BeginnerSimpleXSLT

- FG:FredericGlorieux

Configure your environment

TODO. Text Editor with Mozilla ?

Source example

First of all, you should see a little of how XML looks. So let's take a very short document

```
<article>
  <title>XSLT, in hope to be simple</title>
  <author>
    <firstname>James</firstname>
    <lastname>Clark</lastname>
  </author>
  <author>Kay, Michael</author>
  <abstract>
Sorry, James Clark and Michael Kay to use your names for a so dummy example.
Please, consider that  as a tribute.
  </abstract>
  <section>
    <title>section 1</title>
    <para>
I don't now the exact story of <concept>XSL</concept>,
but the guys behind that are very clever.
    </para>
  </section>
</article>
```

This doc is more expressive than HTML, especially as I can change the names of elements (Schema/DTD) to express more precisely what I want (<abstract/>, <author/>). This is a not so bad way to store my texts, because they are structured, and quite self documented. Now, how do we take advantage of this? XSLT, of course.

Root

This will be the root XSL document in which all snippets will now go.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes" encoding="UTF-8"/>
  <!-- here put your templates -->
</xsl:transform>
```

`<?xml version="1.0" encoding="UTF-8"?>` This is the first declaration that all XML documents should have. Note the encoding precision: UTF-8 is default for XML. Languages other than English may need a different encoding. For an XSLT, leave it as UTF-8: your XML editor should know it.

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

Root element with a namespace declaration (XML-spec). All elements beginning with xsl:* are mapped as the namespace-uri "http://www.w3.org/1999/XSL/Transform". This is the only string identifying your tags as XSLT instructions to process. This means you can also say

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

This could be useful when an XSLT should output another XSLT. However, for now let's keep it simple: it will be easier for copy/paste. (By the way, also keep the @version attribute).

```
<xsl:output method="xml" encoding="UTF-8"/>
```

An xsl:transform is a common XML document, that will be processed. All xsl:* elements will now be instructions to transform from an input to an output. Input should be valid XML, output could be text, xml, html (with unclosed tags, character entities like ...). This is the reason for @method attribute. Note @encoding, one more time, use UTF-8 for XML usages, ISO-8859-1 could be useful if you have a strange browser display.

Pull

As I've put some nice tags in my XML document, it's probably best to use them. For example, I want to extract title|author|abstract of my articles to have a short version. So let's write my first template to **pull** what I need in the source to output it. The pull method is the more intuitive for developpers, but definetely no the more efficient for XSL. But we need to begin ?

match, select, XPath

```
<!--
this template supposed to be in a xsl:transform described upper
test it fastly, but delete it to continue this page
--->
<xsl:template match="/">
<!-- here, we are at root of the xml input, before the first element -->
  <xsl:copy-of select="."/>
</xsl:template>
```

`<xsl:template match="/">`
The XSLT engine is a kind of filter, processing an XML input, with the stylesheet instructions. Imagine a complex search/replace, except instead of working on flat text it's working on a tree of elements. At first, the engine searching for a template matches the root. That's what is provided here, in the @match, with the "/" value. Unix users will quickly understand this syntax. Now, the current node will be the root of the XML input (like after a "change directory").

In fact, this very complex template is doing: **nothing**. Output should be almost exactly the same as the input in XML terms, except the encoding or some other xsl:output effects.

It's also a fast way to debug XSL, to say "Where am I?", "What's in?". The instruction *copy-of* outputs the nodes selected by the expression in @select, here: ".". The dot expresses the current node. It's another important characteristic of expressions, called XPath syntax.

But don't forget what we really want. From the input given upper, imagine you want short output for metadata extracting, to put in your database, or your search engine. You handle some precise and simple type of fields and documents.

```
<record>
  <title>XSLT, in hope to be simple</title>
  <description>
Sorry, James Clark and Michael Kay to use your names for a so dummy example.
Please, consider that  as a tribute.
  </description>
  <creator>Clark, James</creator>
  <creator>Kay, Michael</creator>
</record>
```

Take notice that order is a bit different from the source (a real reason to use Pull method), some names are different, you want to normalize the creator field.

```
  <xsl:template match="article">
    <record>
      <!-- xsl:copy-of, not a good xsl practice -->
      <xsl:copy-of select="title"/>
      <!-- Better here, open an element, and put the value (text only) -->
      <description>
        <xsl:value-of select="abstract"/>
      </description>
      <!-- Begin a push logic, see below a template to handle <author/> -->
      <xsl:apply-templates select="author"/>
    </record>
  </xsl:template>
```

Now, I'm matching XML input with an `<article/>` root element (happily, it's my input). This will change the current node inside the template for all new XPath expressions (ex: @select). Note also the <record/> element, which is not in the XSLT namespace (no xsl: prefix), so the engine will interpret it as output. And now point the three ways to get content from source.

- xsl:copy-of, working in the article context, so the @select is catching and ouputting the `<title/>` node.
- xsl:value-of, same context as the copy-of but output only text.
- xsl:apply-templates, here we are in more tricky, where XSLT power is. This mean, let the hand to some one who will handle specifically <author/> element.

Push

Push method means, process source document as it is, handle nodes by specific templates, and in context, stop, continue, or modify what you want. The most beautiful (and not so easy to understand) push stylesheet is the identity transformation. Instead of a big copy-of all input, imagine to process all nodes by default, and copy each one by one. Seems not very efficient, OK, but very powerful, let's see. Add this template to

The identity transformation. Simple and tricky, like nice computing concepts. It match and process all *node()* (<element/>, but also text(), <!- - comment() - ->...) and | attributes @* (see xsl:copy, xpath short

```
<xsl:template match="@*|node()">
  <!-- each node, copy it, only him, not his children -->
  <xsl:copy>
    <!-- process all children, handle by this template, or others :o) -->
    <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
</xsl:template>
```

This a good start to now see how is processed your document. This will output all the node from source, so you will be able to verify if your stylesheet is handling what you want, like you want. At this step, if you have the `match="article"` and the `match="@*|node()"` templates, your output should be :

```
<record>
  <title>XSLT, in hope to be simple</title>
  <description>
Sorry, James Clark and Michael Kay to use your names for a so dummy example.
Please, consider that  as a tribute.
  </description>
  <author>
    <firstname>James</firstname>
    <lastname>Clark</lastname>
  </author>
  <author>Kay, Michael</author>
  <author>
    <firstname>Dummy</firstname>
  </author>
</record>
```

Indentation of your output depend on your xsl engine, but you see that the <article/> from the source is handle to write a <record/>, <title/> and <description /> are correctly handled, and the effect {{{<xsl:apply-templates select="author"/>
}}} is now, copy each with children, like it is in the source. It means that the generic copy template has less priority than more precise matching `match="article"`. This is a good way to begin to work and see what is going on. If you add this, all `<author>` (even where they are) will be rename to `<creator`:
.

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

Remember, we don't to output `<firstname/> and <lastname/>` elements, but they are nice way to normalize name strings. So try to add these templates. They are not the most efficient in the world, but could give idea of how powerful could be to deal with priority of templates (implicit tests).

```
<!-- other processes could be added here one day --->
<xsl:template match="lastname | firstname">
  <xsl:value-of select="."/>
</xsl:template>
<!-- in case of author witn lastname *and* fistname, we can reorder and add a comma -->
<xsl:template match="author[lastname and firstname]">
  <creator>
    <xsl:value-of select="lastname"/>
    <xsl:text>, </xsl:text>
    <xsl:value-of select="firstname"/>
  </creator>
</xsl:template>
```

HTML

TODO

References

- XPath
- XSLT
- XSL in Docbook let masters talk

See Also

about xsl	**for beginner**
<<PageList(xsl)>>	<<PageList(beginner)>>
cited by	**from authors**
<<FullSearch>>	<<FullSearch('FredericGlorieux')>>

CHANGES

- 2004-07-16:FG more and refactoring
- 2003:FG creation

TODO

- an HTML section
- an easy and tested config for first step in XSL
- correct mistakes