XSLT Tutorial

1. Introduction

XML is a standard which, amongst other things, can be used to record the structure of a text. Encoding a text in XML can be highly useful for a number of reasons. One of the most important advantages is that it enables computers to search the text for its contents beyond free text searches. XML tags are added primarily to enable machines to process the text more intelligently. For human beings, such explicit encoding is normally not needed. Human readers prefer to read texts with a standard lay-out, such as HTML documents in a web browser. If a text is laid out using traditional typographical means, human readers can usually access the contents of the text without any conscious efforts. Whereas XML elements and attributes can capture many semantic and logical aspects of a text, they may appear distracting to human beings who simply want to read a text. For this reason, XML files normally need to be transformed to something else to render them more readable or more usable for human readers. To do this, we can make use of the eXtensible Stylesheet Language for Transformations (XSLT). Like XML, XSLT was developed by the World Wide Web Consortium (W3C). This tutorial contains step-by-step instructions on how to write an XSLT stylesheet.

2. Framework

XSLT can be used to transform XML files into other XML files. An XML file is essentially a collection of data which have been structured according to a certain logic. XSLT can be used to change the structure of these data, or to re-build the document using another logic. For instance, data can be sorted alphabetically of numerically. It is also possible to add certain data to the document or to filter the contents of the document, on the basis of a given criterion. XSLT is used most frequently to transform the XML file into an (X)HTML file so that these data can be presented on-line.

The data in an XML file are always structured hierarchically. A well-formed XML file can always be represented as a tree diagram. The result of the transformation will also be a document that is structured hierarchically. It is often said that XSLT transforms a source tree into a result tree.

source tree
XSLT
result tree

The program that performs the transformation is called an XSLT processor. Such a program reads both the XML source file and the XSLT script and produces a result on the basis of these two. A great number of XSLT Processors can be downloaded free of charge. Examples include XALAN and SAXON. The XML editor Oxygen includes a numer of buit-in XSLT processors. Appendix A contains instructions on how such XSLT processors can be used.

To explain how XSLT works, this tutorial shall make use of a relatively simple XML document as a source tree. The name of this documents is *collection.xml*. It is a brief document that describes three letters from the collection of Leiden University

Library. The actual bibliographic information has been simplified slighty for pedagogical purposes. This source XML document is provided below.

Listing 1. Collection.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE collection SYSTEM "collection.dtd">
<collection>
<head>
   <title>Letter Collection of the University of Leiden
   </title>
</head>
<body>
<letter>
   <author  type="company">De Erven F. Bohn</author>
   <recipient type="company">William Blackwood &amp;
   Sons</recipient
   <place>Haarlem</place>
   <year>1873</year>
   <language>English</language>
   <annotations>
   <note>Minute</note>
   <note>Also available in microform</note>
   </annotations>
   <callnumber>BOH C 7 fol. 306</callnumber>
</letter>
<letter>
   <author type="person">Fuhri, Koenraad </author>
   <recipient type="person">Kruseman, Arie Cornelis
   </recipient>
   <year>1858</year>
   <language>Dutch</language>
   <annotations>
    <note>Also available in microform</note>
   </annotations>
   <callnumber>LTK 1795: 14</callnumber>
</letter>
<letter>
   <author type="company">De Erven F. Bohn</author>
   <recipient type="person">Lytton, Lord</recipient>
   <place>Haarlem</place>
   <language>English</language>
   <annotations><note>Minute</note>
   <note>Also available in microform</note>
   </annotations>
   <callnumber>BOH C 70, fol. 307</callnumber>
</letter>
</body>
</collection>
```

Next, we are going to write a file that can transform *collection.xml* into an HTML file.

3. Root element

An XSLT stylesheet is an XML file itself and should therefore comply with all the rules that govern the well-formedness of XML files. All the tags must be properly nested, for instance, and an XML-declaration should be present. In addition, it must also have one root element, and the name of this root element should be `<stylesheet>`.

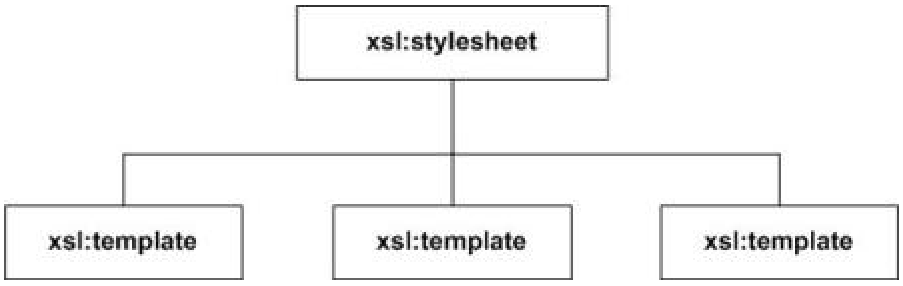
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">

</xsl:stylesheet>
```

The document root, `<stylesheet>`, contains a reference to a so-called namespace. In the example above, you can see that there is a reference to the official W3C XSLT recommendation namespace (http://www.w3.org/1999/XSL/Transform). An XML namespace is a collection of elements and attributes which are identified by a Uniform Resource Identifier (URI). Recall that XSLT is often used to transform an XML that is based on a particular DTD to an XML document that is based on another DTD. The tranformation can be, for instance, from TEI to XHTML. In such situations, name conflicts can occur if these two encoding schemes contain elements with exactly the same names. XML namespaces provide a solution for this problem. The namespace declaration in the example above specifies that each XSLT element must be qualified by the 'xsl' prefix. All the element names that are preceded by the xsl: prefix are thus linked to the URL http://www.w3.org/1999/XSL/Transform. As you can see, the element <stylesheet> also has the xsl-prefix. This way, the qualified elements and attributes can be distinguished from elements and attributes from other encoding schemes. If there are two elements with the same name, they can always be distinguished on the basis of the prefixes in front of them.

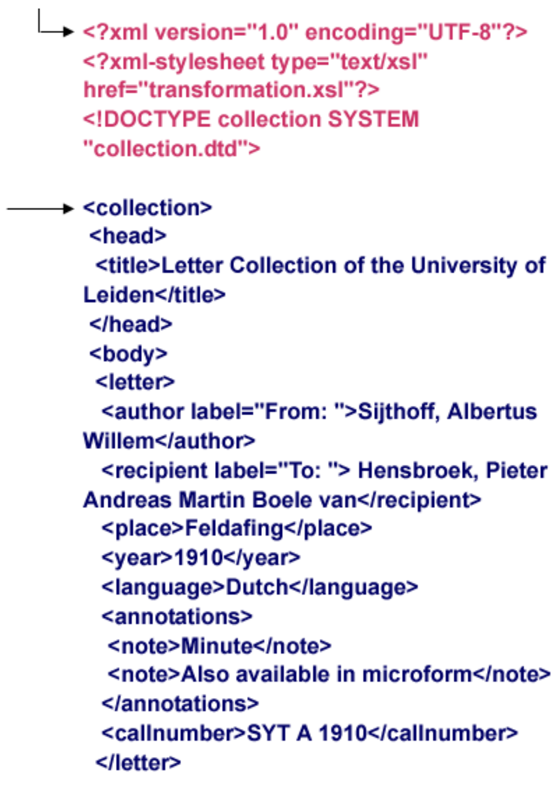
4. Templates

XSLT is not a programming language in which all the commands are simply executed one after the other. An XSLT stylesheet can be seen as a collection of scenarios which will only be carried out in the case of a certain event. Each scenario is described in a `<template>` element. They are all given directly under the <stylesheet> root. A tree diagram of an XSLT stylesheet would look as follows:.



Each <template> is in turn a collection of instructions. To activate a specific template, we must make use of the match attribute of the <template> element. The value of this attribute is normally the name of one of the elements in the XML source tree. At the start of the transformation, the XSLT processor will compare the elements in the source tree with the element names that are mentioned in the match attributes in the various templates. If a match is found, that template will be invoked and its instructions will be carried out.

To make sure that the transformation is carried out in a controlled manner, there must always be one template with a match attribute that points either to the document root or the root node. The document root is the very first element in the document that encapsulates all the other elements in the XML file. The term root node is used to refer to the absolute beginning of the file. This is in fact the very first character of the document. In an XML-file, the document root is normally preceded by a number of other lines. There is usually an XML declaration, and sometimes the document may also include certain processing instructions, such as a reference to a stylesheet. The information preceding the root of the XML tree may, under certain circumstances, also be relevant. The difference between the document root and the root node is also clarified in the illustration below.



```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="transformation.xsl"?>
<!DOCTYPE collection SYSTEM
"collection.dtd">

<collection>
 <head>
  <title>Letter Collection of the University of
Leiden</title>
 </head>
 <body>
  <letter>
   <author label="From: ">Sijthoff, Albertus
Willem</author>
   <recipient label="To: "> Hensbroek, Pieter
Andreas Martin Boele van</recipient>
   <place>Feldafing</place>
   <year>1910</year>
   <language>Dutch</language>
   <annotations>
    <note>Minute</note>
    <note>Also available in microform</note>
   </annotations>
   <callnumber>SYT A 1910</callnumber>
  </letter>
```

The first template that we include should thus contain a reference either to the document root or to the root node. If we point to the document root of collection.xml, the first template will look as follows:

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

</xsl:template>
```

The root node, on the other hand, is represented by a forward slash (“/”). If we choose to refer to this point, the first template will look as follows:

```
<xsl:template match=”/”>

</xsl:template>
```

In this tutorial, we shall choose the first option.

When the XSLT processor finds a “match” between the XML source tree and an XSLT template, this establishes a certain context. This means that the attention of the XSLT processor is focused on a particular location in the source tree, namely the element that is mentioned in the `match` attribute. All the instructions in this template will be interpreted within this context.

5. Selecting text

This tutorial will explain how the file `collection.xml` can be transformed into an HTML document. To do this, we are going to select some text from the XML source tree and combine it with HTML encoding. The easiest way to select data from an XML document is by making use of the `<xsl:value-of>` element. This element needs to be used with a select attribute which refers to the element that contains the data that we want to display. In the result tree, the entire `<xsl:value-of>` element will be replaced with the contents of the element that is mentioned in the select attribute.

If we want to display the title of the document from `<head>` in an HTML `<h2>`, the following stylesheet can be used:

Listing 2.

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

<xsl:template match=”collection”>
   <html>
        <head><title>XSLT transformation</title></head>
        <body>
        <h2><xsl:value-of select=”head/title”/></h2>
        </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

The XSLT processor will perform the instructions between the opening and closing `<template>` tags if in the source tree an element is found with the name `<collection>`. If a match can indeed be established, a number of HTML codes are created first. The contents of the `<title>` element under `<head>` will be inserted into the HTML page that has just been made, at the location of the `<xsl:value-of>`-element. The context is set at the element that is mentioned as value of the match attribute. In this case, this is the element `<collection>`. If we need the contents of elements that are not in the immediate context, we need to tell the XSLT parser where this element can be found in the tree by providing the path to this element. Different levels in the XML document are separated using the forward slash ("/").This is in fact a rule from the XPath language, which will be discussed in more detail in a later section of this tutorial. XPath, in short, is a technique which enables us to refer to specific locations in XML documents. The XPath language The paths that are given must depart from the element that is mentioned in the match attribute. From listing 1, we can see that get to `<title>`, we need to go to `<head>` first. The path that is mentioned in `<xsl:value-of>` is thus head/title. `<collection>` itself does not need to be mentioned, as this element is already mentioned in the `match` attribute.

Transorming collection.xml on the basis of listing 2 would result in the following HTML code:

```
<html>
   <head>XSLT Transformation</head>
   <body>
   <h2>Letter Collection of the University of Leiden
   </h2>
   </body>
</html>
```

Next, we are going to create a HTML list that contains al the letters that are described in the source file. These letters can all be found within the `<body>` element, underneath `<collection>`. When we use `<xsl:value-of select="body">`, however, this will have the disadvantage that the ENTIRE <body> is selected. This includes every single element between the opening and the closing tag of `<body>`. In this example, we want to be able to select specific elements within `<body>`, such as the name of the sender of a letter and the date of the letter. Since these elements are not in the direct context, we need to formulate a path towards these elements again. To select the author of a letter, for example, the following expression can be used:

```
<value-of select="body/letter/author">
```

Listing 3 is a stylesheet that can select the sender, recipient, place, year and callnumber from the letter. As can be seen, the XSLT elements can be combined quite easily with HTML tags. The names of the author and the recipient, for instance, are given in bold. It is important to realise that XHTML should be used, instead of regular HTML. XHTML is, simply put, the variant of HTML that is based on XML. This language is more strict and more consistent than 'normal' HTML. An important difference is that each element that is opened must also be closed. Some elements, such as <META>, <HR> (horizontal line) and <BR> (line break), are often not closed correctly in HTML. However, make sure that you do this when you use XHTML in a XSLT stylesheet. For this reason, <br/> must be used instead of only <br>.

Listing 3.

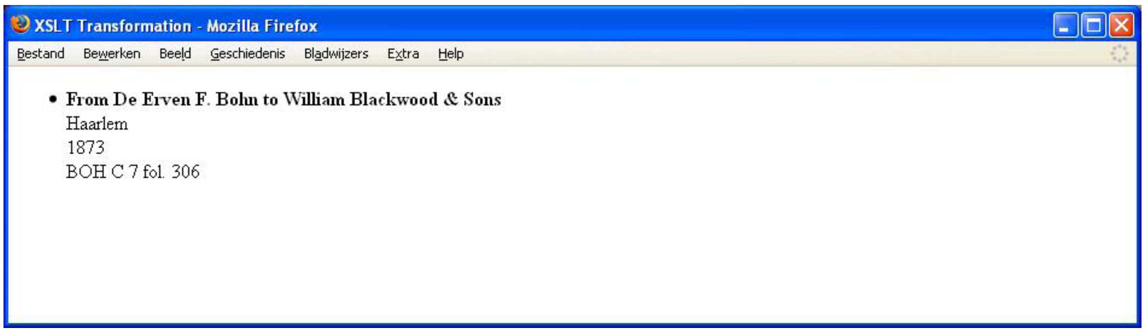
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">

<xsl:template match="collection">

<html>
   <head><title>XSLT Transformation</title></head>
<body>
<p>
   <ul>
   <li>
        <b>From <xsl:value-of select="body/letter/
        author"/> to <xsl:value-of select="body/
        letter/recipient"/> </b><br/>
        <xsl:value-of select="body/letter/place"/>
        <br/>
        <xsl:value-of select="body/letter/year"/> <br/>
        <xsl:value-of select="body/letter/callnumber"/>
   </li>
   </ul>
</p>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

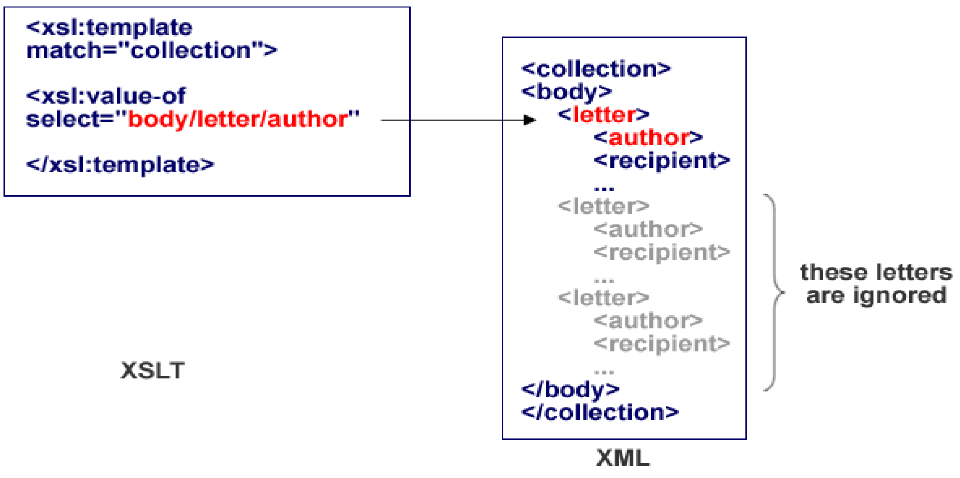
6. Iteration

If the file `collection.xml` is transformed using the stylesheet that is provided as listing 2, the result will look as follows in an internet browser:

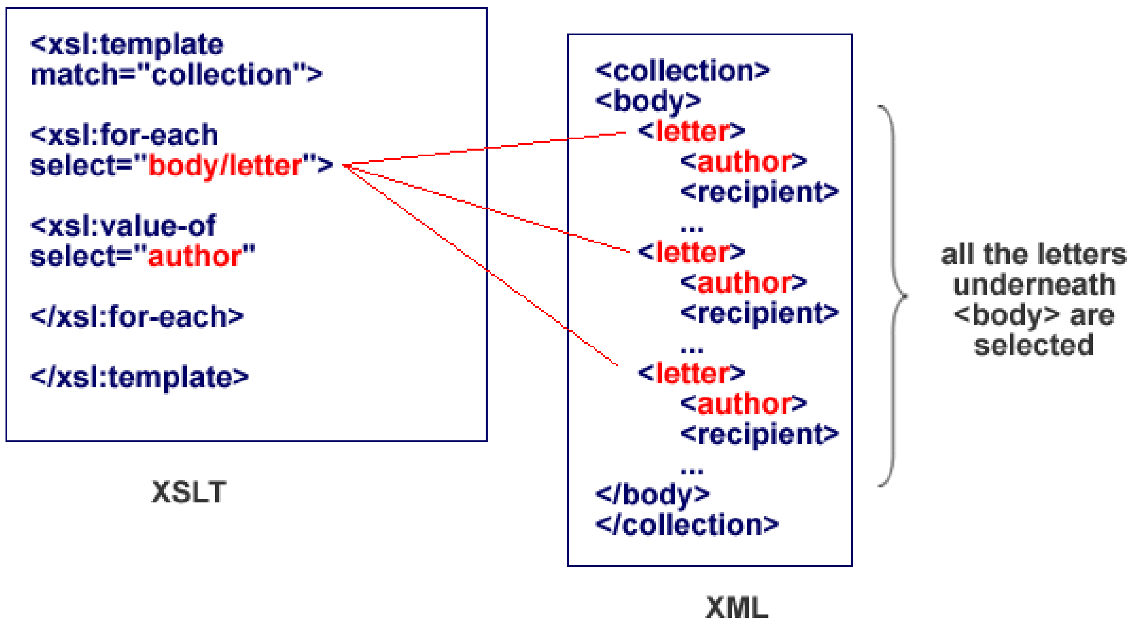

XSLT Transformation - Mozilla Firefox
Bestand Bewerken Beeld Geschiedenis Bladwijzers Extra Help

- **From De Erven F. Bohn to William Blackwood & Sons**
  Haarlem
  1873
  BOH C 7 fol. 306

As you can see, only one letter is shown, whereas the XML file actually describes three different letters. The `<xsl:value-of>` element is used to navigate to specific elements, but when different elements can be addressed with the same path, only the first element will be selected.



If all the elements on this level need to be shown, the element `<xsl:for-each>` will provide a solution. This element has a select attribute which refers to the element that occurs more than once on a certain level in the tree. The instructions which must be performed for each element must be included in between the opening and closing tag of the `<xsl:for-each>` element. The situation can be visualised as follows:



Note that there is an important difference between the `<xsl:value-of>` element and the `<xsl:for-each>` element: `<xsl:for-each>` changes the context. The paths within the opening and closing tag of `<xsl:for-each>` do not longer depart from `<collection>`, but from <letter>, which is the element which is mentioned in the select attribute of the `<xsl:for-each>` element. Within `<xsl:for-each>`, `<xsl:value-of select=”author”>` thus needs to be used instead of `<xsl:value-of select=”body/letter/author”/>`. Listing 3 shows how our stylesheet in progress can be improved.

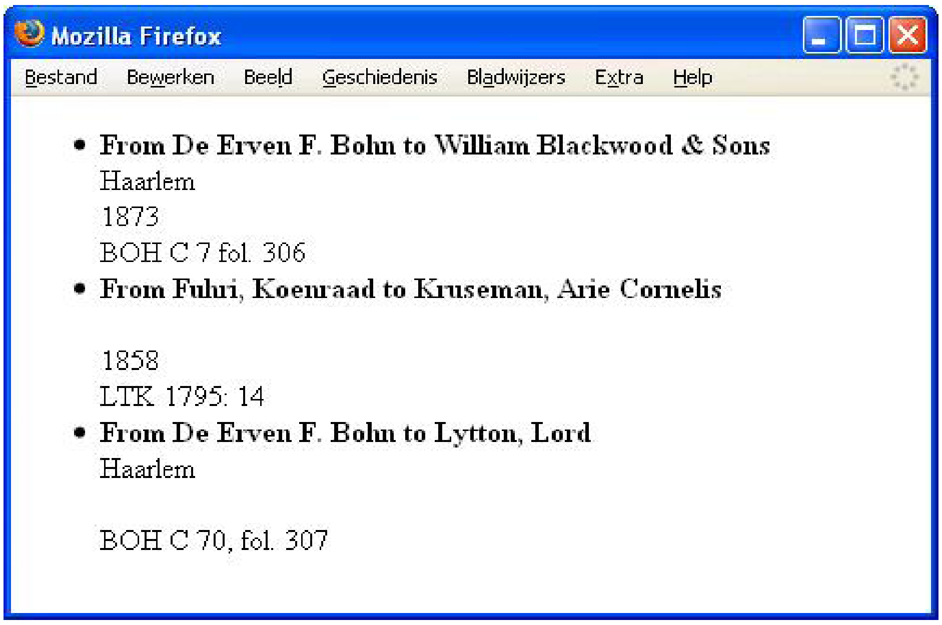
Listing 3.

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

<xsl:template match="collection">
<html>
<head></head>
<body>
<p>
   <ul>
   <xsl:for-each select="body/letter">
        <li>
        <b>From <xsl:value-of select="author"/>
        to <xsl:value-of select="recipient"/>
        </b><br/><xsl:value-of select="place"/>
        <br/><xsl:value-of select="year"/><br/>
        <xsl:value-of select="callnumber"/>
        </li>
   </xsl:for-each>
   </ul>
</p>
</body>
</html>
</xsl:template>

</xsl:stylesheet>
```

If the XML source, *collection.xml* is tranformed on the basis of the stylesheet that is given as listing 3, all the letters will be shown:



- **From De Erven F. Bohn to William Blackwood & Sons**
  Haarlem
  1873
  BOH C 7 fol. 306
- **From Fuhri, Koenraad to Kruseman, Arie Cornelis**

  1858
  LTK 1795: 14
- **From De Erven F. Bohn to Lytton, Lord**
  Haarlem

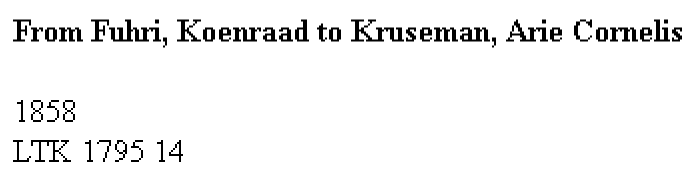
  BOH C 70, fol. 307

7. Setting conditions

The stylesheet that was discussed at the end of section 6 contains the following fragment:

```
From <xsl:value-of select=”author”> to
<xsl:value-of select=”recipient”/><br/>
<xsl:value-of select=”place”/><br/>
<xsl:value-of select=”year”/><br/>
<value-of select=”callnumber”/><br/>
```

According to the DTD on which this XML file is based, none of the elements within <letter> are mandatory. It may be the case that certain elements are missing. What will happen when, for example, `<place>` is missing? In the code above, it can be seen that a line break (`<br/>`) is inserted after each `<xsl:value-of>` element. If collection.xml is transformed with this stylesheet, there will be a slight problem with the second letter. Apparently, the place in which this letter was written could not be established. The stylesheet cannot select a value, and, since a line break is printed after each command (`<br/>`), an empty line will be printed. This results in a somewhat confusing HTML code:



From Fuhri, Koenraad to Kruseman, Arie Cornelis

1858
LTK 1795 14

This problem can be solved by selecting certain elements only under a certain condition. To define a condition, use the element `<xsl:if>`. The instruction between the opening and closing tag of this element will be performed only if this condition can be evaluated as true. The condition must be included in the test attribute of `<xsl:if>`. Our current stylesheet can be improved by surrounding all the optional elements by `<xsl:if>`, as follows:

```
...
<xsl:if test=”place”>
<xsl:value-of select=”place”/>
</xsl:if>
...
```

This script only shows the contents of the `<place>` element if this element is actually present.

8. Sorting

The letters in the file collection.xml are not sorted in any particular way. Sometimes, however, you may need to sort data alphabetically or chronologically. This can be achieved by making use of the `<xsl:sort>` element. This element can only be used as a direct child of `<xsl:for-each>` and `<xsl:apply-templates>` (the latter element will be discussed later on in this tutorial). <xsl:sort> requires a select attribute which indicates the element that needs to be used as a basis for the sorting operation. In this select attribute, you include an XPath expression.

`<xsl:sort>` can sort data both alphabetically and numerically. By default, data are sorted alphabetically. If you want to make this explicit, you can use the `data-type` attribute with the value “text”. To sort nodes numerically, use the `data-type` attribute with the value “number”.

In addition, the elements can be sorted in a descending and an ascending order. This can be specified with the order attribute, which can take either the value “ascending” or “descending”. The ascending order is the default value. Note that the select attribute can only refer to elements that can be reached from the current context. If <sort> is included as a direct child of `<xsl:for-each>`, the path that you mention in the <sort> element must depart from the element that is mentioned in `<xsl:for-each>`.

The following stylesheet will sort the letters chronologically. Note that the elements within the `<xsl:for-each>`-loop are selected only if they are actually present. This is achieved with `<xsl:if>`.

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

<xsl:template match=”collection”>

<html><head/>
<body>
<p>
   <ul>
        <xsl:for-each select=”body/letter”>
        <xsl:sort select=”year”/>
        <li>
             <b>From <xsl:value-of select=”author”/>
             to <xsl:value-of select=”recipient”/>
             </b> <br/> <xsl:if test=”place”>
             <xsl:value-of select=”place”/>
             <br/> </xsl:if>
             <xsl:if test=”year”>
                  <xsl:value-of select=”year”/><br/>
             </xsl:if>
             <xsl:if test=”callnumber”>
                  <xsl:value-of select=”callnumber”/>
             </xsl:if>
        </li>
   </xsl:for-each>
</ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

9. Navigation

During the explanation of the element <xsl:value-of>, it was explained that when elements are outside of the context, these elements can still be selected by making use of the forward slash. The value of the select attribute follow the rules that are defined in the XPath language. XPath is a language that is maintained by W3C. It enables us to navigate to different points in the XML Document. XPath represents the XML document as a collection of nodes. All of these nodes can be reached by making use of special symbols. A number of these symbols and their meaning are explained in the table below.

Symbol	**Description**
`/`	The forward slash is used to navigate to elements which are outside of the context. Example: `collection/body/letter` Using the forward slash at the beginning of an expression means that the path that follows departs from the root node.
`//`	The double forwards slashes indicate that the exact hierarchical position is irrelevant. If the two slashes are followed by the name of an element, this expression will return all the elements that can be reached from that context, even if they are are on different levels
`.`	The full stop is used to refer to the highest element in the context that has been established. This symbol can be used, for example, in a <xsl:for-each> element, when the element occurs more than once and also needs to be displayed more than once. Example: <xsl:for-each select=”annotations/note”> <xsl:value-of select=”.”> </xsl:for-each>
`*`	The asterisk represents any direct child of an element. In the example below, the asterisk refers to the first subelements of the element “letter”. Example: collection/body/letter/*
`../`	Two full stops followed by a forward slash will instruct the XSLT processor to jump to a higher level in the tree. When the context is set at <body>, the next expression can be used to jump to the element <head>, which is on the same level. Example: ../head
`@`	The ‘at’ is used to navigate to the value of an attribute. The symbol is followed immediately by the name of the attribute whose value you want to select. Example: collection/body/letter/author/@type

[]	The result of the transformation can be filtered by making use of a criterion, which must be placed within square brackets.

The possibility to add criteria enables XSLT programmers to influence the result of the transformation very specifically. Imagine, for instance, that we want to select letters only if the place of the sender is known. We can do this by using the following XPath expression:

```
<xsl:value-of select=”letter[place]”>
```

With this addition, the letter will only be selected if an element with the name `<place>` is present. In addition, it is possible to compare the contents of an element to a certain value. The following operators can be used:

&gt;	Greater than
lt;	Less than
=	Equal to
!=	Nor equal to

Note that the symbols used for “less than” and for “greater than” are XML entities. We cannot use the characters “<” and “>” themselves, because otherwise a parser might confuse these with the end or the start of an XML element.

When something needs to be compared with a string (a fragment of text), you need to include the string within single quotes. The double quotes are reserved in XML to indicate the value of an attribute. Using a condition within the square brackets, directly after the name of an element, can filter the result of the transformation very effectively. We could, for instance, produce selection such as the following:
- all letters written after 1870
- all letters written in English
- all letters NOT written in German .

When the context has been set at the element <body>, the above selections can be made on the basis of the following criteria:

```
<xsl:value-of select=”letter[year > 1870]”>
<xsl:value-of select=”letter[language = ‘English’]”>
<xsl:value-of select=”letter[language != ‘German’]”>
```

Finally, XPath also provides a number of function which can give information about the data in the XML document. An overview of the most important XPath functions is provided below.

Symbol	**Parameter**	**Description**
count()	nodeset	counts the number of results in a certain search action

name()		returns the name of an element (instead of the value)
position()		indicates the hierarchical position of an element.
contains()	string, string	checks if the string that is mention first is contained within the second string that is given.

10. Multiple templates

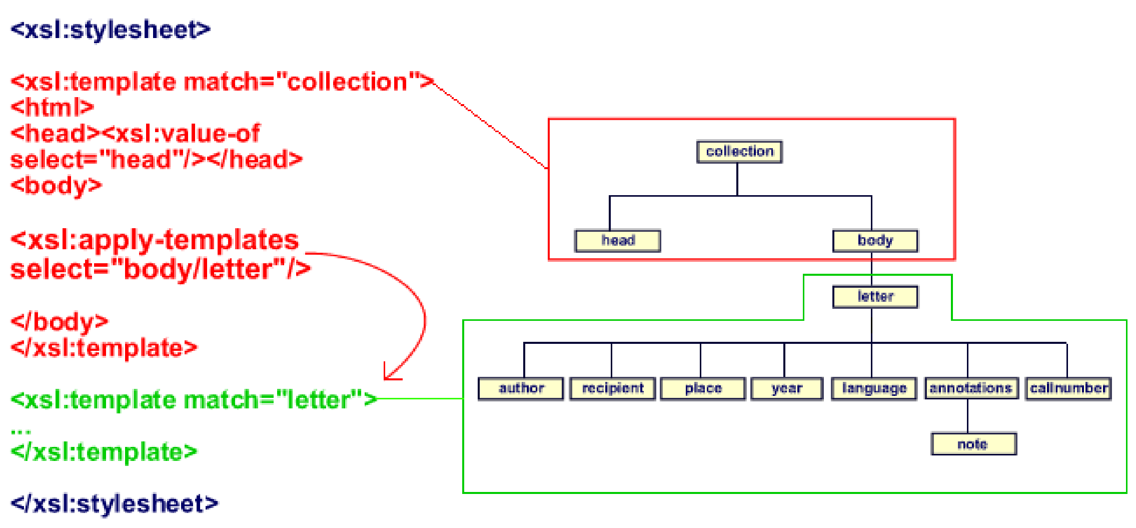
The stylesheet that was discussed in sction 2.5 consisted of a single template only. In general, XSLT stylesheets contain several templates. Using multiple template can yield a number of advantages.

-	The structure of the stylesheet as a whole will be more clear and easier to understand. Different tasks and different aspects of the transformation can be regulated in different dedicated templates. If an error arises somewhere it will also be much easier to locate that error and to correct it.
-	The stylesheet will be more efficient, because templates can be used several times. When a certain task needs to be executed repeatedly, if will be more efficient to invoke a template repeatedly than to repeat the same instructions over and over again.

Earlier in this tutorial, it was explained that the XSLT processor follows a certain procedure at the onset of a transformation. In short, the XSLT processor firstly considers the XML element that are available. Next, the XSLT stylesheet will be consulted to see which elements are mentioned in the match attributes of the various templates. If a match is found, the actions in that particular template will be executed. This will set the transformation in motion.

It is also possible to instruct the XSLT processor to look once more for matching templates from within a template that is already activated. The element that is used for this purpose is `<xsl:apply-templates>`. Using `<xsl:apply-templates>` will instruct the XSLT processor to look for templates once more. The `<xsl:apply-templates>` element can be used either with or without select attribute. When a select attribute is included, the context will be changed. This will force the XSLT processor to look for templates that match the element that can be found in the subsection of the tree that is headed by the element that is mentions after select.

The image below visualises the usage of `<xsl:apply-templates>`.



```
<xsl:stylesheet>

<xsl:template match="collection">
<html>
<head><xsl:value-of
select="head"/></head>
<body>

<xsl:apply-templates
select="body/letter"/>

</body>
</xsl:template>

<xsl:template match="letter">
...
</xsl:template>

</xsl:stylesheet>
```

It can be seen in this illustration that the two templates each have their own responsibility. The first template (which is rendered red) firstly serves to "start up" the transformation. This template contains a number of actions which need to be carried out in any case, such as setting up the basic structure of the HTML file. Within the HTML <body>, this template uses the `<xsl:apply-templates>` element. The select attribute refers to "body/letter". This first template hands the responsibility for the transformation over to another template in the stylesheet. The XSLT processor is asked to look for templates for the element `<letter>` and for all the subelements of that `<letter>`. If a match is found, these templates will be executed. The transformation of the individual letters takes place in a template which is included specifically for this purpose. In the illustration above, this template is drawn in green.

The first template in the example, the one that refers to the document root, will be invoked only once. The reason for this is that a valid XML document can only contain one document root and only one root node. For elements that are at a lower level in he hierarchy, the situation is different. These elements may occur many different times. The template will be invoked each time the element is found in the template. This mechanism is also clarified in the illustration below. The template that contains the match attribute with the value "letter" is invoked for each `<letter>` element that is found at the place that is mentioned. If the source tree contains four `<letter>` elements, the template with attribute `match="letter"` will also be invoked four times. For this reason, the element `<xsl:apply-templates>` can also be used as an alternative to `<xsl:for-each>`!

The following stylesheet will produce the exact same result as the stylesheet that is discussed in section 6.

Listing 4.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
Transform" version="1.0">
<xsl:template match="collection">
        <html>
            <head/>
            <body>
        <ul>
        <xsl:apply-templates select="body/letter"/>
        </ul>
        </body>
    </html>
    </xsl:template>

    <xsl:template match="letter">
        <li>
            <b>From <xsl:value-of select="author"/> to
             <xsl:value-of select="recipient"/>
            </b>
            <br/>
            <xsl:if test="place">
                <xsl:value-of select="place"/>
                <br/>
           </xsl:if>
            <xsl:if test="year">
                <xsl:value-of select="year"/>
                <br/>
            </xsl:if>
            <xsl:if test="callnumber">
                <xsl:value-of select="callnumber"/>
            </xsl:if>
        </li>
    </xsl:template>

</xsl:stylesheet>
```

11. <xsl:apply-templates> and <xsl:value-of>

In the previous section, it was explain that `<xsl:apply-templates>` can be used as an alternative for `<xsl:for-each>`. In addition, `<xsl:apply-templates>` can also be used as an alternative for `<xsl:value-of>`. In listing 4, all <xsl:value-of> elements in the second <template> can be replaced with <xsl:apply-templates>, as follows:

`Listing 5.`

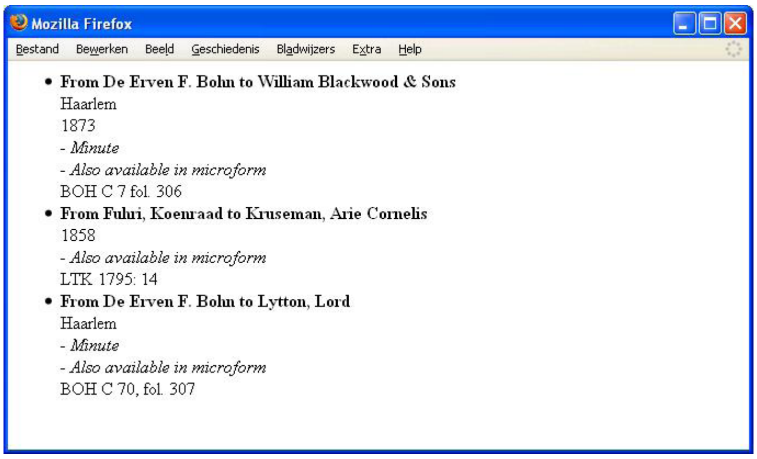
```
<xsl:template match="letter">
    <li>
     <b>From <xsl:apply-templates select="author"/>
     to <xsl:apply-templates select="recipient"/></b>
     <br/>
     <xsl:if test="place">
          <xsl:apply-templates select="place"/>  <br/>
     </xsl:if>
```

```
 <xsl:if test="year">
     <xsl:apply-templates select="year"/><br/>
</xsl:if>
 <xsl:if test="callnumber”>
     <xsl:apply-templates select="callnumber"/><br/>
</xsl:if>
</li>
</xsl:template>
```

The result of the transformation will be exactly the same, despite the fact that no templates have been included for the elements that are referred to in the `<xsl:apply-templates>` elements. These elements are transformed nevertheless. The reason for this is that when a template is missing for a certain element, the XSLT processor will use default template. This default template does not contain any specifications for a specific presentation. The only thing that happens is that the contents of this element are displayed. Using `<xsl:apply-templates>` instead of `<xsl:value-of>` thus has the exact same effect in this particular stylesheet.
There is one important difference. `<xsl:value-of>` simply selects the contents of the element, whereas `<xsl:apply-templates>` will primarily instruct the XSLT processor to search for matching templates for the elements it comes across. In the absence of such a template, the default template will be used.
The big advantage of using `<xsl:apply-templates>` can be illustrated by adding another template specifically for the `<note>` element. As can be seen in the tree diagram in listing 1, this is an element that can be found at a lower branch of the XML tree.

```
<xsl:template match="note">
<i>
<xsl:text>- </xsl:text>
<xsl:value-of select="."/>
</i> <br/>
</xsl:template>
</xsl:stylesheet>
```

The transformed file will look as follows:



- **From De Erven F. Bohn to William Blackwood & Sons**
  Haarlem
  1873
  - *Minute*
  - *Also available in microform*
  BOH C 7 fol. 306
- **From Fuhri, Koenraad to Kruseman, Arie Cornelis**
  1858
  - *Also available in microform*
  LTK 1795: 14
- **From De Erven F. Bohn to Lytton, Lord**
  Haarlem
  - *Minute*
  - *Also available in microform*
  BOH C 70, fol. 307

Note that the HTML file does not simply contain the text of the element `<annotation>`. The contents of the subelement `<note>` are displayed in a different way. This presentation is based on the instructions in the template with the attribute `match="node"`. If we replace the element `<xsl:apply-templates select="annotations"/>` with `<xsl:value-of select="annotations">`, no template will be invoked for <note>. The transformed file will then look as follows:

The conclusion should be that, if we want to be able to influence the appearance of certain subelements, we should make dedicated templates for these elements and make use of `<xsl:apply-templates>` to invoke these templates.