1 XSL Transformations (XSLT)

Sammanfattning: Lektionen behandlar standarden XSLT som används för att formatera och transformera XMLdokument. Detta är en avancerad standard som används tillsammans med XML-dokument. XSLT ger oss betydligt fler möjligheter än vad CSS ger. Standarden XPath är en viktig standard som används tillsammans med XSLT. Vi går igenom XSLT och XPath i tre lektioner, där denna lektion tar upp det mest grundläggande.

Innehåll

1.1 Inledning

XSL är en förkortning av Extensible Stylesheet Language. I likhet med CSS definierar XSL stilmallar, men skillnaden ligger i att XSL är baserat på XML. XSL delas in i:

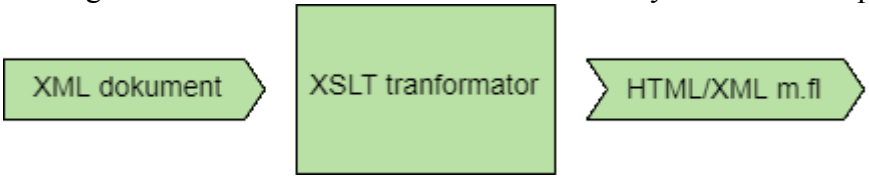
- XSLT: transformerar ett XML-dokument till ett annat dokument
- XSL-FO: är ett märkspråk för att presentera XML-dokument på framförallt pappersmedia

Et exempel på användning kan vara att använda XSLT för att transformera ett XMLldokument till ett annat XML-dokument som använder XSL-FO-taggar. Det nya dokumentet kan därefter köras genom en XSL-FO-tolk och omvandlas till olika typer av standardformat (PDF, PS och liknande). Ett XSL-FO-dokument anger i korthet hur layouten ska vara. I denna kurs tittar vi enbart på XSLT och inte på XSL-FO.

XPath används för att navigera i XML-dokument och är en väldigt viktig del när XSLT används. Vi går kort igenom XPath i kapitel 1.4 för att i nästa lektion göra en mer grundlig genomgång av XPath.

1.2 XSLT

XSLT är som sagt ett språk för att transformera XML-dokument till ett nytt dokument (se figur nedan). Det vanligaste användningsområdet är att transformera ett XML-dokument till ett HTML/XHTML-dokument. Många av de senare versionerna av dagens webbläsare har stöd för XSLT. Det betyder att vi kan öppna



ett XML-dokument med tillhörande XSLT-stilmall i en webbläsare varpå XML-dokumentet transformeras och resultatet visas i webbläsaren.

Det finns även andra transformerare (externa transformerare) som skapar en helt ny fil som vi därefter kan använda till det vi vill. I denna lektion räcker det att använda en webbläsare med stöd för XSLT.

En av fördelarna med XSLT är att det gör dataöverföring enkelt. Tänk dig ett antal verksamheter som lagrar data på olika sätt och format och som då och då behöver utbyta data med varandra. Då kan XML användas som ett format vid överföringen och XSLT för att transformera XML till det specifika formatet en viss verksamhet behöver.

XSLT skrivs som sagt i XML och därför gäller samma regler för ett välutformat dokument. En stilmall skriven i XSLT sparas i en egen fil, med filändelsen `xsl`, och för att knyta stilmallen till XML-dokumentet måste vi använda en processinstruktion i XML-dokumentet. Denna liknar till stor del den som användes för CSS:

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

Som du ser är det endast en liten förändring. Attributet `type` måste ha värdet `text/xsl`. I attributet `href` anges filnamnet som i detta exempel är `personregister.xsl`. För att ge ett kort exempel på användning av XSLT använder vi XML-dokumentet från förra lektionen:

```
<?xml version="1.0" encoding=" UTF-8"?>
<?xml-stylesheet type="text/xsl" href="personregister.xsl"?>

<personregister>
<person>
      <namn>Karin Nordin</namn>
      <telefon>11111111</telefon>
      <civilstatus>Gift</civilstatus>
      <titel>Chefsredaktör</titel>
 </person>
</personregister>
```

Som du ser på rad 2 anger processinstruktionen att stilmallen `personregister.xsl` ska användas.

Denna stilmall ser ut så här:

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

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transfor
m">   <xsl:template match="/">
<html>
   <body>
      <h1>Lista över personer</h1>
      <p>Namnet är <xsl:value-of select="//namn"
/></p>    </body>
</html>
 </xsl:template>
</xsl:stylesheet>
```

Det är inte mening att du redan nu ska förstå vad ovanstående XSLT-dokument gör. Det illustrerar enbart hur ett en stilmall i XSLT kan se ut. I kommande kapitel går vi

igenom vad de olika delarna betyder. Öppnar vi nu XML-dokumentet i en webbläsare ser det ut enligt bilden nedan:

1.3 Hur XSLT fungerar

XSLT består av en uppsättning mallar (på engelska: templates) som bestämmer hur till exempel enskilda element ska processas. Nedan följer i korta drag hur XSLT arbetar:

1. Det skapas en trädliknande struktur, bestående av noder, av angivet XML-dokument.
2. Det skapas en trädliknande struktur av stilmallen (XSLT-dokumentet).
3. XSLT-tolken börjar processa XML-dokumentets trädstruktur med början i rotnoden. I stilmallen kontrolleras det om det finns någon mall som passar till rotnoden. Om en mall finns appliceras denna på rotnoden och resultatet hamnar i resultatträdet. På liknande sätt gås nu alla noder i hela källdokumentet igenom. Finns ingen mall för en given nod hoppas denna över och nästa nod gås igenom.
4. Resultatträdet sparas till ett nytt dokument.

En mall i XSLT är uppbyggt enligt:

```
<xsl:template match="/">
 <!— Bearbeta barnelement som matchar mallen -->
</xsl:template>
```

Elementet `xsl:template` används för att skapa en mall. Detta element har ett attribut med namnet `match` för att beskriva vilka noder i XML-dokumentet mallen gäller för. Attributets värde ska vara en sökväg uttryckt i XPath. Denna sökväg pekar ut olika noder i källträdet.

1.4 XSLT transformation

Nedan går vi nu igenom XSLT-dokumentet från första kapitlet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform
">
<xsl:template match="/">
<html>
    <body>
        <h1>Lista över personer</h1>
        <p>Namnet är <xsl:value-of select="//namn"
/></p>
     </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

På rad 1 börjar vi med en XML-deklaration. Detta eftersom alla XSLT-dokument i sin tur är ett XML-dokument. På rad 2 och 3 finns rotelementet där även en namnrymd anges. Ett XSLT-dokuments rotelement kommer alltid att vara `xsl:stylesheet` eller `xsl:transform`. Båda är giltiga att använda. Som du ser används namnrymdsprefixet `xsl` som knyts till namnrymdsnamnet: `http://www.w3.org/1999/XSL/Transform`

Denna namnrymd måste användas för att dokumentet ska betraktas som ett XSLT-dokument. Om du anger fel URI kommer elementen i XSLT inte att fungera. Det är alltså ett sätt för transformeraren att veta att det är ett XSLT dokument det är frågan om och vad de olika elementen betyder. Alla element som hör till XSLT-standarden måste därför använda prefixet `xsl`. Vi har även med ett attribut i rotelementet som visar vilken version av XSLT som ska användas, nämligen version `1.0`.

Alla mallar (templates) som ska vara med måste nästlas innanför rotelementet. På rad 4 till 11 skapar vi en mall med hjälp av elementet `xsl:template`. I exemplet ser vi att attributet `match` har värdet /. Detta är ett uttryck i XPath som pekar ut dokumentnoden (inte rotnoden, utan hela dokumentet). Tanken med stilmallen i exemplet är att transformera XML-dokumentet till ett HTML-dokument. Allt från rad 5 till 10 är ren HTML-kod och börjar med taggen `html` på rad 5. Undantaget är:

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

på rad 8. Här används elementet `xsl:value-of` som är ett XSLT-element (vilket vi ser genom prefixet `xsl`). Detta används för att hämta ett värde från XML-dokumentet och överföra detta till HTML-dokumentet. Attributet `select` har ett XPath-uttryck som värde (en sökväg).

Denna sökväg pekar ut ett `namn`-element och innehållet i detta element hämtas och skrivs ut. Vi ser alltså att XPath-uttryck används på flera ställen i dokumentet.

Efter denna transformation kommer vi att få följande HTML-kod:

```
<html>
 <body>
      <h1>Lista över personer</h1>
      <p>Namnet är Karin Nordin</p>
 </body> </html>
```

Innan vi går vidare med XSLT måste vi ta en titt på XPath, något som man måste behärska för att kunna använda XSLT. I nästa lektion tittar vi närmare på den funktionalitet som XPath erbjuder.

1.5 XML Path Language (XPath)

XPath är ett språk som framför allt används för att navigera i en nodhierarki. En nod i ett XML-dokument kan till exempel vara en rotnod, elementnod eller attributnod.

I XPath skriver man olika uttryck där den viktigaste komponenten är sökvägar. Dessa sökvägar används för att peka ut de olika noderna. Resultatet av en sökväg är ett nodset.

Sökvägar liknar till stor del de vi använder för vanliga filer på en hårddisk som till exempel:

```
J:\xml\Lektioner\Lektion 7
```

På samma sätt anger vi sökvägar i ett filsystem för att navigera bland innehållet i hårddisken, används XPath för att navigera i ett XML-dokument. Vi kan använda ett XPath-uttryck för att navigera till `namn`-noden i vårt exempel:

```
/personregister/person/namn
```

För att använda ett XPath-uttryck måste en utgångspunkt i nodträdet definieras. Vanligast är något av följande uttryck i elementet `xsl:template`:

1. Starta från dokumentroten. Som nämnts är detta inte det samma som rotelementet i XMLdokumentet, utan representerar hela dokumentet (med prologen). Då används uttrycket:
   ```
   /
   ```
   Det är vanlig använda detta uttryck i den första mallen i XSLT-dokumentet. En mall som matchar hela dokumentet ser med andra ord ut så här:
   ```
   <xsl:template match="/">
   ```

2. Det är även möjligt att ha en utgångspunkt längre ner i XML-dokumentets nodhierarki. Då kommer den nod vi utgår från att kallas för kontextnoden (på engelska context node). Vi kan till exempel skriva:

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

   Nu kommer denna mall att gälla för `person`-elementet i exemplet ovan. I denna mall kommer elementet `person` vara kontextnoden. Kontextnoden används som utgångspunkt när nya XPath-uttryck används i mallen.

Om elementet `person` är satt som kontextnod i en mall kan vi använda XPath-uttrycket `namn/förnamn` i stället för `/personregister/person/namn/förnamn`. Detta kan till exempel användas i elementet `xsl:value-of` för att hämta personens förnamn.

Om XPath-uttrycket börjar med tecknet `/` innebär det att sökvägen är absolut och den utgår därför inte från kontextnoden. Det är viktigt att man tänker sig för en extra gång när absoluta sökvägar används. Tänk dig att vi har följande mall för elementet `person`:

```
<xsl:template match="person">
 <h1>Namn: <xsl:value-of select="/namn" /></h1>
</xsl:template>
```

Det vi vill är att innehållet i elementet `namn` ska skrivas ut, men det kommer inte att fungera så som vi har tänkt. Anledningen är att en absolut sökväg till elementet `namn` används. Det som sker är att det letas efter ett rotelement med namnet `namn`, vilket inte existerar. Om vi i stället vill att innehållet i det första `namn`-elementet i dokumentet ska skrivas ut kan vi skriva:

```
//namn
```

Detta uttryck väljer det första elementet med namnet `namn` från XML-dokumentet. Mer om XPath kommer i nästa lektion.

1.6 Element i XSLT

I exemplen ovan användes några av de element som finns i XSLT. Nedan ges en kort översikt över några av de mest vanliga elementen som används i XSLT. Elementet `xsl:stylesheet` används som rotelement i alla XSLT-dokument (alternativet är `xsl:transform`):

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

Detta element kommer minst att innehålla ett attribut, nämligen en namnrymdsdeklaration för själva XSLT-standarden. Alla andra XSLT-element kommer att nästlas innanför detta element.

Nedan följer nu en lista över de element som används mest och vad de används till (observera att detta inte är en komplett lista). Några av dessa element går vi igenom i denna lektion, medan några inte kommer att tas upp i denna kurs.

element	*beskrivning*
`xsl:template` `xsl:apply-templates` `xsl:call-template`	Definierar mallar och hur de anropas.
`xsl:stylesheet` `xsl:transform` `xsl:includes` `xsl:import`	Definierar stilmallsstrukturen.
`xsl:value-of` `xsl:element` `xsl:attribute` `xsl:comment` `xsl:processinginstruction` `xsl:text`	Element som genererar text som ska visas i det transformerade dokumentet.

element	*beskrivning*
`xsl:variable` `xsl:param` `xsl:with-param`	Element som definierar variabler och parametrar.
`xsl:copy` `xsl:copy-of`	Element som kopierar från källdokumentet till resultatdokumentet i transformationen.
`xsl:if` `xsl:choose` `xsl:when` `xsl:otherwise` `xsl:for-each`	Element för val och sekvens.
`xsl:sort`	Element för sortering.
`xsl:output`	Element för att kontrollerar resultatdokumentets format.

Vi ska nu titta närmare på några av dessa element.

1.6.1 xsl:output

`xsl:output` är ett direkt barnelement till `xsl:stylesheet` och anger för XSLT-tolken vilket format resultatdokumentet ska ha. I exemplet ovan skulle vi kunna ange att resultatdokumentet ska vara ett HTML-dokument:

```
<xsl:output method="html" />
```

Standardvärde för attributet `method` är XML. Ett undantag från detta är om det första elementet i resultatdokumentet är `html`, då sätts värdet på attributet `method` till HTML. Just detta sker i exemplen ovan och är anledning till varför vi inte använt detta element.

Utöver attributet `method` kan även attributet `encoding` användas i elementet `xsl:output`. Detta attribut anger vilken teckenkodning som ska användas i resultatdokumentet.

1.6.2 xsl:template

`xsl:template` är mer eller mindre ett nyckelelement i XSLT eftersom den definierar nya
mallar. Elementet används enligt:

```
<xsl:template match="/">
 <!– Bearbeta barnelement som matchar mallen -->
</xsl:template>
```

Attributet `match` består av et XPath-uttryck. Om detta attribut inte används måste attributet `name` användas i stället. Det är då nödvändigt att använda elementet `xsl:call-templates` för att anropa mallen. Vi kommer att hålla oss till mallar som använder attributet `match` och vi använder `xsl:apply-templates` för att anropa en annan mall (kommer snart exempel).

1.6.3 xsl:value-of

Om du vill att innehåll från XML-dokumentet ska skrivas till resultatdokumentet kan du använda elementet `xsl:value-of`. För detta element finns ett attribut med namnet `select` som måste användas. Attributet `select` anger vad som ska skrivas ut. Attributets värde består av ett XPath-uttryck. I exemplen ovan skrevs namnet i XML-dokumentet ut:

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

1.6.4 xsl:apply-templates

Genom att använda elementet `xsl:apply-templates` kan vi hämta ett nodset vars noder ska processas en och en. För att illustrera detta använder vi följande XML-dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="personregister.xsl"?>
<personregister>
<person>
      <namn>Karin Nordin</namn>
      <telefon>11111111</telefon>
      <civilstatus>Gift</civilstatus>
      <titel>Chefsredaktör</titel>
```

```
 </person>
<person>
      <namn>Lisa Olsson</namn>
      <telefon>22332233</telefon>
      <civilstatus>Sambo</civilstatus>
      <titel>Sjuksköterska</titel>
 </person>
</personregister>
```

Som du ser finns här två `person`-element. Använder vi samma XSLT-dokument som tidigare kommer vi inte att få det resultat vi önskar (vilket är att båda namnen skrivs ut):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transfor
m">   <xsl:template match="/">
       <html>
        <body>
            <h1>Lista över personer</h1>
            <p>Namnet är <xsl:value-of select="//namn"
            /></p>
        </body>
       </html>
 </xsl:template>
</xsl:stylesheet>
```

Här använder vi elementet `xsl:value-of` för att skriva ut innehållet i elementet `namn`.

Det HTML-dokument som skapas kommer att se ut enligt nedan:

```
<html>
 <body>
      <h1>Lista över personer</h1>
      <p>Namnet är Karin Nordin</p>
 </body>
</html>
```

Som du ser skrivs namnet på den första personen i XML-dokumentet ut, men inte namnet på personen `Lisa Olsson`.

Här kan vi använda elementet `xsl:apply-templates` för att gå i igenom alla noder i nodsetet `person`. Vi måste då först skapa en mall som gäller för varje `person`-element och som skriver ut innehållet i elementet `namn` (förändringar visas i fet stil):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
       <html>
```

```
        <body>
            <h1>Lista över personer</h1>
            <xsl:apply-templates select="//person" />
         </body>
          </html>
</xsl:template>

<xsl:template match="person">
      <p>Namnet är <xsl:value-of select="namn" /></p>
</xsl:template>
</xsl:stylesheet>
```

På rad 13-15 har vi skapat en mall för elementet `person` och vi har flyttat raden som skrev ut innehållet för elementet `namn` hit. I den första mallen (rad 4-11) har vi använd elementet `xsl:apply-templates` (rad 8) för att gå igenom alla `person`-element. Att det är just elementet `person` som gås igenom ser du på värdet för attributet `select`. Detta attribut består av ett XPath-uttryck. Lägg märke till att vi endast behöver skriva `person` som värde för attributet `match` på rad 13. Anledningen är att tolken kommer att leta efter en mall för elementet `person` när den kommer till rad 8.
I en webbläsare kommer det nu att se ut som nedan:

Vad skulle ha skett om vi inte hade upprättat den sista mallen, men likväl använt `xsl:applytemplates`? Prova detta! Det som sker är att en standardmall i stället används, som enbart skriver ut allt innehåll i elementet utan några formateringar.

1.6.5 xsl:for-each

Elementet `xsl:for-each` är ett element som utför samma operation på alla noder i ett nodset. Ett användningssätt kan vara som ovan, att skriva ut innehållet i elementet `namn` för alla personer. I attributet `select` anger vi ett XPath-uttryck som pekar ut det set med noder som ska loopas igenom. I vårt fall är det `person`:

```
<xsl:for-each select="//person">
```

```
 <p>Namnet är <xsl:value-of select="namn" /></p> </xsl:for-each>
```

Om vi nu byter ut `xsl:apply-templates` i exemplet ovan mot `xsl:for-each` och dessutom tar bort den andra mallen kommer vi att få följande (förändringar i fet stil):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/"> <html>
        <body>
           <h1>Lista över personer</h1>
           <xsl:for-each select="//person">
             <p>Namnet är <xsl:value-of select="namn" /></p>
           </xsl:for-each>
        </body>
     </html>
 </xsl:template>
</xsl:stylesheet>
```

Här ser du att efter rubriken används elementet `xsl:for-each` som går igenom alla `person`element. For varje person kommer nu namnet att skrivs ut.

Resultatdokumentet efter denna transformering kommer att se exakt likadant ut som i fallet med `xsl:apply-templates`.

Elementet `xsl:for-each` kan vara användbart när tabeller i ett HTML-dokument ska upprättas. Vi kan använda samma XML-dokument som ovan (den med två personer) och skriva ut innehållet i en tabell enligt:

I det XSLT-dokument som används för att generera denna tabell börjar vi med att skriva ut en rubrikrad med information om vad som ska finnas i de olika kolumnerna. Därefter skrivs information ut från varje `person`-element på var sin rad:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
    <body>
    <h1>Lista över personer</h1>
    <table border="1">
    <tr>
       <th>Namn</th>
        <th>Telefon</th>
       <th>Civilstatus</th>
       <th>Titel</th>
    </tr>
    <xsl:for-each select="//person">
    <tr>
       <td><xsl:value-of select="namn" /></td>
       <td><xsl:value-of select="telefon" /></td>
       <td><xsl:value-of Select="civilstatus" /></td>
       <td><xsl:value-of select="titel" /></td>
    </tr>
    </xsl:for-each>
    </table>
    </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

På rad 8 upprättas en tabell som avslutas på rad 24. Först skapas en rubrikrad på rad 9-14. På rad 15 till 23 kommer själva loopen som börjar med elementet `xsl:for-each`. Inuti denna upprättas ett `tr`-element för att skapa en rad för varje `person`-element. Inuti `tr`-elementet upprättas fyra `td`-element för vart och ett av elementet `namn`, `telefon`, `civilstatus` och `titel` i XML-dokumentet. Notera att innanför elementet `xsl:for-each` är elementet `person` kontextnod varför relativa sökvägar räcker i elementen `xsl:value-of`.

1.6.6 xsl:sort

Många gånger kan det vara bra att sortera elementen i XML-dokumentet på ett visst sätt. För detta kan elementet `xsl:sort` användas. De attribut som kan användas i detta element är:

- `select`. Värdet är ett element i XML-dokumentet för vilket sortering ska göras.
- `lang`. Det språket som ska användas under sorteringen. Standardvärdet är engelska.
- `data-type`. Vilken datatyp som ska sorteras. Värdet kan vara antingen `text` eller `number`. Standardvärde är `text`

- `order`. I vilken ordning sorteringen ska ske. Värdet kan vara `ascending` (stigande) eller `descending` (sjunkande). Standardvärde är `ascending`.
- `case-order`. Värdet kan här vara antingen `upper-first` eller `lower-first`. Standardvärde är `upper-first`. Ett exempel på användning är om du i ett element har värdet abc och ett annat element har värdet Abc. Vilken av dessa ska då sorteras först? Detta attribut används inte allt för ofta.

För att sortera ett `telefon`-element som består av siffror kan vi skriva:

```
<xsl:sort select="telefon" data-type="number" />
```

För att sortera vår tabell, i exemplet ovan, efter namn i sjunkande ordning kan vi skriva (förändringar i fet stil):

```
<xsl:for-each select="//person">
   <xsl:sort select="namn" order="descending" />
 <tr>
      <td><xsl:value-of select="namn" /></td>
      <td><xsl:value-of select="telefon" /></td>
      <td><xsl:value-of select="civilstatus" /></td>
      <td><xsl:value-of select="titel" /></td>
 </tr>
</xsl:for-each>
```

Nu kommer raderna i tabellen att skrivas ut i omvändordning mot för bilden ovan. Som du ser placeras elementet `xsl:sort` direkt innanför elementet `xsl:for-each`. Lägg även märke till att det är ett tomt element och att resten av innehållet inte nästlas innanför detta element. `xsl:sort` kan även användas tillsammans med `xsl:apply-templates`. Då nästlas elementet `xsl:sort` innanför `xsl:apply-templates` enligt:

```
<xsl:apply-templates select="//person" >
   <xsl:sort select="namn" />
</xsl:apply-templates>
```

Detta gör att `xsl:apply-templates` inte längre är ett tomt element och en sluttagg måste därför användas.