlkursus/asp/telefonliste.xml");
xml.SortRecordsByContent("efternavn",true);
```

Der medfølger en del kode **eksempler** på forskellige løsninger i XML.

En primitiv script parser:

Nedenstående er et eksempel på en meget primitiv og **ufuldstændig** parser skrevet i script kode!

Det skal straks siges at der **faktisk** findes rigtige XML parsere skrevet i script kode – de kan også findes på Internettet!

Eksemplet skal kun illustrere hvad det **egentligt** er parseren gør. Men det er vigtigt at forstå at dette eksempel blot kan **udbygges** mere og mere – og til sidst ender vi med en 'rigtig' parser.

Det er vigtigt at erindre sig at forskellen på f. eks. **Notesblok** og en **parser** er, at parseren kontrollerer de tegn den læser og opretter en data **struktur** mens den læser teksten igennem! Notesblok læser blot tegn for tegn og viser resultatet!

1. Parseren kontrollerer om visse **ugyldige** tegn anvendes på forskellige steder – dels i tekst noder – dels i element navne – f. eks. at et element navn ikke må indeholde et '<' eller et mellemrum!
2. Parseren **opdeler** XML dokumentet i dele. At parse er at **dele** eller opdele – helt bogstaveligt!

Denne parser læser – som **alle** parsere eller XML readere – simpelthen teksten **tegn** for tegn. Her er det implementeret i en løkke!

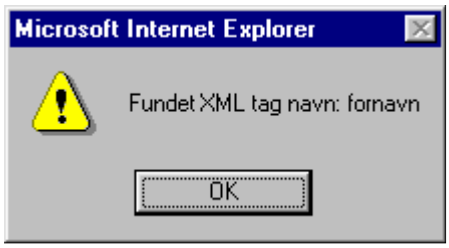
Vi vil kun se på den grundlæggende struktur i parser eksemplet – i nedenstående eksempel er nogle detaljer klippet fra!

```
<script>

//kan kun klare dette slags hierarki!:
parse("<data><navn><fornavn>Rosa & Lina</fornavn></navn></data>");
```

```
function parse(x) {
                    start=0;
                    slut=0;
                    navn="";
                    tekster=new Array();
                    index=0;
                    tagnavne=new Array();
                    tagindex=0;
                    niveau=0;
                    var xml=new String(x);

for(i=0;i<xml.length;i++) {
   c=xml.substr(i,1);
   if(c=='<'){
      start=start +1;
      if(navn.length>0) {
      alert("Fundet #PCDATA: "+navn);
      if(navn.indexOf('<')!=-1) {fejl1();return;}
      if(navn.indexOf('&')!=-1) {fejl1();return;}
      if(navn.indexOf('>')!=-1) {fejl1();return;}
      if(navn.indexOf('"')!=-1) {fejl1();return;}
      tekster[index]=navn;
      index=index+1;
      navn="";
      }
   }

   else if (c!='>'){
       navn+=c;
   }
//parseren læser tegnet '>':

   else if(c=='>'){
      alert("Fundet XML tag navn: "+navn);

      if(navn.substr(0,1)=='/'){
      niveau=niveau-1;
      }

   else niveau=niveau+1;
   alert("Dybde: "+niveau);


   //ulovligheder i tagname – flere kunne tilføjes her:

   if(navn.indexOf(' ')!=-1) fejl();
   if(navn.substr(0,1)=='-') fejl();
   if(navn.substr(0,1)=='.') fejl();

   //starter med plads 0:
   tagnavne[tagindex]=navn;
    tagindex=tagindex+1;
      navn="";
      slut=slut+1;
   }
```

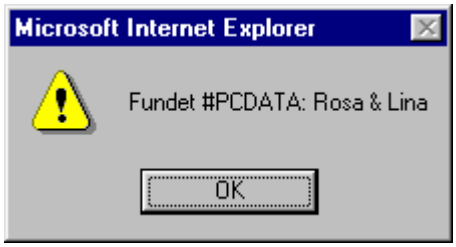
```
}//slut med for loop!

//er dokumentet velformet?

if (start==slut)alert("Gyldigt XML dokument: < >");
if (start!=slut)alert("Ugyldigt XML dokument: < >");

//vis den ene data struktur: tekstnoder:

for(m=0;m<index;m++){
alert("tekster: "+tekster[m]);
}

}
}

</script>
```

Dette script eksempel fungerer rent faktisk neb kun på begrænsede XML dokumenter som vist i eksemplet:



Parseren kan holde rede på i hvilken dybde i træet vi befinder os:

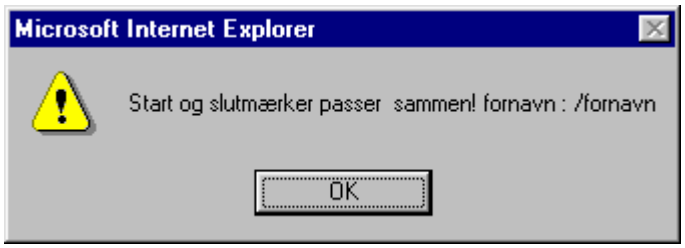
Parseren kan skelne mellem tekstnoder og tag navne og opbygger en liste over dels element navne dels tekst noder:



Den kontrollerer tekstnoderne for ugyldige tegn:

Den tæller simpelt hen hvor mange '<' tegn og hvor mange '>' tegn der er i dokumentet og melder fejl hvis der ikke er lige mange! Dette er selvfølgeligt IKKE nogen fuld kontrol af dokumentet – kun et nemt eksempel! Dette eksempel viser også hvad der er typisk for en sådan SAX parser: at den simpelt hen læser tegn for tegn!

Parseren foretager også en kontrol af at elementer åbnes og lukkes på den korrekte måde:



Vi kan indføre en bevidst fejl sådan her:

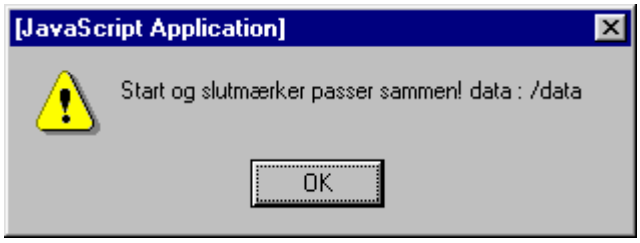
parse("<data><navn><.fornavn>Rosa & Lina</fornavn></navn></data>");

Et **element** navn må i XML ikke starte med et punktum:

Parseren kan også klare mixed content XML dokumenter f. eks. denne type:

parse("<data>Data<navn>Navn<fornavn>Rosa & Lina</fornavn></navn></data>");

Den opretter korrekt en data struktur over alle 3 tekstnoder i dokumentet.



Om brug af Java klasser (programmer):

I dette materiale er der nogle steder vist eksempler fra programmering i Java. Java har spillet en meget stor rolle i XML fra starten. W3C Konsortiet har udgivet officielle **language bindings** for Java – men ikke for andre sprog (bortset fra Java script – ECMA script)! SAX – som vi skal se senere – blev opfundet til Java. Java spiller en stor rolle i programmering af XML dokumenter.

For mange vil Java være en fremmed verden! Vi skal derfor her se på **hvordan** man kan anvende Java – **uden** at have særligt meget forstand på Java!

Applets:

For det første anvendes Java i form af Java **applets** som er en del af en web side eller HTML side. Disse applets kan anvendes af alle – de fungerer blot som **komponenter** når man åbner web siden og kræver sådan set ingen forudsætninger og de kræver heller **ikke** at man har installeret Java eller Java klasser på sin maskine! Applets fungerer uanset styresystemet – Windows, Unix, Linux eller Mac – og helt **uafhængigt** af hvilke programmer man har installeret på maskinen! Applets er virkeligt **portable** – uafhængige.

Vi skal senere se eksempler på applets der kan bruges til XML.

Java klasser i øvrigt:

For det andet kan man anvende færdige Java klasser – programmer. En Java klasse er **ikke** et program på samme måde som f. eks. et Windows program (et .exe program). Java klassen er kompileret **byte** kode som skal køres gennem en Java **fortolker** (java.exe) og **oversættes** til f. eks. Windows kode!

En forudsætning for at kunne køre et Java program – en Java klasse – er at man har **installeret** en Java fortolker d.v.s. i praksis Java Virtual Machine på maskinen! Java kan downloades – i forskellige udgaver - mange steder på Internettet f. eks. fra http://java.sun.com.

Når man har installeret Java installeres det i en bestemt mappe. Det er vigtigt at systemet kan finde java.exe! Systemet skal altså have en path til den mappe hvor java.exe ligger! Man kan teste om dette virker ved at skrive java på en DOS kommando prompt – et DOS vindue.

Java klasser køres derefter nemmest ved at man på Windows skriver en .bat fil med kommandoen der starter Java klassen. Fra denne hjemmeside kan man downloade en lille SAX parser som også validerer om XML dokumentet er gyldigt i forhold til et XSD skema!

De **filer** som skal **downloades** er:

1. **xercesImpl**.jar (indeholder Xerces XML klasserne)
2. **SAXParser**.class (en Java SAX parser)
3. **H**.class (en event handler klasse som parseren skal bruge)
4. **eks8**.xml (et XML dokument)
5. **skema8**.xsd (et XSD skema)
6. **java_**parser.bat (den .bat fil som automatisk kører Java parseren)

Hvis alle disse filer ligger i **samme** mappe kan man blot dobbelt klikke på .bat filen i Windows Stifinder og køre programmet! Programmet producerer en **tekst** fil med resultater fra læsningen af XML dokumentet.

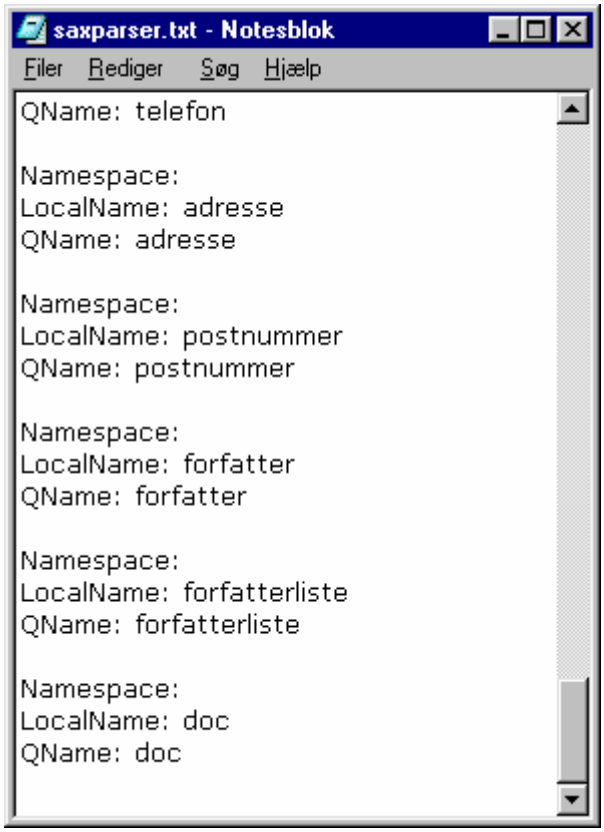
Indholdet af .bat filen er denne kommando:

java -classpath .;xercesImpl.jar;c:\java\java\lib\classes.zip SAXParser eks8.xml > saxparser.txt

Dette kan forklares på denne måde:

1. Output lægges over i saxparser.txt i samme mappe
2. Parseren læser eks8.xml - som har en reference til et skema (som derfor også – automatisk - læses af parseren)
3. Vi sætter en classpath d.v.s. vi angiver hvor Java klasser befinder sig så systemet kan finde dem! Java klasserne ligger i:
    a. punktum (nuværende mappe)
    b. i biblioteket xercesImpl.jar
    c. i biblioteket classes.zip i den nævnte mappe (i **dette** tilfælde!)
4. Vi kalder klassen SAXParser med java (eller java.exe) og et .xml dokument som parameter

På denne måde kan kaldes **alle** mulige Java klasser – programmer – på en nem og sikker måde! Resultaet af at køre denne parser er nogenlunde dette:



Hvis der er skema fejl – altså hvis XML dokumentet er **ugyldigt** – eller hvis XML dokumentet ikke er **velformet** kommer der en fejlmelding

Encoding:

Vi har allerede set på spørgsmålet om et XML dokuments **encoding**. Det er generelt altid en god ide at gemme et dokument med en XML erklæring og en bestemt encoding! Dette er i al fald **nødvendigt** hvis der anvendes tegn som ikke er ASCII tegn (’engelske’ tegn)!

Et eksempel på et dokument **uden** en xml erklæring:

<data>I dette år besøges mange ærtesteder!</data>

Dette dokument – et helt gyldigt XML - kan oftest **ikke** vises i en browser – der er et **’ugyldigt’** tegn i position 15! Dette skyldes ikke at XML eller Unicode ikke forstår tegnet ’å’ - men at dokumentet bliver kodet forkert. Unicode og XML ’forstår’ omkring 90.000 forskellige tegn – altså også ’Æ’!

Hvis vi ændrer dokumentet og skriver en **eksplicit** encoding bliver det vist OK:

Men de forskellige XML parsere 'forstår' kun et begrænset antal kodninger! F. eks. findes en **forbedret** version af ISO-8859-1 som hedder ISO-8859-**15** – men den forstås ikke af f. eks. Internet Explorer – selv om den er en officiel anerkendt encoding!

Et tegn sæt (character set):

Et tegn sæt er en række af gyldige tegn som er definerede i det konkrete tegnsæt! Et tegn kan være et bogstav, et symbol eller et kinesisk skrifttegn! **Unicode** 3.1.1 definerer flere end 90.000 forskellige tegn og de er alle gyldige i et XML dokument! Det er ikke sikkert at de kan skrives i Notesblok – men de er gyldige tegn i et XML dokument!

Alle tegn har en bestemt tal kodning – et såkaldt kode punkt! De første kode punkter er nummer 0 til og med nummer 127. Bogstavet 'a' har kodepunktet **97** i ALLE tegnsæt! Uanset hvordan vi sætter vores encoding i xml erklæringen betyder 97 altid det samme!

Men for værdierne over 127 er de forskellige tegnsæt mere eller mindre forskellige! Dvs. nummer – kode punkt - 4444 er ikke altid det samme tegn – det afhænger af tegnsættet og vores encoding!

UTF-8 og UTF-16 (Unicode):

Hvis der IKKE skrives en encoding erklæring bliver XML dokumentet gemt automatisk i UTF-8 som er en speciel kodning hvor hvert tegn oversættes til mellem 1 og 6 bytes afhængigt af om det er et kinesisk skrifttegn eller et engelsk 'a'! UTF-8 koder altså med et variabelt antal bytes for hvert tegn!

UTF-16 som ofte kaldes **'Unicode'** koder derimod med et fast antal bytes og bits: Hvert eneste tegn oversættes til nøjagtigt 2 bytes eller 16 bits – altid! UTF-16 filer vil derfor ofte fylde mere end UTF-8 eller ISO-8859-1 filer! Hvis teksten kun (eller mest) indeholder engelske tegn – bliver filen **dobbelt** så stor! Dette er baggrunden for at man opfandt kodningen UTF-8!

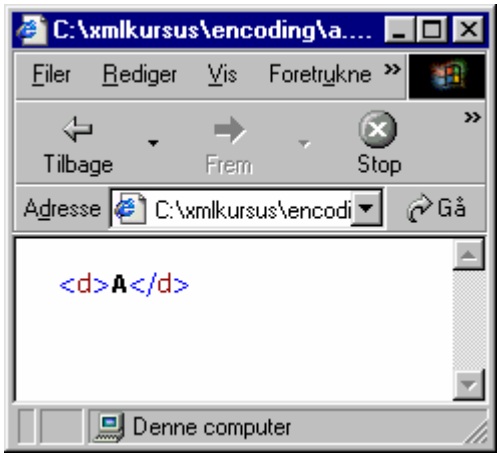
Man kan have problemer med disse formater fordi nogle metoder f. eks. i MSXML automatisk koder i UTF-16 – selv om vi har bedt om noget andet! I så fald får vi en fejl melding i browseren –

at der er skiftet kodning! Hvis et XML dokument først er blevet gemt med en bestemt kodning kan man ikke hen ad vejen ændre denne kodning!

Et eksempel: Vi skriver denne yderst simple tekst og gemmer den som a.xml:

<d>A</d>

Dette XML dokument vises OK i enhver browser!



Vi tilføjer nu en encoding således:

<?xml version='1.0' encoding='utf-16'?>
<d>A</d>

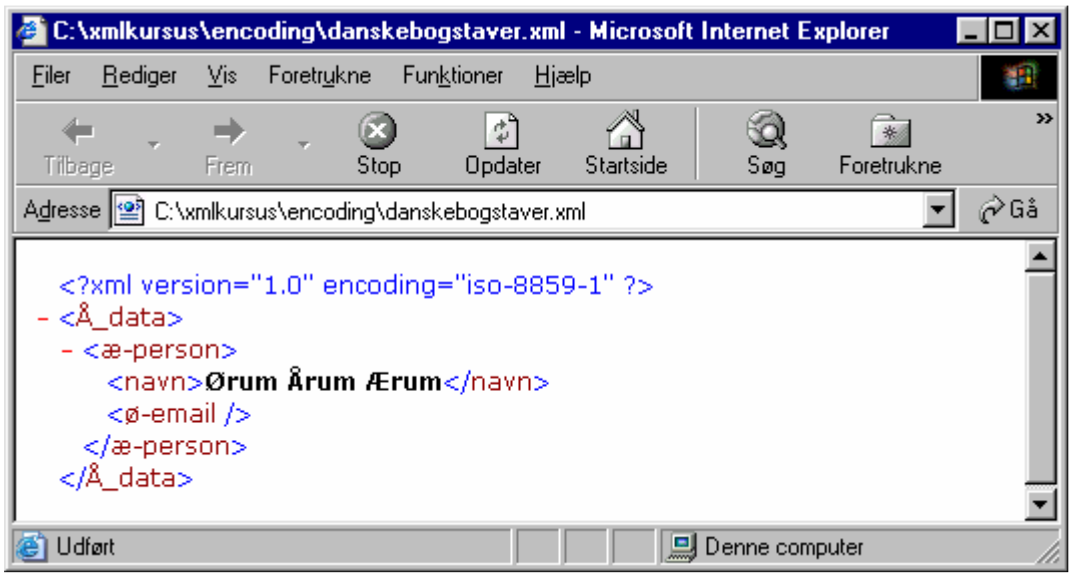
Vi får nu en fejl:

Hvis vi ændrer vores kodning til:

<?xml version='1.0' encoding='utf-8'?>

Vises dokumentet uden problemer! XML dokumenter bliver altså gemt automatisk i UTF-8 hvis intet andet angives!

Det væsentligste er altså at sætte en passende encoding – så opstår ingen problemer:



Som det kan ses er 'æ' osv. OK både i tekstnoden og i definitionen af et element!

Hvis den tekst behandling man arbejder med ikke kan klare 90.000 forskellige tegn kan man altså altid – sikkert – indsætte fremmede tegn med tegn referencer:

ᅜ

XML kan takle alle disse tegn selv ens tekstbehandling eller browser ikke kan!

Man kan vælge forskellige **skrifttyper** i en tekst behandling. Dette er **irrelevant** og har intet at gøre med XML dokumenters **encoding**.

Skift af encoding giver engang imellem problemer. Et eksempel ville være denne fil:

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

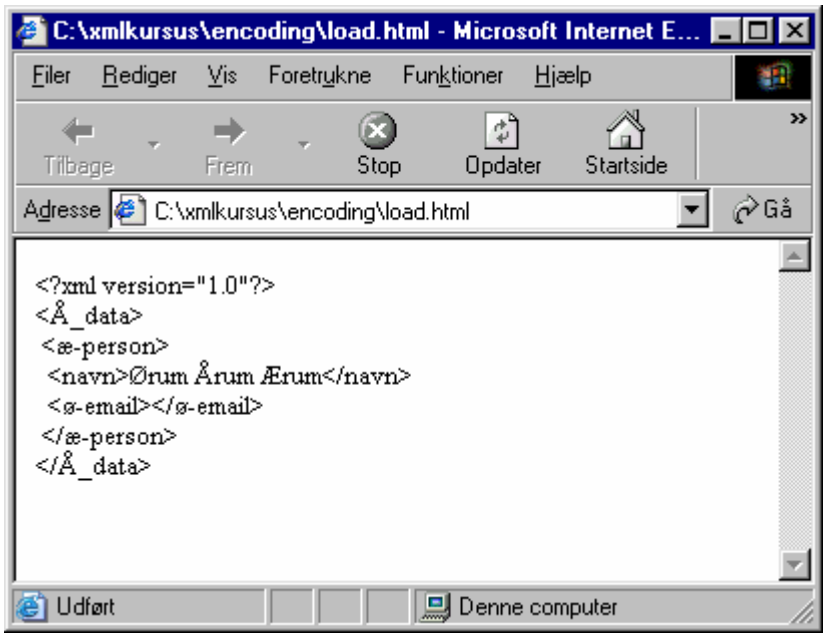
<Å_data>
<æ-person>
<navn>Ørum Årum Ærum</navn>
<ø-email></ø-email>
</æ-person>

</Å_data>

Dette dokument vises OK f. eks. i en browser. Men hvis vi **transformerer** eller bearbejder det i script kode – med et XSLT stylesheet - bliver resultatet **altid** kodet til UTF-16:

```
<body>
<div id="div"</div>
</body>
<script>
load();
function load(){
var doc=new ActiveXObject("MSXML2.DOMDocument.4.0");
doc.load("danskebogstaver.xml");
div.innerText=doc.xml;
}
</script>
```

Når vi loader vores dokument og henter værdien doc.xml bliver resultatet et nyt dokument kodet i UTF-16 og vores encoding bliver **droppet** af parseren!

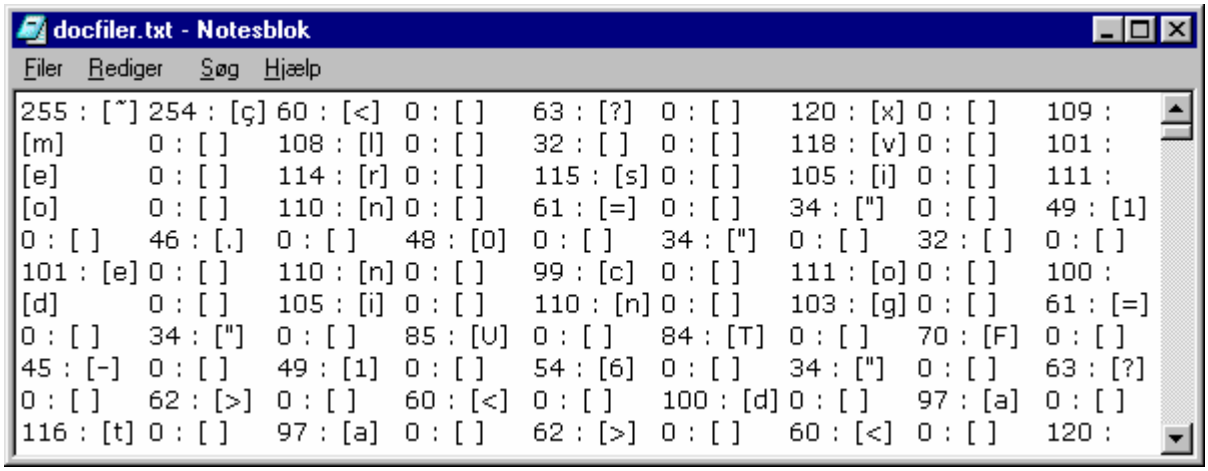


Hvis vi gemmer doc.xml i et nyt dokument dansk.xml og vil åbne det får vi at vide at dokumentet indeholder ugyldige tegn!:

Vi vil senere vende tilbage til spørgsmål og problemer med encoding.

Unicode eller UTF-16 dokumenter er specielle ved at de altid har en **'BOM'** – **byte order mark** – som de første bytes i filen **før** <?xml...?>! Dette BOM indsættes **automatisk** – det skrives aldrig direkte! - og derfor kan det af og til give nogle problemer!

Hvis vi ser på en Unicode tekst starter den således – her er den læst binært d.v.s. vi får de enkelte bytes og deres værdi:



Vi kan se at de to første bytes er **255** (F) og **254** (E). Først **derefter** starter xml erklæringen – som starter med byte nummer 60 ('<').

I dette eksempel kan vi også se, at de fleste bytes er 'spildt' – er sat til 0! Derved bliver Unicode ofte et **tungere** format end UTF-8 eller især iso-8859-1!

Mange **tekstbehandlinger** vil gemme XML dokumenter i Unicode eller UTF-16 – hvis man ikke udtrykkeligt angiver en encoding!