XSLT Stylesheets: 2
Internet Explorer: 3
Amaya: 3
Mozilla: 3
Netscape Navigator: 4
Et XSLT stylesheet for telefonliste: 5
Standard regler for visninger i XSLT: 6
Vis alle elementer: 8
Vis XSL Output: 12
En transformation til tekst formatet: 13
En genvej til en tekst transformation via RTF: 16
Bogstavelige (literale) tegn-data elementer: 19
Et stylesheet med en DTD: 20
De direkte elementer i XSLT under xsl:stylesheet: 21
xsl:include: 21
xsl:import 23
Eksterne og interne parametre: 23
En stylesheet processor: 26
Et stylesheet som anvender extension objekter: 30
Generelt om extension objekter: 32
xsl:fallback: 33
C Sharp eksempel med extension objekt: 33
Redigere i et dokument med XSLT: 35
Look Up Table med et stylesheet: 38
Metoden document(): 40
Hente enslydende poster ind fra mange dokumenter: 42
Ressourcer på Internettet og document() funktionen: 46
Sorter data: 47
Script og formattering af tal: 49
To slags scripts: 52
Rekursive funktioner eller templater i XSLT (tokenizer): 53
Transformere alle attributter til elementer: 55
Alle noder i dokumentet: 57
At indsætte nye id attributter med XSLT: 58
Transformation af ID og IDREF til HTML: 60
Anvendelsen af xsl:key: 63
Debugging af XSLT: 65
Nummerede lister: 68
Integration af XSLT og Cascading Stylesheets: 73
Eksempel på transformation: XML -> RTF dokument: 75
Eksempel på transformation: XML -> Recordset: 77
Eksempel på transformation: XML -> SVG: 78
Eksempel på transformation: SVG -> VML: 80
Eksempel på transformation: XML -> XML: 84
Eksempel på transformation: XML -> XSD: 88
Transformationer uden filer: 90
Transformationer med SAXON XSL processoren: 91
Skriv til flere output dokumenter med SAXON: 95

Gruppering af data med SAXON: 97
SAXON og Java: 100

XSLT Stylesheets:

XSL (eXtensibe Stylesheet Language) består af to dele nemlig:

- **XSLT** (T står for transformation) som bruges til at **transformere** et XML dokument til et andet format f. eks. til en HTML fil, en anden XML fil, SVG grafik, et PDF eller RTF dokument eller en almindelig tekst-fil. Transformationen er en slags **maske** der bliver lagt ned over XML dokumentet. XSL's hovedformål var oprindeligt at udgøre et alternativ til CSS – Cascading Stylesheets. Men i det lange løb bruges XSLT mere til transformationer fra et format til et andet! CSS egner sig fint til narrative dokumenter (rapporter, artikler) men kan ikke sortere eller databehandle som XSLT kan.
- XSL:**FO** (formatting objects) som svarer lidt til CSS og hvis formål er at **formatere** XML filen på en bestemt måde.

Vi skal i dette afsnit se på **XSLT**.

Vi vil starte med denne basis XML fil:

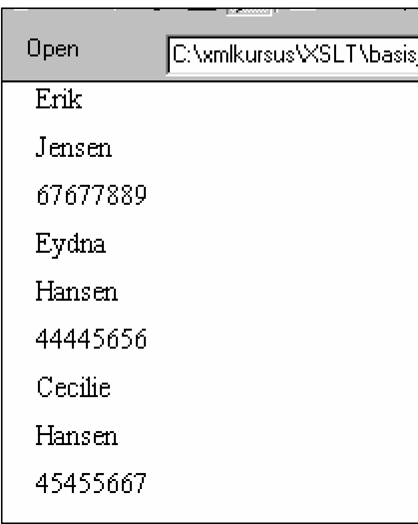
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
<?xml-stylesheet href=".xsl" type="text/xsl" ?>
-->
<telefonliste opdateret="2003-10-1">
<person type="arbejde">
<fornavn>Erik</fornavn>
<efternavn>Jensen</efternavn>
<telefon>67677889</telefon>
</person>

<person type="privat">
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>

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

Dokumentet består af 3 personer og parent telefonliste har en attribut. Dokumentet har ikke nogen CSS eller XML stylesheet. (Er kommenteret ud). Dette dokument vises ret forskelligt i forskellige browsere:

Internet Explorer:

```
Adresse  C:\xmlkursus\XSLT\basis_telefonliste.xml

<?xml version="1.0" encoding="iso-8859-1" ?>
<!-- <?xml-stylesheet href="kun_node.xsl" type="text/xsl" ?>
  -->
- <telefonliste opdateret="2003-10-1">
  - <person type="arbejde">
      <fornavn>Erik</fornavn>
      <efternavn>Jensen</efternavn>
      <telefon>67677889</telefon>
    </person>
  - <person type="privat">
      <fornavn>Eydna</fornavn>
      <efternavn>Hansen</efternavn>
      <telefon>44445656</telefon>
    </person>
  - <person type="privat">
      <fornavn>Cecilie</fornavn>
      <efternavn>Hansen</efternavn>
```

Amaya:

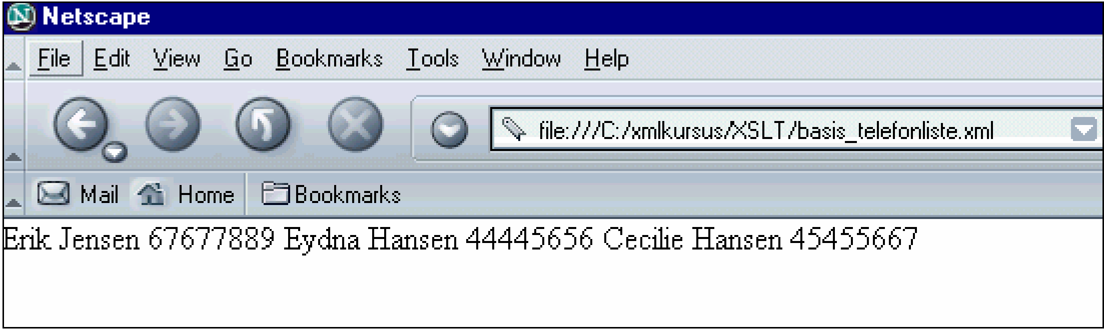


Mozilla:

This XML file does not appear to have any style information associated with it. The document tree is sho

```
- <!--
   <?xml-stylesheet href="kun_node.xsl" type="text/xsl" ?>
  -->
- <telefonliste opdateret="2003-10-1">
  - <person type="arbejde">
      <fornavn>Erik</fornavn>
      <efternavn>Jensen</efternavn>
      <telefon>67677889</telefon>
    </person>
  - <person type="privat">
      <fornavn>Eydna</fornavn>
      <efternavn>Hansen</efternavn>
      <telefon>44445656</telefon>
    </person>
```

Netscape Navigator:



Alle disse browsere anvender i virkeligheden et **default** (standard) stylesheet for at vise XML dokumentet – blot forskellige stylesheets! Netscape viser tekst noderne og kun dem – uden linje skift – men med mellemrum! Amaya viser også tekst noderne (ikke attributterne eller XML erklæringen eller kommentaren) men med linje skift efter hver tekst node. Mozilla og Internet Explorer viser XML dokumentet formatteret så det kan udvides og sammenklappes – Mozilla viser dog ikke XML erklæringen.

Et XML stylesheet er noget helt andet end en CSS fil fordi det fuldstændigt kan transformere, ændre, sortere et XML dokument. CSS og XSLT kan ikke sammenlignes også fordi XSLT ikke direkte kan formatere et XML dokument. For at gøre det suppleres stylesheets oftest med elementer fra CSS og HTML som vi skal se. Men det at transformere et XML dokument til en **web** side eller et HTML dokument er kun een af de mange ting som XSLT kan gøre! Ofte er vi interesseret i at få produceret en tekst fil med indholdet af et XML dokument eller dele heraf! En hel anden opgave.

Et XSLT stylesheet for telefonliste:

Vi kan i første omgang skrive dette stylesheet:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

</xsl:template>

</xsl:stylesheet>
```

I praksis skrives alle XML stylesheets sådan, at de erklærer dette **namespace** (skal staves korrekt!) med en præfiks som så anvendes i hele stylesheet'et. Som det ses er et stylesheet også et XML dokument! D.v.s. det skal være **velformet** og kan anvendes akkurat som enhver anden XML fil. Man kan f. eks. bearbejde, skrive eller læse et stylesheet med **DOM** metoder!

En **template** er i XSLT en **regel**, **skabelon** eller **funktion** som kaldes og som udfører en eller anden **transformation** af XML dokumentets elementer eller attributter. Templaten opretter nogle 'plads holdere' som siden udfyldes af data fra XML dokumentet! Templaten 'ekspanderes'.

En 'template' (skabelon, format) er en form som XML indholdes hældes ned i – som en kageform. At transformere et XML dokument er at køre det igennem en form!

En template har en **match** som angiver den kontekst eller den **node** (element, attribut osv) som funktionen gælder. En template fungerer lige som en 'regular expression' som i en tekst behandling søger igennem teksten for at finde et bestemt mønster, bestemte tegn!

'/' betyder at vi går ind i den kontekst som hedder root eller her elementet **telefonliste** som er roden. Det som står inden i templaten er det som skal ske med alle elementer osv i roden – altså i vores eksempel skal der intet ske med roden og dens subtræ!

En match fungerer på den måde at XML dokumentet læses **forfra** tegn for tegn/ node for node og hver gang parseren støder på en match udfører den det som templaten foreskriver! En match svarer til en match i en **'regular expression'** eller til **Søg** i et tekst behandlings program.

Hvis vi har skrevet 3 templater for forskellige elementer bliver de altså **aktiverede** i den rækkefølge som de optræder i XML dokumentet – **ikke** i den rækkefølge de optræder i XSLT dokumentet!

I Internet **Explorer** og i **Mozilla** vises dette også 'korrekt' ved at der vises en tom side! I de to andre browsere vises det samme billede som før – hvor dokumentet ikke havde noget stylesheet!

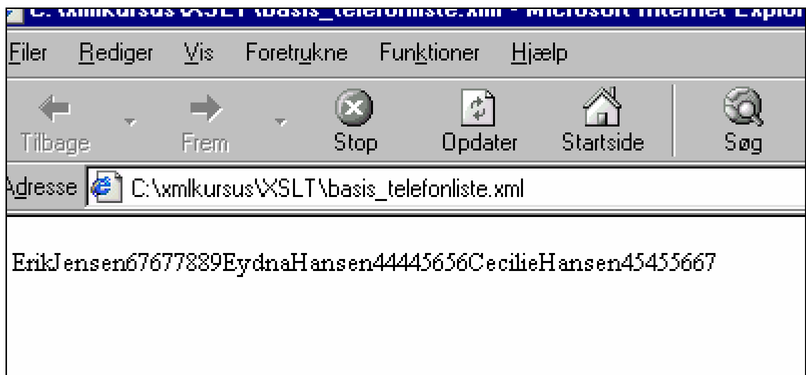
Vi kan nu tilknytte XML dokumentet dette stylesheet således:

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

```
<?xml-stylesheet href="basis1.xsl" type="text/xsl" ?>

<telefonliste opdateret="2003-10-1">
<person type="arbejde">
<fornavn>Erik</fornavn>
...
```

Som vi skal se **behøver** man ikke at anvende en processing instruction for at anvende et stylesheet på et XML dokument! Man kan **programmatisk** style XML dokumentet – f. eks. i et script - eller anvende en XSL transformator som msxsl.exe direkte!

Hvis vi nu – for eksperimentets skyld – sletter vores template så vi har et tomt stylesheet viser Internet Explorer dette billede:



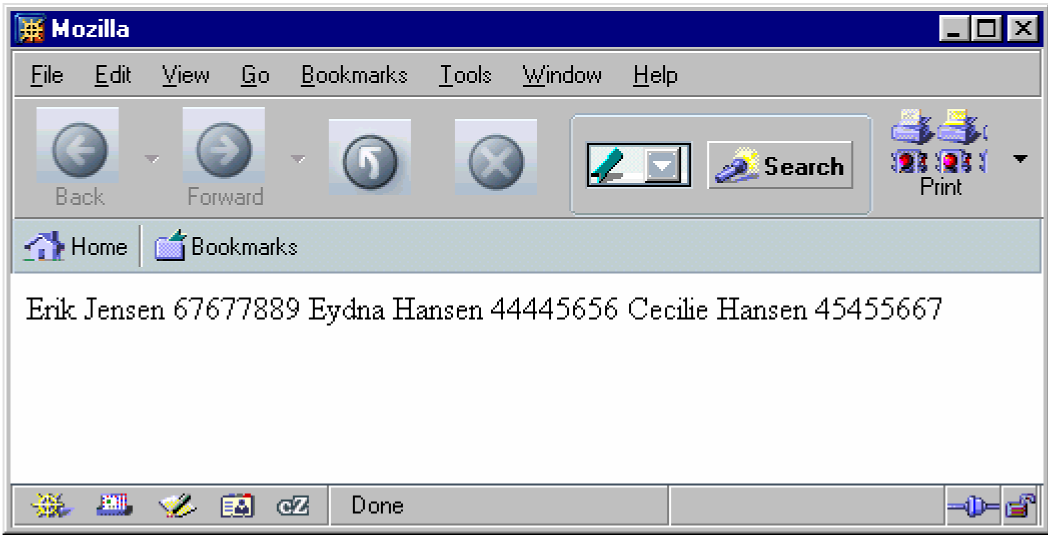
Standard regler for visninger i XSLT:

Internet Explorer vælger altså – i virkeligheden meget logisk - at viser **tekstnoderne** – uden mellemrum og uden linjeskift - i XML dokumentet (**ikke** attributterne osv)! IE anvender her de såkaldte standard eller **default regler** i XSLT som angiver at tekst noders værdi skal vises, men **ikke** attributternes. Vi skal senere se på hvordan disse **standard** regler i XSLT kan udnyttes til at oversætte et XML dokument til RTF formatet (Rich Text formatet)! I Mozilla vises stadig en tom side og i de to øvrige stadig den tidligere side!

XSLT's **default** regler kan vi illustrere ved dette stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
        <xsl:apply-templates />
</xsl:template>

</xsl:stylesheet>
```

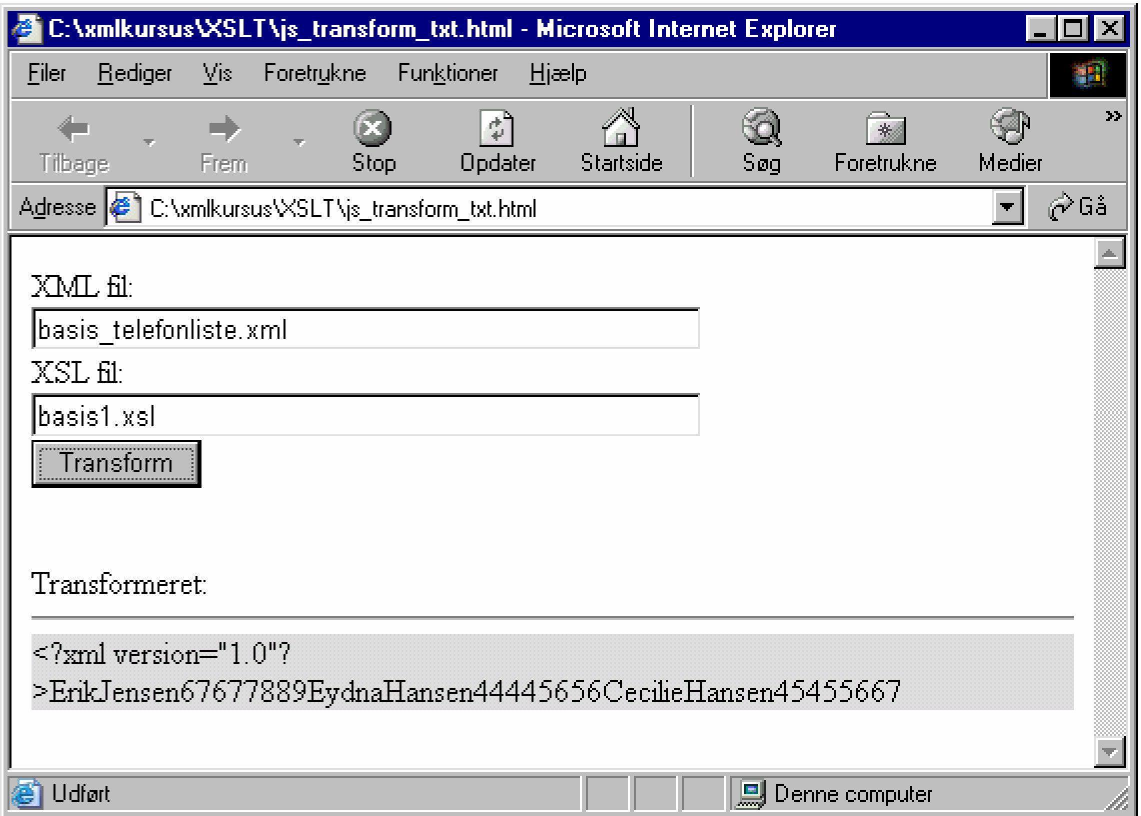
Her starter vi i roden og herfra kaldes uspecificeret **alle** templater! Men der findes jo slet **ikke** nogen andre templater i dokumentet! Dette bevirker at processoren kalder alle **standard** reglerne for visningen dvs. alle elementers tekst noder vises – men **ikke** attributternes! Dette skyldes at i DOM er en attribut **ikke** child af elementet – **ikke** et sub element!



Vi kan se at tekst noderne nu vises med mellemrum imellem!

Når vi anvender et stylesheets sker der det at systemets XSLT **processor** læser dels XSL dels XML filen og opbygger to **træ** strukturer og producerer en output tekst – ofte et output **træ** - af en eller anden slags. Vi skal straks se på hvordan man kan bestemme det output **format** som processoren producerer!

Vi skal senere se på hvad der egentligt produceres i denne transformation, men her er et billede af den fil/tekst som i dette tilfælde produceres af XSLT processoren:



Som det ses produceres en (slags) XML fil – men **uden** root og **uden** elementer – blot med en streng som indeholder tekstnoderne i elementerne!

Det er altså nødvendigt at skrive mere konkrete stylesheets for at sikre at XML dokumentet vises på en rimelig måde – i dette tilfælde – som web side.

Vis alle elementer:

Vi kan skrive et stylesheet som viser elementernes værdier i en HTML tabel således:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" encoding="iso-8859-1" omit-xml-declaration="yes" indent="yes" />

<xsl:template match="/telefonliste">
<html>
<head><title>XSLT Stylesheet.</title></head>
<body>
<table border="1">
<xsl:apply-templates select="person" />
</table>
</body>
</html>
</xsl:template>
```

```
<xsl:template match="person">
<tr>
<td><xsl:value-of select="efternavn" /></td>
<td><xsl:value-of select="fornavn" /></td>
<td><xsl:value-of select="telefon" /></td>
</tr>

</xsl:template>


</xsl:stylesheet>
```

Dette stylesheet giver i Internet Explorer og i de øvrige browsere (dog ikke Amaya!) dette resultat:

Det er vigtigt at forstå, at XSLT processoren læser XML filen i 'document **order'** – dybde først - altså altid **forfra –** node efter node **-** men at vi **kan** bestemme **hvilke** data som skal vises og på hvilken **måde**. Her har vi som et meget lille eksempel ombyttet rækkefølgen af fornavn og efternavn.

Vi har nu defineret **output** formatet som html og dette får visse konsekvenser – fx indsættes en **meta** tag i head. XSLT processoren prøver at producere et HTML dokument som er mest muligt acceptabelt for de fleste browsere!

output formatet kan sættes til **html**, **xml** eller **text**. Vi skal senere se på hvad det betyder i praksis. Det er vigtigt at man gør sig klart **hvad** det er man ønsker at transformere XML dokumentet til!

Ellers er metoden nu at vi grundlæggende indsætter html mærker eller tags i vores stylesheet. Når vi indsætter **'literal'** tekst (altså almindelig tekst) bliver det sendt **direkte** videre til **output-træet**! Det vi skriver bliver simpelt hen skrevet i output objektet!

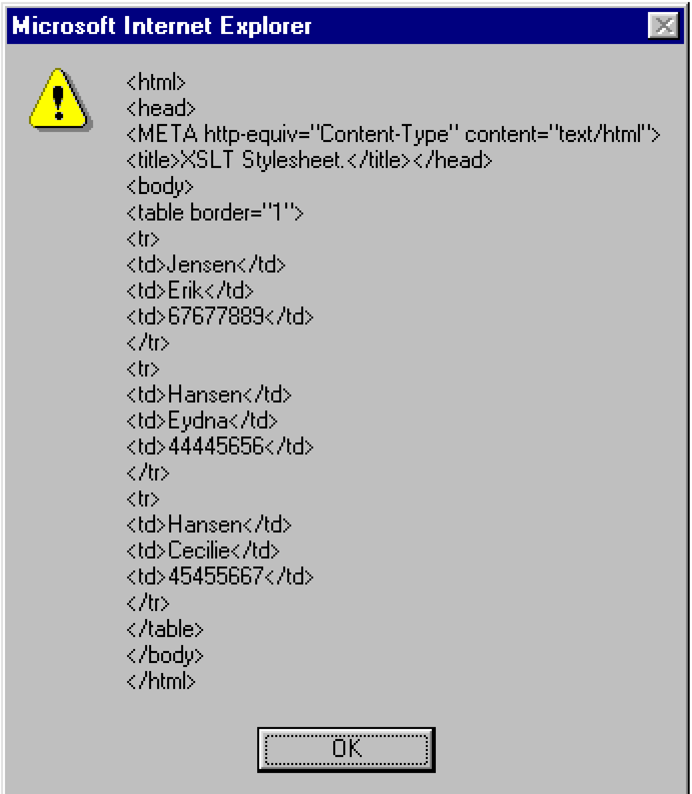
Vi sætter her en start **kontekst** til /telefonliste – dvs. vi går ned i det element som hedder telefonliste. Inden i templaten **kalder** vi så den templat som gælder elementet person. I den sidste templat person er **konteksten** altså skiftet!

Derfor kan vi blot skrive fornavn og ikke telefonliste/person/fornavn!

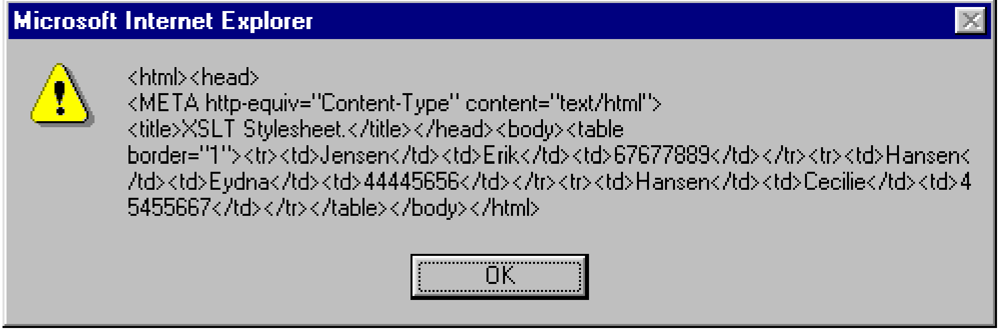
value-of select bruges – normalt - til at hente **tekstnoden** ('value') for det pågældende element! Men man kan også bruge value-of select til at indsætte en literal tekst i output træet eller til at beregne et resultat som f. eks. 44 * 44!

apply-templates kan sammenlignes med at **kalde** en funktion eller sub rutine i et programmeringssprog. Når templaten er færdig vendes **tilbage** til det sted hvor templaten blev kaldt!! Derfor skal vi skrive slut mærker som fx </table> **EFTER** at templaten (funktionen) er blevet kaldt og har returneret!

XSLT processoren producerer i virkeligheden denne tekst:



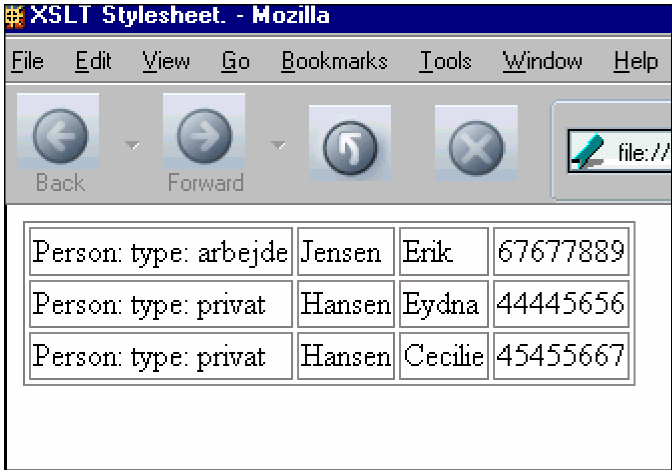
Hvis vi havde valgt **indent**="no" i xsl:**output** elementet var resultatet blevet mindre læseligt, men selvfølgeligt grundlæggende det samme:



I **CSS** kan vi ikke – direkte – vise attributter fra XML dokumentet, men det kan nemt lade sig gøre i XSLT. Vi kan indsætte en ekstra celle i vores templat således:

```
<xsl:template match="person">
<tr>
<td>Person: type: <xsl:value-of select="@type" /></td>
<td><xsl:value-of select="efternavn" /></td>
<td><xsl:value-of select="fornavn" /></td>
<td><xsl:value-of select="telefon" /></td>
</tr>
</xsl:template>
```

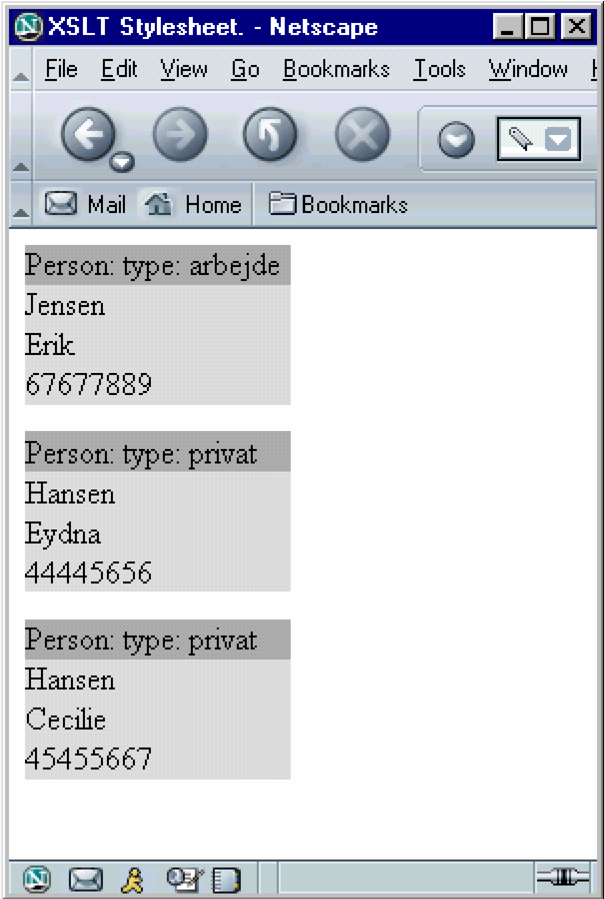
Formlen @attributnavn henter så værdien af den pågældende attribut:



I stedet for at formattere HTML filen som en tabel kan data vises med HTML mærket <**div**> og templaten kan skrives på denne måde (alt andet uændret):

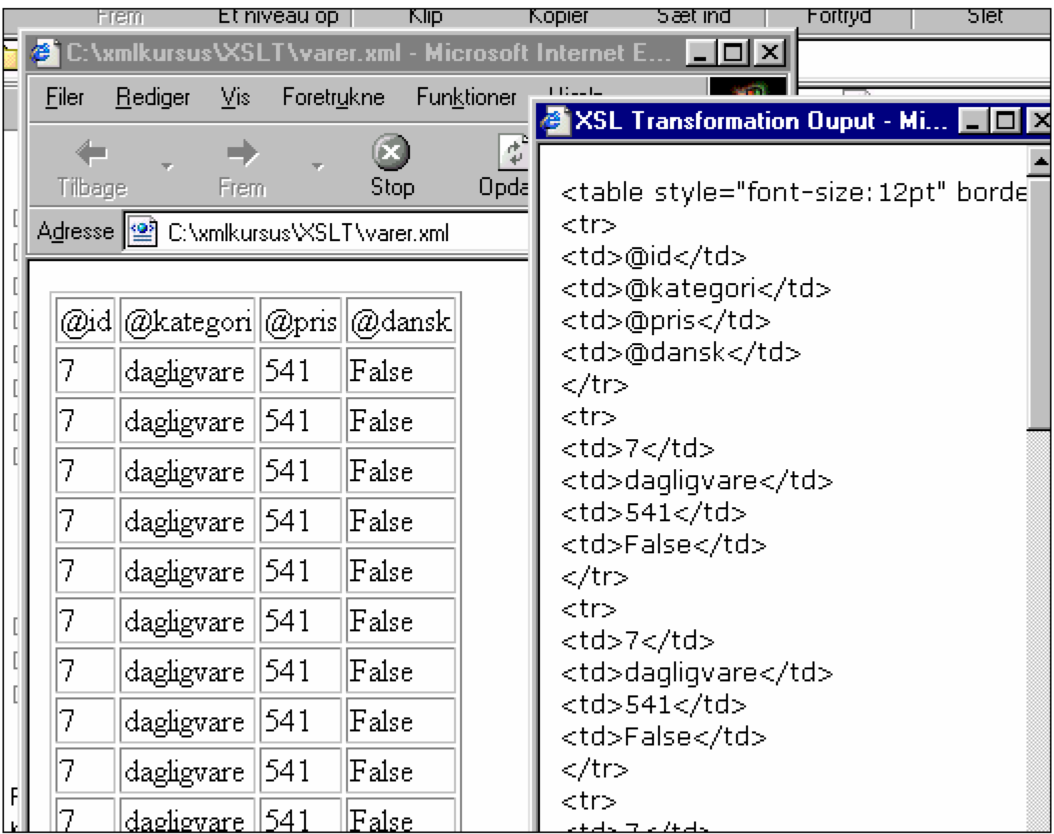
```
<xsl:template match="person">
<div style="width:100pt;background-color:#aeaeae">
Person: type: <xsl:value-of select="@type" />
</div>
<div style="width:100pt;background-color:#dedede">
<xsl:value-of select="efternavn" />
</div>
<div style="width:100pt;background-color:#dedede">
<xsl:value-of select="fornavn" />
</div>
<div style="margin-bottom:10pt;width:100pt;background-color:#dedede">
<xsl:value-of select="telefon" />
</div>
</xsl:template>
```

Dette er en meget almindelig form for formatering af XML **data centriske** dokumenter. Data vises nu i hver sin <**div**> som har fået en style. Resultatet er nogenlunde sådan:



Vis XSL Output:

Man kan fra http://msdn.microsoft.com downloade to værktøjer som gør at man – efter en transformation – kan højre klikke på siden i Internet Explorer og få vist XSL Output altså den tekst som Microsoft XSLT processoren producerer. Værktøjerne – der også omfatter et værktøj til at validere XML dokumenter – kan hentes som IEXMLTLS eller XSL Viewer Tools. Det drejer sig om to inf filer (bl. a msxmlvw.inf) som skal installeres på Windows (højre klik på filerne og vælg Installer!).



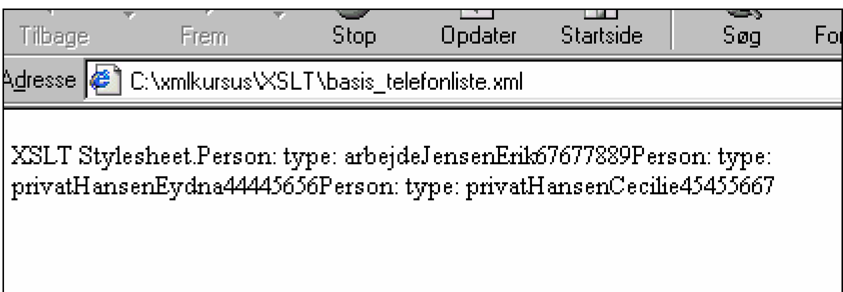
En transformation til tekst formatet:

Ofte er vi interesserede i at omsætte en XML fil – eller dele heraf - til tekst. Dette kan gøres på følgende måde:

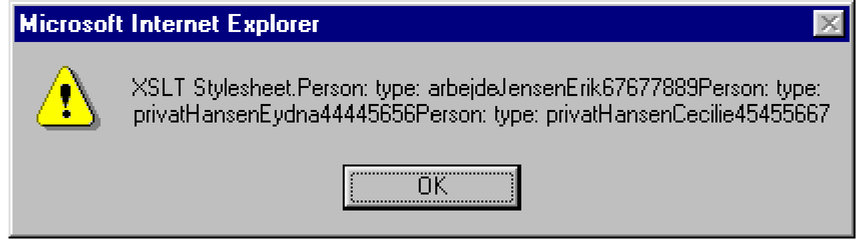
Hvis vi blot – som det **eneste**! - omdefinerer vores xsl:**output** til dette:

<xsl:output method="**text**" **encoding**="iso-8859-1" omit-xml-declaration="yes" indent="yes" />

Får vi dette resultat i Internet Explorer:

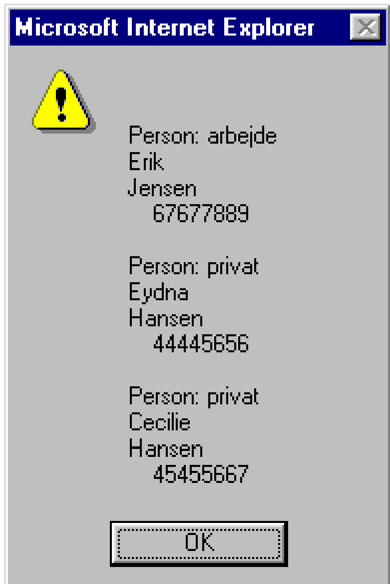


Selve det at vi ændrer en attribut i xsl:output skaber altså et **helt** andet output 'træ'! Den producerede tekst ser nu således ud:



Alle vores HTML mærker osv. er **ignoreret** af XSLT processoren!! Teksten bliver ikke indented og tekstnoderne blot udskrevet uden mellemrum!

Vi er interesseret i noget i stil med dette:



For at opnå dette er vi nødt til at skrive vores stylesheet om f. eks. på denne måde:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="text" encoding="iso-8859-1" omit-xml-declaration="yes" indent="yes" />

<xsl:variable name="linjeskift">
<xsl:text disable-output-escaping="yes">&#10;</xsl:text>
</xsl:variable>

<xsl:variable name="indryk">
<xsl:text disable-output-escaping="yes">&#32;&#32;&#32;&#32;</xsl:text>
</xsl:variable>

<xsl:template match="/telefonliste">
<xsl:apply-templates select="person" />
</xsl:template>

<xsl:template match="person">
<xsl:value-of select="$linjeskift" />
```

```
Person: <xsl:value-of select="@type" />
<xsl:apply-templates />
</xsl:template>

<xsl:template match="efternavn">
<xsl:value-of select="$linjeskift" />
<xsl:value-of select="." />
</xsl:template>

<xsl:template match="fornavn">
<xsl:value-of select="$linjeskift" />
<xsl:value-of select="." />
</xsl:template>

<xsl:template match="telefon">
<xsl:value-of select="$linjeskift" />
<xsl:value-of select="$indryk" />
<xsl:value-of select="." />
</xsl:template>

</xsl:stylesheet>
```

Vi kan her se flere nye XSLT elementer:

Vi har **opdelt** vores templater sådan at **hvert** element fornavn, efternavn osv får **hver** sin templat. Dette er ikke strengt nødvendigt lige her, men er ofte en god ide, som gør stylesheet'et mere fleksibelt.

Vi kalder disse sub templater blot med **apply**-templates **uspecificeret**! Når vi gør det kalder vi de relevante templater fra **kontekten** person i den **rækkefølge** de forekommer i XML dokumentet! Parseren læser **altid** objekterne (noderne) i den **rækkefølge** de står i dokumentet med algoritmen 'dybde først' – som er det vi også selv gjorde da vi skrev dokumentet!

Vi har oprettet 2 **variable** som kan **formatere** teksten: en ny linje og en indryk funktion. Begge indsætter de **ASCII/**Unicode tegn (entiteter) som betyder ny linje (nummer 10 eller hexadecimalt: #xA) og mellemrum (nummer 32 eller hex 20). Tekst kan indsættes i output dokumentet med xsl:**text** som også kan anvende en **disable**-ouput-escaping, som gør at koderne virker. **Værdien** af en variabel hentes ved at bruge:

```
<xsl:value-of select="$linjeskift" />
```

$ tegnet betegner **tekstværdien** af denne variabel modsat selve navnet på variablen! En variabel i XSLT kan rumme meget forskelligt også f. eks. en **funktion** som finder en bestemt værdi eller et stykke XML **kode**! Vi skal siden se eksempler på brug af variable.

Variable og parametre er interessante fordi de kan sættes **udefra** fx fra en XSLT processor og fordi de kan få et og det samme stylesheet til at producere helt **forskellige** output dokumenter med forskellige **parametre**!

Endeligt var vi anvendt '.' som betegner **tekstnoden** til kontekst noden d.v.s. tekst værdien af den node som vi nu befinder os i! Anvendelsen af '.' er meget almindelig i stylesheets. Tekstnoden er egentlig lig med **firstChild** eller **childNodes**(0) af kontekst noden!

Det producerede dokument i Notesblok:

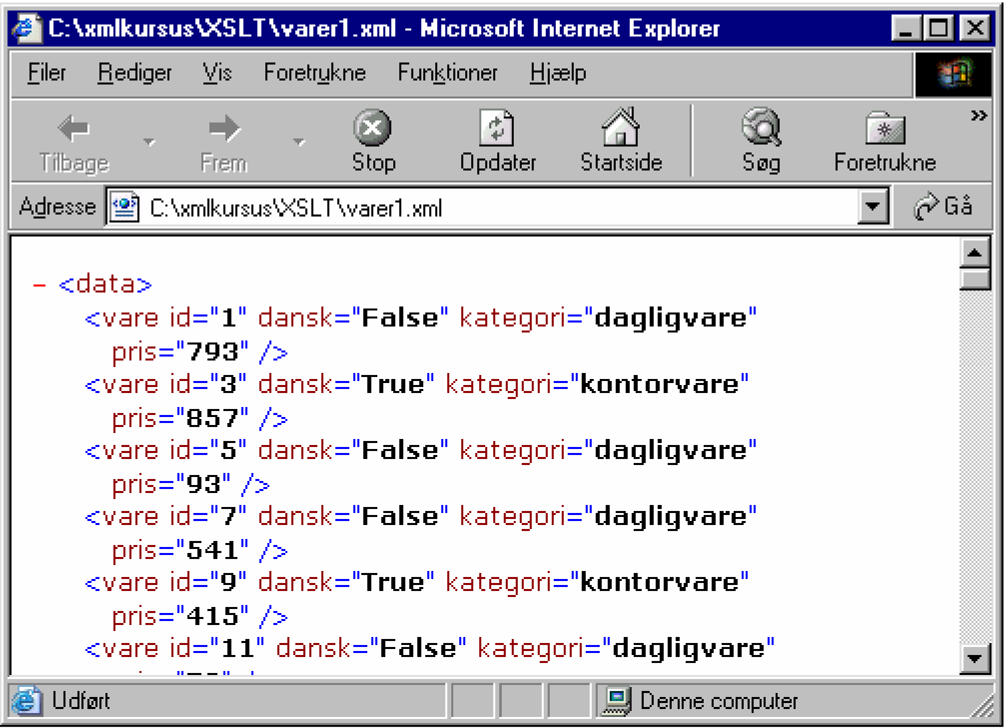
Et generelt stylesheet til tekst formatet:

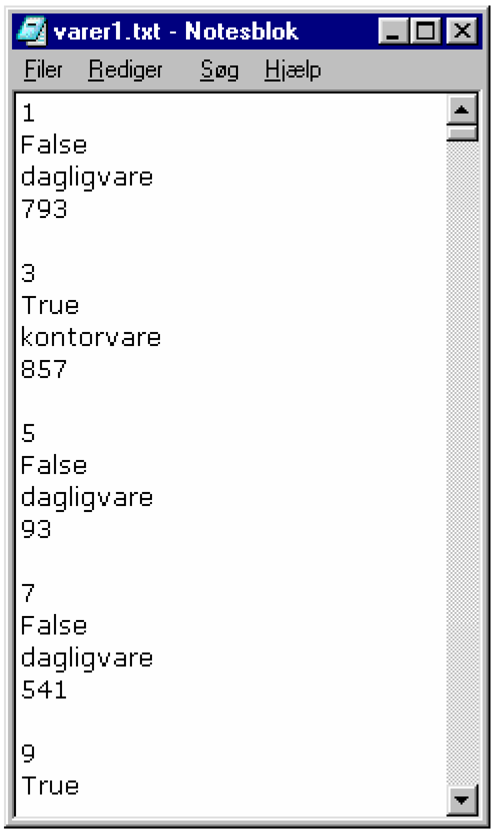
Der er mange måder at komme til nogenlunde det samme resultat. Vi kan også skrive dette generelle stylesheet der omsætter ethvert XML dokument til tekst formatet:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output method="text" encoding="iso-8859-1" />

<xsl:template match="*">
<xsl:text>
</xsl:text>
  <xsl:element name="{name()}">
    <xsl:for-each select="@*">
      <xsl:element name="{name()}">
<xsl:text>
```

```
</xsl:text>
            <xsl:value-of select="." />
            </xsl:element>
        </xsl:for-each>
        <xsl:apply-templates select="*|text()" />
    </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Vi er især – som før – interesseret i de rigtige linje skift i tekst filen! Her opnår vi dem med xsl elementet <text> som altid udskriver til output træet **præcist** hvad der står inden i <text>! Her er der et linjeskift inden i <text>! Elementet <text> kan være nødvendigt når vi **præcist** skal udskrive en bestemt tekst til output!



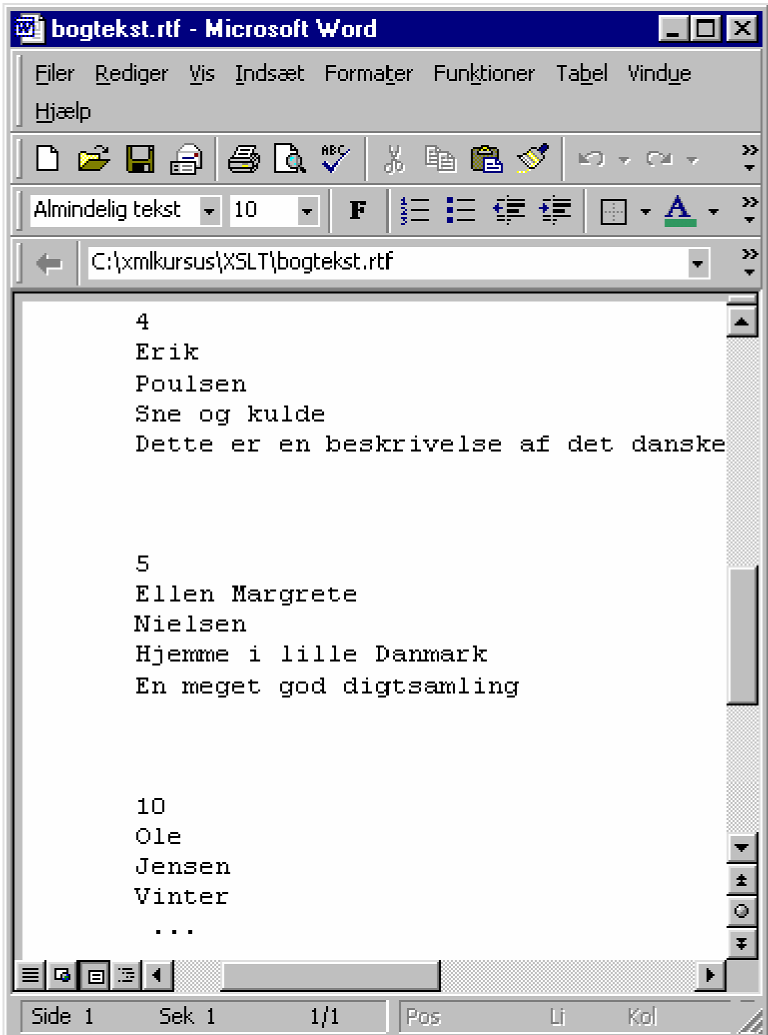


En genvej til en tekst transformation via RTF:

Man kan omsætte et XML dokument **direkte** til RTF formatet (**uden** formateringer som skriftstørrelser eller farver osv) via et **meget** simpelt stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" />

</xsl:stylesheet>
```

Ved at transformere med f. eks. **msxsl**.exe til et RTF output format – altså med f.eks. –o nytekst.rtf kan vi direkte få XML dokumentet omsat til **RTF** formatet!:



Bogstavelige (literale) tegn-data elementer:

Når vi skriver tekst eller elementer **direkte** ind i et stylesheet bliver det direkte sendt videre til output træet. Vi kan f. eks. skrive dette stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body bgcolor="#669966">
<h3>Dette er simpelt hen en literal eller bogstavelig tekst i XSLT.</h3>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Dette stylesheet vil producere det **samme** output træ **uanset** hvilket XML dokument det anvendes på! Den beskrevne html side bliver simpelt hen 100 % **defineret** i vores stylesheets skabelon!!

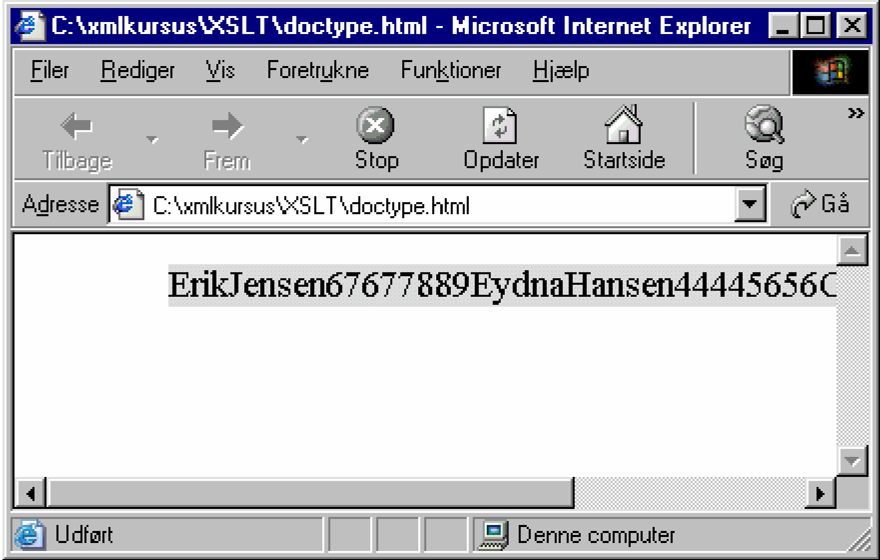
Denne metode kan anvendes til producere alle mulige forskellige **formater** – vi skal senere se eksempler herpå!

Et stylesheet med en DTD:

Eftersom et XSL dokument **også** er et XML dokument **kan** det indeholde en DOCTYPE – selv om det ikke er særligt almindeligt!

```
<!DOCTYPE xsl:stylesheet [
<!ENTITY dokument SYSTEM "telefonliste.xml">
]>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version='1.0'
>

<xsl:template match="/">
<div style="margin-left:50pt;font-size:14pt;background-color:#dedede">
&dokument;
</div>
</xsl:template>

</xsl:stylesheet>
```

Vi kan altså erklære en DOCTYPE xsl:**stylesheet** f. eks. med det formål at definere nogle **entiteter** – her en **reference** til en fil. Også dette stylesheet er et eksempel på anvendelsen af **'literaler'**! Det producerer det **samme** output træ **uanset** hvilket input dokument som anvendes!



De direkte elementer i XSLT under xsl:stylesheet:

Disse såkaldte 'Top Level' elementer er direkte børn af elementet xsl:stylesheet og har en generel interesse. De omfatter først og fremmest:

- xsl:import
- xsl:include
- xsl:strip-space
- xsl:preserve-space
- xsl:key
- xsl:variable
- xsl:param
- xsl:decimal-format
- xsl:template

xsl:include:

xsl:include kan anvendes til at inkludere et andet stylesheet som gemmer elementer, funktioner m.v. som ofte bruges og genbruges – f.eks. sidehoved og sidefod på websider:

Her er oprettet en **header** og en **footer** som så kan inkluderes og kaldes med funktionen xsl:call-template:

Vores **include**-stylesheet ser sådan ud:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" encoding="iso-8859-1" omit-xml-declaration="no" indent="yes" />

<xsl:template name="header">
<div style="font-size:16pt;font-family:Verdana;width:200pt;background-color:#aeaeae">
Dette er et resultat af en transformation i XSLT.
</div>
<div style="margin-bottom:15pt;width:200pt;background-color:#dedede">
Antal noder i dokumentet: <xsl:value-of select="count(//node())" />
</div>
</xsl:template>

<xsl:template name="footer">
<div style="margin-top:15pt;width:200pt;background-color:#dedede">
```

```
Dette dokument blev leveret af XComputerAps <a href="abc">http://www.xcomputeraps.dk</a>
</div>
</xsl:template>

</xsl:stylesheet>
```

Vores hoved stylesheet kan så bruge dette andet stylesheet på denne måde:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:include href="include_header.xsl" />
<xsl:output method="html" encoding="iso-8859-1" omit-xml-declaration="no" indent="yes" />

<xsl:template match="/telefonliste">
<html>
<head><title>XSLT Stylesheet.</title></head>
<body>
<xsl:call-template name="header" />
<xsl:apply-templates select="person" />
<xsl:call-template name="footer" />
</body>
</html>
</xsl:template>
```

Man kan på denne måde nemt lave web sider som har et ensartet præg! Det stylesheet der importeres har nøjagtig samme **prioritet** som det stylesheet som importere det! I virkeligheden har vi blot med klip, kopier og sæt ind flyttet dele af vores stylesheet til et selvstændigt stylesheet! Men include er nyttig fordi den kan **modularisere** stylesheets og på den måde fremme **genbrug**! At skabe templater og stylesheets der kan genbruges igen og igen er et meget vigtigt mål i arbejdet med XSLT!

xsl:import

importerer et andet stylesheet ind i det nuværende stylesheet. xsl:import SKAL stå som det første element efter xsl:stylesheet (skal være firstChild).

Der er ret afgørende forskel på xsl:include og xsl:import!

Eksterne og interne parametre:

Man kan definere en variabel eller parameter og på den måde gøre transformationen mere fleksibel. Globale variable eller parametre (der er egentligt ingen forskel) kan sættes som Top Level elementer direkte under roden. Globale parametre kan sættes udefra f. eks. med en XSLT processor. Lokale parametre gælder i et begrænset scope f. eks. inden for en template.

Dette eksempel viser hvordan man kan søge på en 'tekst database' ved at søge på et bestemt ord. Dette ord sættes i stylesheet'et eller udefra:

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

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:param name="key" select="'XML'" />

<xsl:template match="tekster">
  <html>
        <head>
          <title>Ekstern parameter i XSLT.</title>
        </head>
        <body>
        <xsl:for-each select="//tekst[contains(.,$key)]">
                  <xsl:call-template name="tekst" />
        </xsl:for-each>
        </body>
  </html>
</xsl:template>

<xsl:template name="tekst">
   <xsl:value-of select="." />
</xsl:template>


</xsl:stylesheet>
```

Vi definerer vores parameter eller variabel key med en default værdi som den beholder HVIS ikke andet angives. Den afgørende linje i arket er følgende:

```
<xsl:for-each select="//tekst[contains(.,$key)]">
```

Her sætter vi kontekten til alle elementer som hedder tekst. Af disse henter vi så kun dem som indeholder de tegn og bogstaver som står i vores variabel: $key. Kun disse tekst elementer vises!

Vi kan bruge et XML dokument af denne type:

```
<tekster>
<tekst>
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er et XML eksempel.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
</tekst>
<tekst>
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er et XSL eksempel.

Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
```

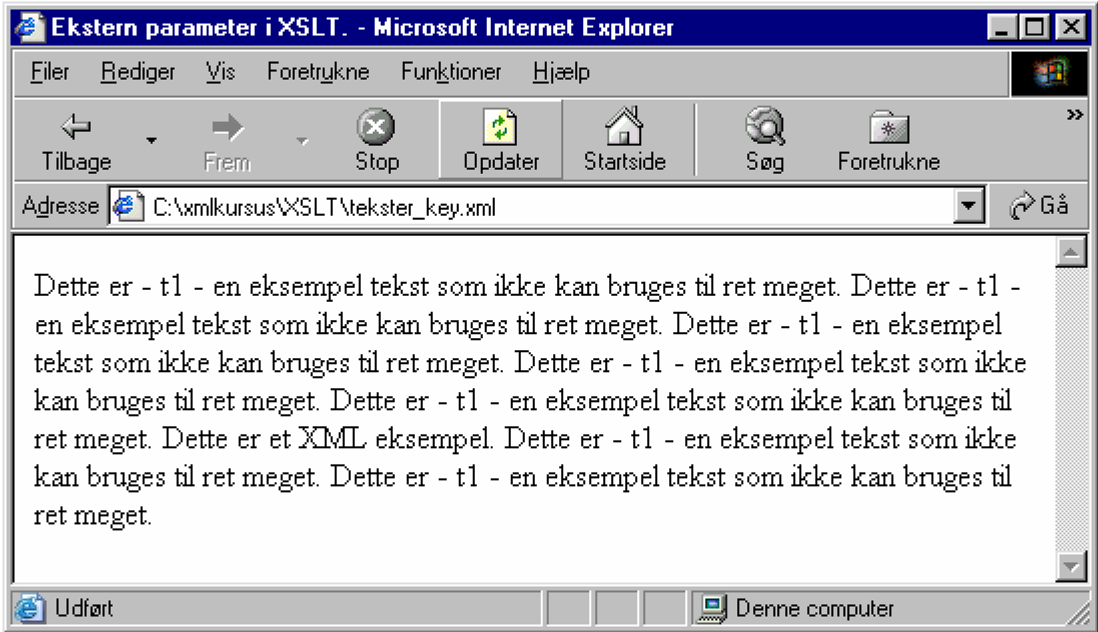
</tekst>

<tekst>
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.

</tekst>

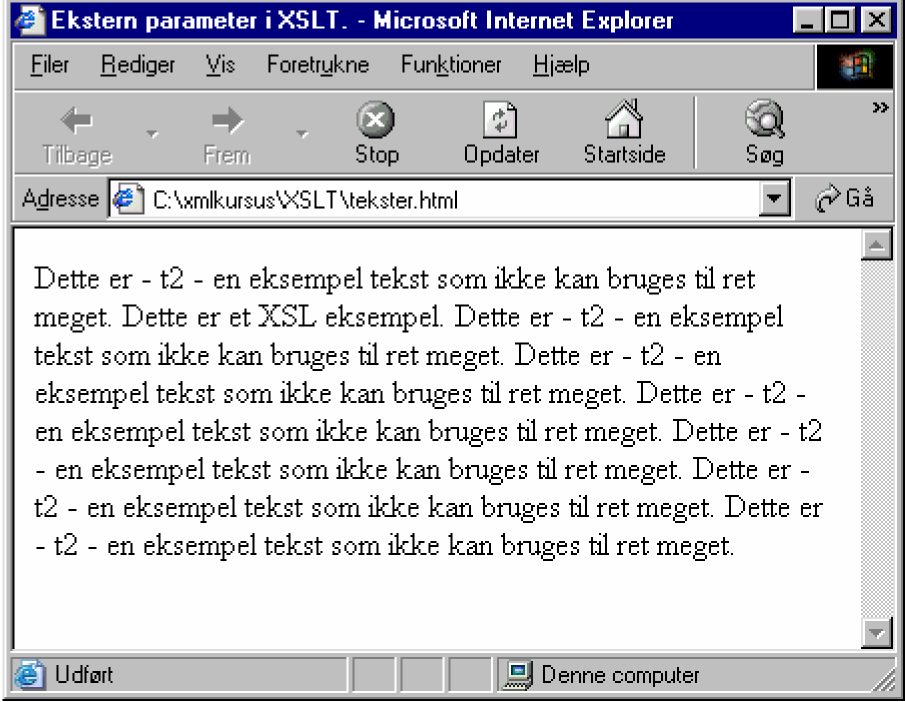
</tekster>

Hvis vi kører transformationen **uden** at sætte parameteren får vi dette resultat:



Vi kan se at processoren kun henter t1 – den tekst som indeholder ordet ’XML’!

Hvis vi sætter parameteren med en XSLT processor til f. eks. ’XSL’ får vi dette resultat:



Vi kan f. eks. sætte parameteren således i msxsl.exe:

msxsl tekster.xml –pi –o tekster.html key=XSL

Andre XSL processorer bruger stort set den samme syntaks. I det efterfølgende gives flere eksempler på variable og parametre.

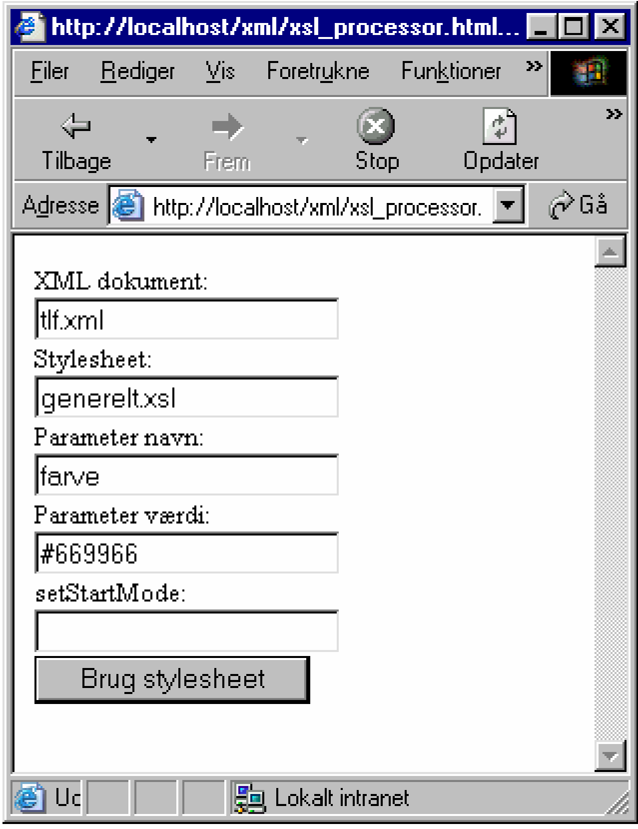
En stylesheet processor:

Man kan i MSXML instantiere en stylesheet **processor** og herigennem få en meget mere detaljeret **kontrol** over hvordan stylingen foregår.

Systemet kræver at vi instantierer dels en såkaldt **Template** dels et **FreeThreadedDOMDocument**. Det sidste er et XML dokument som kan anvendes af mange processer samtidigt - således som det sker på en server.

Der findes to slags XML dokumenter i DOM: Almindelige dokumenter er **'rental** threaded' dvs. de kan kun bruges af een proces og dør når processen stopper. **FreeThreaded** dokumenter er permanente og lever så længe applikationen lever. På en server gør dette en stor forskel.

Vi kan skrive et bruger program der ser nogen lunde sådan ud:



Vi kan skrive følgende **script** der implementerer en XSL processor og som reagerer på brugerens indtastninger:

```
<form>
XML dokument:
<br><input type="text" name="xml" id="xml" value="tlf.xml">
<br>Stylesheet:
<br><input type="text" name="xsl" id="xsl" value="generelt.xsl">
<br>Parameter navn:
<br><input type="text" name="p_name" id="p_name" value="farve">
<br>Parameter værdi:
<br><input type="text" name="p_value" id="p_value" value="#669966">
<br>setStartMode:
<br><input type="text" name="mode" id="mode">

<br><input type="submit" name="submit" id="submit" value="Brug stylesheet" onClick="processor();">


</form>

<script>
function processor(){
var xsl = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.4.0");
var doc = new ActiveXObject("Msxml2.DOMDocument.4.0");
var docout = new ActiveXObject("Msxml2.DOMDocument.4.0");
var fso = new ActiveXObject("Scripting.FileSystemObject");
var fil = fso.createTextFile("c:/xmlkursus/styletxml.html");

doc.load(document.all("xml").value);
```

```
xsl.async=false;
xsl.load("generelt.xsl");

var template=new ActiveXObject("Msxml2.XSLTemplate.4.0");
template.stylesheet=xsl;

var processor=template.createProcessor();
processor.addParameter(document.all("p_name").value,document.all("p_value").value);
processor.setStartMode(document.all("mode").value,"");
processor.input= doc;
processor.output= docout;
processor.transform();
alert(docout.xml);
fil.write(docout.xml);
fil.close();
}
</script>
```

Vores stylesheet dokument **SKAL** være af typen **FreeThreaded**!

Ved hjælp af en **Template** kan vi oprette en **processor**, som kan gemme til en stream – f. eks. Response på en server – eller til et DOM dokument eller til en fil.

Denne processor kan så konfigureres til at starte med visse **parametre** og med en bestemt **mode**. Dette kan selvfølgeligt kun fungere hvis vores stylesheet er skrevet med disse globale parametre. F. eks. man skrive et stylesheet på denne måde (dette stylesheet er helt generelt og han bruges til alle XML dokumenter!):

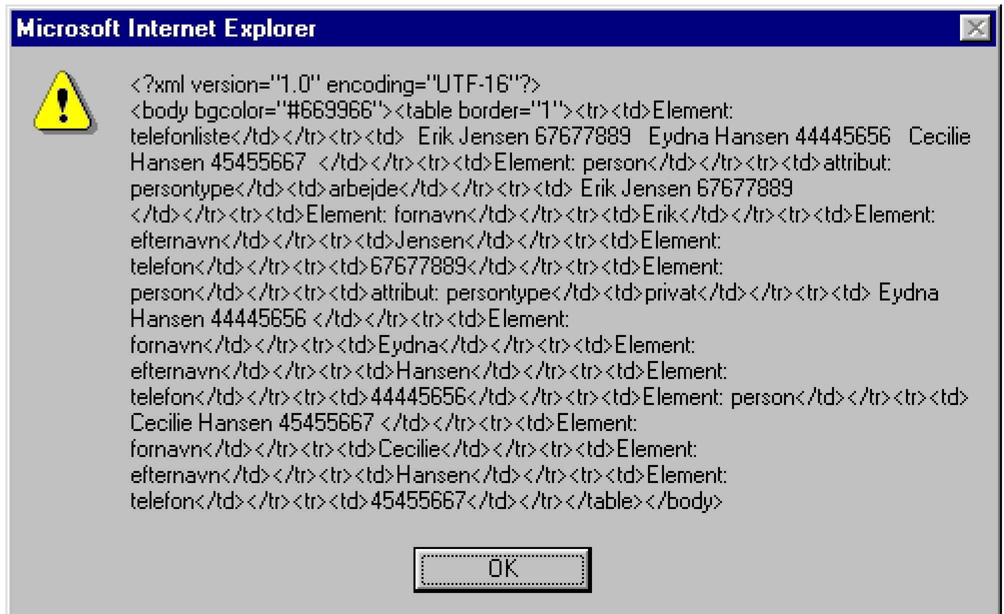
```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:param name="farve"  />

<xsl:template match="/" mode="tabel">
<body bgcolor="{$farve}">
<table border="1">
<xsl:for-each select="//*">
<tr>
<td><xsl:value-of select="concat('Element: ',name())" /></td>
</tr>
<xsl:for-each select="@*">
<tr>
<td><xsl:value-of select="concat('attribut: ',name())" /></td>
<td><xsl:value-of select="." /></td>
</tr>
</xsl:for-each>
<tr>
<td><xsl:value-of select="." /></td>
</tr>
</xsl:for-each>
</table>
</body>
</xsl:template>

<xsl:template match="/">
<xsl:for-each select="//*">
```

```
<div style="background-color:{$farve}">
<xsl:value-of select="concat('Element: ',name())" />
</div>

<xsl:for-each select="@*">
<div style="background-color:{$farve}">
<xsl:value-of select="concat('attribut: ',name())" />
</div>
<div style="background-color:{$farve}">
<xsl:value-of select="." />
</div>
</xsl:for-each>
<div>
<xsl:value-of select="." />
</div>
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Vi har her defineret en global parameter som kan sættes udefra nemlig **baggrundsfarven**. Samtidigt har vi defineret to templater med hver sin **mode**.

At anvende modes eller start modes – det kan også gøres med msxsl.exe – er meget brugt fordi stylesheet'et på den måde bliver mere **fleksibelt**. F. eks. kan man let style til forskellige modtagere – f. eks. to forskellige browsere!

Hvis vi vælger **mode** = tabel og en farve bliver resultatet:



OBS! Dokumentet bliver med denne metode **ALTID** gemt som UTF-16 eller såkaldt **'Unicode'** – hvilket kan give problemer nogle gange hvis man ikke er opmærksom på XML dokumenternes **encoding**!

Et stylesheet som anvender extension objekter:

Vi kan på næsten den samme måde – som ovenfor – inkludere et objekt i et stylesheet, et såkaldt 'extension objekt'. Ovenstående metode med parametre og start mode kan faktisk også gennemføres med msxsl.exe – **men** extension objekter kan **ikke** inkluderes ved hjælp af msxsl.exe – kun f. eks. ved et script eller i et rigtigt programmerings sprog som C eller Java!

Et **extension** objekt er et objekt som 'extender' dvs. forøger de muligheder og funktioner som allerede findes i XSLT.

Vores script kan nu se således ud:

```
<script>
processor();

function processor(){
var xsl = new ActiveXObject("Msxml2.FreeThreadedDOMDocument.4.0");
var doc = new ActiveXObject("Msxml2.DOMDocument.4.0");
var docout = new ActiveXObject("Msxml2.DOMDocument.4.0");
var fso = new ActiveXObject("Scripting.FileSystemObject");
var fil = fso.createTextFile("c:/xmlkursus/xslt/ext_obj_stylet.html");
doc.load("bog.xml");
xsl.async=false;
xsl.load("xyz.xsl");
var template=new ActiveXObject("Msxml2.XSLTemplate.4.0");
template.stylesheet=xsl;
var processor=template.createProcessor();
var obj=new ActiveXObject("Komponent.WSC.1.00");
processor.addObject(obj,"uri:Komponent");
processor.input = doc;
processor.output = docout;
processor.transform();
alert(docout.xml);
fil.write(docout.xml);
fil.close();
}
</script>
```

Det nye i forhold til før er altså at vi kan tilføje et objekt med et namespace til XSLT processoren. Vi **instantierer** et objekt (som vi selv har skrevet!) med objektets såkaldte **progid** – jvf. om lidt!

Det **samme** namespace (vi bestemmer naturligvis selv hvordan dette namespace skal lyde!) skal så anvendes igen i stylesheet'et således:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:k="uri:Komponent"
>
<xsl:output method="html" indent="yes" omit-xml-declaration="yes" />

<xsl:template match="/">
    <h1><xsl:value-of select="k:retur()" /></h1>
```

```
</xsl:template>

</xsl:stylesheet>
```

I dette tilfælde har vi valgt et rent **demo** eksempel – for det eneste dette stylesheet gør er at kalde funktionen **retur**() som findes i det namespace som hedder 'k' eller 'uri:**Komponent'**!

Funktionen retur() returnerer **kun** her en simpel streng:

Dette kan kun lade sig gøre fordi vi med scriptet har **tilføjet** objektet Komponent med det rette namespace til stylesheet'et!

Tilbage er hoved opgaven: at producere **komponenten** som i virkeligheden er en **klasse** lige som alle andre klasser f. eks. klassen 'Msxml2.DOMDocument.4.0' som vi kender!

Det interessante er at denne komponent kan defineres i **XML** på denne måde:

```
<component>
   <registration
      description="Komponent"
      progid="Komponent.WSC"
      version="1.00"
      classid="{220ca5ae-dded-4d25-b5a5-b43b71541944}"
   />
   <public>
   <method name="retur" />
   </public>
   <implements type="ASP" id="ASP" />

   <script language="JScript">
      function retur(){
      return "Dette er en Windows Script Komponent.";
      }
   </script>
</component>
```

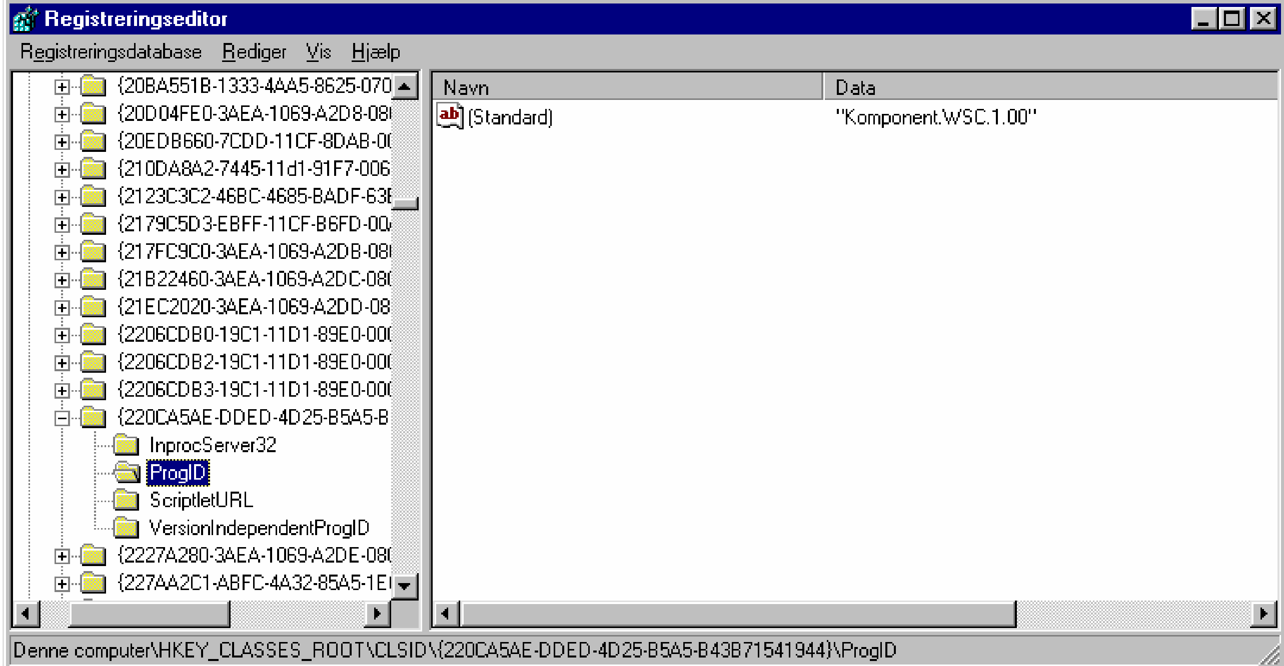
Komponenten gemmes som **'Komponent**.wsc' altså som en Windows Script komponent.

I Windows Stifinder kan man **højreklikke** på filen og vælge Registrer! (På tilsvarende måde kan man også af-registrere komponenten!). Der sker så det at komponenten (klassen) **registreres** i System registrerings databasen. Man kan også på en DOS linje bruge kommandoen:

regsvr32 Komponent.wsc

Hvilket har samme virkning.

Når komponenten er registreret kan den anvendes **overalt** i **alle** programmer! Som enhver anden **COM** eller COM+ komponent (Common Object Model). Hvis man søger i System registrerings databasen efter 'Komponent' finder vi den nye klasse:



I dette eksempel består **extension** objektet kun af en enkelt meget simpel metode, men den kunne bestå af **mange** og meget **indviklede** metoder. Der **er** altså visse fordele ved at bruge extension objekter til styling af XML dokumenter. Et objekt kan også **genbruges** mange gange og skal ikke skrives ind i selve stylesheet'et hver gang!

Der er på adressen http://msdn.microsoft.com/scripting en detaljeret redegørelse for scripting og extension objekter.

Generelt om extension objekter:

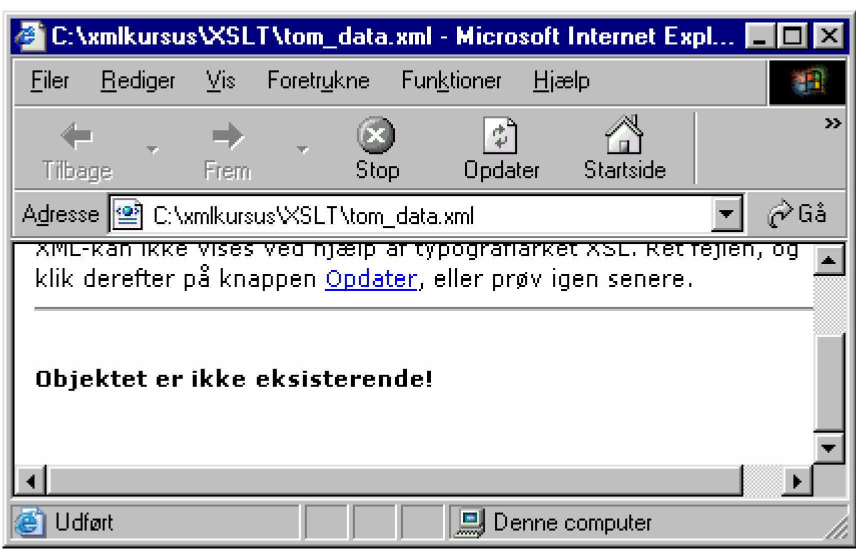
Det er meget principielt altid en tvivlsom ide at anvende extension funktioner og objekter! De vil altid gøre stylesheet'et mindre 'portable' – mindre anvendeligt på forskellige styresystemer og installationer! Ofte kan resultatet opnås med kun metoder i XSLT men lige så ofte er det nødvendigt

at bruge extension objekter! Man skal så bare være klar over at andre brugere af stylesheet'et muligvis ikke har adgang til funktionen!

xsl:fallback:

Det netop gennemgåede eksempel peger på et problem – hvad sker der hvis extension funktionen eller extension objektet ikke er til rådighed!? I XSLT er defineret et element xsl:**fallback** som er ment som en slags xsl:otherwise – altså det som skal ske hvis extension elementet ikke findes eller er til rådighed! Elementet fallback bør altid anvendes hvis der kan være tvivl om hvorvidt funktionen er til rådighed! På samme måde findes en xsl:**message** der terminerer transformationen hvis der opdages alvorlige fejl:

<xsl:if ...
<xsl:**message** terminate="yes">Objektet er ikke eksisterende.</xsl:message>



C Sharp eksempel med extension objekt:

Hvis man i stedet anvender et 'rigtigt' programmeringssprog som f. eks. **C Sharp** på **.NET** platformen bliver mulighederne næsten ubegrænsede for at indlægge objekter i et stylesheet.

Som et meget lille **eksempel** har vi her skrevet en lille C Sharp klasse **Eksempel**:

```
public class Eksempel {

        public string metode(){

                return "XComputerXYZ, Bredgade 44";

        }
}
```

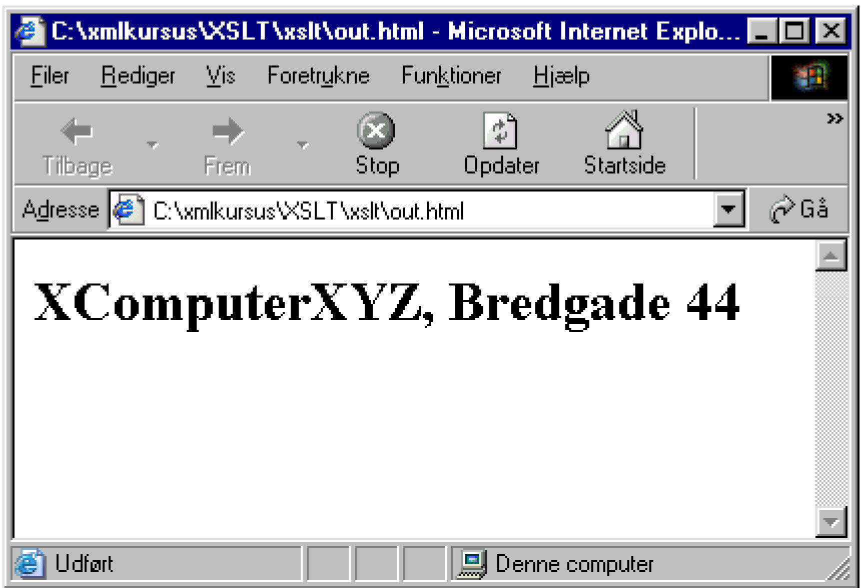
Denne klasse kompileres og gemmes dom **Eksempel**.dll. Vi kan nu skrive et rigtigt C Sharp program som indlægger dette objekt i stylesheet'et:

```
using System;
using System.Xml;
using System.Xml.Xsl;
using System.IO;

class MainClass
{
		public static void Main(string[] args)
		{
				//OBS 3 argumenter: doc, xsl, output

				try{
						XslTransform x=new XslTransform();
						XsltArgumentList objekter=new XsltArgumentList();
						Eksempel obj=new Eksempel();
						objekter.AddExtensionObject("uri:Komponent",obj);
						//xslt doc:
						x.Load(args[1]);

						XmlDocument doc=new XmlDocument();
						doc.Load(args[0]);

						TextWriter writer=new StreamWriter(args[2]);
						//doc, objekt, output
						x.Transform(doc, objekter, writer);
				}
				catch(Exception e) {
						Console.WriteLine(e.Message);
						Console.WriteLine(e.GetType().ToString());
				}
		}
}
```

Vores stylesheet er det samme som før – bortset fra den nye metode:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:k="uri:Komponent"
>
<xsl:output method="html" indent="yes" omit-xml-declaration="yes" />

<xsl:template match="/">
   <h1><xsl:value-of select="k:metode()" /></h1>
</xsl:template>

</xsl:stylesheet>
```

Vi kan nu **transformere** med brug af vores egen C Sharp klasse til dette resultat:



Efter disse principper kunne vi anvende alle klasser og objekter fra C Sharp – eller et andet programmeringssprog! F. eks. kunne vi – hvis det i nogle tilfælde kan være passende! - anvende standard klasser fra C som **MessageBox'e**:

Redigere i et dokument med XSLT:

Man kan på mange forskellige måder **redigere** i et XML dokument med XSLT og – **ikke** ændre selve XML dokumentet! – men kreere et **nyt** dokument der mere eller mindre er en kopi. Man kan påsætte en **processing** instruction, **bortsortere** visse elementer, f. eks. kun vise de første 10 poster i et meget langt dokument, indsætte en encoding erklæring og man kan indføre **namespaces** i et dokument uden namespaces!

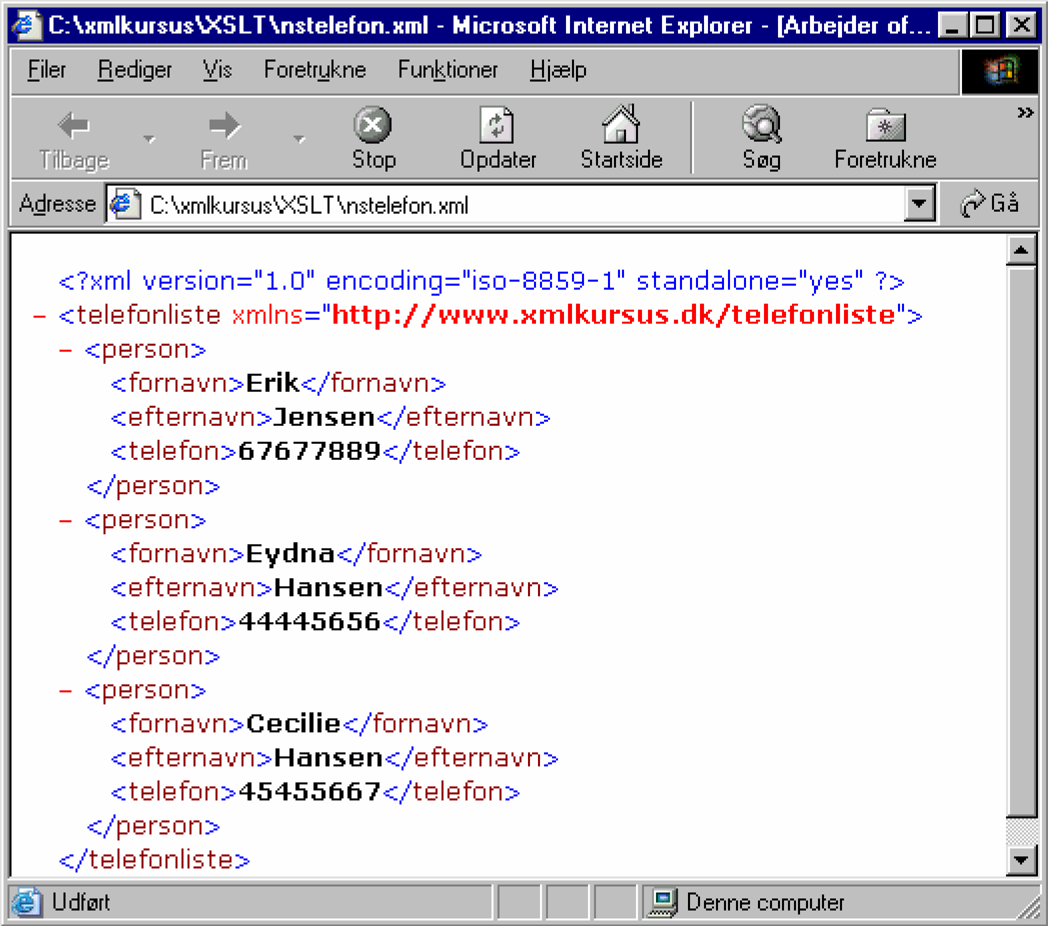
Nedenstående viser en mulig 'redigering' af et XML dokument:

Vores **udgangspunkt** er dette dokument som **ikke** har en xml erklæring, **ingen** encoding og **ikke** har et namespace:

```
<telefonliste>
<person>
<fornavn>Erik</fornavn>
<efternavn>Jensen</efternavn>
<telefon>67677889</telefon>
```

```
</person>
<person>
<fornavn>Eydna</fornavn>
<efternavn>Hansen</efternavn>
<telefon>44445656</telefon>
</person>
<person>
<fornavn>Cecilie</fornavn>
<efternavn>Hansen</efternavn>
<telefon>45455667</telefon>
</person>

</telefonliste>
```

Vi kan nu transformere dette dokument til et – **andet** – dokument der ser således ud:



Her er der sket de ønskede **ændringer**:

1. Dokumentet har fået et default standard **namespace**
2. Det har fået en **xml** erklæring
3. Det har fået en **encoding**
4. Det har fået en **standalone** erklæring

Dette kan vi gennemføre på denne simple måde:

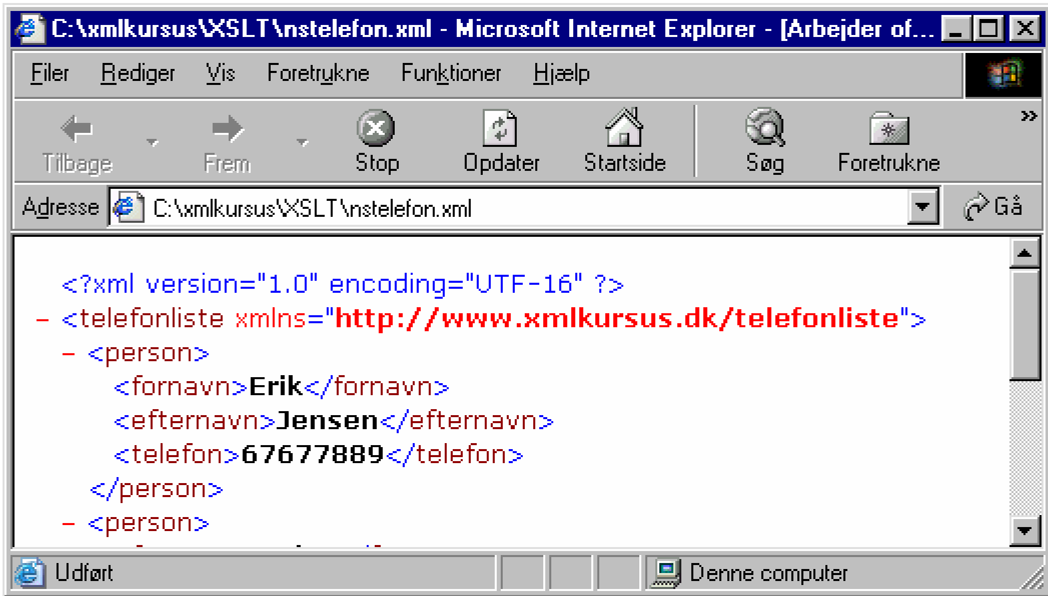
```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output
method="xml"
indent="yes"
encoding="iso-8859-1"
standalone="yes"
/>
<xsl:template match="/">
   <xsl:apply-templates select="*" />
</xsl:template>

<xsl:template match="*">
   <xsl:element name="{name()}" namespace="http://www.xmlkursus.dk/telefonliste">
   <xsl:apply-templates select="*" />
   <xsl:apply-templates select="text()" />
   </xsl:element>
</xsl:template>

</xsl:stylesheet>
```

Læg mærke til at vi også her bruger en rekursiv metode til igen og igen at kalde templaten '*'! den kaldes hver gang på det nuværende elements børn – altså hele tiden en ny kontekst!

Læg også mærke til at processoren bruger dette namespace som default namespace for hele dokumentet – anbragt i roden. Det erklæres ikke i hvert element!

Med denne metode og med metoden document() – omtalt andet steds – kan man sætte et nyt namespace på flere hundrede dokumenter på en gang!

Hvis vi IKKE angiver en encoding i XSLT output elementet – bliver det nye dokument ALTID kodet som Unicode eller UTF-16 – ikke altid det vi ønsker!

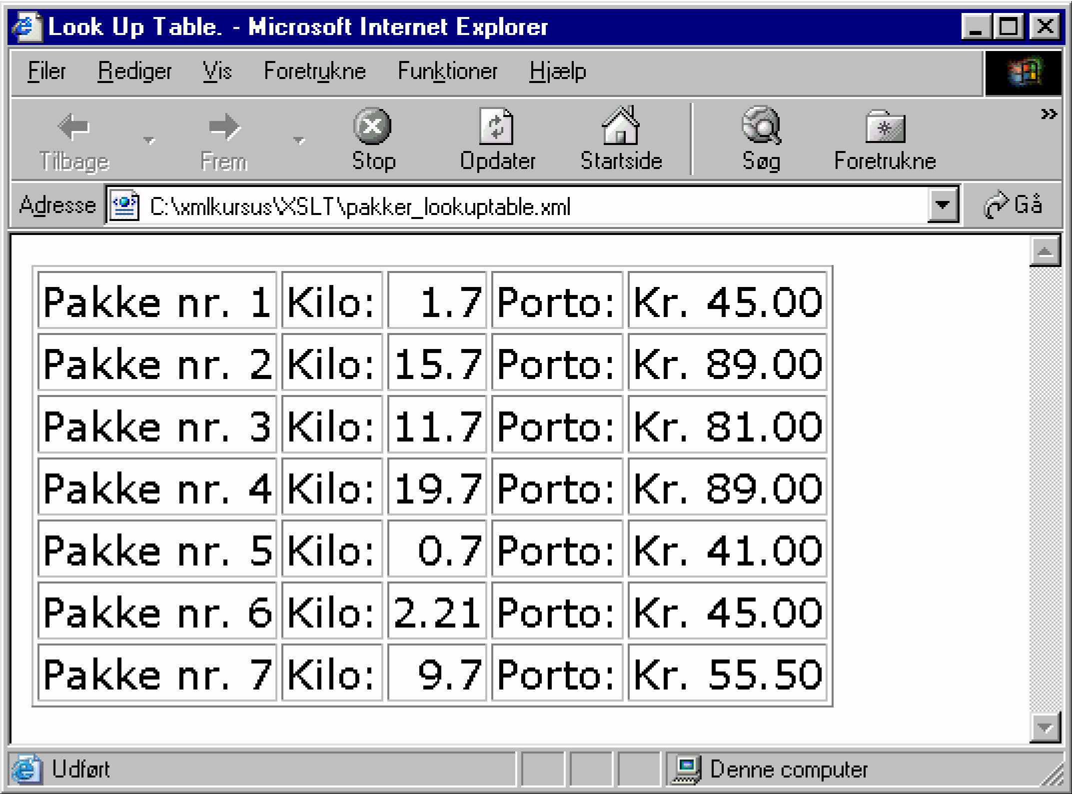


Look Up Table med et stylesheet:

Vi kan tage udgangspunkt i dette XML dokument som simpelt hen opregner en række pakker med deres vægt – attributten v:

```
<?xml-stylesheet type="text/xsl" href="pakker_lookuptable.xsl"?>

<pakker>
  <pakke v="1.7" />
  <pakke v="15.7" />
  <pakke v="11.7" />
  <pakke v="19.7" />
  <pakke v="0.7" />
  <pakke v="2.21" />
  <pakke v="9.7" />
</pakker>
```

Vi kan anvende – og det er meget **almindeligt** at gøre – XSLT til at formatere disse pakker så vi også viser hvad pakkerne vil koste i porto! Dette gøres ved at selve stylesheet'et opretter en såkaldt look up tabel med visse værdier vi kan bruge i beregningen! Det skal straks siges at dette kunne selvfølgeligt også gøres i et eksternt XML dokument! Men dette eksempel viser hvordan man kan anvende interne namespaces i et XSL dokument!



Pakke nr. 1	Kilo:	1.7	Porto:	Kr. 45.00
Pakke nr. 2	Kilo:	15.7	Porto:	Kr. 89.00
Pakke nr. 3	Kilo:	11.7	Porto:	Kr. 81.00
Pakke nr. 4	Kilo:	19.7	Porto:	Kr. 89.00
Pakke nr. 5	Kilo:	0.7	Porto:	Kr. 41.00
Pakke nr. 6	Kilo:	2.21	Porto:	Kr. 45.00
Pakke nr. 7	Kilo:	9.7	Porto:	Kr. 55.50

XSL dokumentet kan skrives på denne måde:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:pakker="uri:pakker"
>
<!-- look up table for porto til pakker -->

<pakker:porto max="1">41.00</pakker:porto>
<pakker:porto max="5">45.00</pakker:porto>
<pakker:porto max="10">55.50</pakker:porto>
<pakker:porto max="15">81.00</pakker:porto>
<pakker:porto max="20">89.00</pakker:porto>

<xsl:template match="/">
<html>
<head>
<title>Look Up Table.</title>
<style>td{text-align:right}</style>
</head>
<body>
<table border="1px" style="font-family:Verdana;font-size:15pt">
  <xsl:apply-templates select="//pakke">
</xsl:apply-templates>
</table>
</body>
</html>
</xsl:template>
```

```
<xsl:template match="pakke">
<tr><td><xsl:value-of select="concat('Pakke nr. ',position())" /></td>
<td>Kilo:</td>
<td><xsl:value-of select="@v" /></td>
<td>Porto:</td>
<td><xsl:value-of select="concat('Kr. ',document('')/*/pakker:porto[@max >current()/@v][1])" /></td>
</tr>
</xsl:template>
</xsl:stylesheet>
```

Vi er **nødt** til at oprette et særligt nyt **namespace** til vores look up tabel pakker:pakke! **Ellers** kan disse elementer ikke stå som direkte børn under roden stylesheet!

Det mest interessante er i øvrigt den sidste templat - pakke:

```
<td><xsl:value-of select="concat('Kr. ',document('')/*/pakker:porto[@max >current()/@v][1])" /></td>
```

Vi ønsker udskrevet hvad portoen er for de individuelle pakker. Dette kræver at vi slår op til tabellen. Vi skal finde en værdi der er lige over pakkens vægt!

Metoden **document**('') henter det **nuværende** dokument altså **dette** stylesheet! I dette dokument søger vi så på alle pakker:porto hvor attributten max er større end vores aktuelle pakkes vægt!

current() bruges her om det aktuelle objekt – det skifter jo når vi gennemløber listen! Vi henter det første af de elementer vi har fundet!

Formlen her er indviklet - **men** den samme formel vil trods alt altid blive brugt når man anvender sådanne interne tabeller til yderligere at formatere XML dokumentet!

Metoden document():

Det er muligt at hente tekster og værdier fra mange forskellige dokumenter ind i XSL scriptet. Dette sker med metoden document().

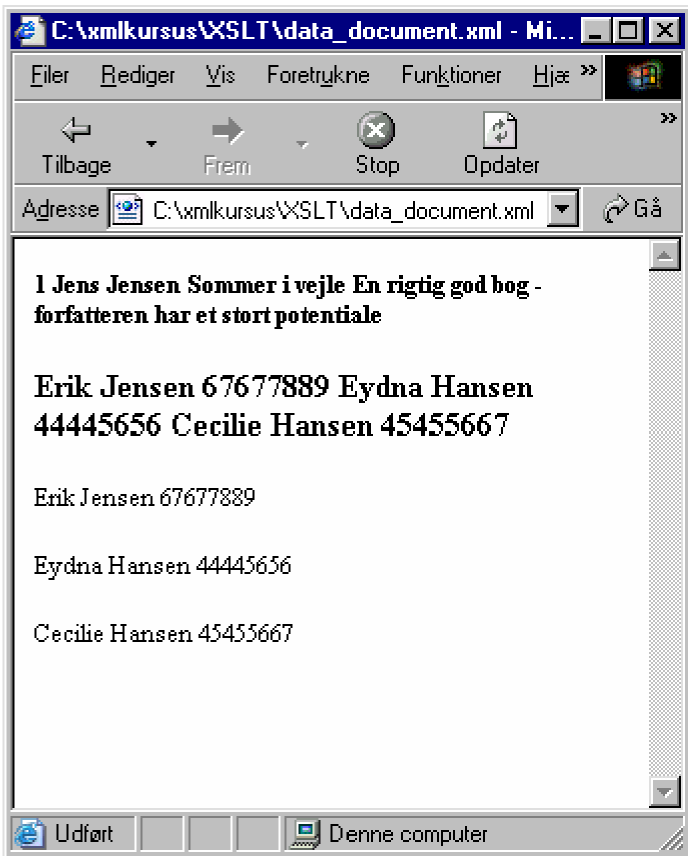
document() kan hente XML dokumenter lokalt eller fra en hvilken som helst server!

document() er på den måde også et eksempel hvordan et XSLT script kan være ret uafhængigt af XML dokumentet!

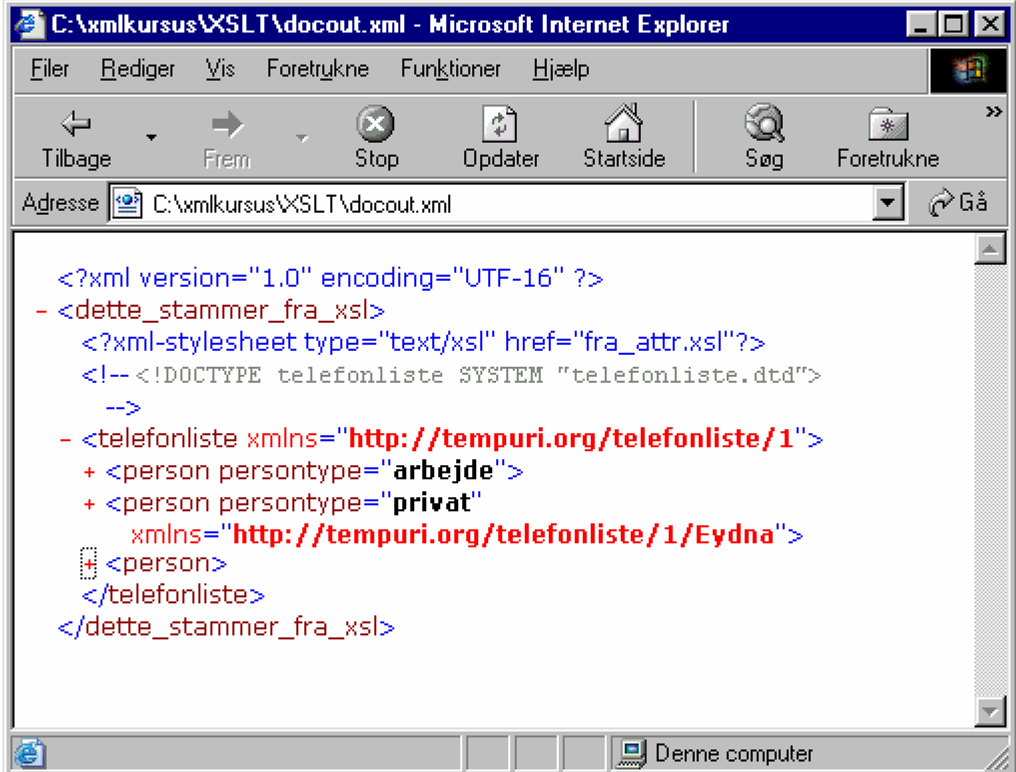
Nedenstående viser hvordan man kan kalde document() med en XML fil og derefter anvende XPATH og DOM metoder på det hentede dokument. På denne måde kan mange forskellige data mange steder fra anvendes i stil arket:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" />

<xsl:template match="/">
<h5>
```

```
<xsl:value-of select="document('bog.xml')/*/*" />
</h5>
<h3>
<xsl:copy-of select="document('tlf.xml')" />
</h3>
<xsl:for-each select="document('tlf.xml')/*/*">
<p><xsl:value-of select="." /></p>

</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```



I dette specielle tilfælde producerer XSL scriptet det **samme** output - **uanset** hvilket XML dokument det anvendes på! Funktionen **copy**-of kopierer hele dokumentet over i output træet! Hvis vi havde produceret XML kunne vi se at **ikke** bare indholdet men også **mærkerne** i XML kopieres med over med **copy**-of!:



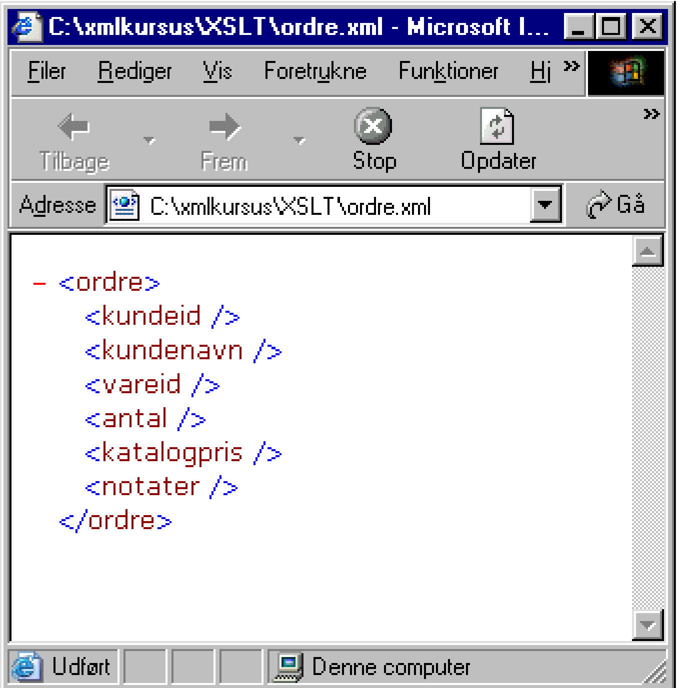
Læg mærke til at XML dokumentet der er produceret med **msxsl**.exe automatisk bliver kodet i UTF-16!!

Vi har indsat en **ny** tag (dette_stammer_fra_xsl) for at vise **hvad** der er kommet ind takket være copy-of!

Hente enslydende poster ind fra mange dokumenter:

Funktionen document() kan f. eks. bruges til at hente en række ensartede data ind i et samlet dokument. Vi kan forestille os en mængde ordrer afgivet til firmaet og gemt i forskellige filer. Disse dokumenter kan så samles og bearbejdes med document().

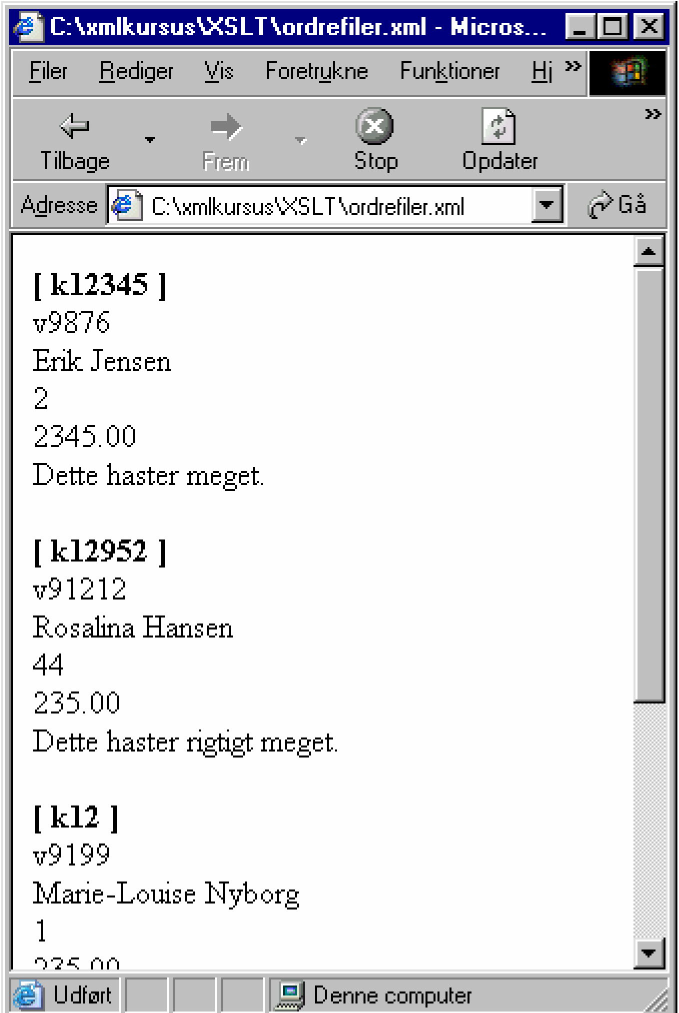
Alle ordrer bliver indgivet efter dette skema:



Vi kan nu referere en række forskellige dokumenter og samle resultatet i et dokument. Vi kan f. eks. skrive et XML dokument med disse referencer:

```
<?xml-stylesheet href="ordrefiler.xsl" type="text/xsl"?>
<ordrefiler>
<o id="ordre1.xml" />
<o id="ordre2.xml" />
<o id="ordre3.xml" />
<o id="ordre4.xml" />
</ordrefiler>
```

Nu kan de forskellige poster samles i et samlet dokument og manipuleres:



Dette kan gøres med dette XSLT ark:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
   <xsl:for-each select="//o">
      <xsl:apply-templates select="document(@id)/ordre" />
   </xsl:for-each>
</xsl:template>

<xsl:template match="ordre">
   <xsl:for-each select=".">
      <b>
      <xsl:value-of select="concat('[ ',kundeid,' ]')" />
      </b>
      <br />
      <xsl:value-of select="vareid" />
      <br />
      <xsl:value-of select="kundenavn" />
      <br />
      <xsl:value-of select="antal" />
      <br />
      <xsl:value-of select="katalogpris" />
```

```
      <br />
      <xsl:value-of select="notater" />
      <br />
      <br />
   </xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Disse referencer kunne selvfølgeligt også bare være i stilarket! Metoden her er meget brugt når der skal samles data fra mange steder.

Det virkeligt interessante er jo at bortset fra det første XPATH trin:

document(@id)/

så er al resten af koden fuldstændig den **samme** som hvis vi kun refererede eet enkelt dokument! Vi bruger **præcist** de samme formler! Vi har – med document() – simpelt hen lagt et ekstra lag **oveni** vores almindelige XPATH stier! En ekstra sti. Når vi henter data kan vi ikke længere **skelne** mellem hvor disse data kommer fra! Der er meget store **muligheder** i dette system! Kun fantasien sætter grænser!

F. eks. kan man samle data fra en række **stylesheets** som vist her – metoden er den samme som ovenfor:

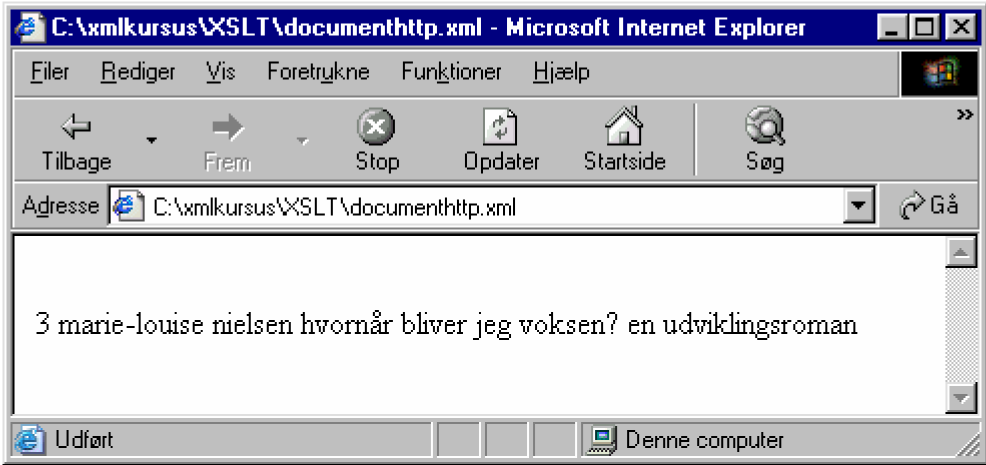
```
C:\xmlkursus\XSLT\docfiler.xml - Microsoft Internet Explorer
Filer  Rediger  Vis  Foretrukne  Funktioner  Hjælp
Tilbage  Frem  Stop  Opdater  Startside  Søg
Adresse  C:\xmlkursus\XSLT\docfiler.xml  Gå

  <?xml version="1.0" encoding="UTF-16" ?>
- <data>
  + <xsl:call-template name="rekursiv"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  + <xsl:when test="contains($streng,$deler)"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  + <xsl:otherwise
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  + <p
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  + <xsl:for-each select="@*"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:apply-templates select="*|text()"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform" />
    <xsl:value-of select="k:retur()"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:k="uri:Komponent" />
    <xsl:value-of select="name()"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform" />
    <xsl:value-of select=": "

Denne computer
```

Ressourcer på Internettet og document() funktionen:

Med document() kan man også frit referere til noder eller sub træer i XML dokumenter på Internettet! Prøv f. eks. denne formel:

```
<xsl:template match="/">
<data>
      <xsl:apply-templates
select="document('http://csharpkursus.subnet.dk/boghandel.xml')/*/bog[3]" />
</data>
</xsl:template>

<xsl:template match="bog">
  <xsl:for-each select=".">
  <br />
    <xsl:copy-of select="." />
  </xsl:for-each>
</xsl:template>
```

XSLT processoren kalder simpelt hen op til Internettet! Det er nu muligt at hente bestemte værdier fra dokumentet boghandel.xml!:



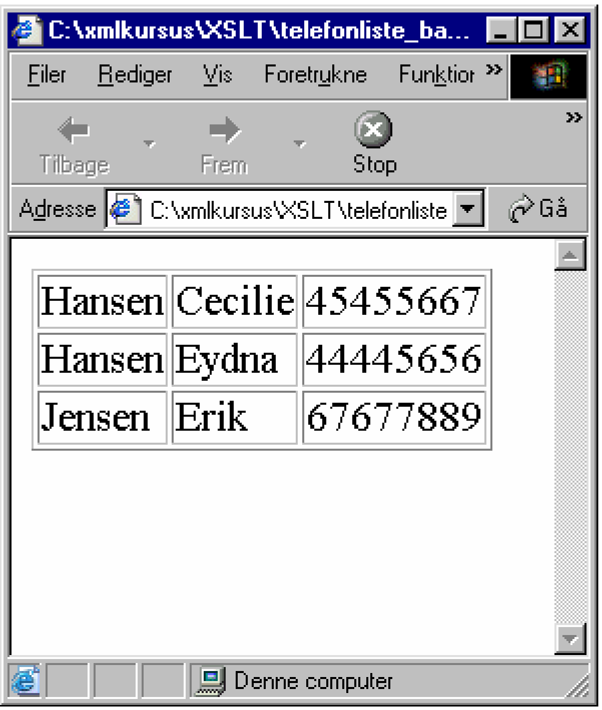
Sorter data:

Det er nemt at sortere data fra et XML dokument. Man kan i XSLT sortere efter forskellige data **typer** og efter lige så mange **kriterier** som man vil! F. eks. er der stor forskel på om vi sortere med **typen** streng eller typen heltal! Hvis vi sortere '1', '2', '111' efter datatypen **streng** eller tekst – som er **default** – bliver de sorteret sådan: '1', '111', '2' !

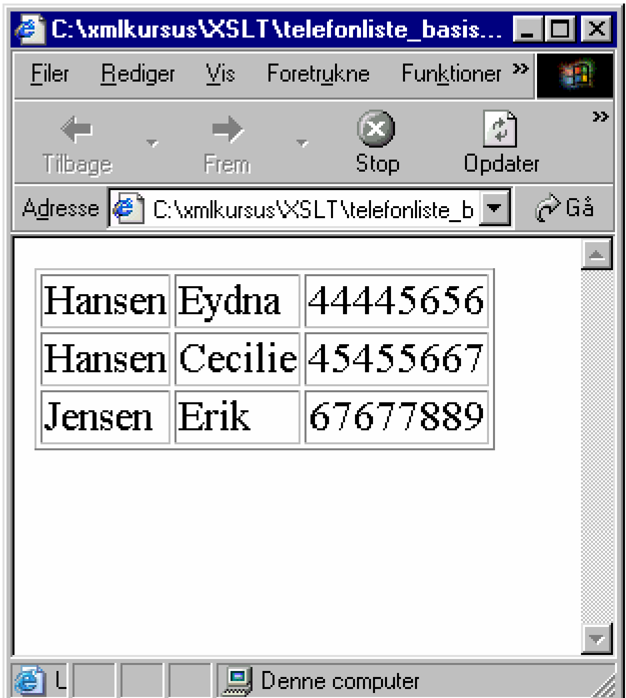
Hvis vi forestiller os en simpel telefonliste kan den sorteres på denne måde:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<table style="font-size:16pt" border="1">
<xsl:for-each select="//person">

<xsl:sort select="efternavn" data-type="text" order="ascending" />
<xsl:sort select="fornavn" data-type="text" order="ascending" />
<xsl:sort select="number(telefon)" data-type="number" order="ascending" />

<tr><td><xsl:value-of select="efternavn" /></td>
<td><xsl:value-of select="fornavn" /></td>
<td><xsl:value-of select="telefon" /></td>
</tr>
</xsl:for-each>
</table>
</html>
</xsl:template>
</xsl:stylesheet>
```

Vi vælger her at sortere efter **først** tekstnoden i efternavn **derefter** i fornavn og **hvis** der er flere ens par fornavn-efternavn (!) **også** efter telefonnummeret:



Hvis vi sætter **telefonnummeret** som kriterium nr 1 fås en anden sortering:



Dette system er altså meget **fleksibelt**. Man kan også sortere efter **sprog** idet nogle sprog f. eks. spansk sortere anderledes end på f. eks. engelsk! Rækkefølgen kan være **ascending** eller **descending**.

Script og formattering af tal:

Man kan anvende og kalde et **script** i XSLT og på den måde få adgang til et næsten ubegrænset antal nye **funktioner** – som f. eks. at hente et tilfældigt tal eller dags dato!

Man kan anvende et <script> på mange måder – her er et eksempel som anvender metoder fra **Microsofts** særlige namespace. Vi skal siden se på andre eksempler.

Dette eksempel viser også hvordan man kan **formatere** tal i XSLT:

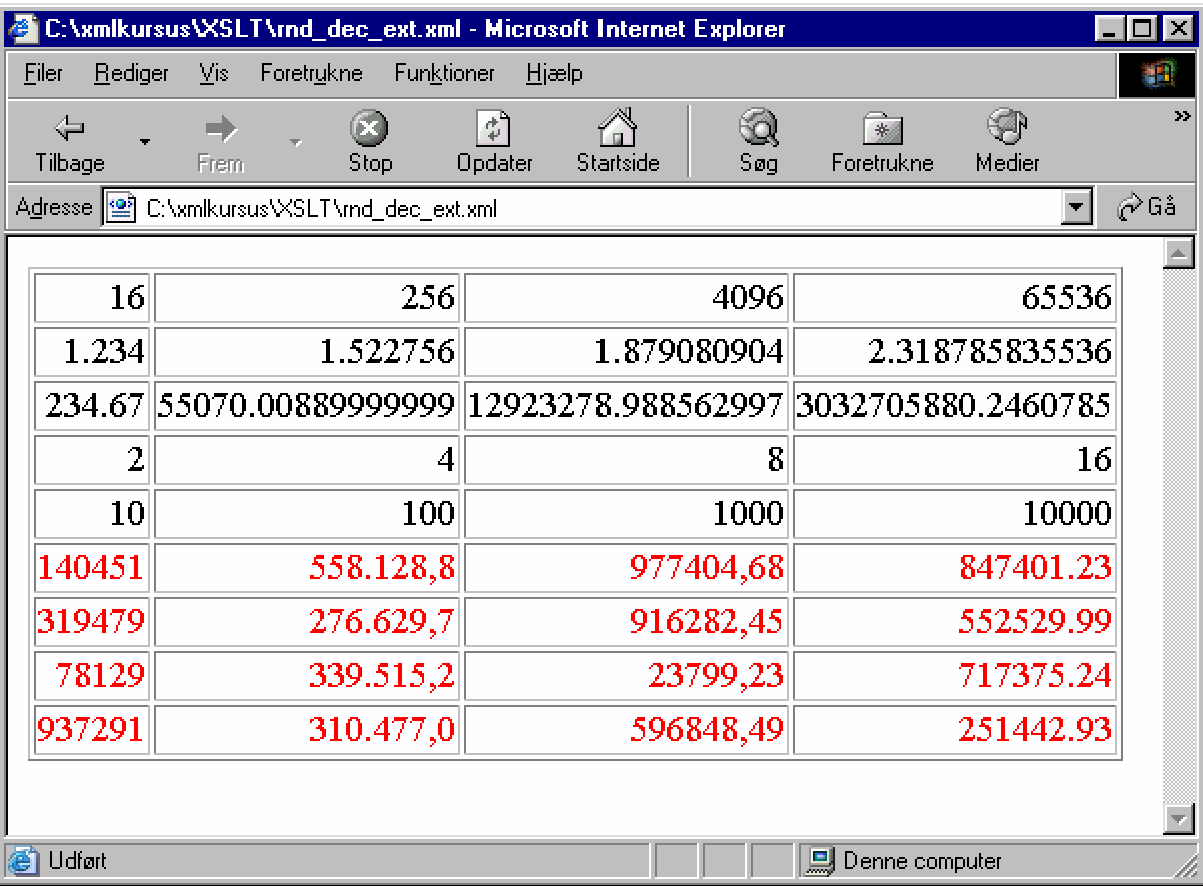
```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:tal="urn:tal.1"

>
<xsl:decimal-format name="DK" decimal-separator="," grouping-separator=".">
</xsl:decimal-format>

<ms:script language="JScript" implements-prefix="tal">
<![CDATA[
function i_anden(t){
return t*t;
}
function i_tredie(t){
return t*t*t;
}
function i_fjerde(t){
return t*t*t*t;
}
function rnd(){
return (Math.random()*1000000);
}
]]>
</ms:script>

<xsl:output method="html" indent="yes" omit-xml-declaration="yes" />


<xsl:template match="/">

<table border="1">
<xsl:for-each select="//tal">
<tr style="font-size:14pt;text-align:right">
<td>
<xsl:value-of select="."></xsl:value-of>
</td>
<td>
<xsl:value-of select="tal:i_anden(number(.))"></xsl:value-of>
</td>
<td>
<xsl:value-of select="tal:i_tredie(number(.))"></xsl:value-of>
</td>
<td>
```

```
<xsl:value-of select="tal:i_fjerde(number(.))"></xsl:value-of>
</td>

</tr>

</xsl:for-each>

<xsl:for-each select="//rnd_tal">
<tr style="font-size:14pt;text-align:right;color:red">
<td>
<xsl:value-of select="format-number(tal:rnd(), '.')"></xsl:value-of>
</td>
<td>
<xsl:value-of select="format-number(tal:rnd(), '###.###,0','DK')"></xsl:value-of>
</td>
<td>
<xsl:value-of select="format-number(tal:rnd(), '###,00','DK')"></xsl:value-of>
</td>
<td>
<xsl:value-of select="format-number(tal:rnd(), '.00')"></xsl:value-of>
</td>

</tr>

</xsl:for-each>

</table>

</xsl:template>

</xsl:stylesheet>
```

Vi har skrevet dette simple demo XML dokument:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="msextensions.xsl"?>

<matematik>

<tal>16</tal>
<tal>1.234</tal>
<tal>234.67</tal>
<tal>2</tal>
<tal>10</tal>

<rnd_tal></rnd_tal>
<rnd_tal></rnd_tal>
<rnd_tal></rnd_tal>
<rnd_tal></rnd_tal>

</matematik>
```

Vi kan nu bearbejde XML dokumentet til noget der ligner dette – men som jo er forskelligt **hver** gang vi kører transformationen!:



16	256	4096	65536
1.234	1.522756	1.879080904	2.318785835536
234.67	55070.00889999999	12923278.988562997	3032705880.2460785
2	4	8	16
10	100	1000	10000
140451	558.128,8	977404,68	847401.23
319479	276.629,7	916282,45	552529.99
78129	339.515,2	23799,23	717375.24
937291	310.477,0	596848,49	251442.93

De mange eksempler fra dette stylesheet skal ikke gennemgås her! Det væsentligste ved denne metode er følgende punkter:

1. Vi kan skrive lige så mange **funktioner** vi orker og på den måde gøre vores styling langt mere fleksibel
2. Et hvert script anbringes i et **namespace** som også bruges når vi **kalder** metoden i value-og select!
3. scriptet er anbragt i en **CDATA** sektion for at undgå at java funktionerne bliver parsede – for det er der ingen grund til!
4. Man kan i XSLT definere et f. eks. dansk **decimal format** med komma og punktum som vi plejer at bruge dem! I selve XML dokumentet kan vi **ikke** bruge et dansk format! 25,50 kr skal altid skrives som 25.50 kr i XML! XML er **altid** uafhængigt af hvad der som regel kaldes 'locale'! 12.145 betyder i XML 12 **komma** 145! 12.145 **skal** skrives **uden** separator sådan: 12145!! Det danske komma skal altid skrives med et punktum – som på engelsk!!
5. Man kan formatere alle tal som vist og herved bestemme hvor mange **decimaler** tallet skal rumme f. eks.:

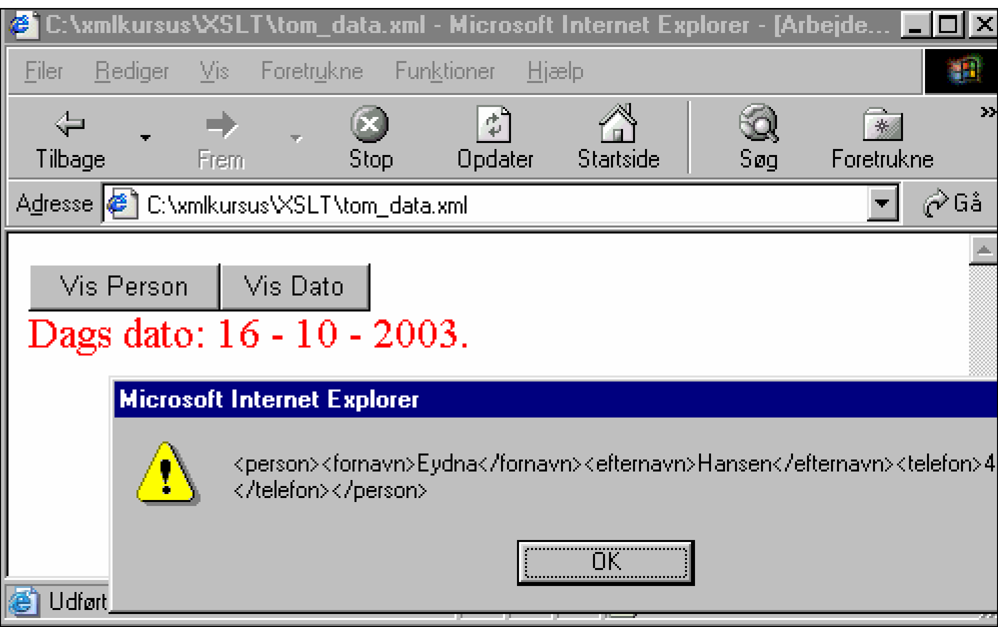
<xsl:value-of select="format-number(tal:rnd(), '###.###,0','DK')"></xsl:value-of>

Denne formel betyder altså at tallet skal vises med **dansk** notation og at der **altid** skal vises 1 og **kun** 1 decimal! (Se **eksemplet** i 2. kolonne i de røde tal!).

To slags scripts:

En server kan sende en side tilbage til en klient med et script! Vi skal altså skelne mellem scripts som anvendes af XSLT transformationen og scripts som ikke skal aktiveres nu men evt. siden kan aktiveres! De sidste scripts får så en attribut defer = true!

Vi kan transformere til en HTML side som f. eks. kan sendes til en klient eller modtager:



Modtageren får en tom side med to knapper! De to knapper viser så en message boks eller en tekst med dags dato.

Vi kan se på hvordan **forskellige** slags scripts bruges til dette i følgende stylesheet:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ms="urn:schemas-microsoft-com:xslt"
xmlns:s="urn:scripts"
xmlns:x="uri:ex"
>

<!-- eks script som bruges af XSLT -->
<ms:script language="JScript" implements-prefix="s">
<![CDATA[
function dato(){
var d=new Date();
return d.getDate()+" - "+ (d.getMonth()+1)+" - "+ d.getYear();
}
]]>
</ms:script>

<ms:script language="JScript" implements-prefix="x">
```

```
<![CDATA[
function retur(){
return "Dags dato: ";
}
]]>
</ms:script>

<xsl:output method="html" indent="yes" omit-xml-declaration="yes" />

<xsl:template match="/">

<!-- eks script direkte til klienten -->

<input type="button" value="Vis Person" onclick="vis();" />
<input type="button" value="Vis Dato" onclick="div.style.visibility='visible';" />
<div id="div" name="div" style="font-size:16pt;color:red;visibility:hidden"><xsl:value-of select="x:retur()"/>
<xsl:value-of select="s:dato()" />.</div>

<script defer="true">
function vis() {
var
person="<person><fornavn>Eydna</fornavn><efternavn>Hansen</efternavn><telefon>44445656</telefon></person>
";
alert(person);
}
</script>

</xsl:template>

</xsl:stylesheet>
```

Som det står i kommentarerne er der tale om helt **forskellige** typer af scripts! Den sidste function vis() der har en attribut **defer** skal selvfølgeligt først kaldes hvis modtageren klikker på knappen – den skal ikke kaldes af XSL processoren! Funktionen vis() er netop **ikke** en **extension** funktion!

Efter dette princip er der ingen grænser for hvad der kan sendes af funktioner og scripts til en klient f. eks. sammen med XML materiale!

Rekursive funktioner eller templater i XSLT (tokenizer):

En tokenizer kan være en funktion som opdeler en tekst i ord f. eks. Vi kan illustrere brugen af såkaldt rekursive funktioner i XSLT med følgende eksempel. Stylesheet'et virker på et hvilket som helst XML dokument fordi det selv skriver hele output træet! Vi tager udgangspunkt i en lille tekst og ønsker at få den opdelt i enkelt ord!

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- eks rekursiv 'funktion' der deler i ord -->

<xsl:template match="/">
<table border="1" style="font-size:14pt">
  <xsl:call-template name="rekursiv">
      <xsl:with-param name="streng" select="'Der var engang en lille dreng, som hed Kurt.'" />
```

```
        <xsl:with-param name="deler" select="' '" />
  </xsl:call-template>
</table>
</xsl:template>

<xsl:template name="rekursiv">
<xsl:param name="streng" />
<xsl:param name="deler" />


<xsl:choose>
   <xsl:when test="contains($streng,$deler)">
      <tr><td><xsl:value-of select="substring-before($streng,$deler)" /></td></tr>
       <xsl:call-template name="rekursiv">
          <xsl:with-param name="streng">
                    <xsl:value-of select="substring-after($streng,$deler)" />
          </xsl:with-param>
          <xsl:with-param name="deler" select="$deler" />
       </xsl:call-template>
   </xsl:when>
   <xsl:otherwise>
   <tr><td><xsl:value-of select="$streng" /></td></tr>
   </xsl:otherwise>
</xsl:choose>


</xsl:template>


</xsl:stylesheet>
```

De centrale passager er her vist med fed skrift! Vi **kalder** templaten rekursiv med en streng og checker om den indeholder et mellemrum. Hvis den gør det tager vi bort det første ord og kalder funktionen rekursiv igen på samme måde men med den nye forkortede streng! Til sidst er der ikke flere mellemrum i teksten og vi er færdige!

En templat kan aktiveres med **call**-template og med **apply**-templates! Med den første metode vender programmet altid **tilbage** til den linje hvor funktionen blevet kaldt! Med apply-templates fortsætter det i en helt ny kontekst!

I rekursive funktioner anvendes altid **call**-template af en template som netop har et **navn** og **ikke** en element match!

Vi kan også se her hvordan man kan kalde en 'funktion' eller templat med visse input **parametre**!

Metoden svarer helt til **kald** af en funktion f. eks. i et programmeringssprog som Java eller C++.

Eksemplet viser hvordan XSLT (eller rettere: **XPATH**) indeholder de vigtige funktioner substring!

Transformere alle attributter til elementer:

En almindelig opgave er tit at transformere et XML dokument til et andet XML dokument hvor alle attributter er transformeret til sub elementer! Vi anvender også her en rekursiv algoritme således at den samme template '*' bliver ved med at blive kaldt med apply-templates! Dette kan gøres på følgende måde:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output method="xml" encoding="iso-8859-1"/>

<xsl:template match="*">
  <xsl:element name="{name()}">
    <xsl:for-each select="@*">
      <xsl:element name="{name()}">
      <xsl:value-of select="." />
      </xsl:element>
    </xsl:for-each>
    <xsl:apply-templates select="*|text()" />
  </xsl:element>
</xsl:template>
```

</xsl:stylesheet>

```
C:\xmlkursus\XSLT\varer1.xml - Microsoft Internet ...
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"
Udført  Denne computer
```

Hvis vi anvender dette stylesheet på vores vare liste hvor hver vare kun består af attributter får vi dette XML dokument:

C:\xmlkursus\XSLT\varer1_ny.xml - Microsoft Intern...

Filer Rediger Vis Foretrukne Funktioner Hjælp

Tilbage Frem Stop Opdater Startside

Adresse C:\xmlkursus\XSLT\varer1_ny.xml Gå

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <data>
  - <vare>
      <id>1</id>
      <dansk>False</dansk>
      <kategori>dagligvare</kategori>
      <pris>793</pris>
    </vare>
  - <vare>
      <id>3</id>
      <dansk>True</dansk>
      <kategori>kontorvare</kategori>
      <pris>857</pris>
    </vare>
  - <vare>
      <id>5</id>
      <dansk>False</dansk>
      <kategori>dagligvare</kategori>
      <pris>93</pris>
    </vare>
  - <vare>
      <id>7</id>
      <dansk>False</dansk>
```

Udført Denne computer

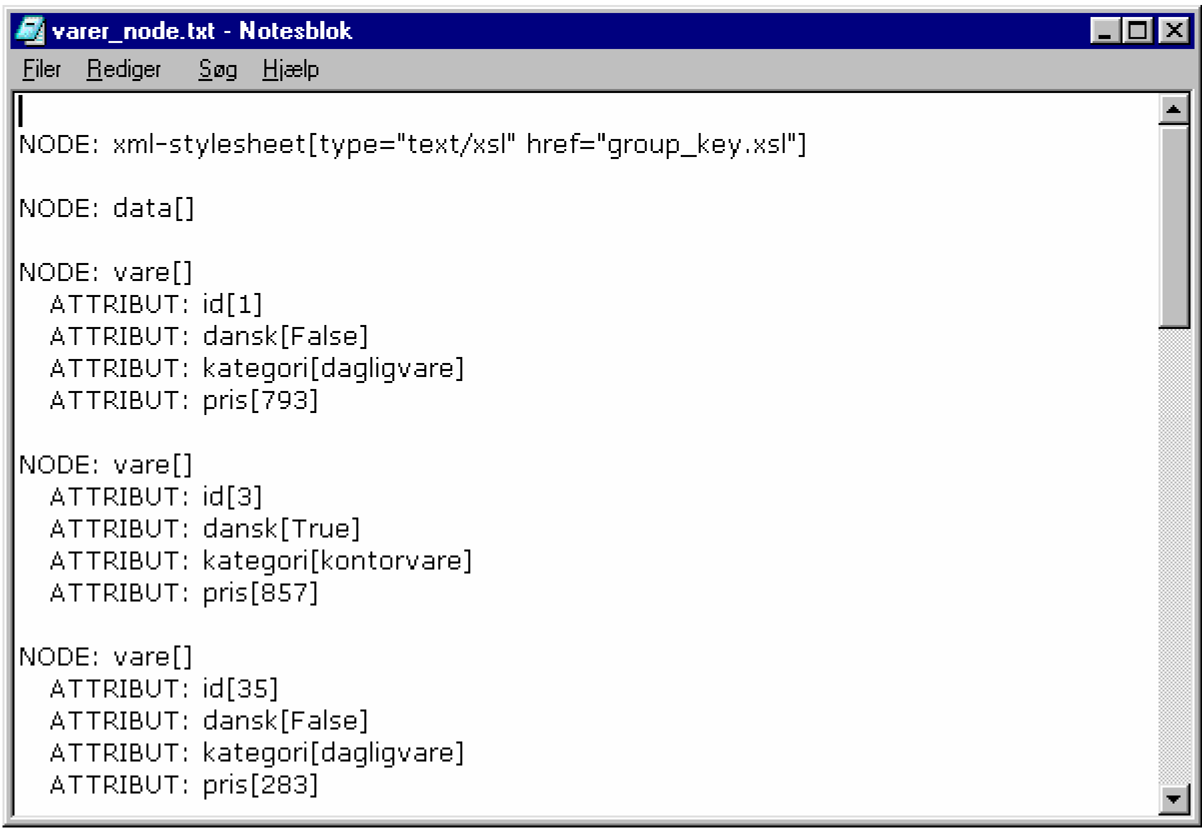
Alle noder i dokumentet - rekursivt:

Følgende stylesheet kan bruges til at vise **ALLE noder** (ikke kun alle elementer!) i et XML dokument inklusive whitespace noder (f. eks. linjeskift), elementer, attributter, tekstnoder, xml-erklæring osv! Også dette stil ark anvender en **rekursiv** metode ved at kalde den **samme** template indtil bunden af træet er nået!:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output method="text" encoding="iso-8859-1"/>

<xsl:template match="node()">
NODE: <xsl:value-of select="name()" />[<xsl:value-of select="." />]
<xsl:for-each select="@*">   ATTRIBUT: <xsl:value-of select="name()" />[<xsl:value-of select="." />]
</xsl:for-each>
<xsl:apply-templates select="node()" />
</xsl:template>
```

</xsl:stylesheet>

Dette kan så producere denne **tekst** (som ikke er et XML dokument). I skarpe **parenteser** vises nodens 'value' altså dens tekst indhold. Læg også mærke til at **navnet** på stylesheet noden simpelt hen er **'xml**-stylesheet'! Her anvendes et XML dokument varer.xml:



varer_node.txt - Notesblok

Filer Rediger Søg Hjælp

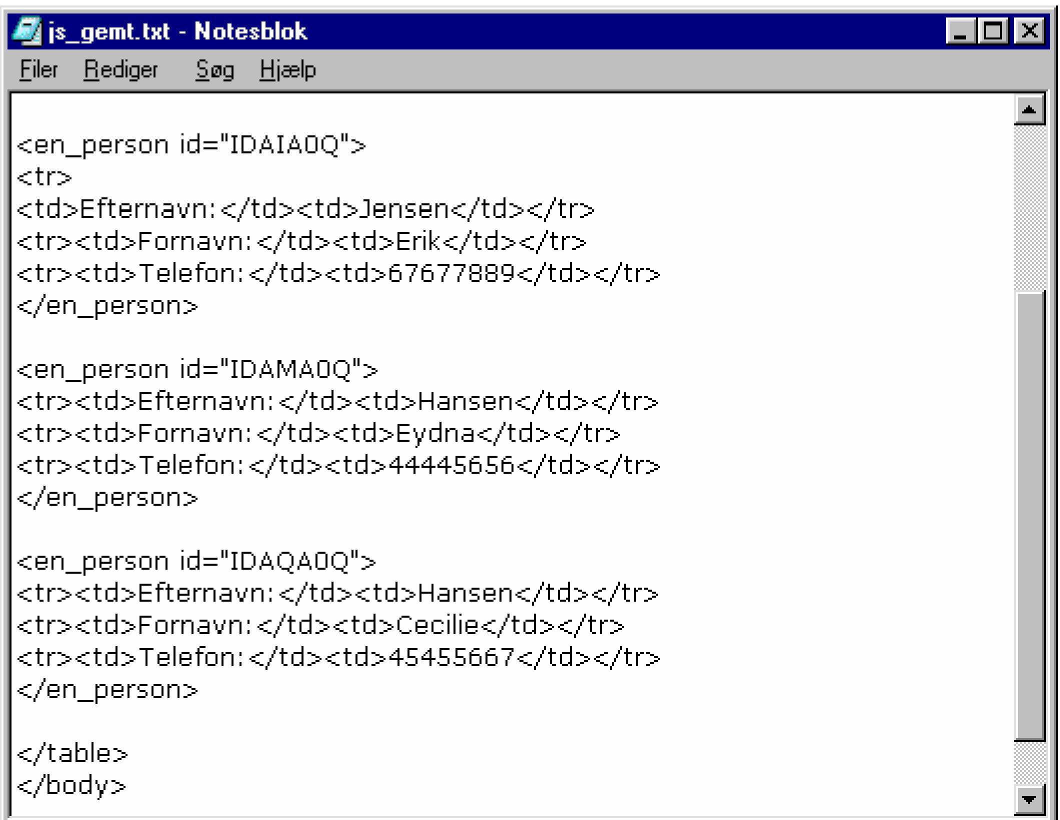
```
NODE: xml-stylesheet[type="text/xsl" href="group_key.xsl"]

NODE: data[]

NODE: vare[]
   ATTRIBUT: id[1]
   ATTRIBUT: dansk[False]
   ATTRIBUT: kategori[dagligvare]
   ATTRIBUT: pris[793]

NODE: vare[]
   ATTRIBUT: id[3]
   ATTRIBUT: dansk[True]
   ATTRIBUT: kategori[kontorvare]
   ATTRIBUT: pris[857]

NODE: vare[]
   ATTRIBUT: id[35]
   ATTRIBUT: dansk[False]
   ATTRIBUT: kategori[dagligvare]
   ATTRIBUT: pris[283]
```

At indsætte nye id attributter med XSLT:

I mange tilfælde ønsker vi at transformere et XML dokument til et andet format hvor visse objekter får tildelt **unikke ID** attributter.

Dette er meget nemt at gøre i XSLT med **XPATH** funktionen **generate**-id() som genererer et tilfældigt ID ud fra det aktuelle objekt og som garanterer at der aldrig skabes to ens ID'er i samme dokument for det samme objekt! Og garanterer at det samme objekt i dokumentet altid genererer det samme ID!

Vi har her taget vores telefonliste og – for øvelsens skyld ændret navnet til **<en**_person> og **indsat** et ID:



```
<en_person id="IDAIA0Q">
<tr>
<td>Efternavn:</td><td>Jensen</td></tr>
<tr><td>Fornavn:</td><td>Erik</td></tr>
<tr><td>Telefon:</td><td>67677889</td></tr>
</en_person>

<en_person id="IDAMA0Q">
<tr><td>Efternavn:</td><td>Hansen</td></tr>
<tr><td>Fornavn:</td><td>Eydna</td></tr>
<tr><td>Telefon:</td><td>44445656</td></tr>
</en_person>

<en_person id="IDAQA0Q">
<tr><td>Efternavn:</td><td>Hansen</td></tr>
<tr><td>Fornavn:</td><td>Cecilie</td></tr>
<tr><td>Telefon:</td><td>45455667</td></tr>
</en_person>

</table>
</body>
```

Vi kan se at et sådant ID aldrig starter med et tal (dette er altid reglen for et ID i XML) og at det er **forskelligt** i de 3 tilfælde.

Dette kan opnås ved dette stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>generate-id() stylesheet.</title>
</head>
<body>
<table border="1px" style="font-family:Verdana;font-size:15pt">
<xsl:apply-templates select="//person">
</xsl:apply-templates>
</table>
</body>
</html>
</xsl:template>

<xsl:template match="person">
<en_person id="{generate-id()}" >
<tr><td>Efternavn:</td>
<td><xsl:value-of select="efternavn" /></td></tr>
<tr><td>Fornavn:</td>
<td><xsl:value-of select="fornavn" /></td></tr>
<tr><td>Telefon:</td>
<td><xsl:value-of select="telefon" /></td></tr>
```

```
</en_person>

</xsl:template>
</xsl:stylesheet>
```

Transformation af ID og IDREF til HTML:

Et Id eller et IDREF element defineres i et DTD eller XSD skema. I dette eksempel går vi altså ud fra et XML dokument som allerede har defineret disse relationer og nøgler. Det er ikke relationer som stylesheetet opretter!

Vores eksempel tekst er defineret således:

```
<?xml-stylesheet href="tekster_id.xsl" type="text/xsl"?>
<!DOCTYPE tekster [
<!ELEMENT tekster (tekst*)>
<!ELEMENT tekst (#PCDATA|ref)*>
<!ATTLIST tekst id ID #REQUIRED>
<!ELEMENT ref EMPTY>
<!ATTLIST ref id IDREF #REQUIRED>
]>

<tekster>
<tekst id="t1">
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
<ref id="t2" />
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t1 - en eksempel tekst som ikke kan bruges til ret meget.
</tekst>
<tekst id="t2">
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t2 - en eksempel tekst som ikke kan bruges til ret meget.
<ref id="t1" />
</tekst>

<tekst id="t3">
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
```

```
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
Dette er - t3 - en eksempel tekst som ikke kan bruges til ret meget.
<ref id="t1" />
<ref id="t2" />

</tekst>

</tekster>
```

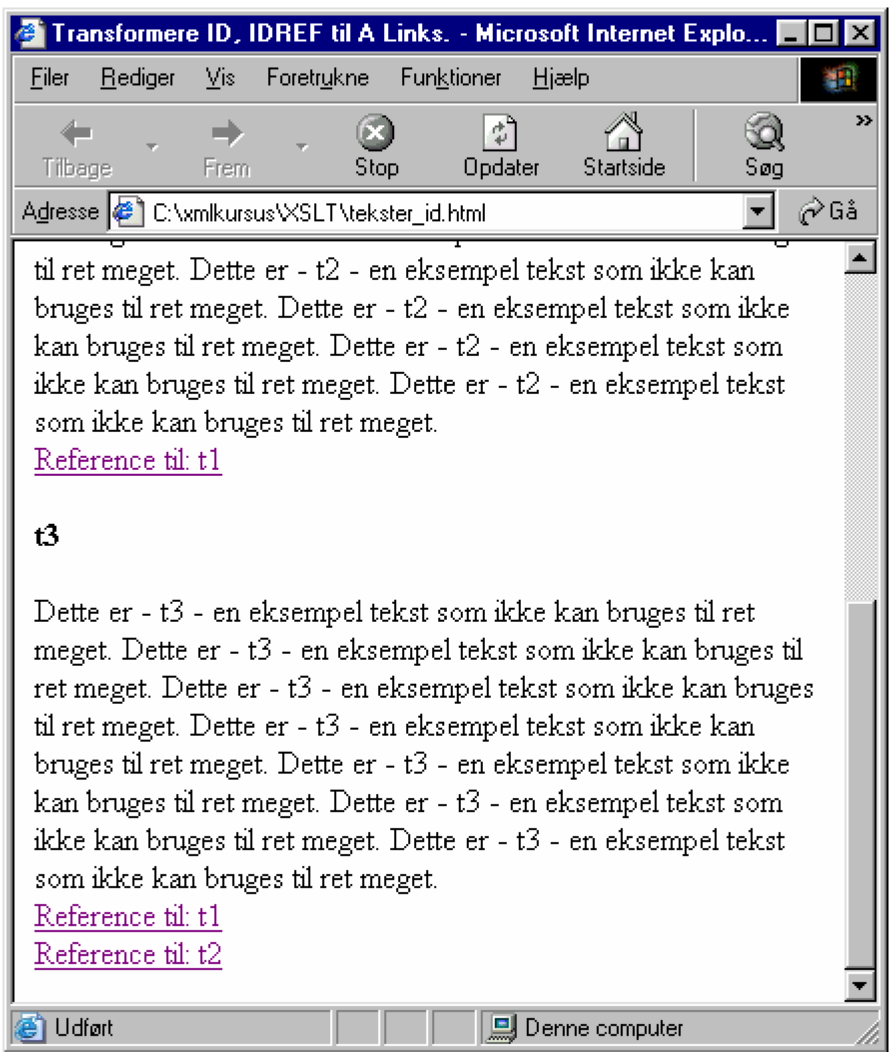
Vi har gjort teksterne lidt lange for at kunne se hvordan der kan springes rundt i dokumentet siden hen!

Vi kan nu transformere disse XML ID koder til a links i HTML på denne måde:

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="tekster">
  <html>
      <head>
        <title>Transformere ID, IDREF til A Links.</title>
      </head>
      <body>
      <xsl:apply-templates select="tekst" />
      </body>
  </html>
</xsl:template>

<xsl:template match="tekst">
   <p>
   <b>
   <a name="#{@id}"><xsl:value-of select="@id" /></a>
   </b>
   </p>
   <xsl:value-of select="." />
   <xsl:for-each select="ref">
      <br />
      <a href="#{@id}">Reference til: <xsl:value-of select="@id" /></a>
   </xsl:for-each>
</xsl:template>


</xsl:stylesheet>
```

Metoden er simpel nok: For hvert element tekst henter vi attributten id (som vi ved er unik) og opretter et <a> direkte i output træet! Dette <a> er et <a name> fordi springer rundt i det samme dokument ikke til andre dokumenter!

På samme måde oprettes en <a href> hver gang vi støder på objektet <ref>! Dokumentet kan nu se sådan ud i en browser:



Man kan nu klikke sig rundt i dokumentet! Vi har her brugt t1, t2 og t3 som en slags overskrifter over teksterne for at kunne skelne dem fra hinanden og fordi et <a name> jo ikke kan ses! Normalt vil man selvfølgeligt vælge en mere naturlig overskrift til tekterne!

Den **simpleste** og **sikreste** måde at oprette links og kryds henvisninger på er at anvende et DTD som her og typerne ID og IDREF. Dokumentet kan så kun valideres hvis:

1. Alle ID'er skal være forskellige
2. Alle IDREF skal henvise til et ID som findes i dokumentet

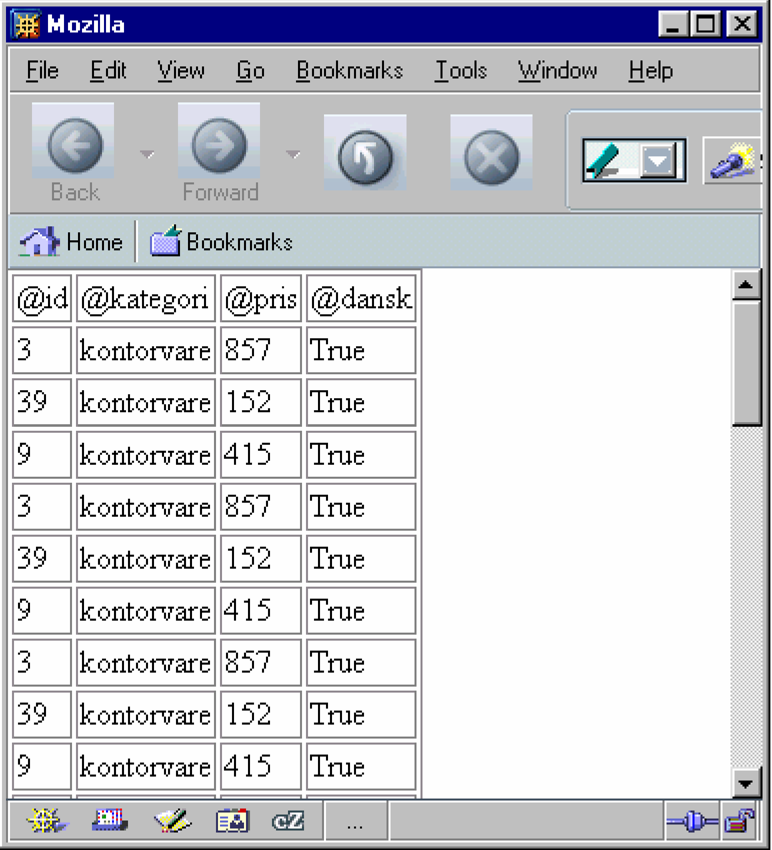
I stedet for typen IDREF kan bruges typen IDREFS som er en liste af ID referencer! Men stadig væk er dette system begrænset og der er derfor brug for at man kan oprette relationer på en anden måde. Derfor kan man i XSLT anvende elementet key og funktionen generate-key(). Den afgørende forskel er at med disse XSLT metoder skaver vi links og relationer i XML dokumentet som slet ikke findes i forvejen! Vi er ikke begrænset af at et objekt kun kan have en ID eller af at der kun må være et ID med samme værdi. Med XSLT kan vi skabe langt mere variable og frie relationer. Vi kan også f. eks. linke til et dokument (del dokument) som indeholder et bestemt ord!

Anvendelsen af xsl:key:

Vi kan oprette en key eller en indeksering af et data XML dokument. Vi kan tage udgangspunkt i dette eksempel som opretter en mængde varer med visse attributter:

```
- <data>
    <vare id="1" dansk="False" kategori="dagligvare"
      pris="793" />
    <vare id="3" dansk="True" kategori="kontorvare"
      pris="857" />
    <vare id="35" dansk="False" kategori="dagligvare"
      pris="283" />
    <vare id="37" dansk="False" kategori="dagligvare"
      pris="18" />
    <vare id="39" dansk="True" kategori="kontorvare"
      pris="152" />
    <vare id="41" dansk="False" kategori="dagligvare"
      pris="420" />
    <vare id="43" dansk="False" kategori="dagligvare"
      pris="548" />
    <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"
```

Hvis vi nu kun er interesseret i at arbejde med visse varer f.eks. kun varer som tilhører kategorien **kontorvare** kan vi **indeksere** på denne attribut. Vi laver på den måde en sub tabel som man også kan gøre i en database tabel! Vores søgeproces bliver nu meget hurtigere.



@id	@kategori	@pris	@dansk
3	kontorvare	857	True
39	kontorvare	152	True
9	kontorvare	415	True
3	kontorvare	857	True
39	kontorvare	152	True
9	kontorvare	415	True
3	kontorvare	857	True
39	kontorvare	152	True
9	kontorvare	415	True

Scriptet ser således ud:

```
<?xml version="1.0"?>
<!-- key som i DB -->

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:key name="kategori_key" match="//vare" use="@kategori" />
<xsl:key name="dansk_key" match="//vare" use="@dansk" />

<xsl:output method="html" indent="yes" />

<xsl:template match="/">
<table style="font-size:12pt" border="1">
<tr>
  <td>@id</td>
  <td>@kategori</td>
  <td>@pris</td>
  <td>@dansk</td>
</tr>

        <xsl:apply-templates select="//vare" >
        </xsl:apply-templates>

</table>
</xsl:template>

<xsl:template match="vare">
```

```
<xsl:for-each select="key('kategori_key','kontorvare')">
<tr>
  <td><xsl:value-of select="@id" /></td>
  <td><xsl:value-of select="@kategori" /></td>
  <td><xsl:value-of select="number(@pris)" /></td>
  <td><xsl:value-of select="@dansk" /></td>
</tr>
</xsl:for-each>

</xsl:template>

</xsl:stylesheet>
```

Det som vi kan opnå med key kan vi også opnå ved almindelige XPATH søge metoder! Ofte er det bare nemmere at gøre med en xsl:key! Koden er nemmere at overskue! Og processen skulle meget gerne gå noget hurtigere fordi key sub tabellen er forberedt i forvejen! Den skal ikke produceres hver gang.

En key svarer delvist til en forespørgsel eller en query i en database.

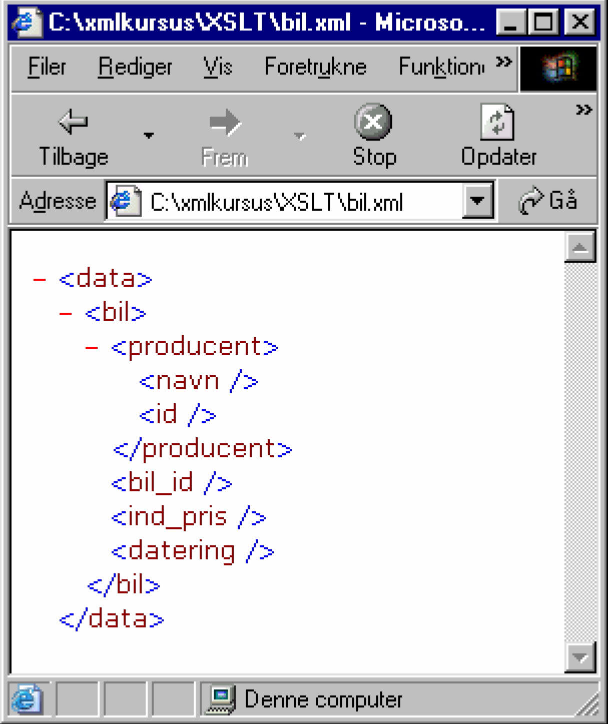
Det er vigtigt at xsl:key elementet og XPATH funktionen key() er to vidt forskellige ting! Funktionen key() returnerer sand eller falsk d.v.s. den undersøger for hver enkelt vare om betingelsen er sand: her at kategorien skal være 'kontorvare'! Er det sandt beholdes denne vare i nodesættet – ellers tages denne vare ikke med.

En parameter kan bruges til at søge i en key og en sådan parameter kan sættes udefra f. eks. med en XSL processor som vi ser på andre steder.

Debugging af XSLT:

Med debugging forstås her at det bliver muligt at trace eller følge transformationen – at kunne følge de enkelte trin som XSL processoren foretager! Formålet med en sådan debugging er at finde evt. fejl i scriptet! F. eks. ved rekursive funktioner – men i det hele taget – er det tit ret svært at se hvad processoren egentligt laver! Forskellige XSL processore handler heller ikke helt ens! Resultatet skulle dog meget gerne være det samme stort set!

Vi tager udgangspunkt i dette simple dokument:



Vores stylesheet ser sådan ud:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:template match="*">
        <xsl:message terminate="no">
        XSLT kalder nu en node: <xsl:value-of select="concat('[',name(),']')" />
        <xsl:value-of select="concat('[',position(),']')" />
        </xsl:message>
<xsl:apply-templates select="*" />
</xsl:template>


</xsl:stylesheet>
```

I XSLT er det muligt at erklære et message element som vist. Hvis et message har attributten:

```
        <xsl:message terminate="yes">
```

medfører det at processoren stopper straks når den støder på elementet! Meningen hermed er f. eks. at et extension element eller en speciel funktion måske ikke er til stede – og derfor ønsker vi at transformationen i så fald skal afbrydes. En sådan terminate message lægges altså ind i en if sætning!

Her bruges message til noget andet idet vi ønsker at følge processoren i dens trin! Derfor beder vi processoren om hver gang den støder på elementet at udskrive visse debugging oplysninger! Her ønsker vi udskrevet nodens navn og dens position i nodesættet.

Denne form for message – som ikke er **terminate** – kan ikke lade sig gøre med Microsofts XSL processor **msxsl**.exe! (Men Internet Explorer reagerer på en terminate message!).

Men den kan lade sig gøre med f. eks. **Xalan** XSL processoren som frit kan downloades fra Internettet:

```
MS-DOS-prompt
C:\xmlkursus\XSLT>xalan -a bil.xml
XSLT message:
                XSLT kalder nu en node: [data][2], source tree node: data (file:
///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [bil][1], source tree node: bil (file://
/C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [producent][1], source tree node: produc
ent (file:///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [navn][1], source tree node: navn (file:
///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [id][2], source tree node: id (file:///C
:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [bil_id][2], source tree node: bil_id (f
ile:///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [ind_pris][3], source tree node: ind_pri
s (file:///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
XSLT message:
                XSLT kalder nu en node: [datering][4], source tree node: daterin
g (file:///C:/xmlkursus/XSLT/debugging.xsl, line 7, column 31)
<?xml version="1.0" encoding="UTF-8"?>

C:\xmlkursus\XSLT>_
```

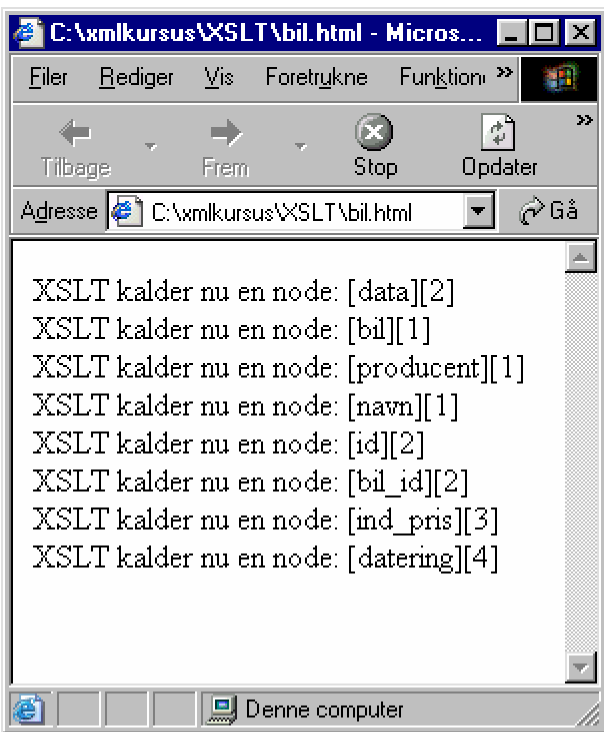
Vi kan se på den sidste linje at det **eneste** som processoren udskriver er en xml erklæring for et tomt dokument – ikke et gyldigt dokument!

Ellers kan vi her følge de enkelte **trin**. Som sagt er dette meget væsentligt hvis vi sidder med stylesheets vi ikke kan få til at fungere! F. eks. ofte med **rekursive** stylesheets som dette – der dog er ret simpelt!

Vi kan se at processoren starter med at kalde **childNodes** nemlig **data** som er nr 2 i sættet (nr 1 er skjult men er xml erklæringen – som vi ikke har skrevet)! I anden omgang kaldes **bil** nr 1 osv. Vi kan tydeligt se at **position** altid er positionen **inden** for et bestemt **nodesæt** – altså på et bestemt **niveau** i træet. **Både** id og bil_id har position 2 – **men** i to forskellige **dybder** eller niveauer i XML træet!

Processoren udskriver også **hvor** i stylesheet'et denne message udløses (linje og kolonne nummer) og i **hvilket** stylesheet det sker! Vi kunne jo nemt anvende flere **forskellige** stylesheets samtidigt!

Rækken af message-elementer kan godt udskrives til f. eks. en HTML fil ved i stedet at sætte <div> elementer uden om de ønskede data:

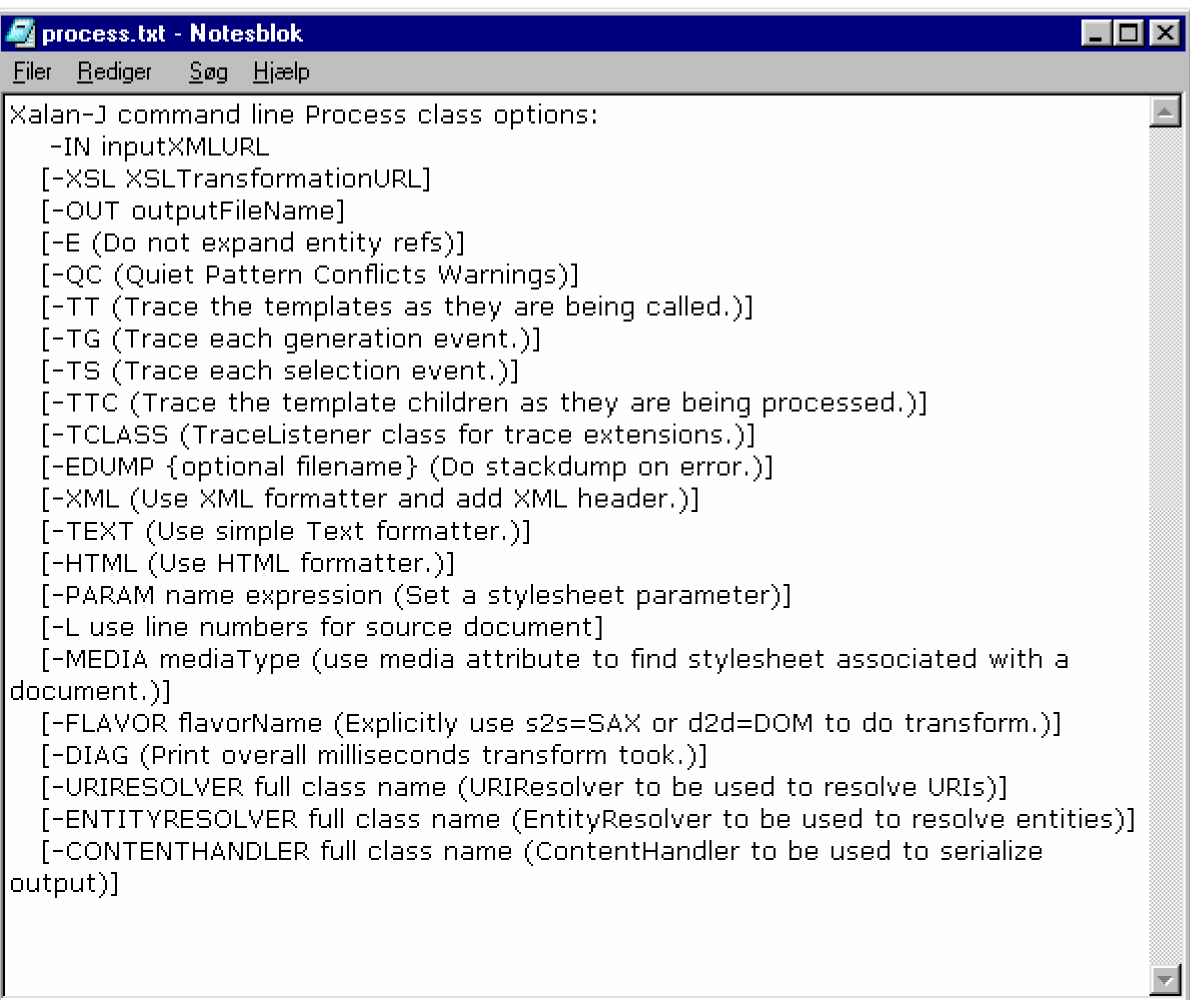


XALAN processoren:

XALAN processoren som vi omtalte i sidste afsnit er egentligt en del af Xerces parserne som kan downloades fra http://www.apache.org. Man downloader herfra en samling af Java klasser. Hvis – i stedet for Windows udgaven som vi brugte i sidste afsnit – bruger den egentlige Java klasse skal man bruge klassen Process. Man kan køre Xerces XALAN processoren ved f. eks. at skrive en lille .bat fil der starter processoren på denne måde:

```
java -classpath .;xercesImpl.jar;c:\java\java\lib\classes.zip org.apache.xalan.xslt.Process
```

Java fortolkeren skal altså kunne finde classes.zip og xercesImpl.jar klasserne! Hvis vi kører dette bat program får vi udskrevet de mange muligheder der er for at bruge Xalan stylesheet processoren:


process.txt - Notesblok

Filer Rediger Søg Hjælp

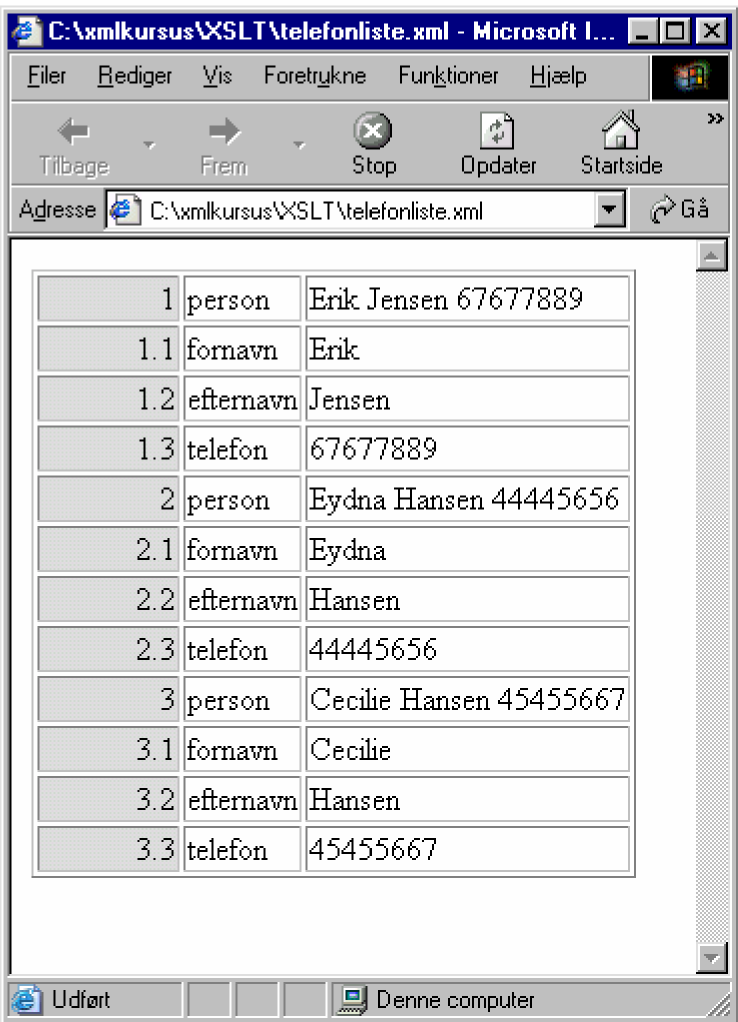
```
Xalan-J command line Process class options:
    -IN inputXMLURL
   [-XSL XSLTransformationURL]
   [-OUT outputFileName]
   [-E (Do not expand entity refs)]
   [-QC (Quiet Pattern Conflicts Warnings)]
   [-TT (Trace the templates as they are being called.)]
   [-TG (Trace each generation event.)]
   [-TS (Trace each selection event.)]
   [-TTC (Trace the template children as they are being processed.)]
   [-TCLASS (TraceListener class for trace extensions.)]
   [-EDUMP {optional filename} (Do stackdump on error.)]
   [-XML (Use XML formatter and add XML header.)]
   [-TEXT (Use simple Text formatter.)]
   [-HTML (Use HTML formatter.)]
   [-PARAM name expression (Set a stylesheet parameter)]
   [-L use line numbers for source document]
   [-MEDIA mediaType (use media attribute to find stylesheet associated with a
document.)]
   [-FLAVOR flavorName (Explicitly use s2s=SAX or d2d=DOM to do transform.)]
   [-DIAG (Print overall milliseconds transform took.)]
   [-URIRESOLVER full class name (URIResolver to be used to resolve URIs)]
   [-ENTITYRESOLVER full class name (EntityResolver to be used to resolve entities)]
   [-CONTENTHANDLER full class name (ContentHandler to be used to serialize
output)]
```

Vi kan altså transformere et XML dokument på denne måde med Xerces Xalan:

java -classpath .;xercesImpl.jar;c:\java\java\lib\classes.zip org.apache.xalan.xslt.Process -IN eks8.xml -XSL kun_text.xsl -OUT eks8.txt

Nummerede lister:

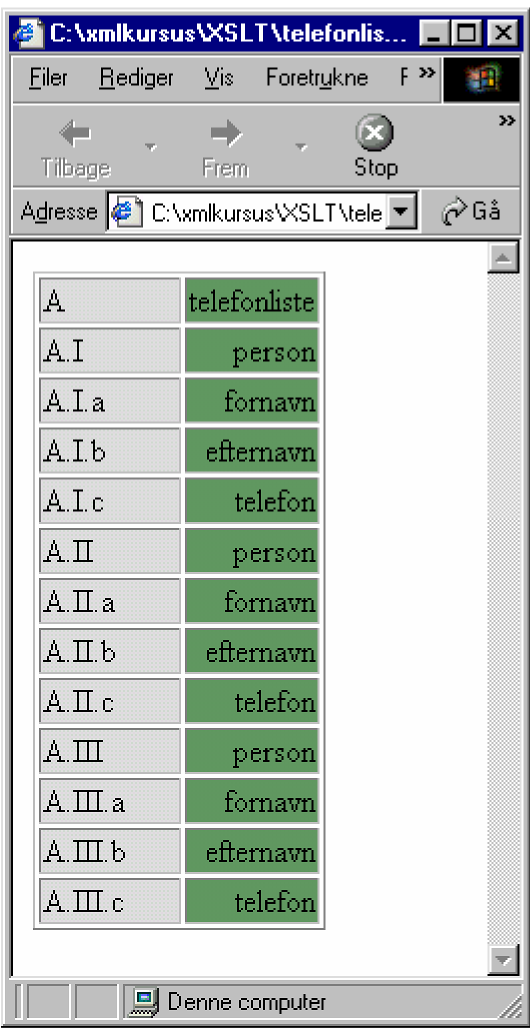
Ved hjælp af et element XSLT xsl:**number** kan vi oprette nummerede lister f. eks. over indholdet af XML dokumentet. xsl:number har en række attributter som kan bruges til at kontrollere hvordan vi vil nummerere eller tælle. Vi kan også på denne måde vise den hierarkiske struktur i XML dokumentet:



1	person	Erik Jensen 67677889
1.1	fornavn	Erik
1.2	efternavn	Jensen
1.3	telefon	67677889
2	person	Eydna Hansen 44445656
2.1	fornavn	Eydna
2.2	efternavn	Hansen
2.3	telefon	44445656
3	person	Cecilie Hansen 45455667
3.1	fornavn	Cecilie
3.2	efternavn	Hansen
3.3	telefon	45455667

Vi kan se at XSL processoren tæller på den rigtige måde! Tallene viser præcist strukturen i dokumentet.

Elementet number kan formateres på mange forskellige måder. Hvis vi i stedet for ønsker romertal kan vi også få det – her kombineret med en optælling eller nummering med små bogstaver!:



Her har vi taget et højere niveau med nemlig roden i dokumentet – telefonliste! Når vi vil nummerere afgør vi hvilke elementer der skal med i nummereringen. Stil arket kan skrives på denne måde:

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

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>
<xsl:output
method="html" indent="yes"
/>
<xsl:template match="/">
<table border="1">

    <xsl:for-each select="telefonliste|.//person|.//fornavn|.//efternavn|.//telefon">
    <tr><td style="text-align:left;width:50pt;background-color:#dedede">
    <!--
    <xsl:number level="multiple" count="person|fornavn|efternavn|telefon"

    format="1.1.1.1" />
    -->
    <xsl:number level="multiple" count="telefonliste|person|fornavn|efternavn|telefon"
```

```
format="A.I.a.1" />

</td >
<td style="text-align:right;background-color:#669966">
<xsl:value-of select="name()" />
</td>
<!--
<td>
<xsl:value-of select="." />
</td>
-->
</tr>

 </xsl:for-each>
</table>

</xsl:template>


</xsl:stylesheet>
```

Et number i XSLT kan sættes på forskellige måder med meget forskellige resultater:

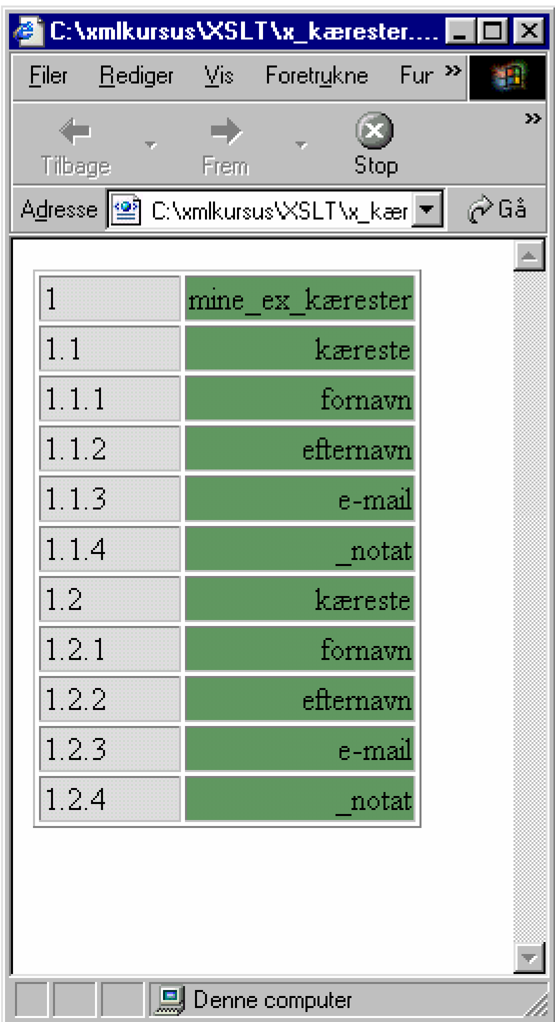
1. count definerer de elementer (attributter) som skal tælles
2. level afgør om vi ønsker flere niveauer eller kun eet (multiple, single)
3. from definerer hvor tælleriet skal starte
4. format bestemmer symbolet for nummereringen: 1.1.1, 1, A, a, I, i

Hvis man vil kan man nummerere også med det danske alfabet (æ, ø, å) eller med hebræiske tegn! Man kan starte en nummerering f. eks. med nummer 1000 ved simpelt hen at skrive tallet i format-strengen!

Man kan gøre dette stylesheet helt generelt så det kan style et hvilket som helst XML dokument og vise dets hierarkiske struktur. Vi skal blot foretage små ændringer:

```
<xsl:for-each select="*|./*|./*/*|./*/*/*">
<tr><td style="text-align:left;width:50pt;background-color:#dedede">
<xsl:number level="multiple" count="*|@*"
format="1.1.1.1" />
</td >
<td style="text-align:right;background-color:#669966">
<xsl:value-of select="name()" />
</td>
<!--
<td>
<xsl:value-of select="." />
</td>
-->
</tr>
 </xsl:for-each>
```

Vi søger nu helt abstrakt på de øverste elementer i dokumentet! På samme måde kunne inddrages attributter. Stylesheet’et kan nu anvendes på ethvert XML dokument:



Integration af XSLT og Cascading Stylesheets:

CSS og XSLT er helt forskellige teknologier og de kan sagtens bruges hver for sig. Men de har hver deres begrænsninger. CSS kan ikke vise attributter – XSLT har meget få muligheder for at formatere teksten i et output dokument! Vi kan bruge de to teknologier på kryds og tværs og vi har set mange eksempler at XSLT bruger CSS til formatering. Dette skyldes primært at stort set alle browsere i dag 'forstår' CSS.

Nedenstående eksempel er et lidt anderledes eksempel der viser hvordan vi også kan integrere de to teknikker.

Vi bruger her en telefonliste som XML dokument. Dette XML dokument bliver så stylet med dette XSLT script:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
```

```
<html>
<head>
   <style>
      table {font-size:16pt;color:blue}
      .x {background-color:red}
   </style>
   <link rel="stylesheet" type="text/css" href="css_xsl.css" />
</head>
<body>
   <table border="1">
   <xsl:for-each select="//person">
      <tr><td>    <xsl:value-of select="fornavn" /></td></tr>
      <tr><td>    <xsl:value-of select="efternavn" /></td></tr>
      <tr><td class="x"><xsl:value-of select="telefon" /></td></tr>
   </xsl:for-each>
   </table>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Der er ikke så mange nye ting her vi ikke har set i forvejen. Men i dette tilfælde skriver XSLT simpelt hen en <style> sektion ind i <head> som man vil gøre det i HTML. I CSS kan man style på 3 forskellige niveauer:

1. Med et eksternt stylesheet (laveste prioritet)
2. Med en style sektion i <head> (højere prioritet)
3. Med en lokal konkret style (højeste prioritet).

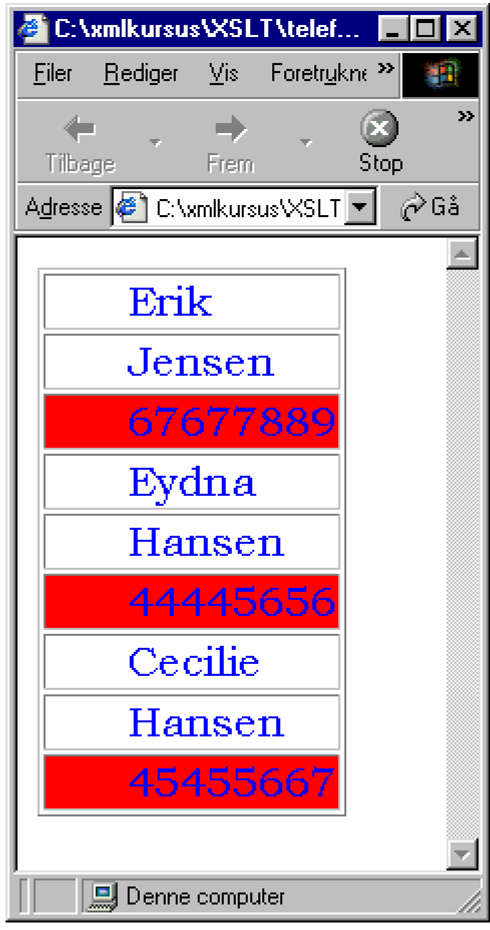
Disse styles er 'cascading' d.v.s. de ned arves fra parent! En style f. eks. i et eksternt stylesheet kan altså 'overrides' eller omdefineres senere!

Her refererer vi til filen css_**xsl**.css som kun rummer en linje:

```
*{font-family:'Bookman Old Style';text-indent:30pt}
```

Denne angiver at ALLE elementer i dokumentet skal formateres med denne skrifttype og med en indrykning!

Ved hjælp af XSLT og CSS kan vi nu transformere til dette dokument:



Vi kan se at vi i virkeligheden kan **blande** elementer fra XML, CSS og HTML som vi har brug for!

Eksempel på transformation: XML -> RTF dokument:

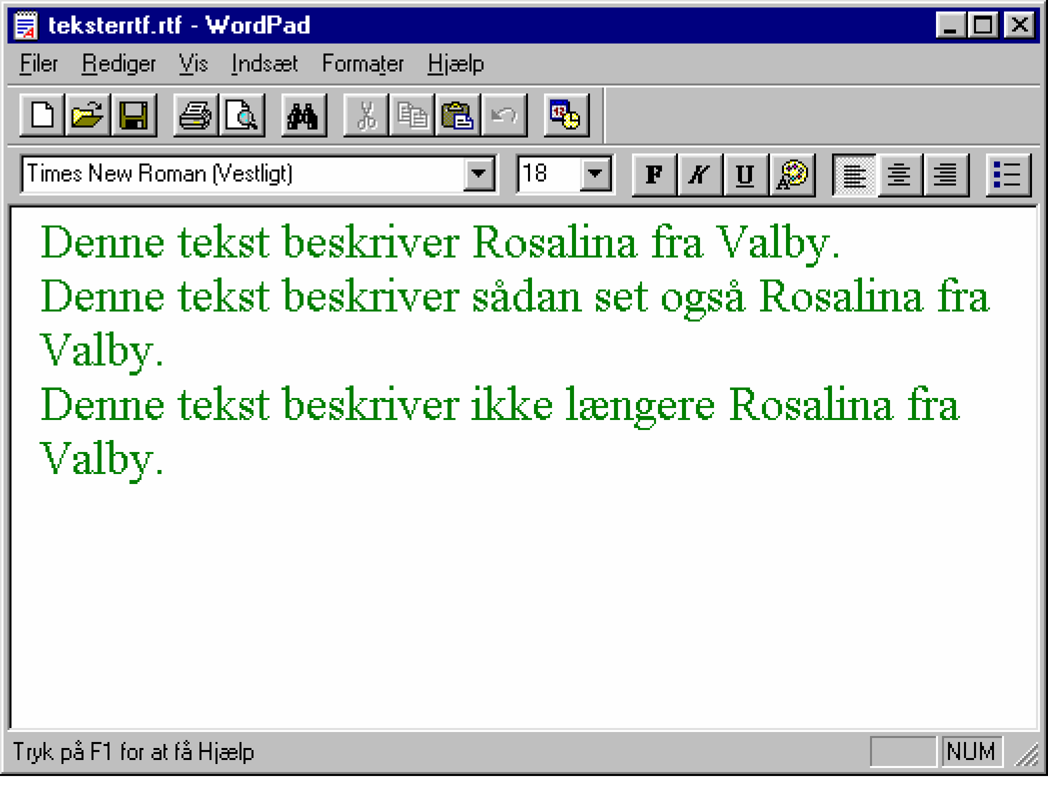
RTF dokumenter er **Rich Text** dokumenter der kan formateres med farver og forskellige skrifttyper. I nogle tilfælde kan det være interessant at transformere et XML dokument til et RTF dokument. Dette sker igen grundlæggende ved at vi sørger for at indsætte **literale** tekst data som normalt bruges i et RTF dokument! Ellers kan vi blot hente informationerne fra et XML dokument!

Vi kan starte med dette simple XML dokument:

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

<tekster>
```

```
    <tekst>
    Denne tekst beskriver Rosalina fra Valby.
    </tekst>
    <tekst>
    Denne tekst beskriver sådan set også Rosalina fra Valby.
    </tekst>
    <tekst>
    Denne tekst beskriver ikke længere Rosalina fra Valby.
    </tekst>
</tekster>
```

Slut resultatet skulle gerne ser nogenlunde sådan ud:



Vi kan **transformere** XML dokumentet til et **RTF** dokument således:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output
method="text"
encoding="iso-8859-1"
omit-xml-declaration="yes"
indent="yes"
/>

<xsl:template match="tekster">

{\rtf1\ansi\ansicpg1252\deff0\deflang1030{\fonttbl{\f0\fnil\fcharset0 Times New Roman;}}
{\colortbl ;\red0\green128\blue0;}
<xsl:apply-templates select="tekst" />
}
```

```
</xsl:template>

<xsl:template match="tekst">

\viewkind4\uc1\pard\cf1\f0\fs36 <xsl:value-of select="." />\cf0\fs20\par
</xsl:template>


</xsl:stylesheet>
```

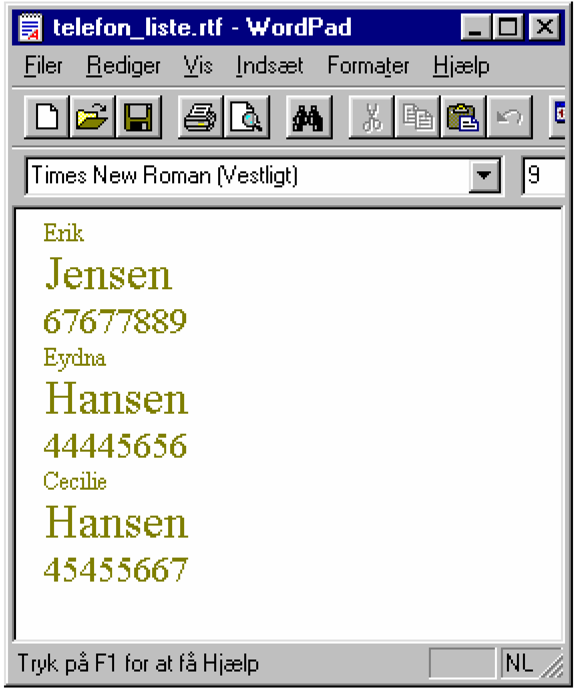
De mange koder er – stort set - alle nødvendige for at formatere RTF dokumentet! Koderne definerer **farver, skriftstørrelsen** og skrift **typen**.

Her er defineret Times New Roman, farven grøn og en skrift størrelse på 18 punkt (nemlig 36 delt med to!).

RTF filen skal altså dels have et **hovede** som kun skal stå en gang og dels en række **paragraffer** eller afsnit som indledes med linjeskift (som i en tekst behandling)!

Det er vigtigt for at denne proces skal lykkes at man transformerer til en tekst fil som f. eks. kaldes fil.**rtf**. Derefter skal denne RTF fil åbnes f. eks. i **Wordpad** og **gemmes** som **tekst** type og **samtidigt** med **filtypen** .rtf altså som **dokument**.**rtf** f. eksempel!

Det er en lidt kompliceret proces, men det kan lade sig gøre og når først stylesheet'et **er** skrevet een gang er det **let** at bygge videre!



Eksempel på transformation: XML -> Recordset:

Eksempel på transformation: XML -> SVG:

SVG er et XML sprog som kan bruges til at definere grafiske elementer. SVG har sin egen DOCTYPE (dtd) som definerer hvad der er et legalt SVG dokument. Vi kan transformere et XML dokument til et SVG dokument f. eks. for at vise data grafisk!

Et simpelt eksempel på et XML dokument kunne være:

```
<?xml version="1.0"?>
<?xml-stylesheet href="resultater_svg.xsl" type="text/xsl" ?>
<data>
<rubrik>Forskellige overskud i firmaet.</rubrik>
<liste>
	<resultat>230</resultat>
	<resultat>130</resultat>
	<resultat>120</resultat>
	<resultat>210</resultat>
	<resultat>530</resultat>
	<resultat>430</resultat>
	<resultat>630</resultat>
	<resultat>130</resultat>
	<resultat>370</resultat>
</liste>
</data>
```

Det vi så ønsker at få vist disse overskud i et **søjlediagram**. Vi kan så skrive dette stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output
method="xml"
encoding="iso-8859-1"
omit-xml-declaration="no"
indent="yes"
doctype-public="-//W3C//DTD SVG 1.0//EN"
doctype-system="http://localhost/svg10.dtd"
/>

<xsl:template match="/">

<svg xmlns="http://www.w3.org/2000/svg" width="400" height="300">

<xsl:for-each select="//resultat">
<rect x="{(position()*50)-50}" y="50" width="48" height="{. div 2}" >
<xsl:attribute name="style">
<xsl:choose>
<xsl:when test="position() mod 2 = 0">
fill:#669966
</xsl:when>
<xsl:otherwise>
fill:#996666
</xsl:otherwise>
</xsl:choose>

</xsl:attribute>
```

```
</rect>
</xsl:for-each>
</svg>
</xsl:template>

<xsl:template match="person">
</xsl:template>


</xsl:stylesheet>
```

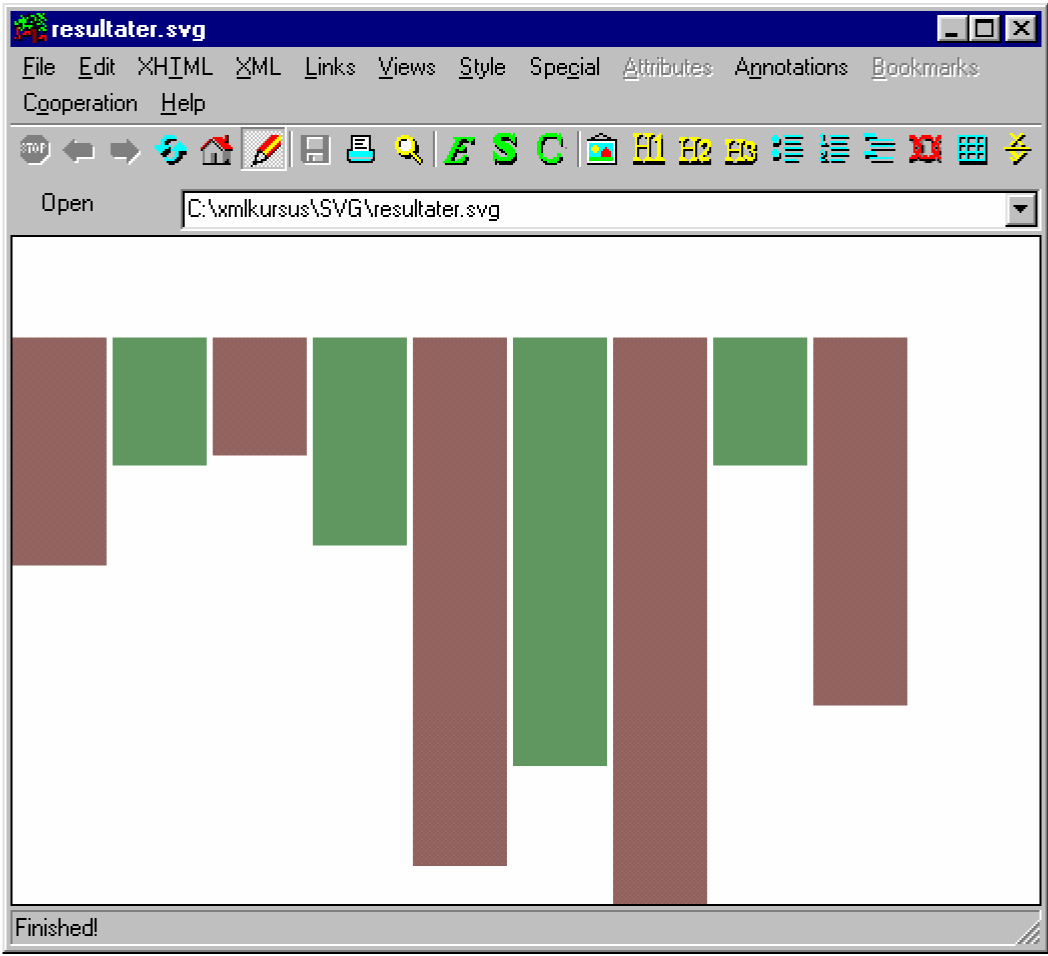
Vi kan som vist indsætte en DOCTYPE i xsl:output – på den måde kan vi også validere at det producerede dokument er gyldigt! Her går vi ud fra at DTD'en gemt lokalt!

```
doctype-public="-//W3C//DTD SVG 1.0//EN"
doctype-system="http://localhost/svg10.dtd"
```

SVG dokumenter skal have en bestemt rod og et bestemt namespace for at være gyldige. dem indsætter vi som direkte tekst i output træet:

```
<svg xmlns="http://www.w3.org/2000/svg" width="400" height="300">
```

Vi viser blot her søjlerne på den **letteste** måde – dette stylesheet kunne let **forbedres** ved at gøre højde og bredde af søjlerne variabel!:



Filen resultater.svg kan vises i f. eks. browseren Amaya fra http://www.w3.org og selve filen kan f. eks. produceres med msxsl:

msxsl resultater.xml –pi –o resultater.svg

Eksempel på transformation: SVG -> VML:

Vi har andetsteds omtalt **SVG** og **VML** som er to **XML** sprog der bruges til at definere vektor grafik. VML er et Markup Language som kun bruges af **Microsoft** produkter – bl. a. Office pakken. Grundlæggende er VML en simplere – og tidligere – udgave af SVG.

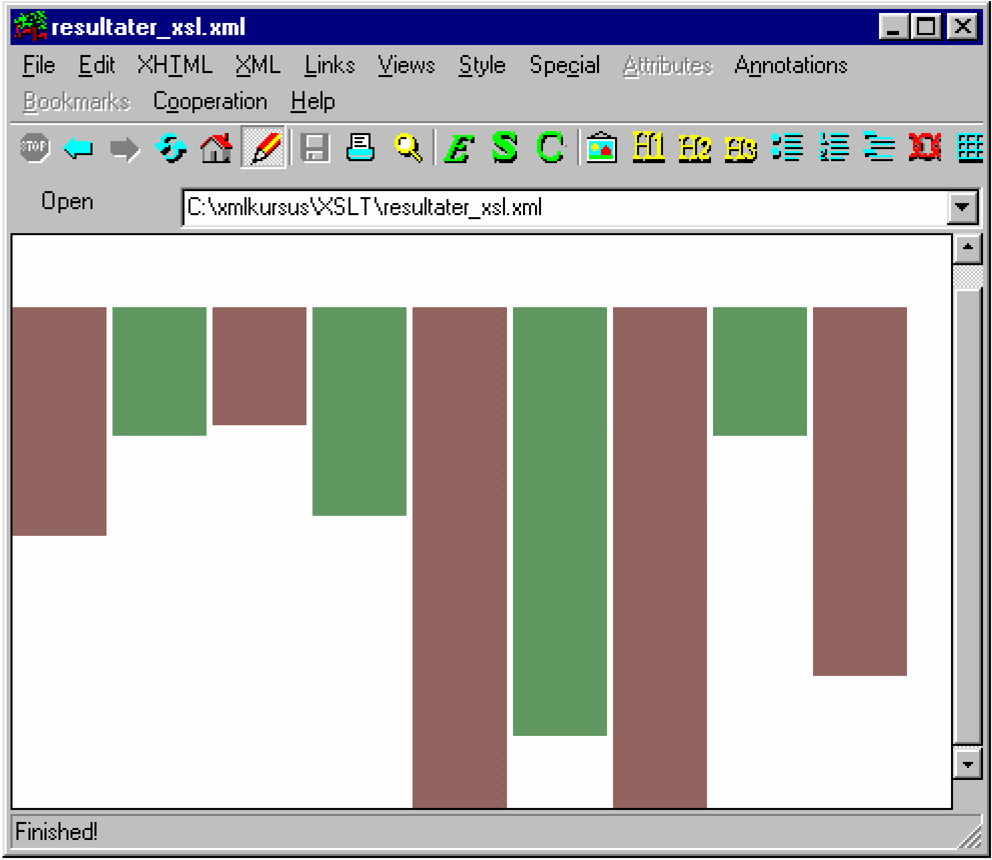
Vi vil i dette eksempel se på hvordan man kan **transformere** et SVG dokument til et VML format.

Vores **SVG** dokument ser således ud:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://localhost/svg10.dtd">
<s:svg width="400" height="300" xmlns:s="http://www.w3.org/2000/svg">
<s:rect x="0" y="50" width="48" height="115" style="fill:#996666" />
```

```
<s:rect x="50" y="50" width="48" height="65" style="fill:#669966" />
<s:rect x="100" y="50" width="48" height="60" style="fill:#996666" />
<s:rect x="150" y="50" width="48" height="105" style="fill:#669966" />
<s:rect x="200" y="50" width="48" height="265" style="fill:#996666" />
<s:rect x="250" y="50" width="48" height="215" style="fill:#669966" />
<s:rect x="300" y="50" width="48" height="315" style="fill:#996666" />
<s:rect x="350" y="50" width="48" height="65" style="fill:#669966" />
<s:rect x="400" y="50" width="48" height="185" style="fill:#996666" />
</s:svg>
```

Vi kunne have skrevet dokumentet lidt lettere – med tanke på at det skal transformeres til VML! – men vi har valgt denne form! Dokumentet er gemt som resultater.xml. Det kan vises i **Amaya** således:



Hvis vi ønsker det vist f. eks. i browseren Mozilla eller i Internet Explorer fås:

Mozilla

File Edit View Go Bookmarks Tools Window Help

Back Forward

Home Bookmarks

```
- <s:svg width="400" height="300">
    <s:rect x="0" y="50" width="48" height="115"
    style="fill:#996666"/>
    <s:rect x="50" y="50" width="48" height="65"
    style="fill:#669966"/>
    <s:rect x="100" y="50" width="48" height="60"
    style="fill:#996666"/>
    <s:rect x="150" y="50" width="48" height="105"
    style="fill:#669966"/>
    <s:rect x="200" y="50" width="48" height="265"
    style="fill:#996666"/>
    <s:rect x="250" y="50" width="48" height="215"
    style="fill:#669966"/>
    <s:rect x="300" y="50" width="48" height="315"
    style="fill:#996666"/>
    <s:rect x="350" y="50" width="48" height="65"
    style="fill:#669966"/>
```

Vi kan nu transformere SVG dokumentet til **VML** med dette stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:s="http://www.w3.org/2000/svg"
xmlns:vml="urn:schemas-microsoft-com:vml"
>
<xsl:output method="xml" omit-xml-declaration="yes" indent="yes" />

<xsl:template match="/">

<html xmlns:vml="urn:schemas-microsoft-com:vml" >
    <head>
        <style>
        vml\:*{behavior:url(#DEFAULT#VML);}
        </style>
        </head>
    <body>
        <xsl:apply-templates />
    </body>
</html>
</xsl:template>
```

```
<xsl:template match="//s:rect">
<vml:rect fillcolor="{substring-after(@style,'fill:')}" style="width:{@width};height:{@height};top:{@y};left:{@x}"/>
</xsl:template>

</xsl:stylesheet>
```

Princippet er altså meget simpelt: **Hver** gang XSL **processoren** finder et element s:**rect** skal den oprette et nyt element ved navn **vml**:rect!

Det mest besværlige her i virkeligheden at vi skal have fat i **farven** som er en del af en style i SVG dokumentet. Derfor bruger vi XSLT (og XPATH) funktionen **substring**-after for at hente det som kommer efter 'fill:' inden i strengen!

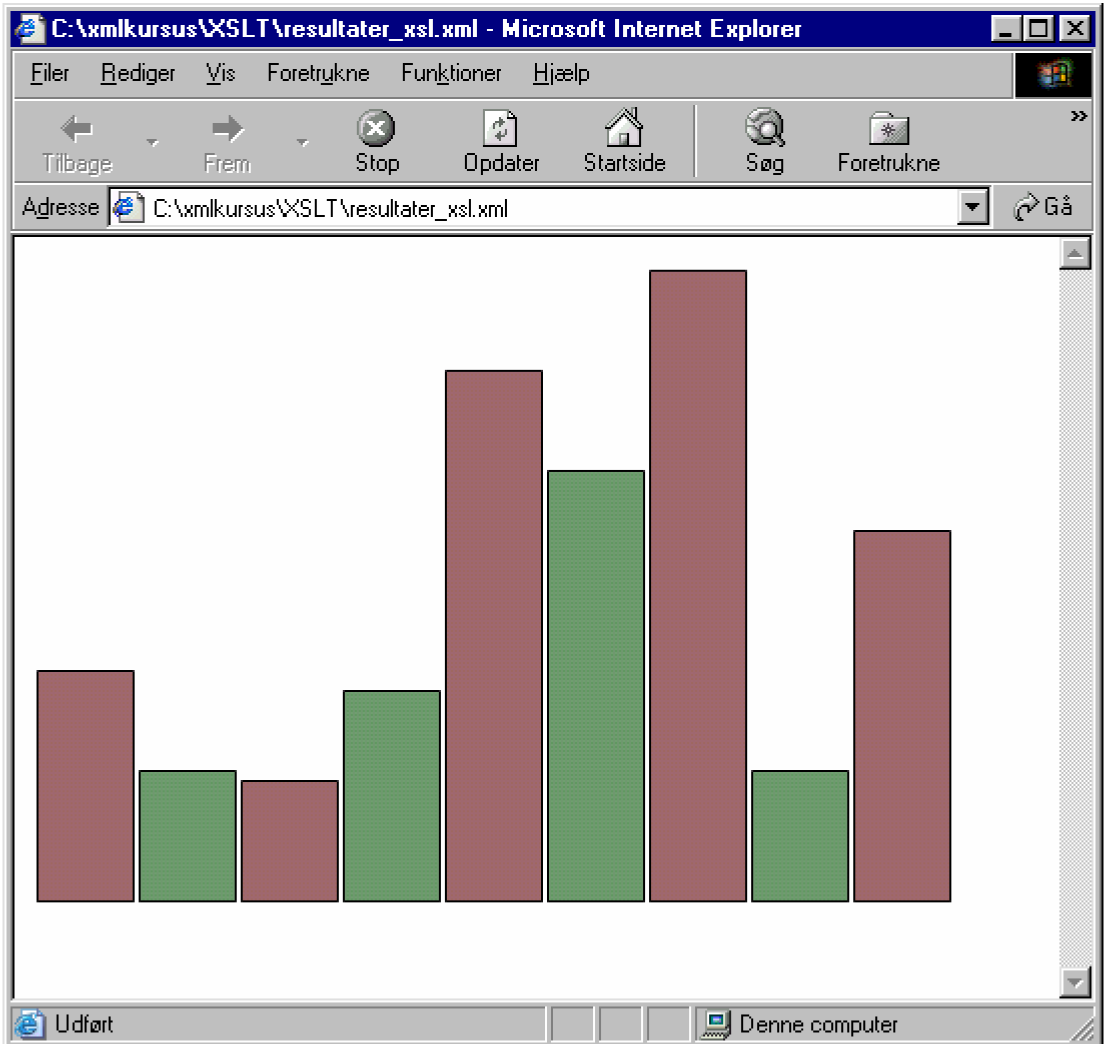
Vi kan se at transformationen i meget høj grad består i at vi indsætter literal tekst i output træet. Dette gælder også <head> som skal indeholde disse VML erklæringer. Vi importerer de to namespaces for VML og SVG og kan på den måde bruge dem i stylesheetet! Vi oversætter derefter værdierne fra SVG til VML ved hjælp af såkaldte attribut value templater – jvf brugen af { og }.

Dette stylesheet kunne gøres noget mere fleksibelt hvis vi inddrog flere elementer – men her har vi blot illustreret processen med et element nemlig rect!

Hvis vi sætter en stylesheet erklæring på XML filen således:

```
<?xml version="1.0"?>
<?xml-stylesheet href="svg_vml.xsl" type="text/xsl"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://localhost/svg10.dtd">
<s:svg width="400" height="300" xmlns:s="http://www.w3.org/2000/svg">
.....
```

kan vi få grafen vist i Internet Explorer:



Vi kan stadig vise XML dokumentet (SVG dokumentet) i Amaya – selv om det nu har fået et stylesheet. Men hvis vi transformerer XML dokumentet til et nyt HTML dokument – kan Amaya ikke vise denne fil!

Mozilla kan heller ikke vise XML (SVG) dokumentet efter at det har fået et stylesheet. Mozilla forstår ikke koderne til VML!

Eksempel på transformation: XML -> XML:

Eksempel på transformation: XML -> PDF:

Vi vil i dette kursus ikke komme meget ind på XSL-FO eller Formatting Objects, som er et XML baseret formateringssprog. FO er ment som et alternativ til CSS især men er langt mere indviklet og langt mere virkningsfuldt! FO er en W3C standard som definerer hvordan man kan opsætte sider og dokumenter til trykning og publishing.

En browser kan normalt ikke vise FO dokumenter så normalt skal XML dokumenter altså først transformeres til XSL-FO og derefter endnu en gang transformeres til f. eks. PDF formatet som er et anerkendt dokument format som f. eks. kan vises i Adope Acrobat Reader! Men FO kan også transformeres til andre dokument og publishing formater.

XSL-FO definerer **meget præcist** hvordan en tekst skal vises og formateres. Som et konkret og rimeligt kortfattet eksempel kan vi se på dette **XSL-FO** dokument:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
   <fo:simple-page-master master-name="nr1"
      margin-right="4cm"
      margin-left="4cm"
      page-width="20cm"
      page-height="20cm">
      <fo:region-body />
   </fo:simple-page-master>
   </fo:layout-master-set>

   <fo:page-sequence master-reference="nr1">
      <fo:flow flow-name="xsl-region-body">
         <fo:block color="#060606" font-size="16pt" font-family="Helvetica">
         Dette er et afsnit skrevet som et eksempel i XSL-FO Formatting Objects. Dette er et afsnit skrevet som et
eksempel i XSL-FO Formatting Objects.Dette er et afsnit skrevet som et eksempel i XSL-FO Formatting Objects.
         </fo:block>
         <fo:block text-align="center" padding="1cm" font-size="12pt" border="1pt solid black" background-
color="#dedede">
         Dette er et afsnit skrevet som et eksempel i XSL-FO Formatting Objects. Dette er et afsnit skrevet som et
eksempel i XSL-FO Formatting Objects.Dette er et afsnit skrevet som et eksempel i XSL-FO Formatting Objects.
         </fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

Vi kan først og fremmest se at dokumentet er et XML dokument og at det overholder alle XMLs krav om at være velformet!

Elementerne fo:root osv. skal have disse navne og typer i et FO dokument! Roden skal altså altid hedde fo:root! Et FO dokument **skal** have et layout-master-set!

Vi kan se at strukturen i et FO dokument består af en **body** (til sidst) der befinder sig inden i en **page-sequence**! En page-sequence skal henvise til en bestemt **master**! **Masteren** – som defineres først – indeholder **generelle** definitioner af sidens udseende - f. eks. dens marginer - som vist her.

Man kan **downloade** FO specifikationen fra http://www.w3.org - og det ER nødvendigt hvis man skal kunne finde rundt i de mange formaterings muligheder!

Selve body eller teksten vises her i en fo:**block** som betyder at blokken starter med et **linjeskift** (lige som <p> i HTML eller display:block i CSS).

En fo:**block** kan defineres med et utal af **attributter**! Men mange af disse attributter er heldigvis de samme som anvendes i Cascading Stylesheets **CSS**!

Dette dokument – **eksempel**.fo – kan så f. eks. transformeres til PDF formatet som kan vise de mange formaterings angivelser.

For at gennemføre denne transformation er det nødvendigt at have et særligt **værktøj**. Der findes **mange** forskellige såkaldte FOP værktøjer som kan downloades fra Internettet.

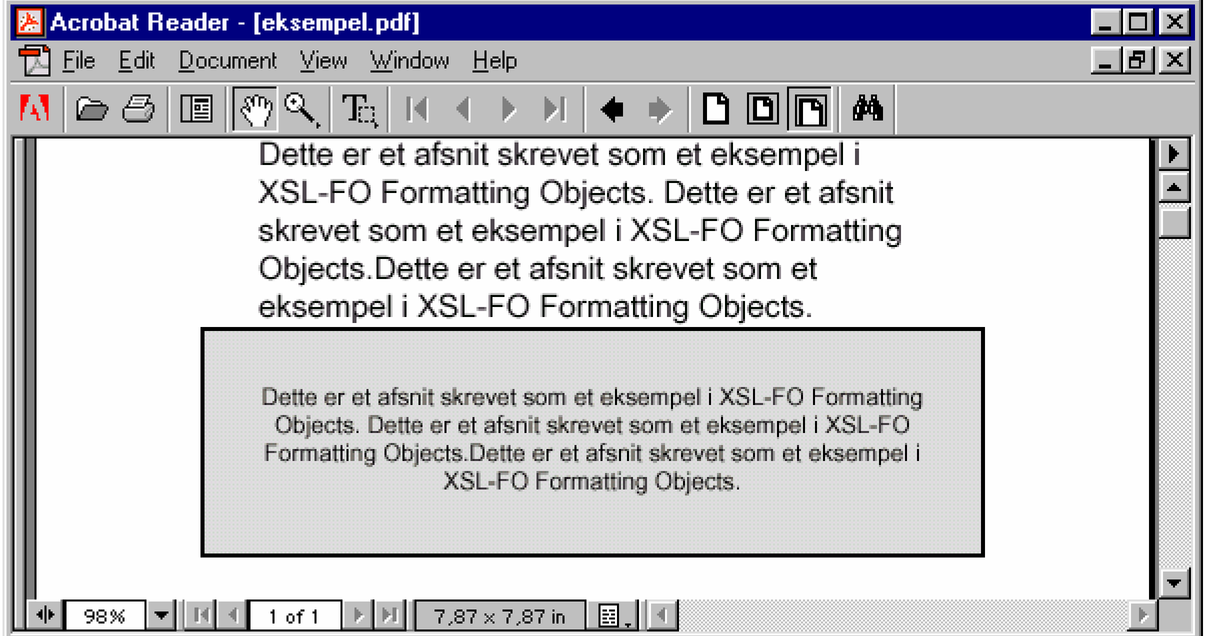
Som et eksempel kan fra adressen http://www.renderx.com downloades værktøjet XEP i en prøve version. XEP - XSL Formatting Engine for Paged Media – kan transformere til **forskellige** formater. Der medfølger mange eksempler og megen dokumentation!

OBS: XEP kræver en speciel licens - som **også** skal downloades!

Vi kan køre **transformationen** med følgende .bat fil når vi har installeret XEP på maskinen. XEP kræver **også Java** biblioteket installeret på maskinen! Den klasse som kaldes er **XSLDriver**:

```
java -classpath .;xep.jar;c:\java\java\classes.zip com.renderx.xep.XSLDriver -fo eksempel.fo -pdf eksempel.pdf
```

Med denne kommando opretter vi **eksempel**.pdf:



PDF formatet egner sig til visning som her i **Acrobat Reader** – men især til **trykte** medier! PDF formatet er et anerkendt – **uafhængigt** format som er portable – uafhængigt af maskine og styresystem!

XEP kan også kontrollere om FO dokumenter er gyldige med klassen – programmet – **Validator**:

```
java -classpath .;xep.jar;c:\java\java\classes.zip com.renderx.xep.Validator eksempel.fo
```

Når en transformation køres med **XSLDriver** fås altid en tilstands rapport som denne:

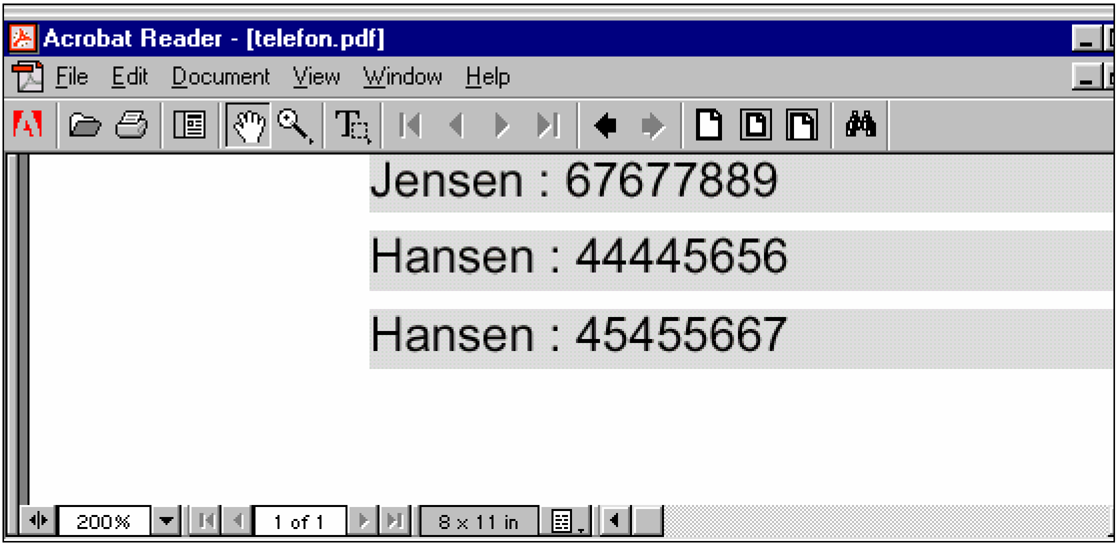
Udført - xep_eksempel

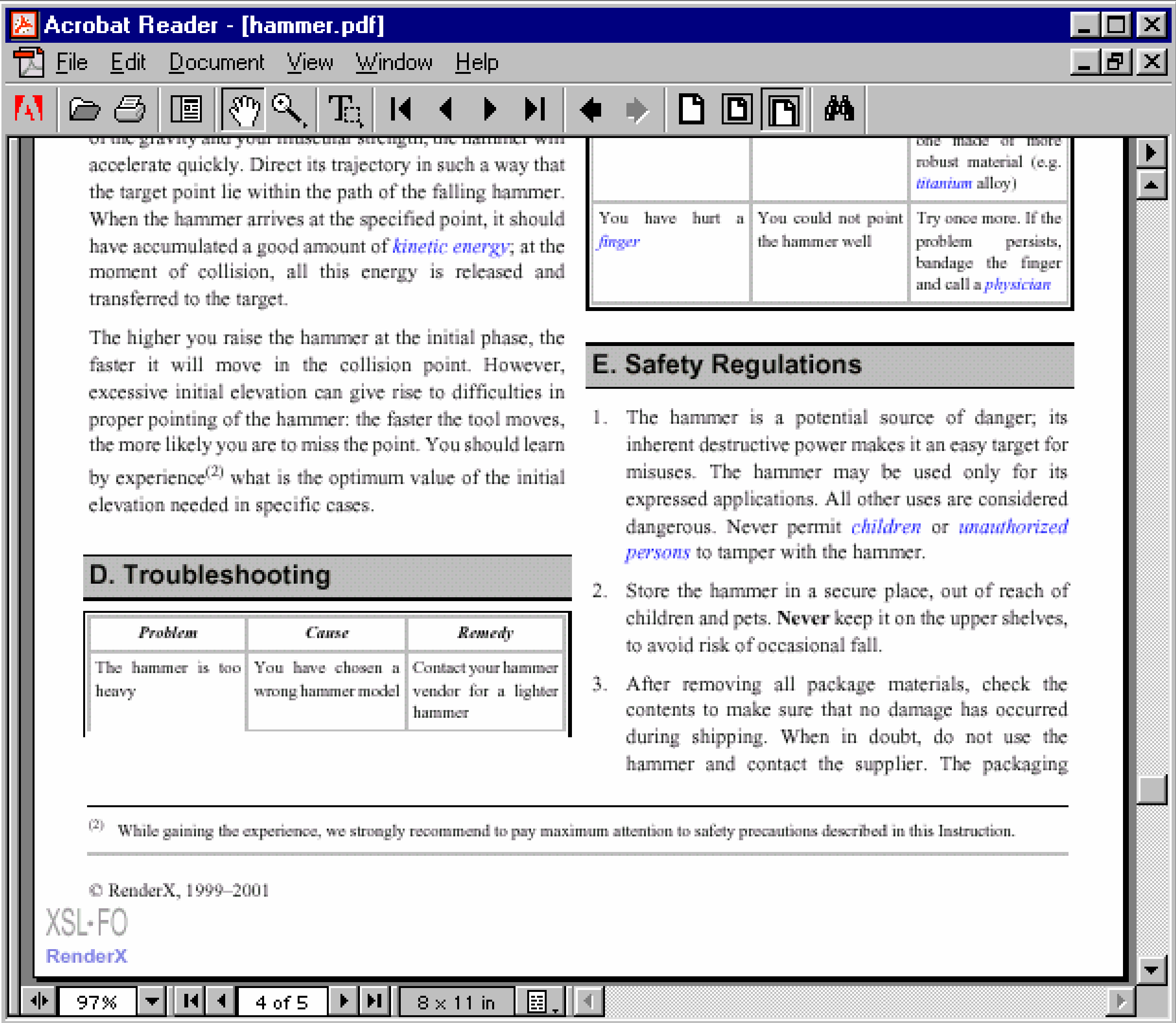
```
C:\xmlkursus\XEP>java -classpath .;xep.jar;c:\java\java\classes.zip com.renderx.
xep.XSLDriver -fo eksempel.fo -pdf eksempel.pdf
(document
(validate [validation OK])
[system-id file:/C:/xmlkursus/XEP/eksempel.fo]
(compile (masters (sequence-master [master-name nr1]))
(sequence [master-reference nr1](flow [flow-name xsl-region-body])))
(format
(sequence [master-reference nr1](flow [page-number 1])
(static-content [page-number 1])
))
(generate [output-format pdf][page-number 1]))
```

Man kan her følge de enkelte trin i transformationen. Transformationen vil selv sørge for at fordele teksten – hvis den er tilstrækkelig lang – på de korrekte sider!

Der følger en række **fonts** eller skrifttyper med XEP og man kan selv definere og registrere sine egne skrifttyper!

Vi vil ikke her demonstrere hvordan man kan bruge et XSLT script til at producere et FO dokument. Blot kan det ses at det meste er **literal** tekst – som i de tidligere eksempler! Men det er en god ide at skrive et **XSLT** stylesheet der kan transformere et XML dokument til en FO tekst således at man kan **automatisere** produktionen af FO og dermed af PDF dokumenter! Ellers kan processen blive lidt langtrukken!




Acrobat Reader - [hammer.pdf]

File Edit Document View Window Help

accelerate quickly. Direct its trajectory in such a way that the target point lie within the path of the falling hammer. When the hammer arrives at the specified point, it should have accumulated a good amount of *kinetic energy*; at the moment of collision, all this energy is released and transferred to the target.

The higher you raise the hammer at the initial phase, the faster it will move in the collision point. However, excessive initial elevation can give rise to difficulties in proper pointing of the hammer: the faster the tool moves, the more likely you are to miss the point. You should learn by experience(2) what is the optimum value of the initial elevation needed in specific cases.

D. Troubleshooting

Problem	*Cause*	*Remedy*
The hammer is too heavy	You have chosen a wrong hammer model	Contact your hammer vendor for a lighter hammer

		one made of more robust material (e.g. *titanium* alloy)
You have hurt a *finger*	You could not point the hammer well	Try once more. If the problem persists, bandage the finger and call a *physician*

E. Safety Regulations

1. The hammer is a potential source of danger; its inherent destructive power makes it an easy target for misuses. The hammer may be used only for its expressed applications. All other uses are considered dangerous. Never permit *children* or *unauthorized persons* to tamper with the hammer.
2. Store the hammer in a secure place, out of reach of children and pets. **Never** keep it on the upper shelves, to avoid risk of occasional fall.
3. After removing all package materials, check the contents to make sure that no damage has occurred during shipping. When in doubt, do not use the hammer and contact the supplier. The packaging

(2) While gaining the experience, we strongly recommend to pay maximum attention to safety precautions described in this Instruction.

© RenderX, 1999–2001

XSL-FO RenderX

97% 4 of 5 8 x 11 in

I et PDF dokument kan man meget præcist definere hvordan teksten og siden skal vises! Dette eksempel følger med pakken fra XEP renderx.com!

Eksempel på transformation: XML -> XSD:

Vi kan ved hjælp af XSLT **automatisk** producere et **XSD** skema ud fra et XML eksempel! Der findes **værktøjer** på nettet som kan gøre dette på en detaljeret og nogenlunde pålidelig måde – f. eks. XMLWriter.

Processen er ret kompliceret fordi et stylesheet 'logisk' skal **slutte** sig frem til skemaet! Men vi vil her kun se på et enkelt eksempel.

Vores mål er at starte med et XML dokument der ser således ud:

C:\xmlkursus\XSLT\xml_til_skema.xml - Microsoft Internet E...

Filer Rediger Vis Foretrukne Funktioner Hjælp

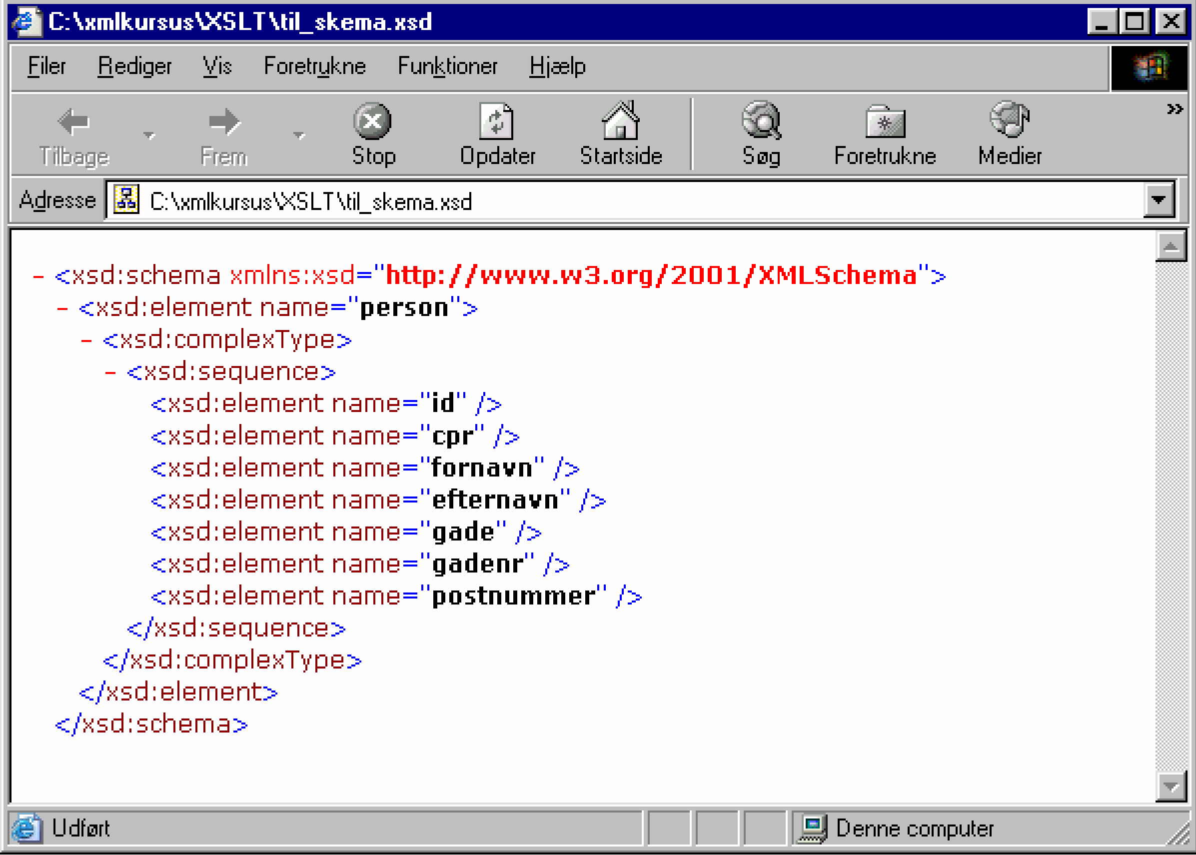
Tilbage Frem Stop Opdater Startside Søg

Adresse C:\xmlkursus\XSLT\xml_til_skema.xml Gå

```
<!-- <?xml-stylesheet type="text/xsl"
href="xml_til_skema.xsl"?>
  -->
- <person
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
    xsi:noNamespaceSchemaLocation="til_skema.xsd">
    <id />
    <cpr />
    <fornavn />
    <efternavn />
    <gade />
    <gadenr />
    <postnummer />
  </person>
```

Udført Denne computer

Skemaet til_skema.xsd skal så se sådan ud – som et minimum:



Dette skema **til_skema.xsd** kan vi producere ved hjælp af følgende XSLT **script**:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" encoding="iso-8859-1" omit-xml-declaration="yes" indent="yes" />

<xsl:template match="/*">

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

<xsd:element name="{name()}">
  <xsd:complexType>
  <xsd:sequence>
  <xsl:for-each select="/*/*">
     <xsd:element name="{name()}"></xsd:element>
  </xsl:for-each>
  </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>

</xsl:template>
</xsl:stylesheet>
```

Også her bruger vi simpelthen **bogstavelige** elementer som f. eks.:

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

Når vi kører vores transform bliver denne linje simpelthen **indskrevet** i output træet. Vi anvender attribut value templater til at hente værdien af name() på elementet i XML dokumentet.

Vi kører en løkke der varer lige så længe som der er sub elementer. Systemet her kan kun klare to niveau'er som det er skrevet her men det kan sagtens udbygges!

Hvis vi prøver at validere dokumentet – bliver det valideret OK!

Transformationer uden filer:

Det er vigtigt at forstå at en XSLT transformation kan foregå på et XML dokument der kun ligger i **hukommelsen** eller i RAM og at stylesheet'et heller ikke behøver at være en fil som er gemt f. eks. på harddisken! Det transformerede dokument kan også blot ligge i **RAM** – det behøver ikke at blive gemt som en fil.

Når en **browser** anvender et stylesheet til at vise et XML dokument bliver det transformerede dokument jo **ikke** gemt som en fil! Det bliver produceret og skrevet af en XSL processor og browseren viser dette RAM dokument! Det transformerede dokument bliver så produceret for **hver** gang browseren skal vise XML dokumentet!

Fordelen ved at gemme det transformerede dokument som en fast fil er derfor at nu kan browseren (eller et andet program) så blot **genbruge** det gemte dokument! D.v.s. at processen går lidt **hurtigere** – XSL processoren skal ikke databehandle de to dokumenter!

En XSL processor som f. eks. Microsofts **msxsl**.exe producerer også det transformerede dokument til RAM som standard! Hvis man blot skriver:

msxsl eksempel.xml –pi

transformerer **msxsl.exe** dokumentet med det stylesheet som findes som processing instruction i selve XML dokumentet – og resulatet bliver **blot** udskrevet på DOS linjen (**stdout**) – altså til **RAM**!

Transformationer med SAXON XSL processoren:

SAXON parseren og processoren 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.

For at kunne bruge disse klasser skal man altså have installeret en udgave af **Java** på maskinen. Forskellige udgaver af Java kan downloades fra mange steder f. eks. fra http://sun.com. Den store **fordel** ved at bruge Java klasser er at de fungerer på **alle** maskiner – Windows, Unix, Linux, Mac!

Med Saxon følger bl a. en række **extension** funktioner som f. eks. matematiske funktioner der ikke findes i XPATH eller XSLT. 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 medfølger i alt cirka 50 forskellige ekstra extension funktioner! Som et eksempel kan nævnes funktionen:

tokenize(string, string)

som opdeler en evt. lang tekst i tokens f. eks. opdeler den i ord adskilt af mellemrum! Funktionen returnerer så et **nyt** node sæt af ord!

En anden funktion er saxon:**output** som skriver output af transformationen til en bestemt fil. På den måde kan produceres mange dokumenter med udgangspunkt i **dele** af et XML dokument! Vi skal snart se et eksempel på dette.

Med SAXON følger også en stribe færdige programmer.

Vi skal i første omgang se på **hvordan** man helt konkret kan bruge SAXON XSL maskinen. 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 x=44

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 allerede har set.

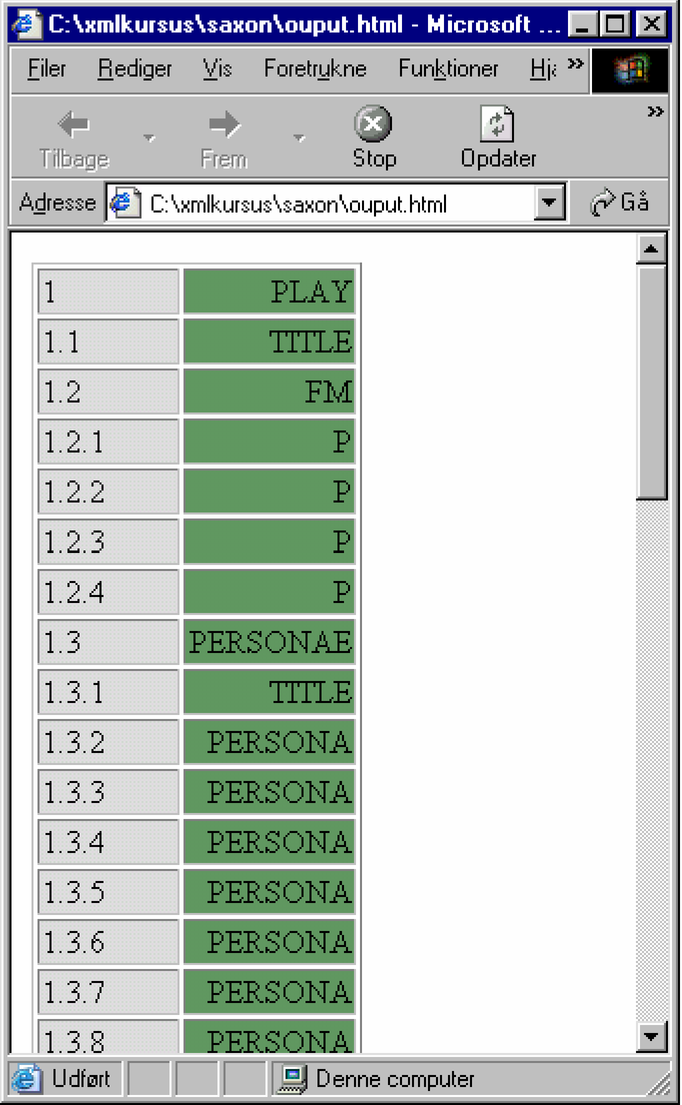
x=44 er KUN her indsat for at vise hvordan man kan indsætte eksterne parametre med SAXON.

Ovenstående **forudsætter** at saxon.jar biblioteket ligger i den **samme** mappe! Java fortolkeren skal kunne finde biblioteket.

Vi kalder 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 ovennævnte indhold (kommandoen) gemt som f. eks. style.**bat**! Man kan også anvende SAXON i **scripts**.

Resultatet af kommandoen er et HTML dokument:



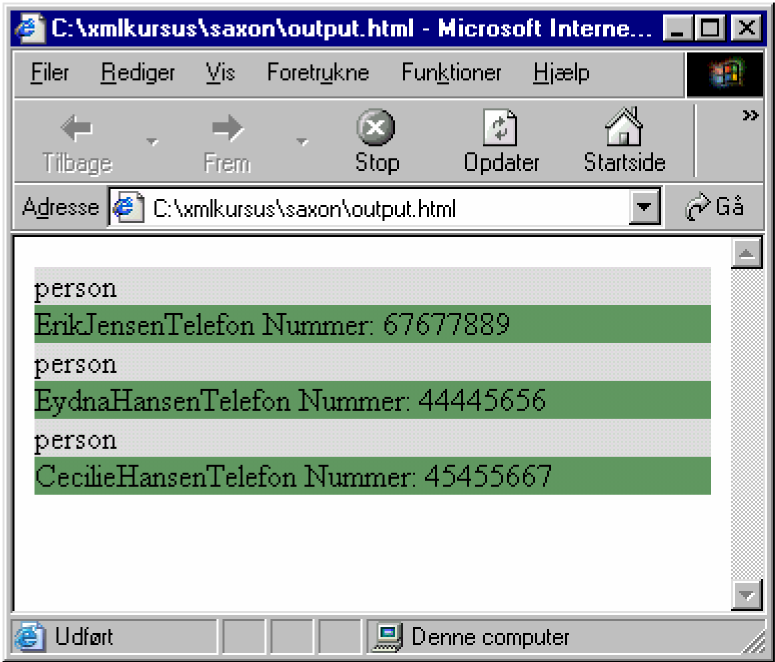
Kommandoen kan hente dokumenter alle steder fra også fra Internettet:

java -classpath .;saxon.jar com.icl.saxon.StyleSheet -o output.html http://localhost/telefonliste.xml node.xsl

NB kommandoen skal ikke indeholde et linjeskift!

Her har vi anvendt denne templat eller skabelon:

```
<xsl:template match="//person">
<div style="background-color=#dedede">
<xsl:value-of select="name()" />
</div>
<div style="background-color=#669966">
<xsl:apply-templates select="*" />
</div>
</xsl:template>
```



Klassen Stylesheet har også en parameter –T der tracer hvad processoren foretager sig! D.v.s. linje for linje udskrives hvilke trin XSLT processoren udfører!

Her er slutningen af en kørsel med –T:

```
Udført - tracer
<Top-level element="xsl:template" line="6" file="file:/C:/xmlkursus/saxon/tracer
.xsl" precedence="0"/>
<Instruction element="xsl:output" line="4">
</Instruction> <!-- xsl:output -->
<Source node="/data[1]" line="1" mode="#default">
 <Instruction element="xsl:template" line="6">
  <Instruction element="xsl:for-each" line="7">
   <Source node="/data[1]/person[1]" line="2" mode="#default">
    <Instruction element="xsl:value-of" line="8">
    </Instruction> <!-- xsl:value-of -->
   </Source><!-- /data[1]/person[1] -->
   <Source node="/data[1]/person[2]" line="3" mode="#default">
    <Instruction element="xsl:value-of" line="8">
    </Instruction> <!-- xsl:value-of -->
   </Source><!-- /data[1]/person[2] -->
  </Instruction> <!-- xsl:for-each -->
 </Instruction> <!-- xsl:template -->
</Source><!-- /data[1] -->
</trace>
<?xml version="1.0" encoding="iso-8859-1"?>
FORNAVN: Rosalina
FORNAVN: Sigurd Erik
C:\xmlkursus\saxon>
```

Vores XML dokument er så simpelt så muligt:

<data>

```
<person fornavn="Rosalina" />
<person fornavn="Sigurd Erik" />
</data>
```

Og vores XSLT script er simpelt – så vi bedre kan følge (trace) transformationen:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
>
<xsl:output encoding="iso-8859-1" />

<xsl:template match="/data">
<xsl:for-each select="person">
FORNAVN: <xsl:value-of select="@fornavn" />
</xsl:for-each>
</xsl:template>

</xsl:stylesheet>
```

Vi kan i SAXON outputtet se hvad processoren gør **linje** for linje! Vi kan se at processoren hele tiden regner med et objekt med en bestemt position (i skarpe parenteser). Dette redskab kan være en stor hjælp i debugging – altså hvis det ikke fungerer som vi ønsker!

SAXON som Windows exe:

SAXON processoren kan også downloades fra den nævnte adresse i form af '**Instant** Saxon' som er SAXON processoren som et Windows .exe program.

Dette program – saxon.exe - kan så køres fra en DOS kommando prompt. Instant SAXON – en såkaldt 'cut down version' - rummer ikke helt de samme muligheder som Java versionen – men er nemmere at bruge for Windows brugere! Og den er i hvert fald ikke langsommere!

MS-DOS-prompt

```
C:\xmlkursus\instantsaxon>saxon varer1.xml
No stylesheet file name
SAXON 6.5.3 from Michael Kay
Usage: saxon [options] source-doc style-doc {param=value}...
Options:
  -a              Use xml-stylesheet PI, not style-doc argument
  -ds             Use standard tree data structure
  -dt             Use tinytree data structure (default)
  -o filename     Send output to named file or directory
  -m classname    Use specified Emitter class for xsl:message output
  -r classname    Use specified URIResolver class
  -t              Display version and timing information
  -T              Set standard TraceListener
  -TL classname   Set a specific TraceListener
  -u              Names are URLs not filenames
  -w0             Recover silently from recoverable errors
  -w1             Report recoverable errors and continue (default)
  -w2             Treat recoverable errors as fatal
  -x classname    Use specified SAX parser for source file
  -y classname    Use specified SAX parser for stylesheet
  -?              Display this message

C:\xmlkursus\instantsaxon>saxon -a varer1.xml
Transformation failed: No matching <?xml-stylesheet?> processing instruction fou
nd

C:\xmlkursus\instantsaxon>_
```

Skriv til flere output dokumenter med SAXON:

Ofte er vi interesseret i at **fordele** indholdet af et XML dokument på **forskellige** filer. Dette kan lade sig gøre i SAXON med funktionen **output**. Denne **extension** funktion sender det som står **inden** i elementet <saxon:output> til den fil som angives med en attribut **href**.

Som vi har set tidligere skal vi erklære et nyt namespace til SAXON! Dette namespace skal have det viste indhold! Vi er også nødt til at have en **attribut** der erklærer præfikset – lige som vi tidligere har set eksempler på.

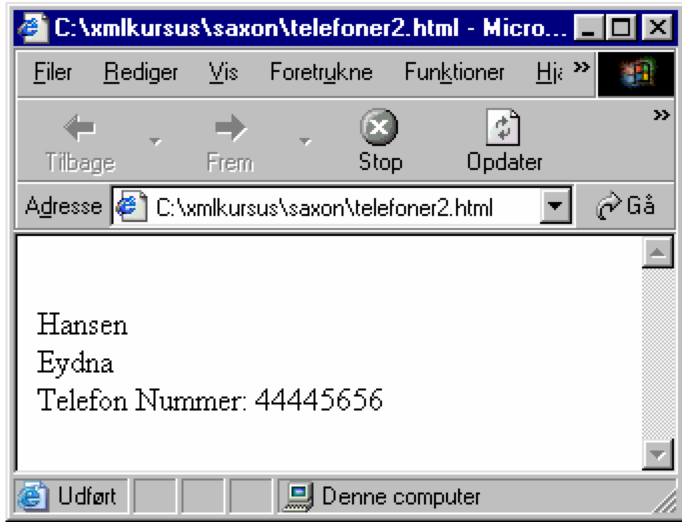
```
<?xml version="1.0"?>

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:saxon="http://icl.com/saxon"
extension-element-prefixes="saxon"
>
<xsl:output
method="html" indent="yes"
/>
<xsl:template match="/">
<xsl:choose>
   <xsl:when test="element-available('saxon:output')">
   <xsl:for-each select="//person">
      <saxon:output href="telefoner{position()}.html">
         <br /><xsl:value-of select="efternavn" />
         <br /><xsl:value-of select="fornavn" />
```

```
            <br /><xsl:value-of select="telefon" />
        </saxon:output>
    </xsl:for-each>
    </xsl:when>

    <xsl:otherwise>
    <xsl:for-each select="//person">
            <br /><xsl:value-of select="efternavn" />
            <br /><xsl:value-of select="fornavn" />
            <br /><xsl:value-of select="telefon" />
    </xsl:for-each>
    </xsl:otherwise>
</xsl:choose>
<xsl:if test="not(element-available('saxon:output'))">
      <xsl:message terminate="no">
      Elementet saxon:output er ikke til stede.
      </xsl:message>
</xsl:if>

</xsl:template>


</xsl:stylesheet>
```

Dette stilark **tester** om elementet – funktionen – output er til stede! Hvis det **er** til stede fordeles telefonlisten i **tre** dokumenter med hvert sit navn! Hvis elementet **ikke** er til stede skrives det hele i een fil! Det er altså ret vigtigt at byde på et **alternativ**!

Hvis vi med andre ord kører denne transformation med en **anden** processor end SAXON – bliver listen skrevet til **een** fil! Det er **kun** SAXON processoren som forstår kommandoen **output**!!

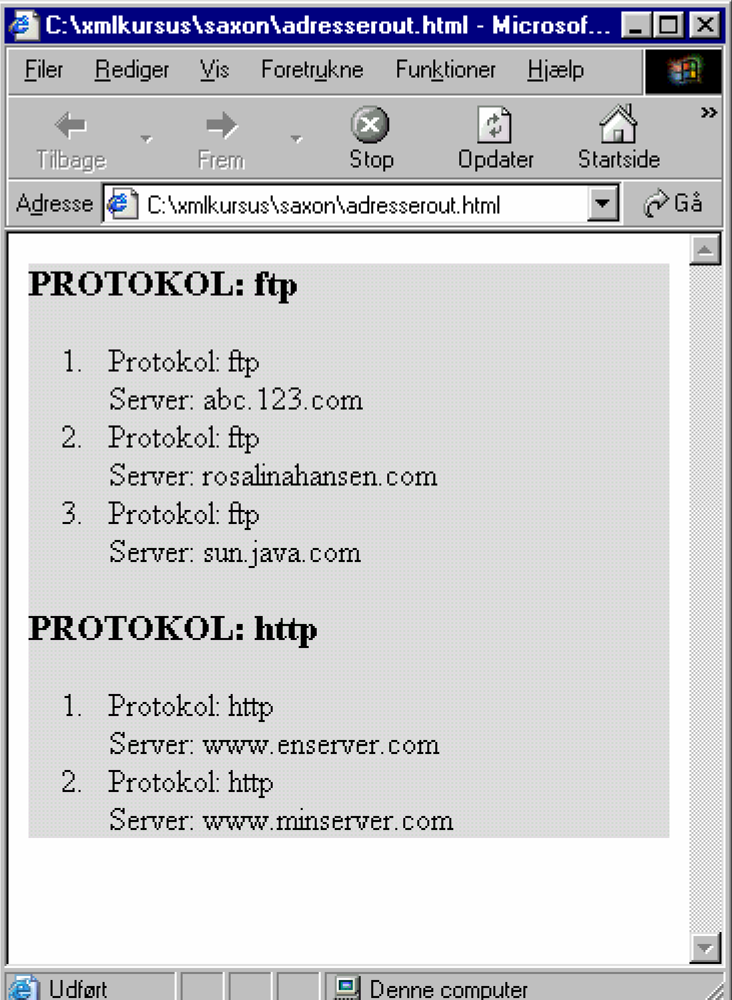
Her er et af de producerede dokumenter:



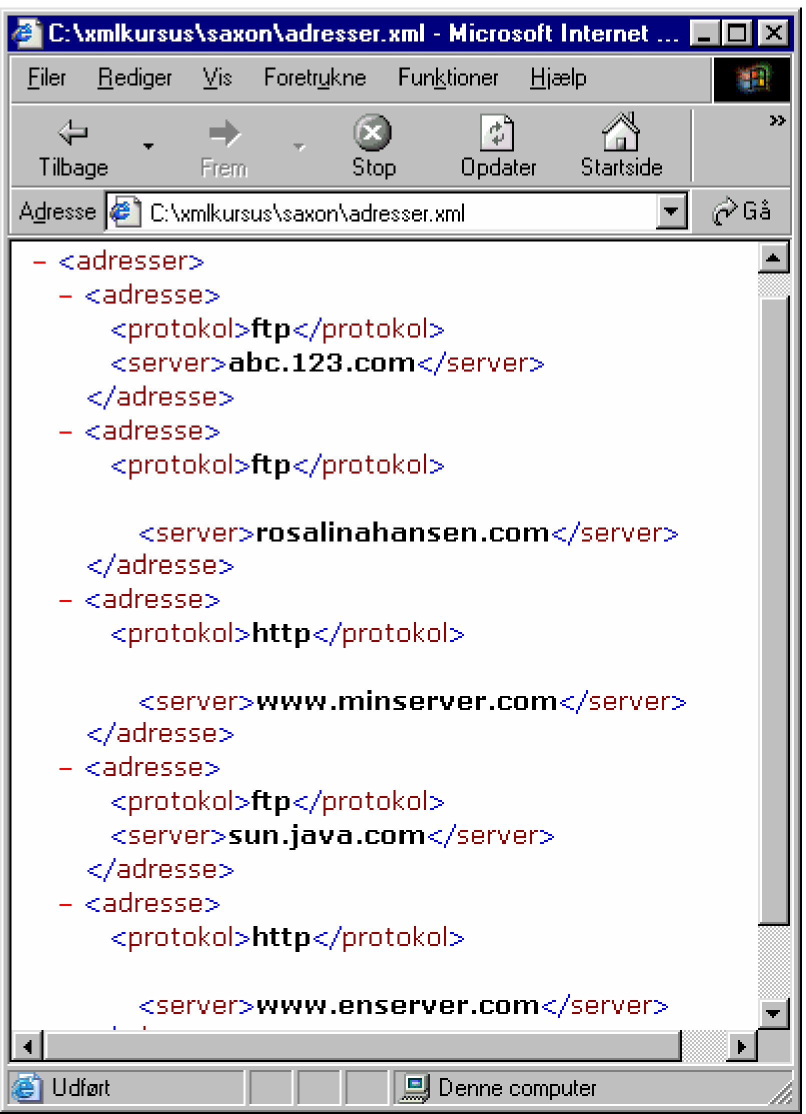
Gruppering af data med SAXON:

Data fra et XML dokument kan sorteres og grupperes. At gruppere data vil sige at samle alle objekter som har en fælles egenskab i en gruppe – og så måske bagefter sortere dem efter et eller andet kriterium. Sortering og gruppering er altså to helt **forskellige** ting!

Man kan – som vi har set – let sortere data med XSLT – men ikke gruppere data så let. I SAXON findes en extension funktion som grupperer data saxon:**group**. Vi vil her give et kortfattet eksempel på brug af saxon:group. Det vi ønsker er en gruppering som denne:



Vores XML dokument ser sådan ud:



Med **SAXON** XSLT processoren kan vi så skrive og anvende dette typografi **ark**:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:saxon="http://icl.com/saxon"
extension-element-prefixes="saxon"
>

<xsl:template match="/">
  <div style="margin-bottom:5pt;background-color:#dedede" xsl:extension-element-prefixes="saxon">
  <saxon:group select="//adresse" group-by="protokol">
  <xsl:sort select="server" order="ascending"/>
  <h3>PROTOKOL: <xsl:value-of select="protokol" /></h3>
    <ol>
    <saxon:item>
      <li>Protokol: <xsl:value-of select="protokol"/><br />
        Server: <xsl:value-of select="server"/></li>
    </saxon:item>
    </ol>
  </saxon:group>
```

```
   </div>
</xsl:template>

</xsl:stylesheet>
```

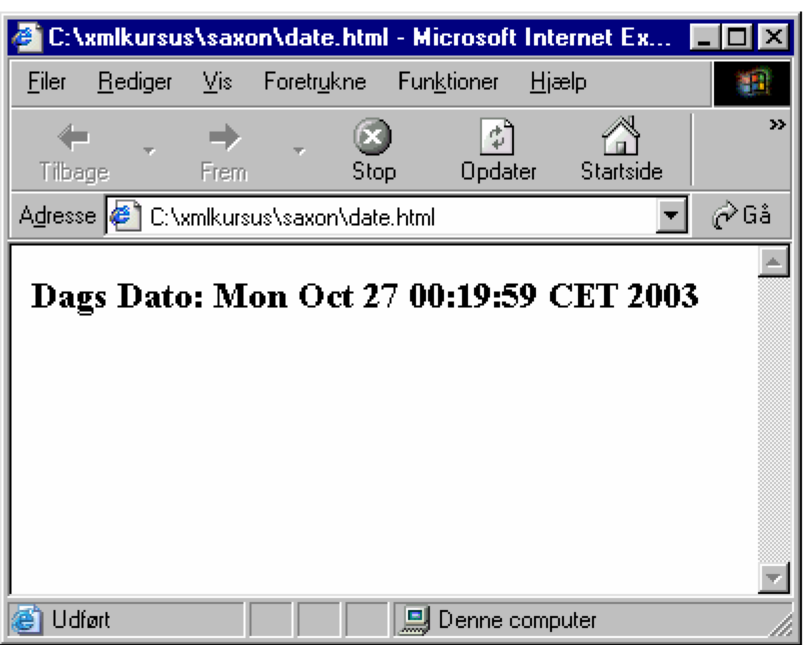
Vi kan godt gennemføre en sådan gruppering uden en extension funktion – men det er meget mere besværligt!

SAXON og Java:

Alle **Java** klasser og funktioner kan uden videre anvendes med SAXON – i modsætning til f. eks. Microsoft XSLT processoren! På den måde kan man også skrive Java klasser selv og anvende dem som ekstension objekter eller elementer.

Et lille eksempel herpå er følgende som kalder klassen **Date** i Java og udskriver **dags dato**:

```
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="http://www.jclark.com/xt/java/java.util.Date">

<xsl:template match="/">
  <html>
    <xsl:if test="function-available('date:to-string') and function-available('date:new')">
      <h3>Dags Dato: <xsl:value-of select="date:to-string(date:new())"/></h3>
    </xsl:if>
  </html>
</xsl:template>

</xsl:stylesheet>
```



Extension objekter eller elementer i SAXON processoren:

Man kan skrive Java klasser som så kan anvendes som extension objekter med SAXON. Vi kan skrive et lille eksempel på en Java klasse der kompileres som ExtKlasse.class:

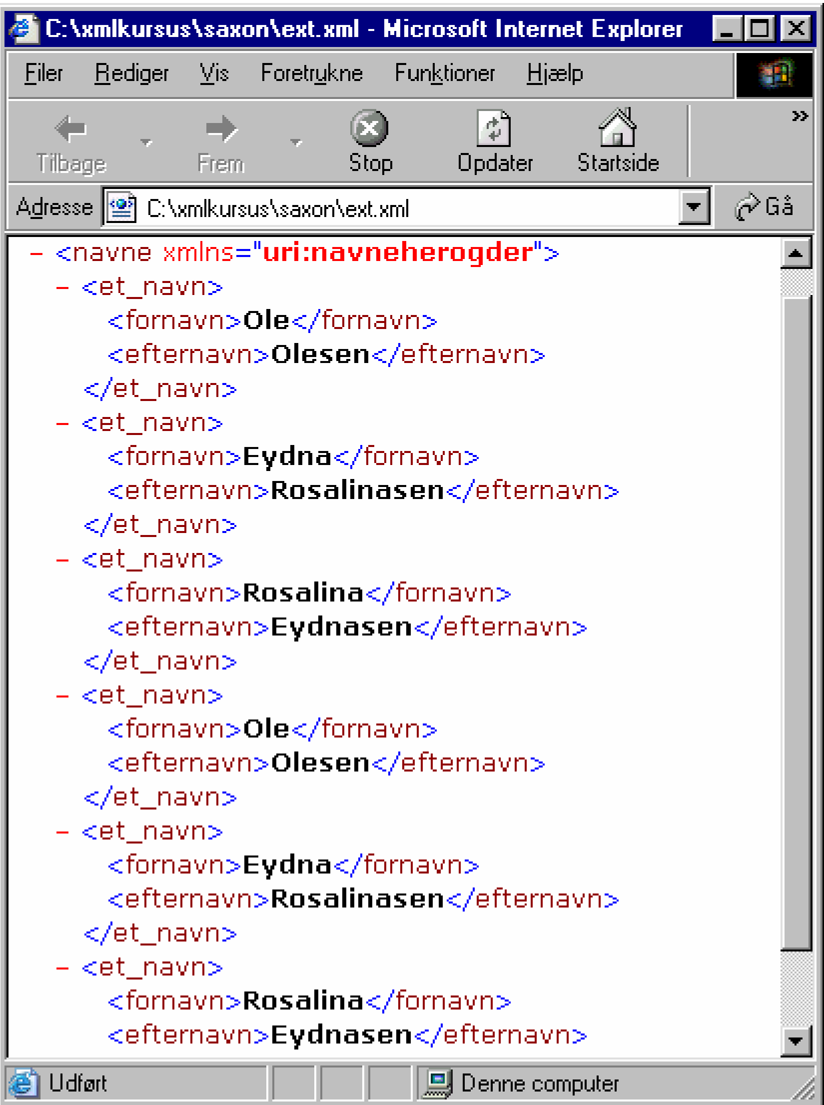
```
public class ExtKlasse {
  public static String retur() {
      String fornavne[]={"Ole","Eydna","Rosalina"};
      String efternavne[]={"Olesen","Eydnasen","Rosalinasen"};

      String ud="<navne xmlns='uri:navneherogder'>";
      for(int i=0,j=6;i<6;i++,j--) {
            ud+="<et_navn><fornavn>"+fornavne[i%3] +"</fornavn>";
            ud+="<efternavn>"+efternavne[j%3] +"</efternavn></et_navn>";
      }
      ud+="</navne>";
      return ud;
  }
}
```

Vi kan nu inddrage denne Java klasse – som blot returnerer et tilfældigt XML dokument eller nodesæt – på denne måde i et stylesheet:

```
<xsl:stylesheet
  version="1.1"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:x="java:ExtKlasse"
  >
  <xsl:output omit-xml-declaration="yes" />
<xsl:script language="java" src="java:ExtKlasse" archive="."  implements-prefix="x">
</xsl:script>
<xsl:template match="/">
  <xsl:value-of disable-output-escaping="yes" select="x:retur()" />
</xsl:template>

</xsl:stylesheet>
```

Læg mærke til følgende:

1. Versionen for dette script **skal** være 1.1 for at dette fungerer i SAXON
2. Vi skal erklære et namespace med en præfiks for siden at kunne bruge Java klassen
3. Indholdet af dette namespace er altid i formen java:klassenavnet
4. Vi inddrager klassen blot ved at oprette et tomt script element
5. Vi kalder klassens metode x:retur() med en disable escaping – ellers bliver tegnene < og > escapede til < og >!



Hvis vi tracer processoren kan vi også se hvordan den hopper rundt i dokumenterne!:

Udført - ext_element_style

Auto

```
C:\xmlkursus\saxon>java -classpath saxon.jar com.icl.saxon.StyleSheet -o ext.xml
  -T tracer.xml ext_element.xsl
<trace>
<Top-level element="xsl:output" line="6" file="file:/C:/xmlkursus/saxon/ext_elem
ent.xsl" precedence="0"/>
<Top-level element="xsl:script" line="7" file="file:/C:/xmlkursus/saxon/ext_elem
ent.xsl" precedence="0"/>
<Top-level element="xsl:template" line="9" file="file:/C:/xmlkursus/saxon/ext_el
ement.xsl" precedence="0"/>
<Instruction element="xsl:output" line="6">
</Instruction> <!-- xsl:output -->
<Instruction element="xsl:script" line="7">
</Instruction> <!-- xsl:script -->
<Source node="/" line="0" mode="#default">
 <Instruction element="xsl:template" line="9">
  <Instruction element="xsl:value-of" line="10">
  </Instruction> <!-- xsl:value-of -->
 </Instruction> <!-- xsl:template -->
</Source><!-- / -->
</trace>

C:\xmlkursus\saxon>
```

Eksempler på extensions med Apache XSLT processoren Xalan:

Xalan kan downloades fra www.apache.org og er den XSLT processor som hører sammen med Apaches Xerces XML parser.