SVG – Scalable Vector Graphics.

SVG er en 'XML applikation', et vektor grafik Markup Language. Det betyder at et SVG dokument ER et XML dokument og skal overholde de samme normer. Et svg dokument skal f. eks. være velformet, alle start mærker skal passe til alle slutmærker. At SVG er en XML applikation betyder også at et SVG dokument kan loades i et XML DOM dokument. SVG er en stanadrd som er defineret af W3C Konsortiet. Svg dokumenter kan valideres overfor et DTD skema – der findes forskellige DTD'er til SVG men her vil vi bruge svg10.dtd, som kan downloades fra http://www.w3.org hjemmesiden.

Hvis man har gemt DTD'en lokalt kan man validere et SVG dokument uden at etablere en netværks forbindelse til www.w3.org!

Alle SVG dokumenter skal have denne struktur – skabelon:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!--
<!DOCTYPE svg
PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd">

<!DOCTYPE svg
PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
-->

<!DOCTYPE svg
PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://localhost/svg10.dtd">


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

</svg>
```

Som det ses ER dette et XML dokument. Det kan gemmes som .xml eller under et andet navn f. eks. .svg. Selve navnet – filtypen – er ikke afgørende.

SVG dokumentet skal have en DOCTYPE hvis det skal kunne valideres – ellers behøver dokumentet ikke nogen DOCTYPE! SVGs DOCTYPE skal have en PUBLIC id og en filplacering for DTD'en. Der er her vist to forskellige DTD'er og vores eget eksempel: at DTD'en gemt lokalt.

Roden i et SVG dokument skal være <svg> og denne rod skal have dette namespace – ellers kan objekterne i SVG ikke bruges. I dette tilfælde erklærer vi et default namespace som kommer til at gælde for alle sub elementer under <svg>.

SVG har en 'tegne flade' (en device context) som markerer grænserne for hvor vi kan tegne. Hvis vi tegner uden for – bliver det ikke vist! Hvis vi ikke angiver en width og height er tegnefladen uendelig!

child noder direkte under <svg>:

For at få et vist begreb om strukturen – skemaet – i SVG kan vi nævne at følgende elementer kan optræde som direkte børn af svg (altså som 'Top Level' elementer):

1. rect
2. circle
3. ellipse
4. polyline
5. polygon
6. text
7. defs
8. title
9. script
10. path
11. linearGradient
12. g

<g> bruges til at samle en række elementer i en gruppe.

Som det ses består disse Top Level elementer bl. a. forskellige figurer som rektangel eller cirkel!

Vektor grafik:

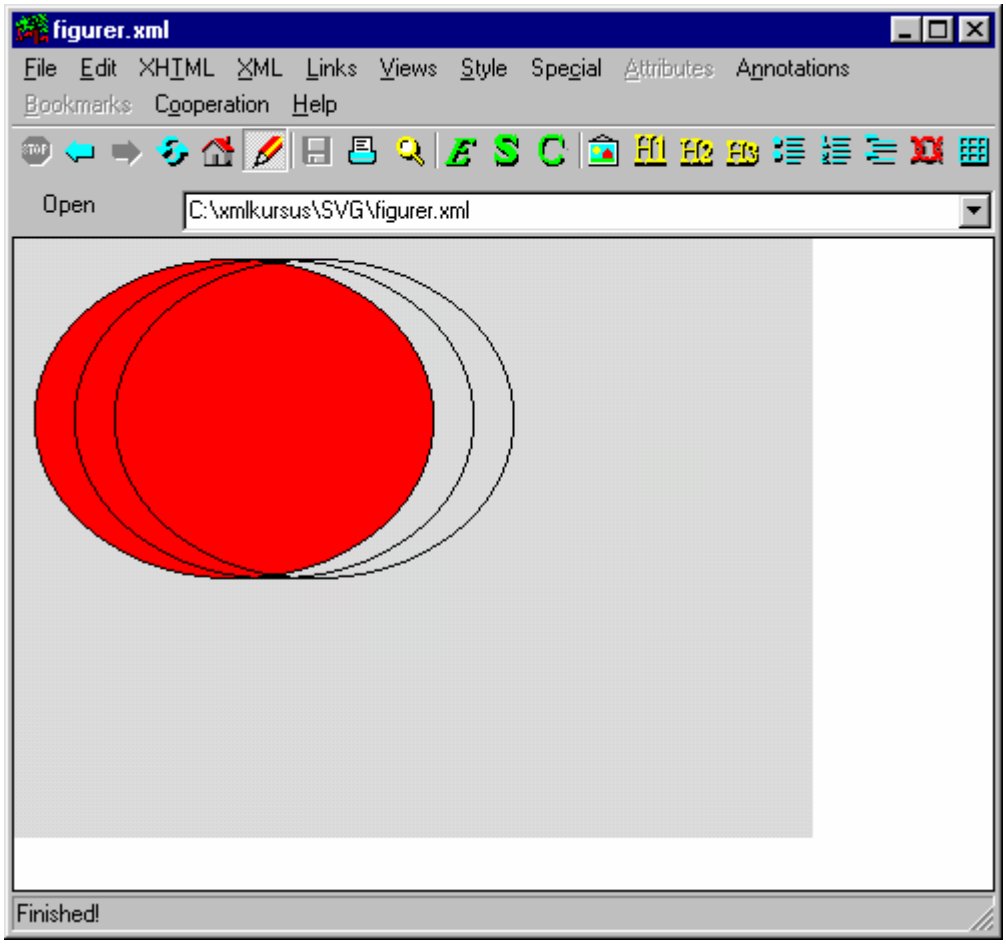
Ved hjælp af SVG kan man definere og 'tegne' **vektor grafik** af punkter, linjer, figurer, tekst osv. SVG grafik er vektor grafik – dvs. formatet er defineret som 'matematiske' formler – IKKE som pixels enkeltvis som i en Bitmap! Vektor grafik kan skabes dynamisk f. eks. i script kode og kan modificeres dynamisk! Vektor grafik fylder meget mindre en bitmap grafik. Et rektangel som er 100 gange 100 'fylder' ikke mere i filen end et rektangel som er 10 gange 10! Det modsatte gælder i en bitmap hvor et billede der er 10 gange 10 fylder – principielt – 100 tegn mens et billede som er 100 gange 100 fylder 10.000 tegn!

SVG og VML (Microsofts 'udgave') bruges i mange forskellige sammenhænge f. eks. i Microsofts Office programmer i tegne funktionerne! Mange tegne programmer kan producere vektor grafik! Her skal vi imidlertid se på SVG dokumenter primært som XML dokumenter og vi skal 'tegne' figurerne manuelt – noget som IKKE er praktisk muligt hvis grafikken bliver rigtig kompliceret!!

I SVG dokumenter angives mange attributter i en måleenhed. Hvis man ikke specificerer måleenheden – blot skriver f. eks. x="44" – oversættes værdien til en monitor afhængig værdi oftest pixels. Man kan også angive måleenheden som fx "44pt", "44px", "2cm" eller "2in".

Eksempel: figurer.xml:

Vi kan f.eks. opnå grafik som dette – vist i browseren Amaya fra www.w3.org som er en af de få browsere som direkte kan vise XML SVG dokumenter:



Vi har skrevet dette XML dokument:

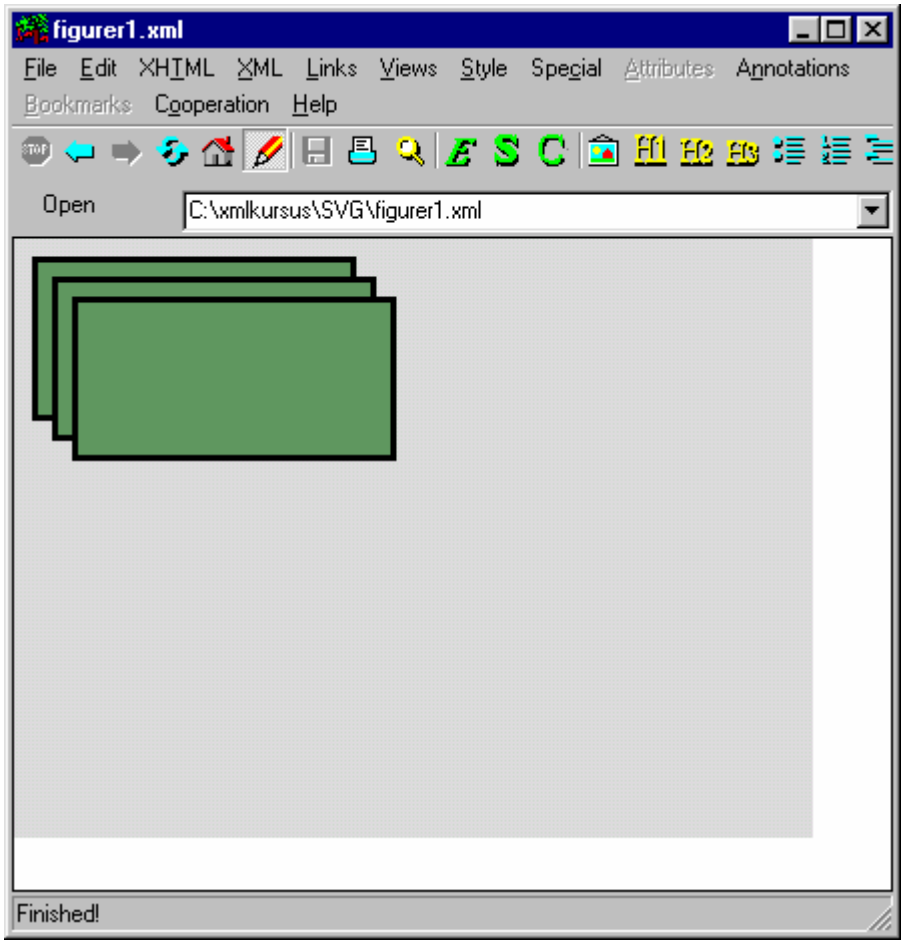
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg
PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://localhost/svg10.dtd" [
<!ENTITY style "fill:none;stroke-width:1;stroke:#030303">
]
>

<svg xmlns="http://www.w3.org/2000/svg" width="400" height="300" style="background-
color:#dedede">

<ellipse style="&style;fill:red" cx="110" cy="90" rx="100" ry="80" />
<ellipse style="&style;" cx="130" cy="90" rx="100" ry="80" />
<ellipse style="&style;" cx="150" cy="90" rx="100" ry="80" />
```

</svg>

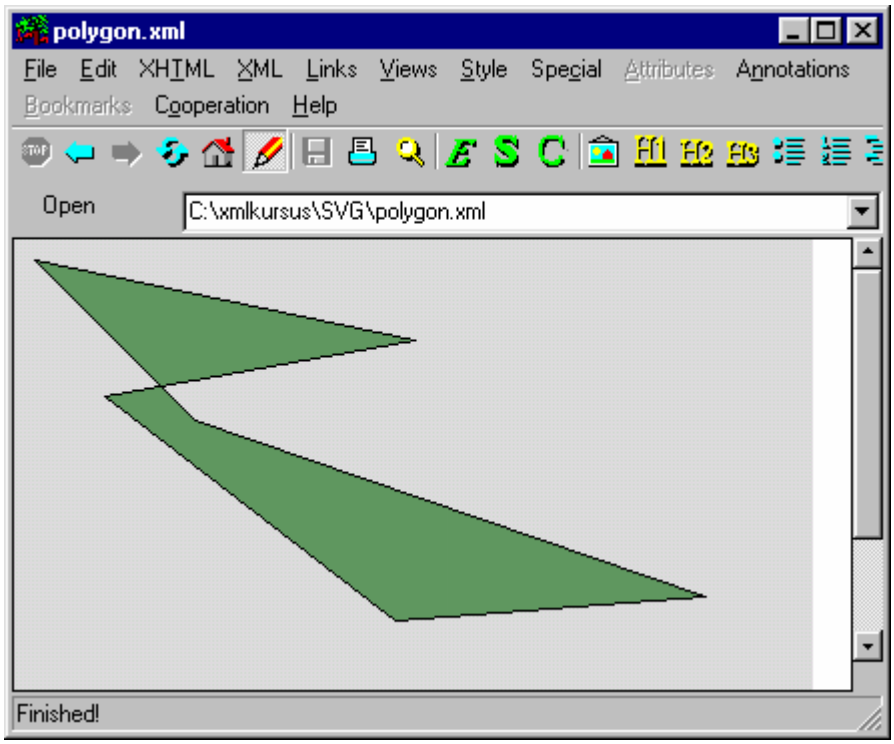
Vi har her brugt elementet ellipse som dels har et centrum (punktet cx, cy) dels har to radier. Vi kan style SVG med elementer fra CSS. Her har vi også brugt en intern entitet til at angive elementets style – hvilket selvfølgeligt i mange tilfælde vil være nemmere end hver gang at skulle definere en style inden i elementet. Vi kan også se at den style som står i elementet kan 'over ride' entitetens style! Den 'sidste' style gælder altid i CSS!

Et eksempel der bruger elementet rect med fyldte rektangler kunne se sådan ud:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE svg
PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://localhost/svg10.dtd" [
<!ENTITY style "fill:#669966;width:160;height:80;stroke-width:3;stroke:black">
]
>

<svg xmlns="http://www.w3.org/2000/svg" width="400" height="300" style="background-
color:#dedede">

<rect x="10" y="10" style="&style;" />
<rect x="20" y="20" style="&style;" />
<rect x="30" y="30" style="&style;" />

</svg>
```



Man kan producere vilkårlige polygoner (og poly lines) som denne:



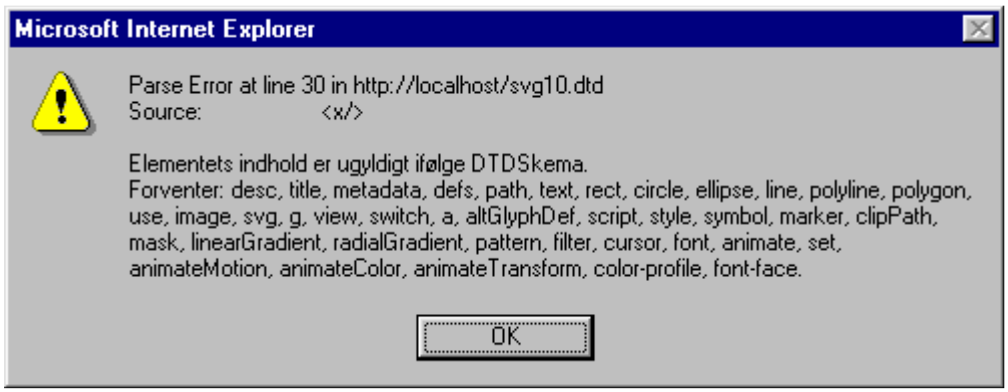
Det nye element skrives så således idet en **polygon** er defineret af et sæt af x-y punkter og linjerne mellem alle punkterne:

<polygon points="10,10, 200,50, 45,78, 190,190, 345,178, 90,90" style="&style;"/>

Polygonen starter i punktet 10 hen ad x-aksen og 10 ned ad y-aksen osv.

Det er almindeligt i SVG at oprette en <defs> sektion som rummer et antal definitioner eller typer – som i XSD kan man oprette typer i SVG.

Hvis vi bevidst indfører en fejl og derefter prøver at validere en <defs> får vi at vide hvordan syntaksen er – hvad en defs kan indeholde af child elementer:

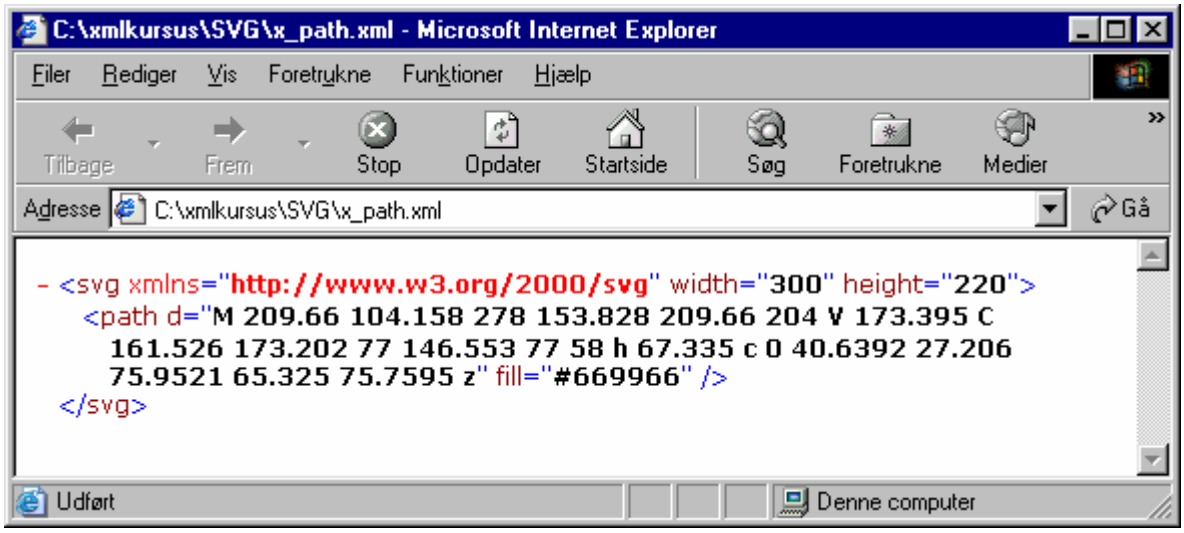


SVG paths eller stier:

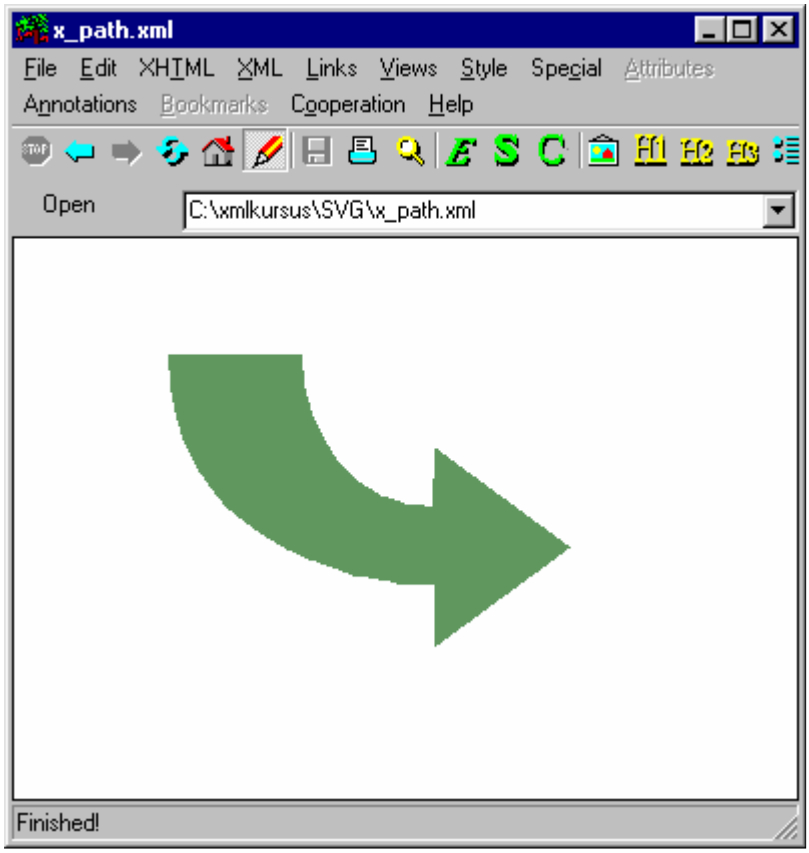
En path er en – meget kompliceret – række af punkter (x,y par) med tilhørende kurve symboler som c eller s, som er to forskellige slags kurver.

En sti er normalt så kompliceret at den sjældent skrives i hånden. Det første tegn i stien er M som betyder 'move' til dette punkt. Det sidste tegn er ofte et Z som markerer at stien skal lukkes d.v.s. at der skal trækkes en linje tilbage til start punktet.

Et eksempel på en path i SVG kunne være:



Denne path giver dette stykke grafik:



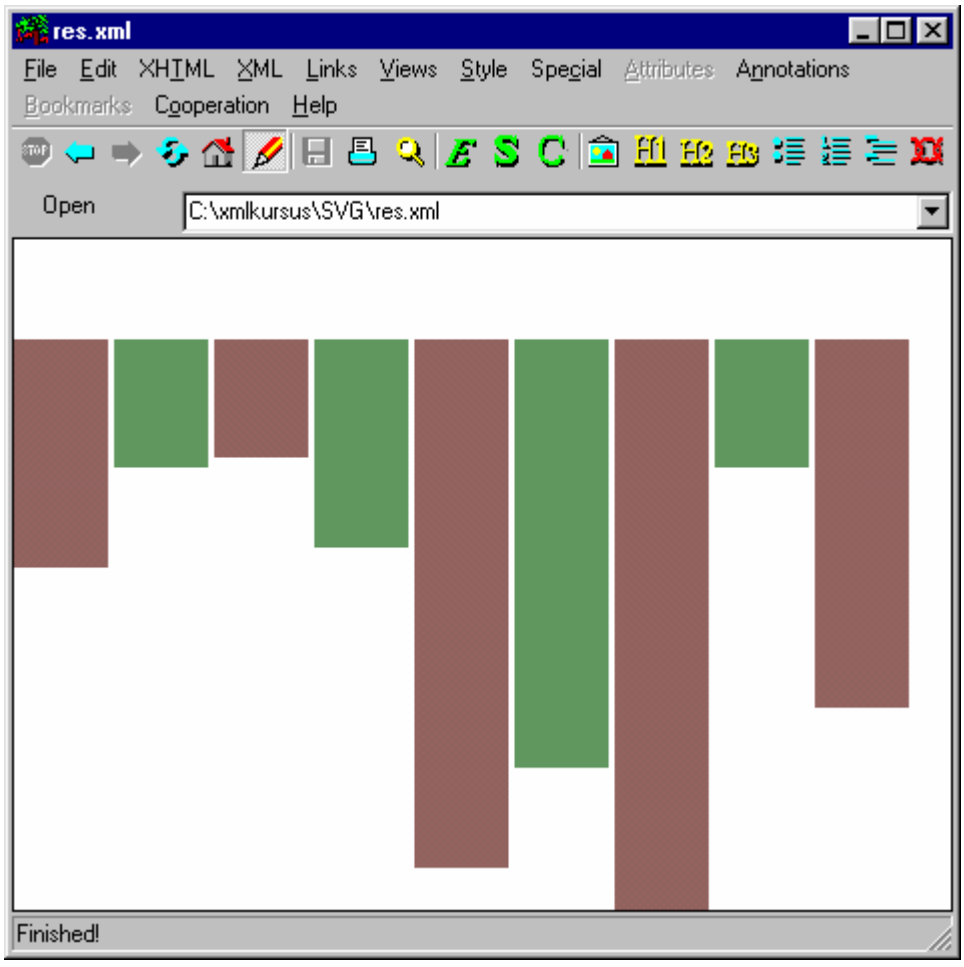
Normalt bliver stier eller paths tegnet i et tegne værktøj!

Skabe grafer i SVG ud fra XML data:

Ved hjælp af f. eks. et stylesheet kan vi vise XML data som en graf eller et diagram. Vi tager udgangspunkt i denne liste af værdier:

```
<?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>
```

Vi kan nu omsætte disse værdier i en SVG **graf** der ligner dette:



Dette kan gøres med følgende XSLT 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:stylesheet>
```

Man kan se at vi **direkte** skriver erklæringerne for SVG XML filen i dette stylesheet:

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

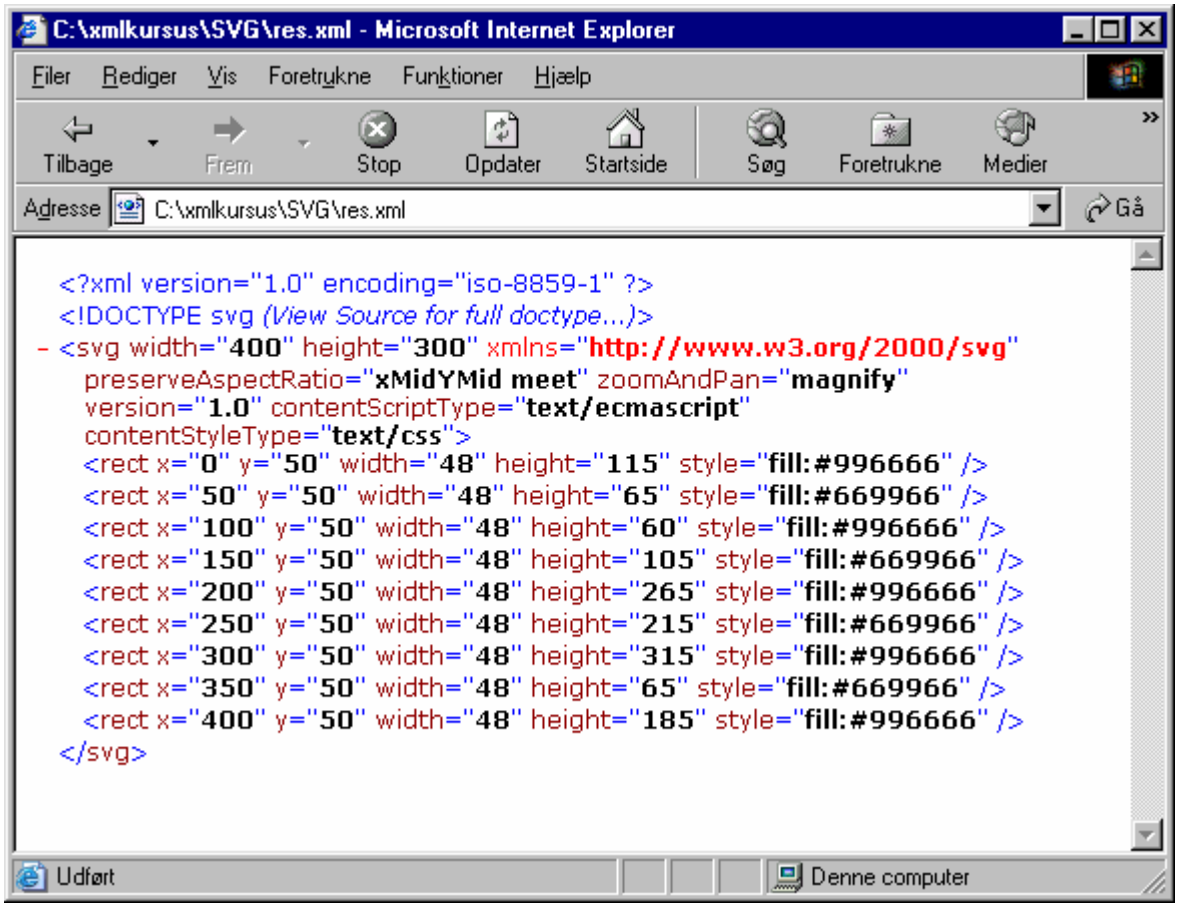
Ellers gennemløber vi simpelt hen værdierne og opretter et rect element for hver som har den rigtige højde (dybde). Dette stylesheet kunne gøres mere fleksibelt hvis man indførte nogle variable der talte på antallet af værdier og deres størrelse!

Vi skifter farven på søjlerne ved hjælp af at måle position() som returnerer 1, 2 osv.

Ovenstående system er i princippet det som alle **regneark** anvender hvis de skal tegne diagrammer eller grafer over værdier! Microsoft Office anvender dog ikke SVG men et andet tilsvarende Markup Language nemlig **VML** – Vector Graphics Markup Language. Dette anvendes overalt i tegnefunktioner og grafik funktioner.

Den producerede output fil gemmes som en .xml fil. Dette kan f. eks. gennemføres med msxsl.exe. Eller i et script - som vi har set eksempler på.

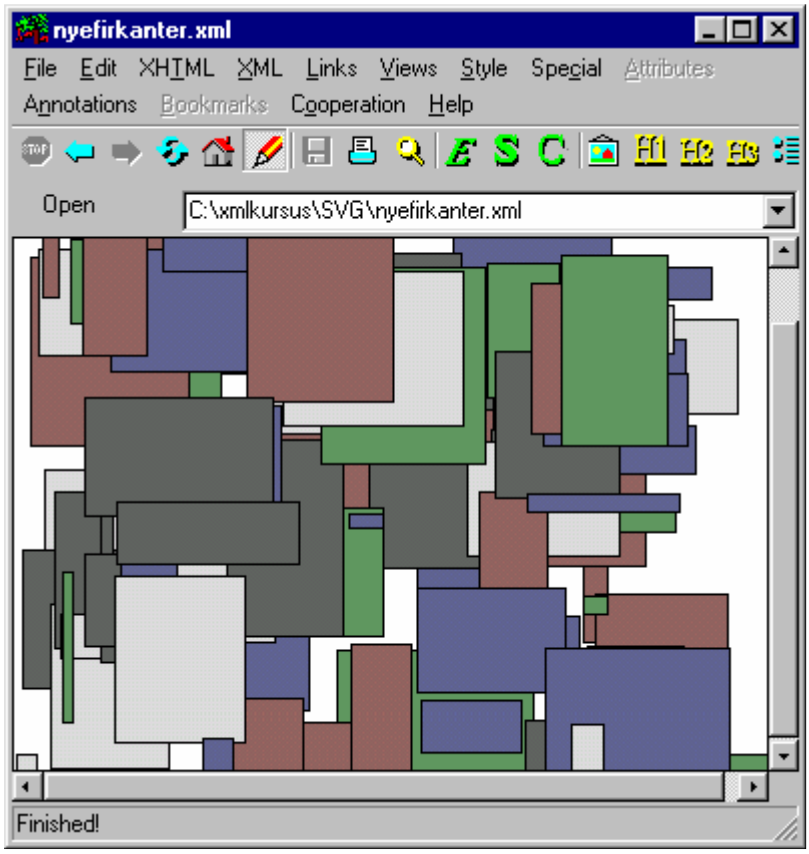
Den producerede fil ser sådan ud i Internet Explorer:



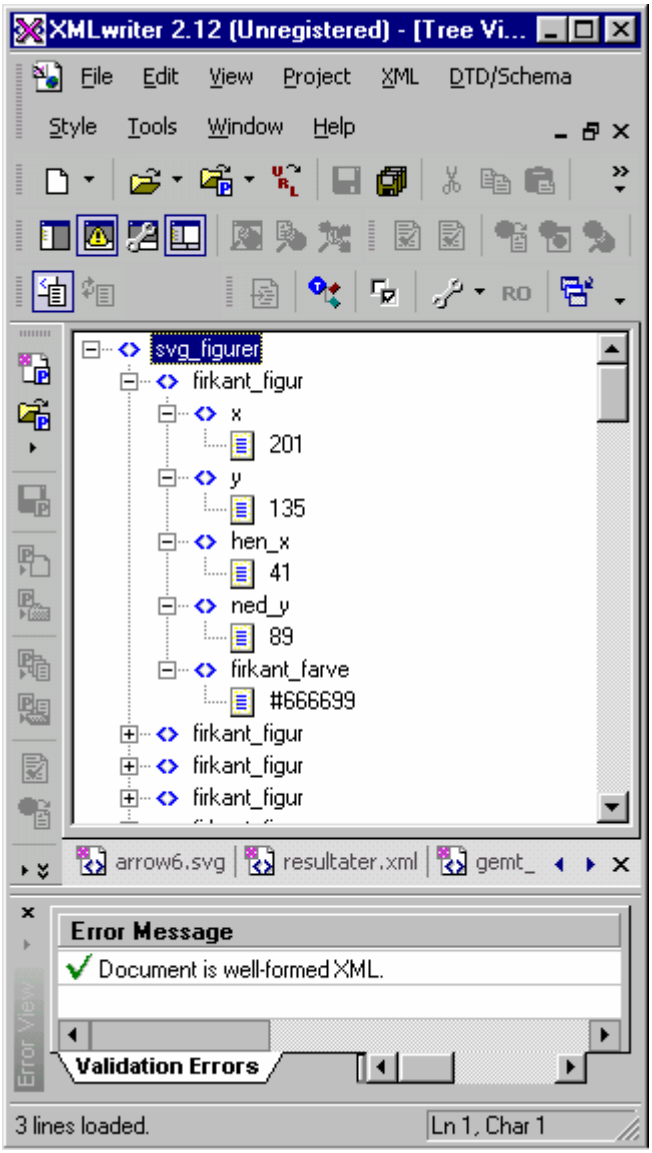
Det kan ses at systemet automatisk tilføjer nye attributter i <svg> roden! Dokumentet kan valideres.

Tilfældige rektangler med SVG:

Vi kan skabe tilfældig grafik i stil med dette:



Vi har her først produceret en tilfældig XML fil som er opbygget på denne måde:



Vi har produceret dette XML dokument med metoder fra W3C **DOM** modellen selv om det lige så godt kunne være foregået med en SAX writer:

```
<script>

var doc=new ActiveXObject("Msxml2.DOMDocument.4.0");
var rod=doc.createElement("svg_figurer");
var pi=doc.createProcessingInstruction("xml-stylesheet","href='firkanter.xsl' type='text/xsl'");
doc.appendChild(pi);
doc.appendChild(rod);

for(i=0;i<100;i++){
var el_rect=doc.createElement("firkant_figur");
var x=doc.createElement("x");
var y=doc.createElement("y");
var width=doc.createElement("hen_x");
var height=doc.createElement("ned_y");
var farve=doc.createElement("firkant_farve");
```

```
var tal=parseInt((Math.random()*300));
txt_x=doc.createTextNode(tal);
x.appendChild(txt_x);
var tal=parseInt((Math.random()*300));
txt_x=doc.createTextNode(tal);

y.appendChild(txt_x);
var tal=parseInt((Math.random()*100));
txt_x=doc.createTextNode(tal);

width.appendChild(txt_x);
var tal=parseInt((Math.random()*100));
txt_x=doc.createTextNode(tal);

height.appendChild(txt_x);

var tal=parseInt((Math.random()*5));
var c="";
switch(tal) {
case 0: c="#dedede";break;
case 1: c="#669966";break;
case 2: c="#996666";break;
case 3: c="#666666";break;
case 4: c="#666699";break;

}
txt_x=doc.createTextNode(c);

farve.appendChild(txt_x);

el_rect.appendChild(x);
el_rect.appendChild(y);
el_rect.appendChild(width);
el_rect.appendChild(height);
el_rect.appendChild(farve);

rod.appendChild(el_rect);

}
var fso=new ActiveXObject("Scripting.FileSystemObject");
var fil=fso.createTextFile("c:/xmlkursus/SVG/gemt_dom_svg.xml");
fil.write(doc.xml);
fil.close();

</script>
```

Disse metoder ligner metoder vi har brugt andre steder. Vi opbygger dokumentet ved at finde tilfældige tal og bruger dem som tekst noder til firkanternes sub elementer.

Ovenstående er egentligt ikke nogen svg fil! For at få den transformeret til et XML SVG dokument kan vi anvende det XSLT **stylesheet** vi brugte før – eller i al fald grundlæggende det samme:

```
<?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="//firkant_figur">
<rect x="{x}" y="{y}" width="{hen_x}" height="{ned_y}" style="fill:{firkant_farve};stroke:black;stroke-width:1;">
</rect>
</xsl:for-each>
</svg>
</xsl:template>

</xsl:stylesheet>
```

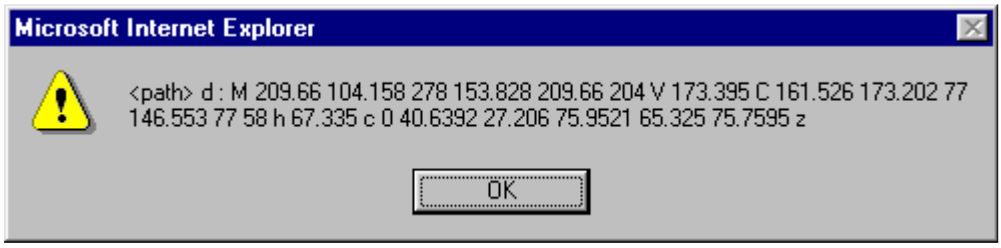
Metoden er ret simpel. Vi anvender blot såkaldte 'attribut value templater' i form af { } for at finde værdien af sub elementerne i XML dokumentet. På den måde opbygges – stort – gyldigt XML dokument der direkte kan læses som SVG grafik.

Dynamisk at bearbejde XML SVG dokumenter:

Man kan arbejde med, analysere, bearbejde, manipulere med SVG dokumenter akkurat som med alle andre XML dokumenter. Nedenstående er et lille eksempel på dette:

```
<xml id="xmldoc">

<svg id="s" xmlns="http://www.w3.org/2000/svg" width="300" height="220">
  <path id="P"
       d="M 209.66 104.158 278 153.828 209.66 204 V 173.395 C 161.526 173.202 77 146.553 77 58 h 67.335 c 0
40.6392 27.206 75.9521 65.325 75.7595 z"
       style="fill:#669966" />
</svg>

</xml>
<script>

var doc=xmldoc.XMLDocument.documentElement;
alert(doc.xml);
var attr=doc.attributes;

for(i=0;i<attr.length;i++){
alert("<svg> "+attr(i).nodeName+" : "+attr(i).nodeValue);
}

var c=doc.childNodes;
for(i=0;i<c.length;i++){
   alert(c(0).xml);
   var attr=c(0).attributes;
   for(j=0;j<attr.length;j++){
```

```
      alert("<path> "+attr(j).nodeName+" : "+attr(j).nodeValue);
    }
}

</script>
```



Vi kan f. eks. hente bestemte værdier ud af SVG dokumentet. Vi kan også indlægge <script> kode på sider og direkte dynamisk ændre et stykke SVG grafik.