Namespaces.....1
Default namespace:.....6
Præfiks:.....7
To slags navne i XML:.....11
Standard namespaces:.....14
RDF Resource Description Framework:.....18
Attributter:.....19
DTD skemaer og namespaces:.....21

# Namespaces.

Et **namespace**, **navnerum** eller **navneområde** er et 'super-element' eller en 'super-klasse' som nøjere definerer et XML element eller en XML attribut.

På adressen http://www.w3.org kan man finde W3C konsortiets specifikation – fra år 2000 - af XML Namespaces! Den oprindelige specifikation af XML fra 1998 – XML 1.0 - indeholdt **ikke** namespaces! derfor findes stadig parsere som ikke forstår namespaces eller navneområder! DTD skemaer og CSS forstår på denne måde heller ikke hvad et namespace er – som vi skal se!

En sammenligning kunne være vejnavne. Navnet 'Sortemosevej' kan forekomme i flere kommuner eller postnumre. Vi kan altså ikke nøjes med at skrive 'Sortemosevej'. I stedet kan vi skrive fx '2730.Sortemosevej' eller '2860.Sortemosevej'.

I Objekt Orienteret Programmering angives et namespace ofte på denne måde (namespace + et punktum + navnet). Et **namespace 'kvalificerer'** et objekt eller element så det er entydigt.

På samme måde kan vi skelne mellem to **filer** der begge hedder 'fil.txt' hvis de ligger i to forskellige **mapper**! Den ene hedder mappe1/fil.txt, den anden hedder mapp2/fil.txt! Et navne rum svarer til mappen.

Namespaces har dog også en langt mere videregående anvendelse – de svarer til biblioteker eller libraries eller 'programmer'. Når man importerer et navnerum importerer man samtidigt ofte en række funktioner.

Vi kan kvalificere elementer på denne måde:

---

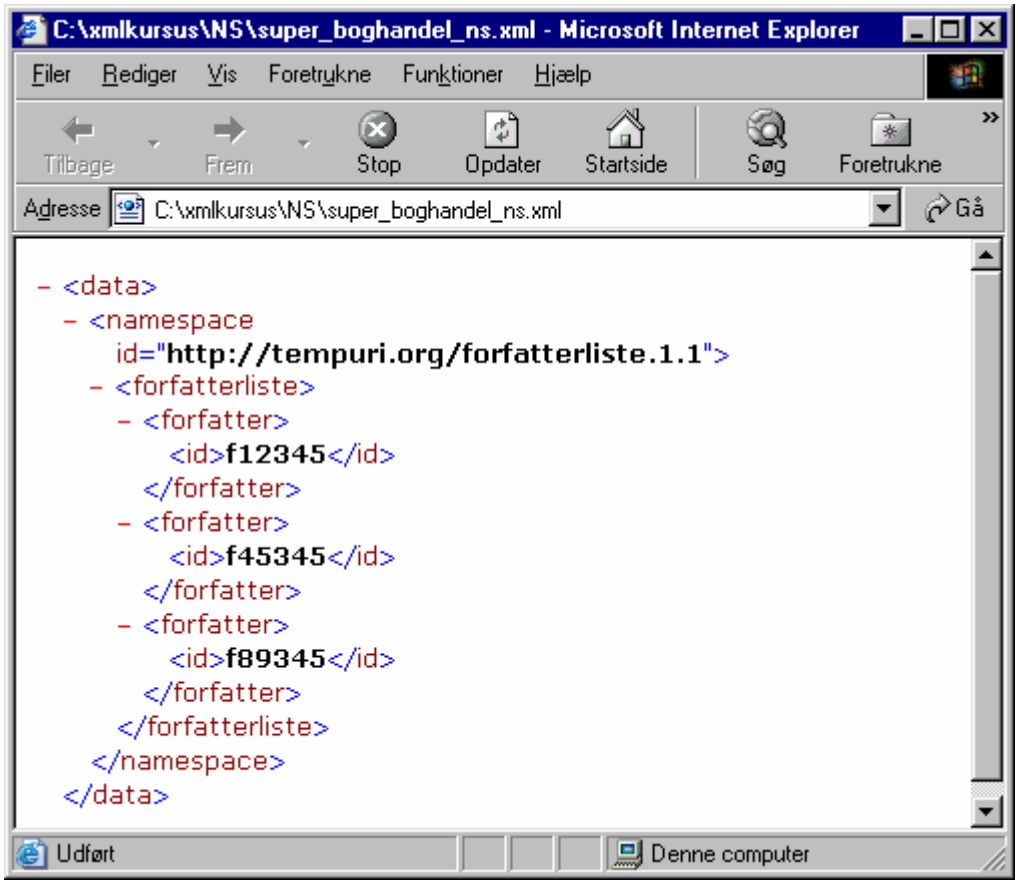
```
<?xml version="1.0" encoding="iso-8859-1"?>

<data>
<forfatterliste xmlns="http://tempuri.org/forfatterliste.1.1">
                    <forfatter>
                    <id>f12345</id>
                    </forfatter>
                    <forfatter>
```

```
                <id>f45345</id>
                </forfatter>
                <forfatter>
                <id>f89345</id>
                </forfatter>
</forfatterliste>

<bogliste xmlns="http://tempuri.org/bogliste.1.1">
                <bog>
                <id>b12345</id>
                </bog>
                <bog>
                <id>b67345</id>
                </bog>
                <bog>
                <id>b45345</id>
                </bog>
</bogliste>
</data>
```

---

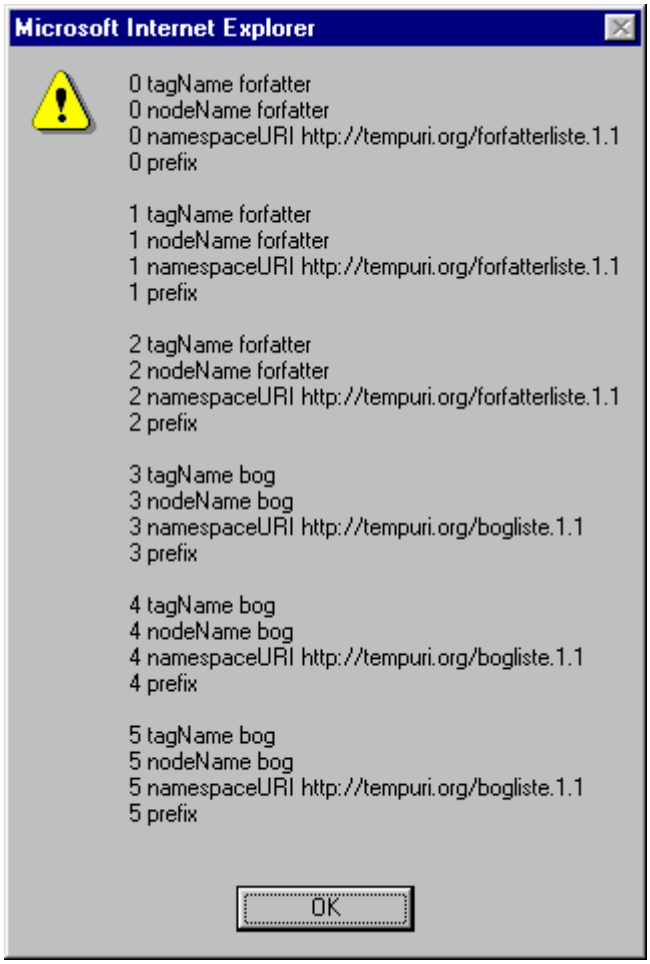
I dette tilfælde indeholder roden 'data' 2 sub-**træer** nemlig forfatterliste og bogliste. Disse to er **kvalificerede** med to namespaces. Når et namespace defineres på denne måde gælder det **hele** det sub **træ** som ligger neden under elementet. De to namespaces er her erklæret med et standard navneområde - et **'default** namespace' dvs at navnerummet gælder default i **hele** sub træet.

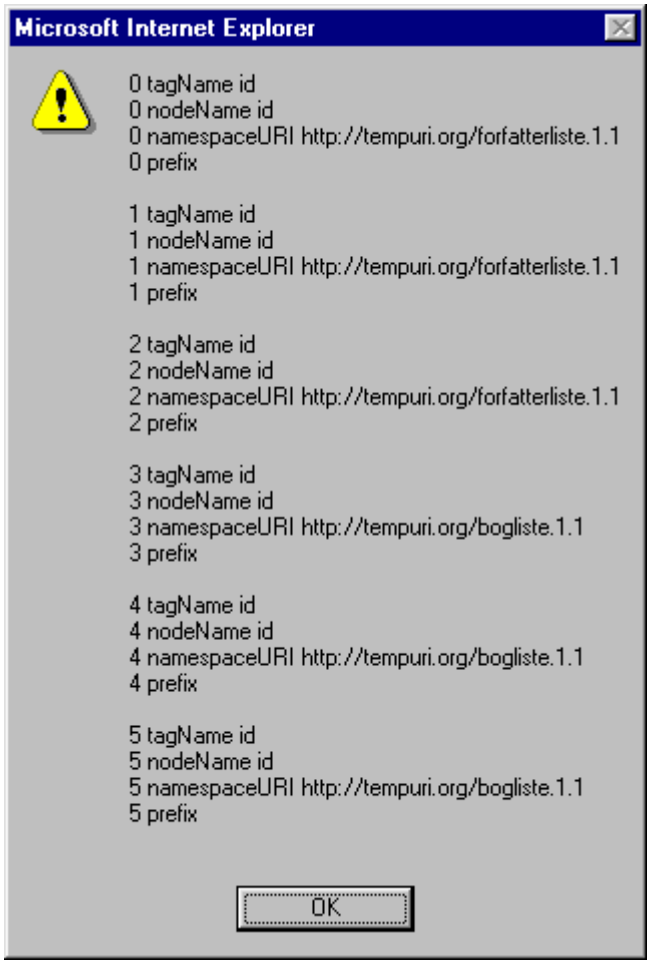
Namespace noden er en **selvstændig** node – ikke en attribut! Logisk set kan man forestille sig at namespace noden ligger som et super element **oven** over elementet som en **container**:




Vi kan på denne måde direkte se at forfatterliste er **indeholdt** i elementet namespace!

Hvis vi analyserer XML dokumentet fås dette resultat:


Microsoft Internet Explorer
0 tagName forfatter
0 nodeName forfatter
0 namespaceURI http://tempuri.org/forfatterliste.1.1
0 prefix
1 tagName forfatter
1 nodeName forfatter
1 namespaceURI http://tempuri.org/forfatterliste.1.1
1 prefix
2 tagName forfatter
2 nodeName forfatter
2 namespaceURI http://tempuri.org/forfatterliste.1.1
2 prefix
3 tagName bog
3 nodeName bog
3 namespaceURI http://tempuri.org/bogliste.1.1
3 prefix
4 tagName bog
4 nodeName bog
4 namespaceURI http://tempuri.org/bogliste.1.1
4 prefix
5 tagName bog
5 nodeName bog
5 namespaceURI http://tempuri.org/bogliste.1.1
5 prefix
OK




Vi kan se at vores **namespaces** også kommer til at gælder **sub** elementerne under bogliste og forfatterliste! Grunden til at namespaces anvendes er bl. a. at vi på denne måde kan skelne mellem elementet **'id'** i de to helt forskellige sammenhænge. Der er forskel på id i bogliste subtræet og id i forfatterliste subtræet!

Dette skrives nogle gange sådan – med parenteser eller **braces** - i forklaringer til namespaces (og af XML processorer fx i fejl meldinger):

{http://tempuri.org/bogliste.1.1}id eller
{http://tempuri.org/forfatterliste.1.1}id

Dette svarer til det tidligere eksempel: '2730.Sortemosevej'. Man kan også sige at vi ikke behøver at kalde de to id elementer to forskellige ting. Begge kan hedde 'id'. Vi kan alligevel **skelne**.

En namespace erklæring - som f. eks. xmlns="uri:bogliste" - er som sagt en slags **super** node som anbringes oven over det element hvori namespacet bliver erklæret. Det kan også se hvis man analyserer et XML dokument med **SAX** metoder – som vi siden skal se. Navneområdet **erklæres** godt nok inden i et element – det kan **aldrig** stå alene - men er egentligt **overordnet** elementet! Når vi læser et XML dokument med SAX metoder bliver namespacenoden kaldt **før end** selve elementet!

Et namespace er et (helst!) **unikt** ID, men som værdi eller tekst **kan** anvendes stort set hvad som helst. Men hvis dette namespace skal anvendes af andre – fx på **Internettet** eller et **intranet** - skal det helst være **unikt** - ellers virker kvalifikationen ikke. Derfor anvendes ofte sider på nettet (http eller ftp adresser) som ID - fordi der ikke findes to web sider som har det samme navn. Men det betyder ikke nødvendigvis at der findes noget i den anden ende af et namespace - selv om det ligner en http adresse! Dette har givet megen **forvirring** – men sådan fungerer systemet altså. Http adresser – f. eks. – vælges kun eller primært fordi de er unikke! En e-mail adresse ville også kunne anvendes – den er altid unik.

Et namespace kan fx være:

1. xmlns=”http://www.computeraps.dk/bogliste.1.1”
2. xmlns=”ftp://167.167.167.167”
3. xmlns=”mailto:jensjensen@mail.dk”
4. xmlns=”uri:12345.bogliste.xyz”

Namespaces bruges ofte sammen med **skemaer** som skal validere XML dokumentet. Skemaet definerer så det særlige sprog – **vocabulary** - eller den særlige XML **applikation**. I vores eksempel kan man sige at vi definerer et ’Data MarkUp Language’, DML ligesom XML eller HTML, med en rod som hedder ’data’!

Logisk set **bør** ALLE XML dokumenter have et (default) namespace – selv om eksemplerne i dette kursus ikke lever op til dette princip!

Der er en vis **tradition** for at angive et namespace som f. eks. http://olenyborg.com/personer som så svarer til en placering af selve skemaet på http://olenyborg.com/personer/personer.xsd - som vi skal se senere i forbindelse med skemaer til XML dokumenter.

Men et XML dokument kan have et skema **uden** at anvende namespaces - og omvendt anvende namespaces **uden** at have et skema! Men som en grundlæggende regel er det naturligt at ethvert XML dokument har et navneområde som definerer dets karakter eller definerer XML applikationen.

## *Default namespace:*

Ofte bruges et **default** namespace i roden (eller i et sub element til roden) til at angive det XML sprog eller den XML **applikation** som dokumentet (eller sub træet) tilhører. Eksemplet kunne også være skrevet sådan:

---

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

<data xmlns="http://tempuri.org/forfatterliste.1.1">
<forfatterliste>
                <forfatter>
                <id>f12345</id>
                </forfatter>
.....
```

---

I dette tilfælde er namespacet i roden så default namespace for HELE dokumentet – hvis intet andet er angivet!

Dette vil typisk gøres hvis dokumentet kun anvender et namespace altså tilhører et XML sprog eller applikation. Eller hvis stort set alle elementer tilhører det samme namespace.

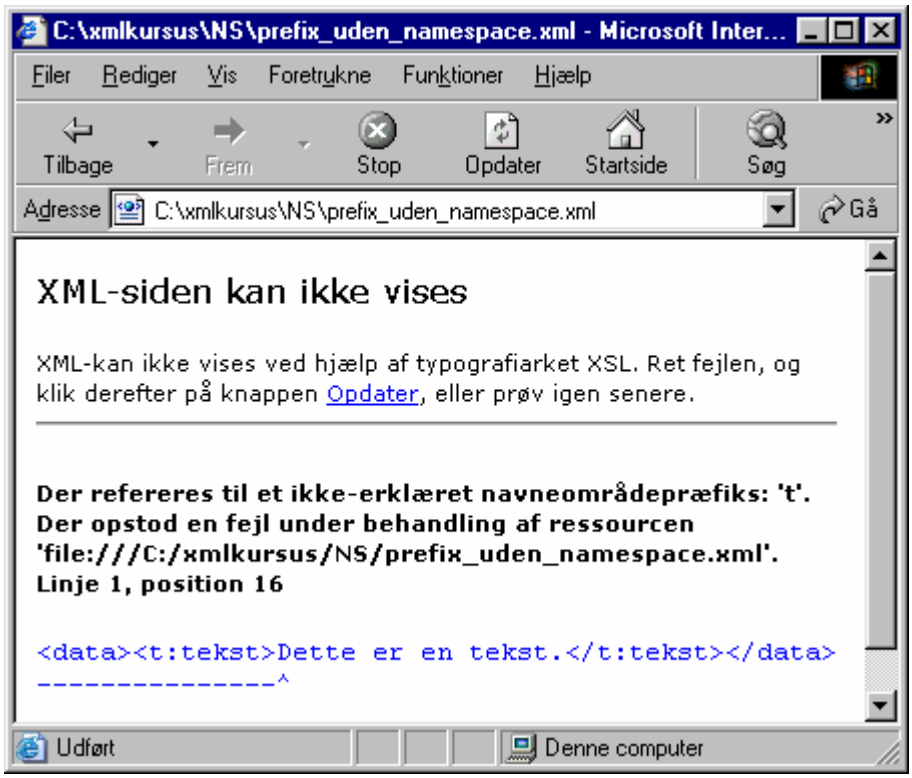
## *Præfiks:*

Et namespace eller navne område kan oprettes **med** eller **uden** en præfiks.

Det er **ikke** tilladt at bruge en præfiks **hvis** den ikke er erklæret sammen med et namespace!
Følgende XML fil:

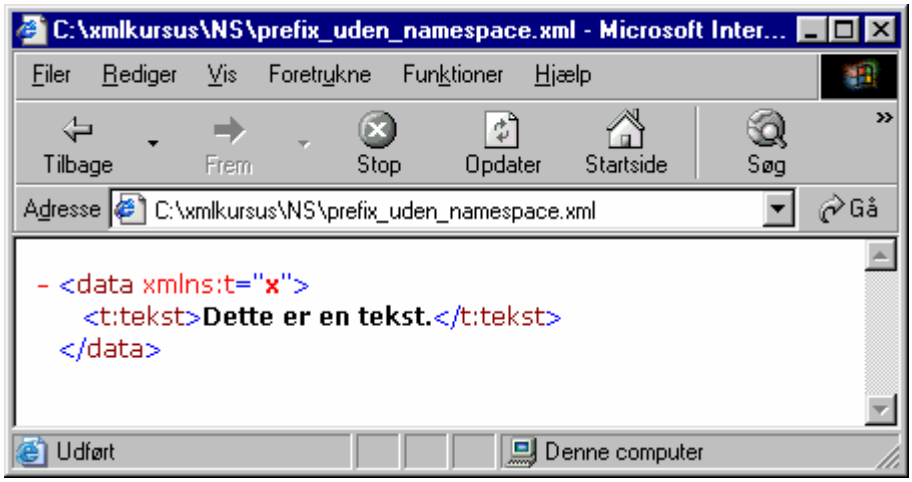
```
<data><t:tekst>Dette er en tekst.</t:tekst></data>
```

giver en fejlmelding:




Men der skal ikke meget til for at gøre t præfikset lovligt:

```
<data xmlns:t="x"><t:tekst>Dette er en tekst.</t:tekst></data>
```

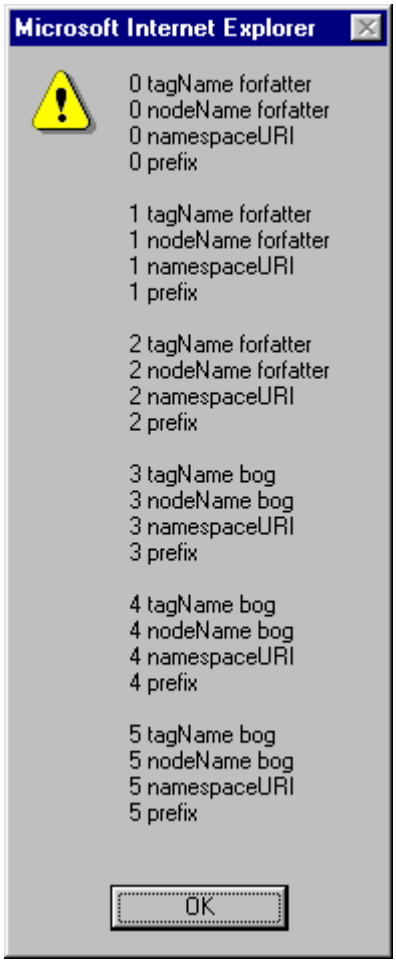



Præfikset er en **vilkårlig** forkortelse for selve namespace-værdien (som jo også kan være helt vilkårlig!). Man kan selv definere dette præfix – dog må man ikke anvendes bogstaverne x, m eller l, som er reserverede til w3.org formål! Præfikset kan ellers – normalt - være hvad som helst! Hvis man importerer standard navneområder som f. eks. XML skema navneområdet kan man også vilkårligt definere præfikset! Præfikset er – normalt – altid vilkårligt. (HTML som namespace er dog nogle gange en undtagelse som vi siden skal se). Vi kan f. eks. gøre således for at indføre præfikser:

---

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

<data
xmlns:forfatter="http://tempuri.org/forfatterliste.1.1"
xmlns:bog="http://tempuri.org/bogliste.1.1"
>
<forfatter:forfatterliste>
                <forfatter>
                <id>f12345</id>
                </forfatter>
                <forfatter>
                <id>f45345</id>
                </forfatter>
                <forfatter>
                <id>f89345</id>
                </forfatter>
</forfatter:forfatterliste>

<bog:bogliste>
                <bog>
                <id>b12345</id>
                </bog>
                <bog>
                <id>b67345</id>
                </bog>
                <bog>
                <id>b45345</id>
                </bog>
</bog:bogliste>
</data>
```

Det er vigtigt at forstå at nu er det **kun** de to elementer bogliste og forfatterliste som er i et namespace. Deres sub elementer er **ikke** i et bestemt namespace!

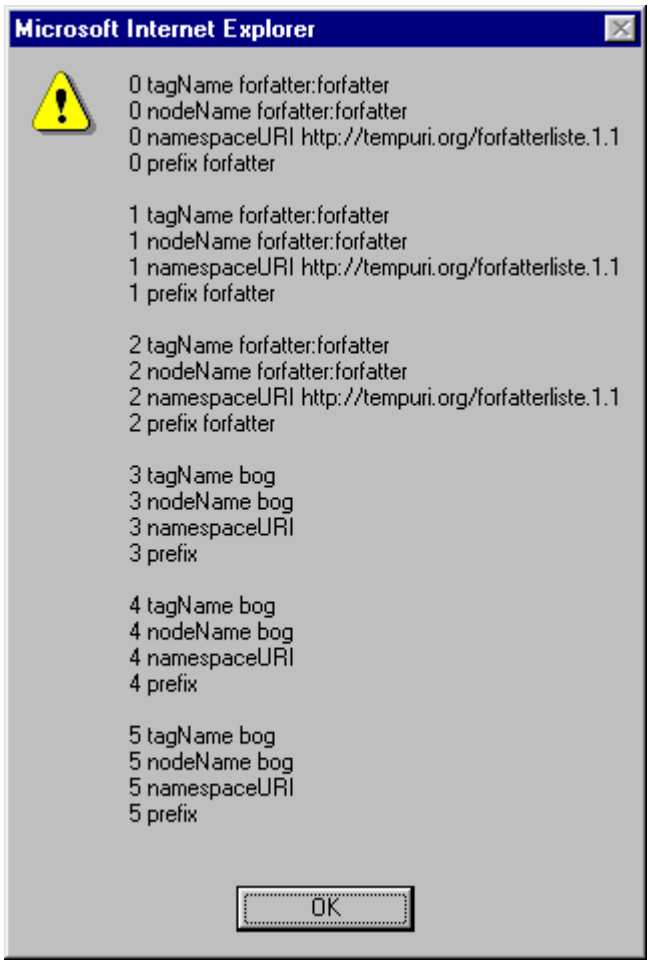




Som det ses er elementerne bog og forfatter nu pludseligt **uden** kvalificering!!

På denne måde kan man frit blande flere forskellige XML sprog eller applikationer. Vi kan nu kvalificere elementet forfatter på denne måde:

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

<data
xmlns:forfatter="http://tempuri.org/forfatterliste.1.1"
xmlns:bog="http://tempuri.org/bogliste.1.1"
>
<forfatter:forfatterliste>
                    <forfatter:forfatter>
                    <id>f12345</id>
                    </forfatter:forfatter>
                    <forfatter:forfatter>
                    <id>f45345</id>
                    </forfatter:forfatter>
                    <forfatter:forfatter>
                    <id>f89345</id>
                    </forfatter:forfatter>
</forfatter:forfatterliste>

.....
```

Nu er elementet **forfatter** kvalificeret (**men** sub elementet id – og bog - har IKKE et namespace!):



## To slags navne i XML:

Før XML Namespaces – altså i den oprindelige udgave af XML – fandtes kun een slags XML navne der skulle overholde visse regler – som vi har set. Et XML name må f. eks. ikke starte med et tal og det må ikke indeholde en '/'! Men den oprindelige definition sagde at navnet godt må indeholde kolon! Følgende var altså fuldt ud **gyldigt** i følge XML standarden fra 1998:

<x:y:z>Dette er x:y:z</x:y:z>

Hvis vi prøver at opnå dette XML dokument i Internet Explorer fås en fejlmelding:

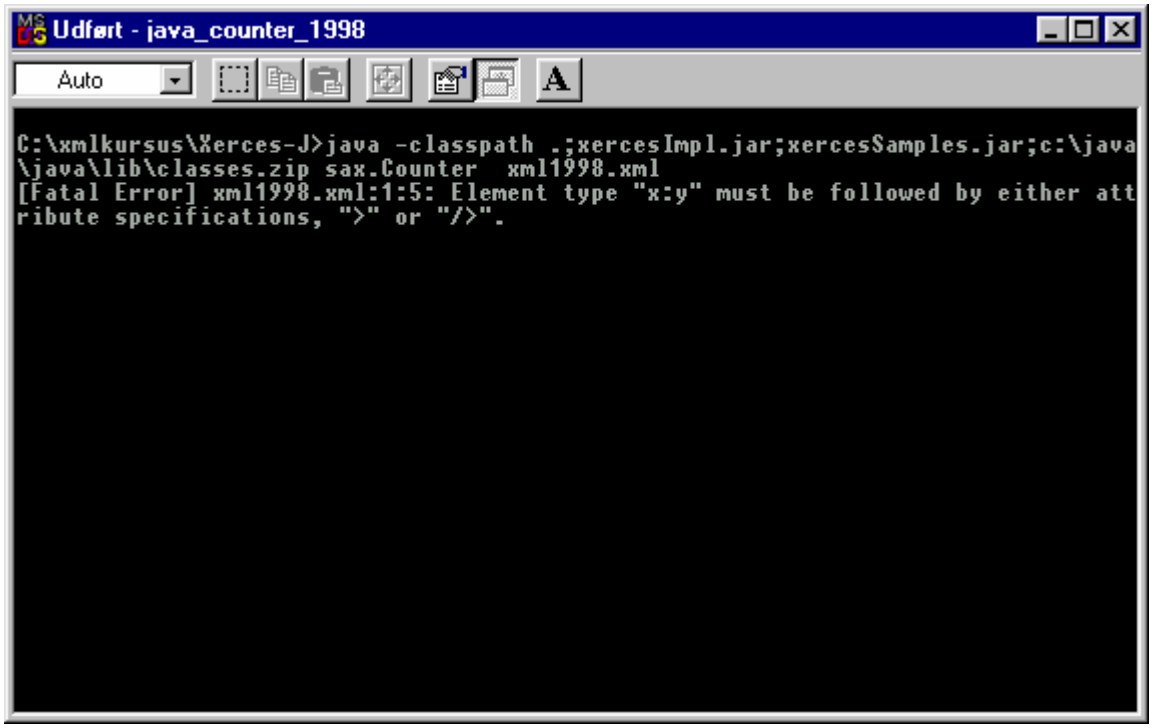

Dette element er ugyldigt – fordi Internet Explorer anvender en Namespace XML parser! I følge den nye – nu helt almindelige XML standard – må et almindeligt XML navn ikke indeholde et kolon! Fordi et kolon nu altid betegner et navneområder og markerer et præfiks!

I dag skelnes derfor mellem **to slags** XML navne:

1. Et localName som aldrig må indeholde et kolon – og som simpelthen er navnet efter et eventuelt kolon
2. Et qualifiedName som maksimalt kan indeholde et kolon således at det som står efter dette kolon er lig med localName!

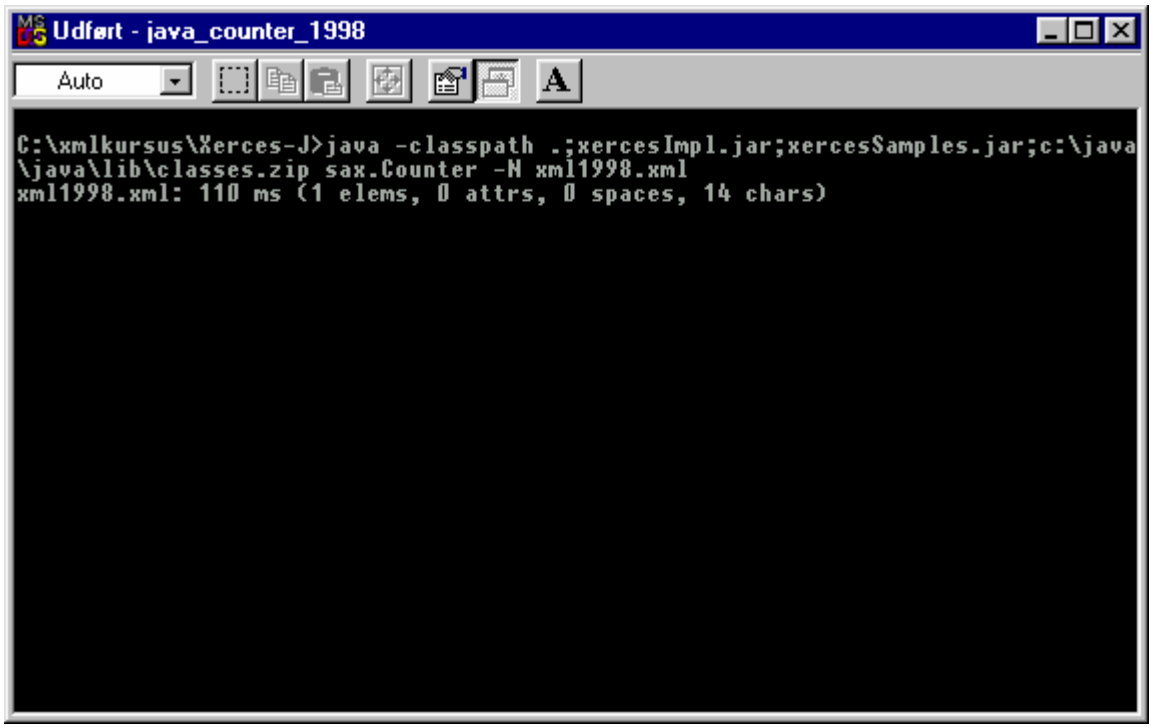
I praksis betyder det at selve navnet på et element eller en attribut aldrig må indeholde et kolon!

Hvis vi prøver at parse ovennævnte XML dokument med x:y:z i Xerces parseren fås også en fejl:




Vi kan se at Xerces opfatter x:y som elementets navn og at den forventer en attribut eller et slut mærke!

I Xerces parseren kan vi midlertid slå namespaces til og fra. Vi kan slå namespaces fra ved en parameter –N og derved kører vi Xerces parseren som om den var en gammeldags 1998 parser!:



Nu er der **intet** galt med vores XML dokument! I nogle tilfælde – med 'gamle' XML dokumenter – er det **nødvendigt** at kunne slå namespaces til og fra for at kunne parse og læse dokumentet!

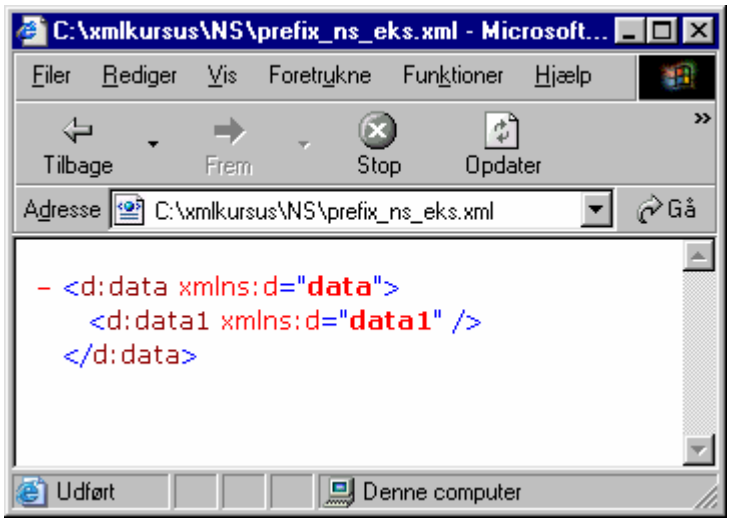
Xerces parseren kan også helt fjerne (ignorere) præfikserne med en parameter –NP.

De nye parsere – som er af typen Namespace **Aware** parser – ved altså hvad de skal gøre når de støder på f. eks.:

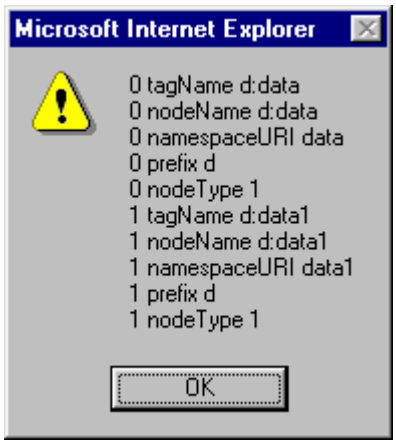
<p:person>

Parseren kan straks se at elementets **navn** er person og at p er en **præfiks**. Det næste som parseren gør er så at kontrollere at der i XML dokumentet - hidtil - har været en xmlns med dette præfiks! Parseren finder altid det **nærmeste** præfiks fordi det **samme** præfiks kan være koblet til forskellige namespaces! Parseren starter i selve elementet – går til parent o.s.v. og leder efter det rette xmlns! Hvis parseren **ikke** kan finde det korrekte xmlns – opgiver den og melder fejl!

Præfikser og navneområder kan **omdefineres** længere nede i træet. Dette er f. eks. fuldt ud gyldigt:




Hvis analyserer dette XML dokument fås dette:



## *Standard namespaces:*

Der findes et meget stort antal forud definerede namespaces som kan anvendes og som forstås af de **fleste** XML processorer.

Eksempler her på er:

1. xmlns:xsd="http://www.w3.org/2001/XMLSchema" (som skal anføres i et XML skema)
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" (som skal anføres i en XML fil som er en instantiering af et skema, dvs som anvender et skema)

Disse standarder er altså helt nødvendige for at få skemaer og stylesheets osv til at fungere efter hensigten!

Vi kan indsætte et **standard** - forud defineret - namespace på denne måde:

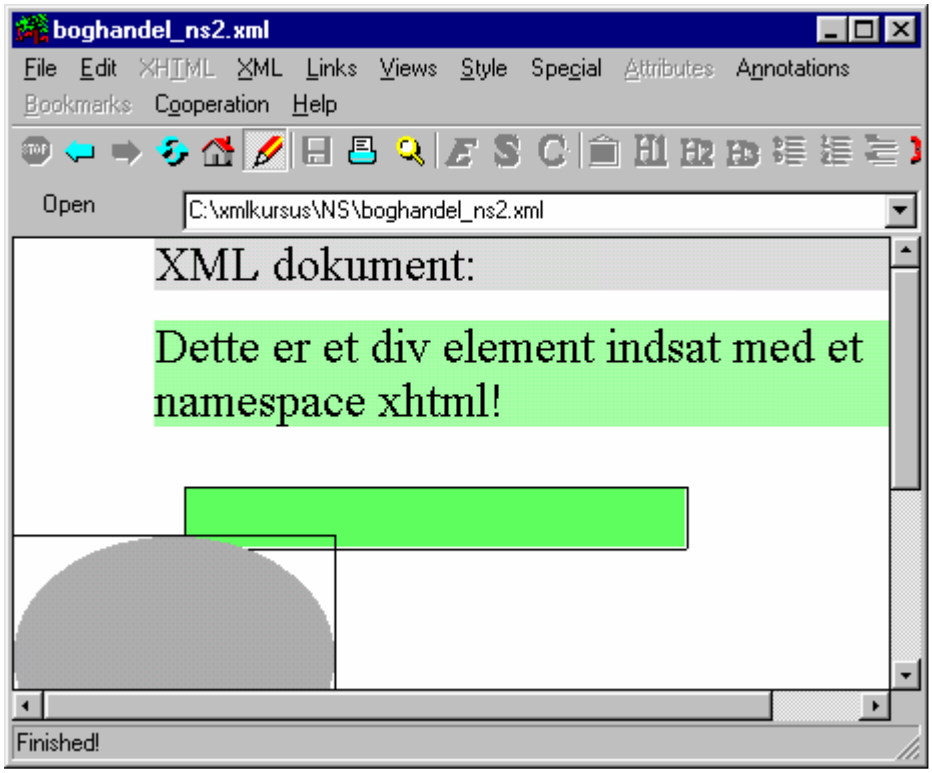
---

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

<data
xmlns:forfatter="http://tempuri.org/forfatterliste.1.1"
xmlns:bog="http://tempuri.org/bogliste.1.1"

xmlns:HTML="http://www.w3.org/Profiles/XHTML-transitional"
xmlns:s="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
>
<HTML:div style="font-size:18pt;background-color:#dedede">XML dokument:</HTML:div>
<HTML:div style="font-size:18pt;background-color:#aaffaa">
Dette er et div element indsat med et namespace xhtml!
</HTML:div>
<s:svg>
  <rect x="15" y="15" style="border:1px solid black" fill="#66ff66" width="250" height="30"/>
  <ellipse fill="#afafaf" style="border:1px solid black" cx="10" cy="100" rx="80" ry="60"></ellipse>
</s:svg>
<forfatter:forfatterliste
xlink:type="simple" xlink:href="information.txt"
>
                    <forfatter:forfatter>
                    <id>f12345</id>
                    </forfatter:forfatter>
                    <forfatter:forfatter>
                    <id>f45345</id>
                    </forfatter:forfatter>
                    <forfatter:forfatter>
                    <id>f89345</id>
                    </forfatter:forfatter>
</forfatter:forfatterliste>
```

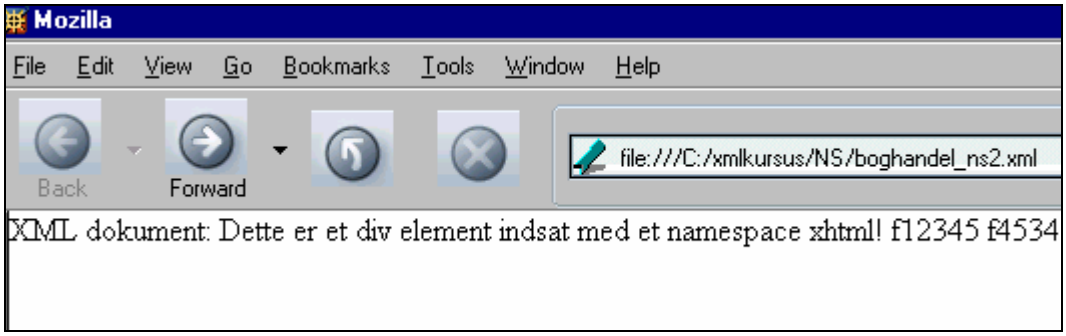
Vi erklærer her nogle standard namespaces og kan så anvende elementer og objekter fra disse namespaces!

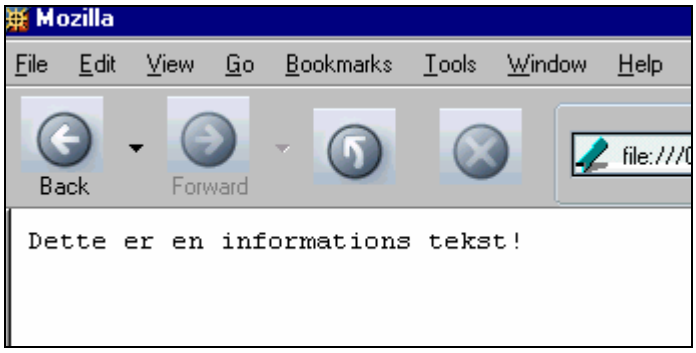
NB: Det er helt afgørende at et standard namespace staves på den korrekte måde – en lille slåfejl og intet virker! Teksten i xmlns er simpelhen et **flag** eller en kode til XML parseren! Parseren forstår kun een bestemt kode!

Vi kan anvende et div element fra html (xhtml) og indsætte en svg-grafik – hvis ellers browseren kan klare det! I browseren Amaya som kan downloades fra http://www.w3.org kan svg-grafik vises:



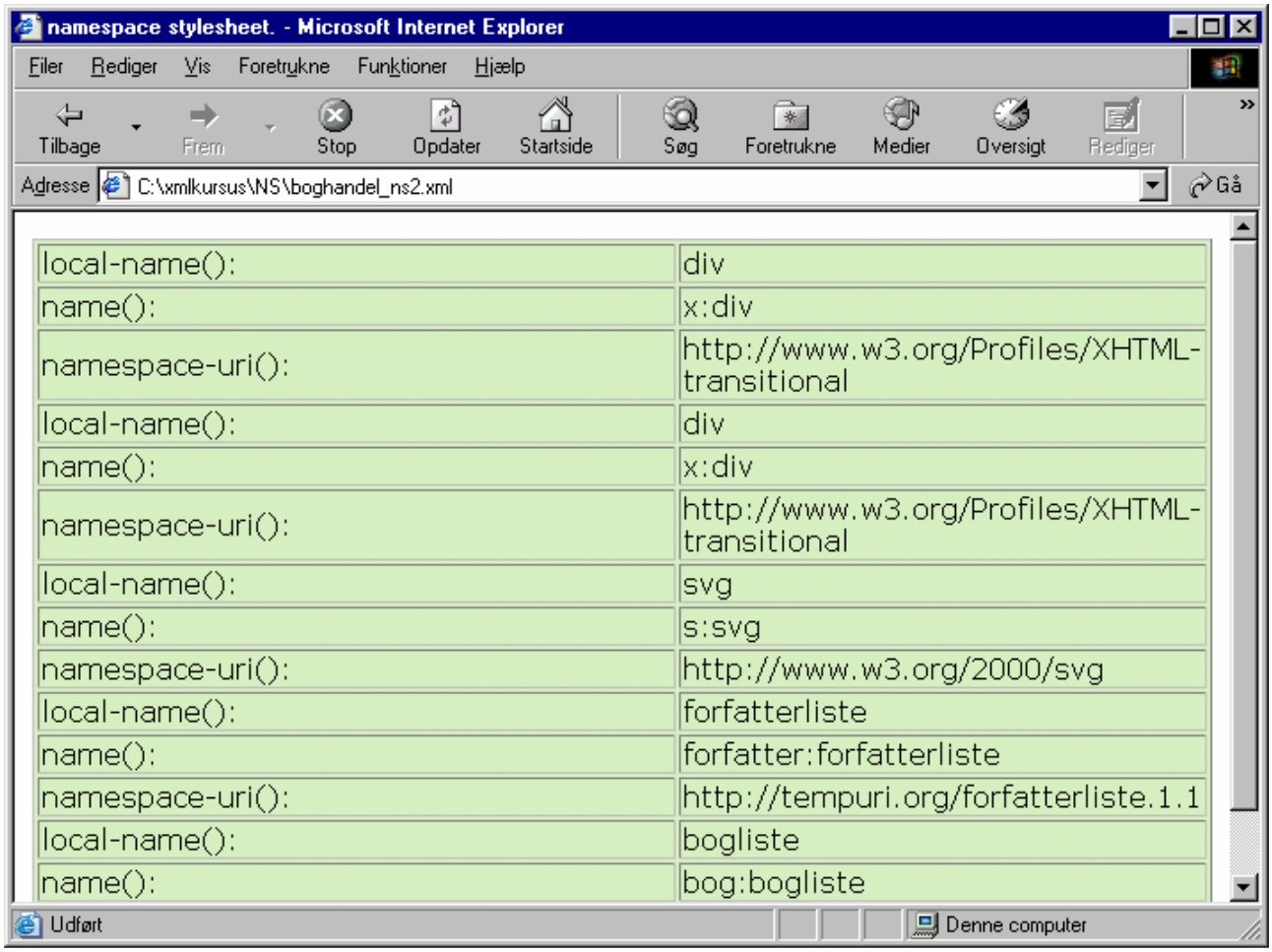

XLinks er en XML teknologi som svarer til <a> elementer i HTML. Hvis XML dokumentet vises i visse browsere kan der klikkes på **linket** og 'information.txt' vises. Dette eksempel viser browseren **Mozilla** (tilsvarende i Netscape, men xlink virker ikke i MS Internet Explorer):






Det væsentlige er lige her ikke de konkrete eksempler men at man kan **importere** et namespace og derefter anvende objekter fra dette namespace eller denne XML applikation!

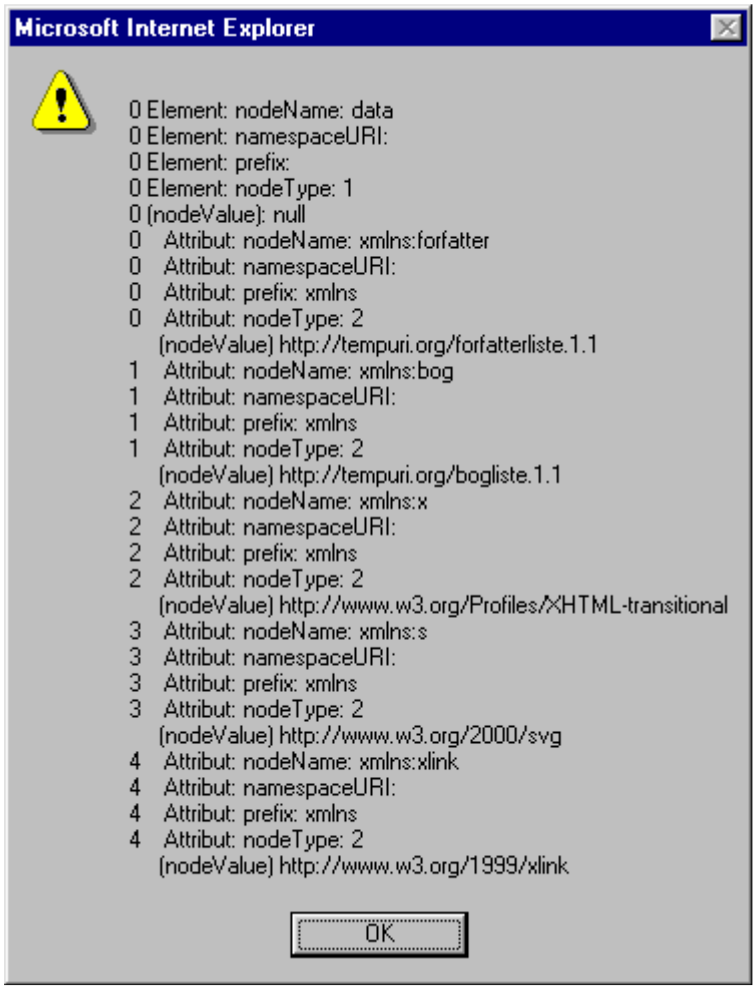
Med et XSLT stylesheet kan vi analysere vores XML dokument således:




| | |
|---|---|
| local-name(): | div |
| name(): | x:div |
| namespace-uri(): | http://www.w3.org/Profiles/XHTML-transitional |
| local-name(): | div |
| name(): | x:div |
| namespace-uri(): | http://www.w3.org/Profiles/XHTML-transitional |
| local-name(): | svg |
| name(): | s:svg |
| namespace-uri(): | http://www.w3.org/2000/svg |
| local-name(): | forfatterliste |
| name(): | forfatter:forfatterliste |
| namespace-uri(): | http://tempuri.org/forfatterliste.1.1 |
| local-name(): | bogliste |
| name(): | bog:bogliste |

Hvis vi vil finde attributterne i rod elementet data ses følgende billede, hvor vi kan se de forskellige attributter i root og deres værdier.

Vi kan også se at elementet data IKKE i sig selv befinder sig i et namespace selv om det erklærer adskillige namespaces! (node type 1 er et element, node type 2 er en attribut):

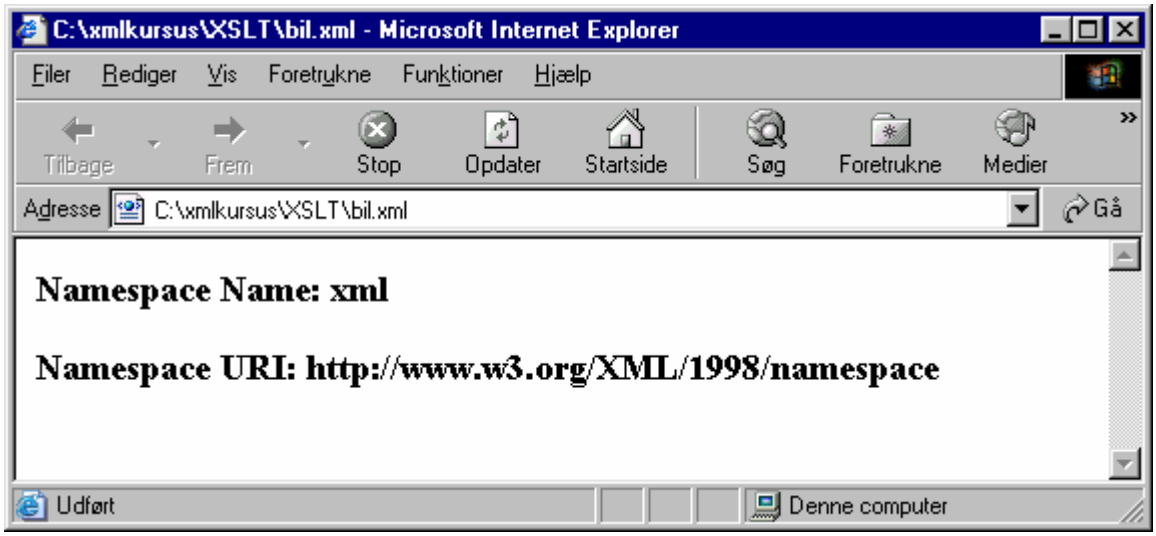



## XML navneområde:

Alle XML dokumenter har et skjult namespace som ikke er skrevet og heller ikke normalt kan ses i et XML dokument. Hvis kan vi tage et hvilket som helst tilfældigt dokument:

```
<data>
<bil>
<producent>
<navn>
</navn>
<id>
</id>
</producent>
<bil_id></bil_id>
<ind_pris></ind_pris>
<datering></datering>
</bil>
</data>
```

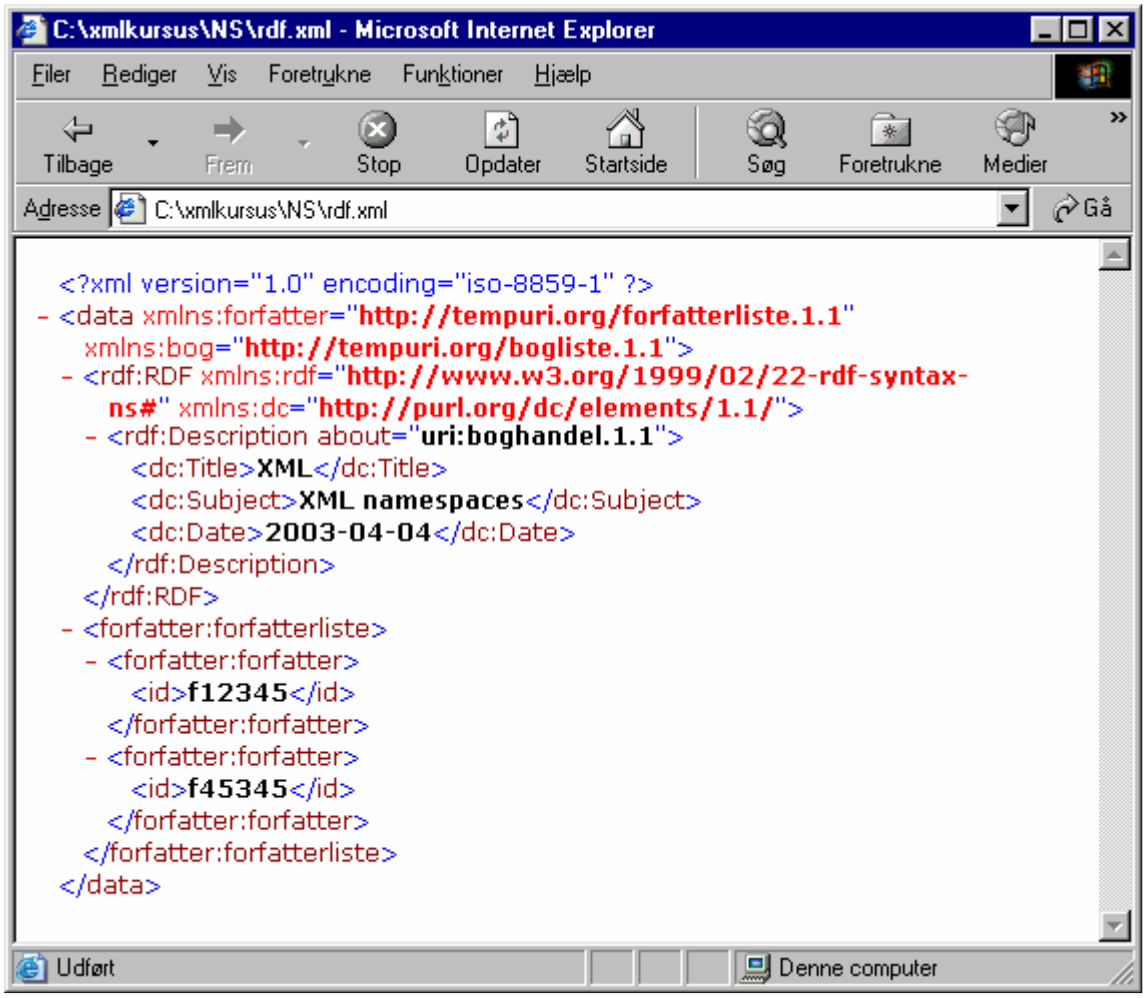
Dette bil dokument har tydeligvis ingen namespaces! Men hvis vi checker namespaces f. eks. med en metode fra XML style sheets – som vi vil vende tilbage til – opdager vi at dokumentet har et standard skjult navneområde – som alle dokumenter har:




Dette navneområde lyder altså xmlns:xml=”http://www.w3.org/XML/1998/namespace”!

## *RDF Resource Description Framework:*

Et eksempel på et standard namespace er **RDF** som er et skema som kan bruges til at definere data om det aktuelle XML dokument svarende til **meta** oplysninger i HTML. Som et supplement bruges et andet namespace kaldet **Dublin Core**. Disse meta data kan lægges ind i en XML fil på denne måde:




Disse **meta** data kan så bruges af **søge** maskiner og robotter på Internettet. Der findes et stort antal elementer i dc – dublin core som f. eks: Title, Creator, Description, Publisher, Contributor, Type, Language og Rights. Disse kan anvendes som i eksemplet.

## *Attributter:*

Attributter – modsat sub elementer - arver ikke fra en **default** namespace. Attributter er nemlig **ikke** 'børn' af elementet! Hvis man anvender metoden i DOM getParent() – jvf. senere - på en attribut får man værdien null!

Vi kan sætte en ny attribut 'opdateret' på vores forfatterliste således:

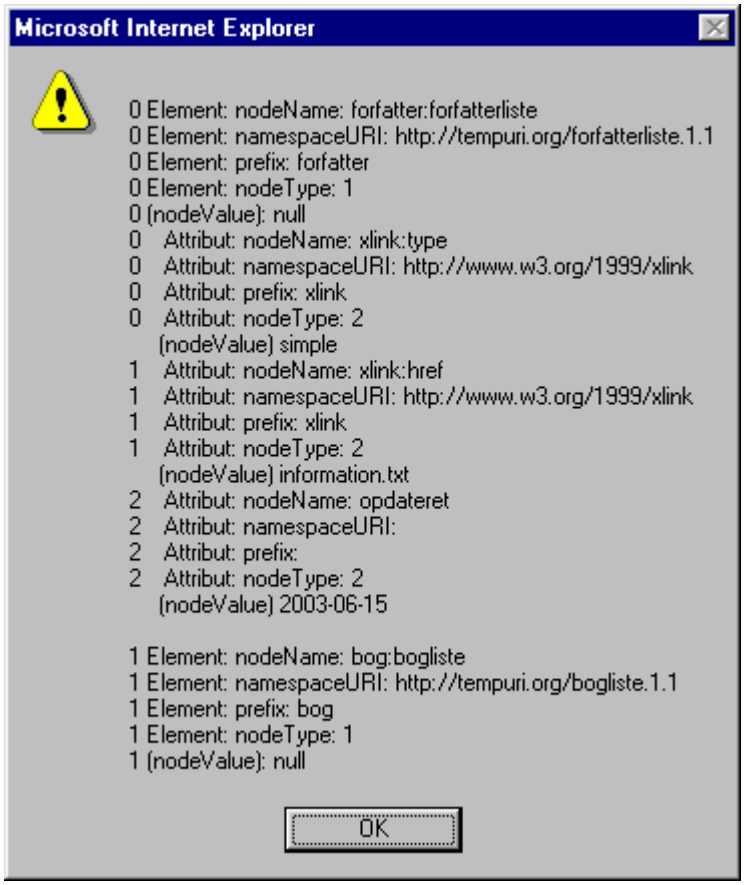
---

<forfatter:forfatterliste
xlink:type="simple"
xlink:href="information.txt"
opdateret="2003-06-15"

>

---

Hvis vi nu viser hvilke attributter der er definerede i objektet forfatter:forfatterliste ses dette:



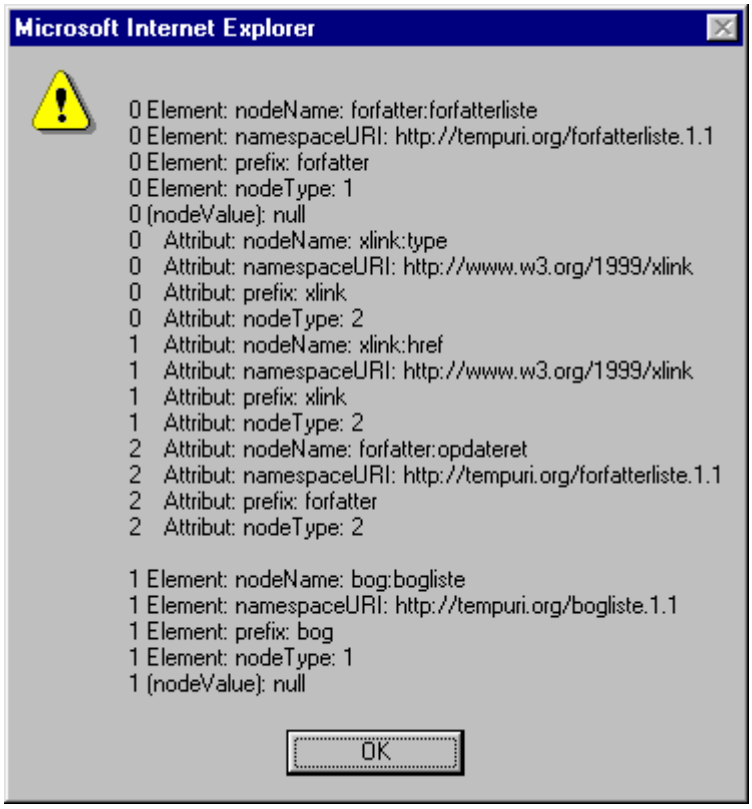

Vi kan se, at forfatterliste har 3 attributter (0, 1 og 2). De 2 første er i et namespace, men vores egen **opdateret** attribut er **ikke** i et namespace (er ikke kvalificeret) – selv om elementet er i et namespace! Denne attribut skal altså kvalificeres **specifikt** for at komme ind under et namespace – hvis vi bruger et default namespace! F. eks. på denne måde:

---

<forfatter:forfatterliste
xlink:type="simple"
xlink:href="information.txt"
forfatter:opdateret="2003-06-15"
>

---

Resultatet er nu helt anderledes:




Nu ses det, at 'opdateret' - attribut nummer 2 - er kommet ind under et namespace nemlig det samme som elementet!

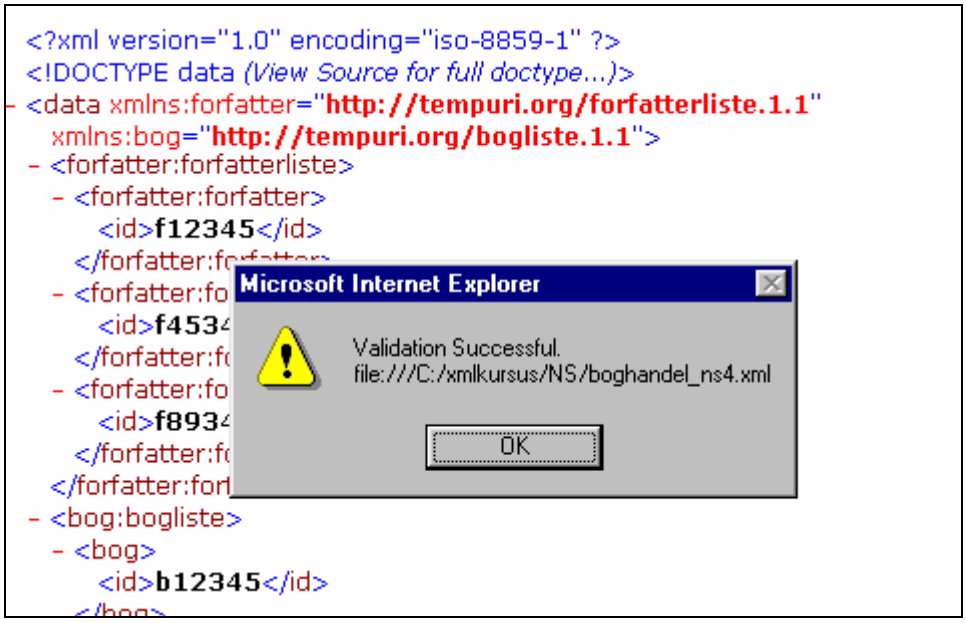
## *DTD skemaer og namespaces:*

DTD skemaer som stammer fra SGML og ikke fra XML 'forstår' ikke namespaces – modsat XSD skemaer.

Vi kan skrive et inline DTD skema (et inline DTD subset) på denne måde:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE data [
<!ELEMENT data (forfatter:forfatterliste,bog:bogliste)>
<!ELEMENT forfatter:forfatterliste (forfatter:forfatter)*>
<!ELEMENT forfatter:forfatter (id)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT bog:bogliste (bog)*>
<!ELEMENT bog (id)>
<!ATTLIST data xmlns:forfatter CDATA #REQUIRED xmlns:bog CDATA #REQUIRED>
]>
<data
xmlns:forfatter="http://tempuri.org/forfatterliste.1.1"
xmlns:bog="http://tempuri.org/bogliste.1.1"
>
...
```

Vi kan se at det er nødvendigt at **erklære** de fulde **kvalificerede** navne på elementerne – ellers kan XML dokumentet ikke valideres med DTD skemaet.




På samme måde er det nødvendigt i et **CSS** – Cascading Style Sheet - dokument at anføre de **fulde** navne – ikke kun selve elementets lokale navn. Vi kan fx skrive et lille CSS dokument således:

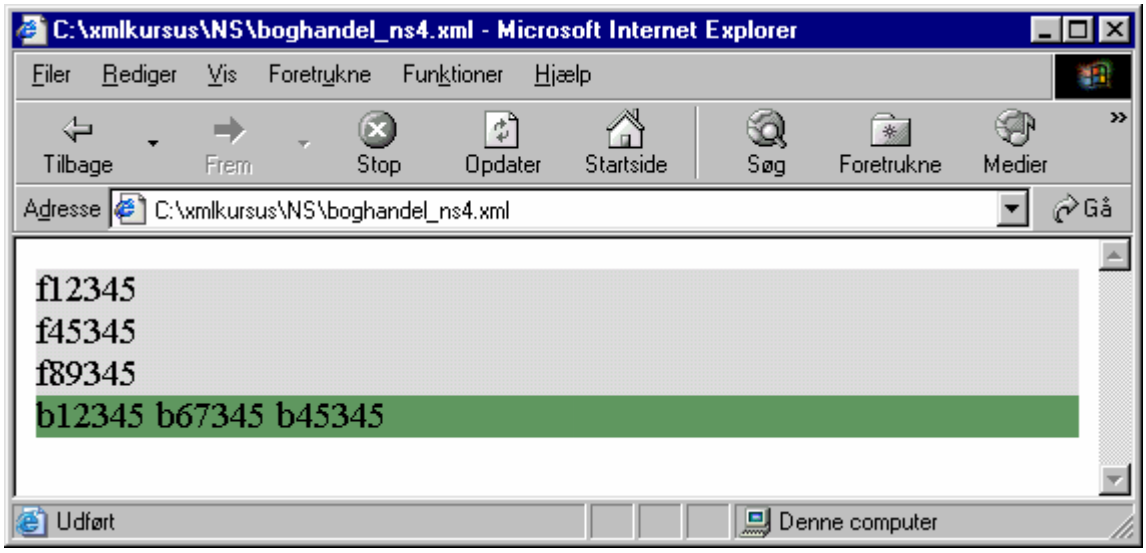
---

```
data {font-size:14pt}
forfatter\:forfatter {display:block;background-color:#dedede}
bog\:bogliste {display:block;background-color:#669966}
```

---

Som det ses er det nødvendigt at **escape** dvs at sætte \ foran kolonnet! Dette stylesheet kaldet boghandel.css sættes på XML filen således:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet href="boghandel.css" type="text/css"?>
...
```


C:\xmlkursus\NS\boghandel_ns4.xml - Microsoft Internet Explorer
Filer Rediger Vis Foretrukne Funktioner Hjælp
Tilbage
Frem
Stop
Opdater
Startside
Søg
Foretrukne
Medier
Adresse
C:\xmlkursus\NS\boghandel_ns4.xml
Gå
f12345
f45345
f89345
b12345 b67345 b45345
Udført
Denne computer