XML: XSLT

1. Introducción

1.1. Ideas básicas

- XSLT significa EXtensible Stylesheet Language Transformations y es parte de XSL.
- XSL (eXtensible Stylesheet Language) es una familia de lenguajes que permiten describir cómo debe ser presentada la información contenida en un documento XML.
- Los ficheros XSLT son ficheros XML.
- Se usa para transformar un documento XML (los datos):
  - en otro documento XML,
  - en otro documento que pueda ser reconocido por un navegador (HTML o XHTML).
  - en texto.
- Se suele decir que XSLT transforma un árbol XML fuente en un árbol XML resultado.
- Con XSLT en la transformación se puede
  - Añadir o eliminar elementos y atributos.
  - Renombrar elementos o atributos
  - También ordenar, realizar test, tomar decisiones sobre qué elementos ocultar o mostrar, etc.

1.2. Ejemplos de uso

- XML a XML

```
~/xml-a-xml$ xmlstarlet tr modifica.xsl catalog.xml
# O redireccionando a fichero
~/xml-a-xml$ xmlstarlet tr modifica.xsl catalog.xml > res.xml
~/xml-a-xml$ more res.xml
```

- XML a texto

```
~/xml-a-texto$ xmlstarlet tr genera-csv.xsl catalog.xml
```

- XML a HTML
  - En línea de comandos

```
~/xml-a-html$ xmlstarlet tr rsspretty.xsl rss.xml
~/xml-a-html$ xmlstarlet tr rsspretty.xsl rss.xml > resultado.html
~/xml-a-html$ firefox resultado.html
```

  - **En navegador**: no es posible realizar directamente las transformaciones XML a HTML en el navegador, es necesario iniciar un servidor web para hacerlas. Para ello
    - inicie Live Server en VScode (*View / Command Palette y busque "Live Preview: Start Server"*).
    - Al no existir el fichero `index.html` el navegador integrado muestra un error.
    - Abra el navegador externo, y si modifica la URL podrá navegar por la estructura de directorios y localizar los ficheros de ejemplo.
    - Ahora si podrá ver el resultado de mostrar el fichero de datos con transformación (`rss_con_xsl.xml`) y sin transformación (`rss.xml`).

2. Formato fichero XSLT

Observe el contenido del fichero `plantilla.xslt`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <!-- Formato de salida. Probar sin indent (o "no") -->
  <xsl:output method="xml" encoding="utf-8" indent="yes"/>

  <!-- Se aplica si encuentra expresión XPATH en match -->
  <xsl:template match="/root">
    <!-- Crea comentario en salida desde XSLT -->
    <xsl:comment> Comentario creado desde XSLT </xsl:comment>
    <!-- creación de elemento con xsl:element -->
    <xsl:element name="elementoroot">
        <!-- creación de hijo "directa" -->
        <hijo1> Contenido textual </hijo1>
        <!-- creación de elemento con xsl:element -->
        <xsl:element name="hijo2">
            <xsl:text> Nodo de Texto </xsl:text>
        </xsl:element>
      </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Se puede apreciar:

- Es un fichero XML.
- El elemento raíz es `xsl:stylesheet`.
- Se especifica formato de salida con `xsl:output`.
- El contenido del elemento `xsl:template` se aplica cuando la expresión XPATH del atributo `match` coincide en la instancia de datos.
- El elemento `xsl:element` genera un elemento XML.
- El elemento `xsl:text` genera un nodo de texto.
- Se pueden generar elementos XML indicando el nombre sin prefijo como en `<hijo1>`.
- El texto sin etiquetar genera un nodo de texto, como en `hijo1`.

Y si usamos la transformación `plantilla.xslt` sobre `plantilla.xml`:

```
~/paso_a_paso$ xmlstarlet tr plantilla.xslt plantilla.xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Comentario creado desde XSLT -->
<elementoroot>
  <hijo1> Contenido textual </hijo1>
  <hijo2> Nodo de Texto </hijo2>
</elementoroot>
```

3. Ejemplo paso a paso

Se usará en este apartado como instancia de datos el fichero `catalog.xml`.

3.1. Paso 01: `template match` y acceso a elementos y atributos

Dado el siguiente código

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

  <xsl:element name="coleccion">
    <xsl:element name="propietario">
      <xsl:value-of select="OWNER"/>
    </xsl:element>
  </xsl:element>

</xsl:template>
```

Observe:

- el `match` se hace con la expresión XPATH `/CATALOG`.
- se crea un elemento `coleccion` y dentro del mismo el elemento `propietario`.

- Se accede al valor de un elemento de la instancia de datos usando `xsl:value-of` y la expresión XPATH `OWNER` en el atributo `select` (tenga en cuenta que el era con `/CATALOG`, luego se accede a `/CATALOG/OWNER`).

Puede ver el resultado procesando `catalog.xml` con `catalog-01.xslt`:

```
$ xmlstarlet tr catalog-01.xslt catalog.xml
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
  <propietario>Jess Arzak</propietario>
</coleccion>
$
```

Si añadimos algo más de código:

```
  <xsl:template match="/CATALOG">
    <xsl:element name="coleccion">
      <xsl:element name="propietario">
        <!-- <xsl:value-of select="OWNER"/>
        <xsl:text>(</xsl:text>
        <xsl:value-of select="OWNER/@country"/>
        <xsl:text>)</xsl:text> -->
        <xsl:value-of select="concat(OWNER, '(',OWNER/@country, ')') "/>
      </xsl:element>
      <localiza>
        <xsl:value-of select="URL/LOCATION"/>
      </localiza>
    </xsl:element>
  </xsl:template>
```

- se puede usar la función `concat()` para simplificar el acceso a los valores (se muestra comentado la forma de hacerlo sin usar `concat()`)
- accedemos al valor de un atributo con la expresión `OWNER/@country`.
- se genera un elemento `localiza` accediendo a `URL/LOCATION`.

Puede ver el resultado procesando `catalog.xml` con `catalog-03.xslt`:

```
$ xmlstarlet tr catalog-03.xslt catalog.xml
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
  <propietario>Jess Arzak(uk)</propietario>
  <localiza>www.pruebas.local</localiza>
</coleccion>
$
```

3.2. Paso 02: bucles

Añadimos algo más de código, el que se muestra tras el comentario:

```
  <xsl:template match="/CATALOG">
    <xsl:element name="coleccion">
     ...
      <!--  Bucle para recorrer el catálogo   -->
      <xsl:for-each select="CD">
        <xsl:element name="disco">
          <xsl:value-of select="concat(ARTIST, ': ', TITLE)" />
        </xsl:element>
      </xsl:for-each>

    </xsl:element>
  </xsl:template>
```

El elemento `xsl:for-each` permite hacer un bucle en XSLT y selecciona cada uno de los elementos de un conjunto especificado de nodos del árbol XML.

En este caso para cada `CD` se genera un elemento de nombre `disco`, y su contenido se construye a partir del `ARTIST` y `TITLE` del `CD`. Observe las expresiones XPATH de

- Atributo `match` del elemento `xsl:template`
- Atributo `select` del elemento `xsl:for-each`
- Atributo `select` del elemento `xsl:value-of`

Puede ver el resultado procesando `catalog.xml` con `catalog-04.xslt`:

```
$ xmlstarlet tr catalog-04.xslt catalog.xml
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
  <propietario>Jess Arzak(uk)</propietario>
  <localiza>www.pruebas.local</localiza>
  <disco>Bob Dylan: Empire Burlesque</disco>
  <disco>Bonnie Tyler: Hide your heart</disco>
  ...
  <disco>Joe Cocker: Unchain my heart</disco>
</coleccion>
$
```

3.2.1. Paso 03: Opciones del bucle

Para ordenar el resultado de la salida basta añadir el elemento `xsl:sort` dentro del elemento `xsl:for-each`, especificando el criterio de ordenación en el atributo `select`:

```
  <xsl:for-each select="CD">
    <xsl:sort select="ARTIST" />
    <xsl:element name="disco">
      <xsl:value-of select="concat(ARTIST, ': ', TITLE)" />
```

```
    </xsl:element>
  </xsl:for-each>
```

Otras posibilidades:

- Especificar el orden con `order="ascending|descending"`.
- Por defecto la ordenación es alfanumérica. Si ordena números debe indicar `data-type="number"`.
- Si desea ordenar por más de un criterio se indican uno tras otro.

```
    <xsl:for-each select="CD">
      <xsl:sort select="PRICE" data-type="number" order="descending"/>
      <xsl:sort select="ARTIST" />
      <xsl:element name="disco">
        <xsl:value-of select="concat(PRICE, ' - ', ARTIST, ': ', TITLE)" />
      </xsl:element>
    </xsl:for-each>
```

Puede ver el resultado procesando `catalog.xml` con `catalog-05.xslt`.

3.2.2. Paso 04: Filtrando la salida del bucle con predicados

Puede filtrar los elementos que se recorren en el bucle `xsl:for-each` usando predicados XPATH en el atributo `select`. Por ejemplo

```
  <!--      Bucle para recorrer el catálogo      -->
  <xsl:for-each select="CD[YEAR &lt; '1984']">
    ...
```

Operadores válidos son:

- `=` (igual)
- `!=` (distinto)
- `<` (menor que)
- `>` (mayor que)

Puede ver el resultado procesando `catalog.xml` con `catalog-06.xslt`.

3.3. Paso 05: Selectivas

Se desea generar salida sólo si se cumple una cierta condición puede usar la selectiva `xsl:if` dónde la condición se especifica en el atributo `test`. Vea un ejemplo donde se añade el texto `OFERTA` al precio sólo si es menor que 8.

```
<xsl:for-each select="CD">
  <disco>
    <xsl:if test="PRICE&lt;8"> ¡OFERTA! </xsl:if>
    <xsl:value-of select="PRICE"/>
```

Si desea una selectiva múltiple (o un if-else) debe usar la selectiva multiple `xsl:choose`. Vea un ejemplo donde cada `when` evalúa una expresión y el `otherwise` sería el caso por defecto:

```
<disco>
   <xsl:choose>
     <xsl:when test="PRICE&lt;8">¡OFERTA!  </xsl:when>
     <xsl:when test="PRICE&lt;10">¡Buen precio! </xsl:when>
     <xsl:otherwise>
           <xsl:text>¡Novedad! </xsl:text>
     </xsl:otherwise>
   </xsl:choose>
   <xsl:value-of select="PRICE"/>
   ...
```

Puede ver el resultado procesando `catalog.xml` con `catalog-07.xslt`.

3.4. Paso 06: Generando atributos

Hasta ahora se han generados elementos y texto en la salida. Para generar un atributo (asociado a un elemento) debe usar `xsl:attribute`, darle un nombre y asignarle un valor.

Vea un ejemplo de cómo añadir el atributo `url` al elemento `localiza`, y asignarle a ese atributo el valor del elemento `URL/LOCATION`:

```
  <xsl:element name="localiza">
    <xsl:attribute name="url">
          <xsl:value-of select="URL/LOCATION" />
    </xsl:attribute>
    <xsl:value-of  select="TITLE" />
  </xsl:element>
```

Puede ver el resultado procesando `catalog.xml` con `catalog-08.xslt`:

```
$ xmlstarlet tr catalog-08.xslt catalog.xml
<?xml version="1.0" encoding="utf-8"?>
<coleccion>
  <propietario>Jess Arzak(uk)</propietario>
  <localiza url="www.pruebas.local"/>
  <disco>
    ...
```

> Nota: puede usar una notación abreviada para definir el elemento y su atributo definiendo el atributo junto con el elemento y delimitando su valor con llaves con `<localiza url="{URL/LOCATION}">`:

```
<localiza url="{URL/LOCATION}">
     <xsl:value-of select="TITLE" />
</localiza>
```

3.5. Paso 07: Más sobre bucles

3.5.1. Recorriendo elementos finales

En los bucles previos el elemento que se repite es un elemento contenedor. Veamos ahora un ejemplo en que el elemento que se repite es simple (no un contenedor). En este ejemplo vamos a recorrer los elementos `LINE` del elemento `DESCRIPTION`:

```
<xsl:for-each select="DESCRIPTION/LINE">
  <xsl:element name="p">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:for-each>
```

Observe que **para mostrar el valor del elemento LINE se usa la expresión XPATH "."** que indica el elemento actual.

Puede ver el resultado procesando `catalog.xml` con `catalog-09.xslt`.

3.5.2. Bucles anidados

En los datos en algunos `CD` aparece un elemento `BAND` con elementos `MUSICIAN` que se repiten. Para recorrerlos tendremos un segundo bucle que recorre los `MUSICIAN` (dentro del bucle que recorre los `CDs`).

```
<xsl:for-each select="CD">
  ...
  <banda>
     <xsl:for-each select="BAND/MUSICIAN">
        <xsl:value-of select="."/>,
     </xsl:for-each>
  </banda>
</xsl:for-each>
```

Puede ver el resultado procesando `catalog.xml` con `catalog-10.xslt`:

> Propuesta: ¿Cómo eliminamos la coma del último elemento? Ver apartados posteriores.

3.6. Paso 08: Funciones auxiliares

Hemos ya usado una función axiliar, `concat()`. Existen otras como `position()` o `last()` que nos pueden ser útiles.

Usando `position()`

```
<!-- Genera número con la posición, nos sirve para numerar -->
<xsl:value-of select="position()"/>
<!-- Trata distinto el primero -->
<xsl:if test="position()=1">PRIMERO</xsl:if>
```

Usando `last()`

```
<!-- Trata distinto el último -->
<xsl:if test="position()=last()">LAST</xsl:if>

<!-- Todos menos el último -->
<xsl:if test="position()!=last()">, </xsl:if>
```

Otra situación habitual es comprobar si un elemento existe: basta indicar el nombre del elemento en el `test` de `xsl:if`:

```
<!-- Comprobar si un elemento o atributo existe: -->
<xsl:if test="price">
 <xsl:value-of select="concat('-',price,'-')"/>
</xsl:if>
```

4. Generando ficheros de texto

Si la salida es texto se debe especificar en `xsl:output`

```
  <xsl:output method="text" encoding="utf-8" />
```

Si desea generar un salto de línea en el texto tienes dos opciones:

```
<!-- a) con salto de línea manual -->
<xsl:text>
</xsl:text>
<!-- b) con entidades: &#xa; o &#10; -->
<xsl:text> &#10; </xsl:text>
```

El caso más frecuente de generación de ficheros de texto es son los fichero CSV:

> Los archivos CSV (del inglés comma-separated values) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma) y las filas por saltos de línea.

Vea un ejemplo de transformación a CSV:

```
<xsl:output method="text" encoding="utf-8" />

<xsl:template match="/CATALOG">
    <xsl:text>Titulo, Autor&#xa;</xsl:text>
    <xsl:for-each select="CD[PRICE &lt;'8']">
        <xsl:value-of select="concat(TITLE, ', ', ARTIST, '&#xa;' )"/>
    </xsl:for-each>
</xsl:template>
```

Puede ver el resultado procesando `catalog.xml` con `catalog-texto-01.xslt`.

5. Generando HTML

Si la salida es HTML se debe especificar en `xsl:output`

```
    <xsl:output method="html" encoding="utf-8" />
    <xsl:template match="/CATALOG">
        <html>
            <head>
        <title>Ejemplo HTML XSLT</title>
      </head>
            <body>
        <h1>colección de <xsl:value-of select="OWNER" ></h1>
                <ol>
                    <xsl:for-each select="CD">
                        <li>
                            <xsl:value-of select="concat(TITLE, ', ',
ARTIST)"/>
                        </li>
                    </xsl:for-each>
                </ol>
            </body>
        </html>
    </xsl:template>
```

Puede ver el resultado procesando `catalog.xml` con `catalog-html-01.xslt`.

Para ver la transformación en un navegador debe incluir una instrucción de proceso en el fichero de datos que enlace con el fichero XSLT:

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

<!-- IP que le indica al navegador fichero a usar en la presentación -->
<?xml-stylesheet href="catalog-html-01.xslt" type="text/xsl"?>

<CATALOG>
  <OWNER country="uk">Jess Arzak</OWNER>
  ...
```