

Transformación de documentos: XSLT

Ha llegado el momento de encarar el hecho que XML no incorpora ninguna semántica intrínseca de presentación, por lo que sin una tecnología apropiada ningún procesador sabría como representar el contenido de un documento, como no fuera en forma de una cadena indiferenciada de caracteres. De hecho, cuando XML empezó su camino no existía ningún navegador capaz de mostrar sus documentos como no fuera en forma de un simple texto, lo que suponía un paso hacia atrás respecto a HTML, que si interpreta el documento de una página Web y lo traduce en texto e imágenes formateadas.

Por ello manejar documentos XML, bien por un humano bien por una aplicación software, conlleva que difícilmente se podrán usar en la forma en que se presentan al ser accedidos y en consecuencia inevitablemente estos documentos deberán ser objeto de algún tipo de transformación para adaptarse y ser útiles en cada contexto concreto.

En este tema se presentan las técnicas que permiten transformar un documento en otro, de forma que el resultado que se obtenga se adapte a los distintos usos y necesidades, que pueden ser muy numerosas. Como simple reflexión considérese la siguiente relación, que no pretende ser exhaustiva de posibilidades:

- Pasarlo a HTML para mostrarlo en un determinado tipo de terminal en la Web.
- Ponerlo en formato MIF (Maker Interchange Format) para posteriores intercambios.
- Darle un estilo para su presentación y publicación impresa.
- Pasarlo a WML para presentarlo sobre periféricos WAP.
- Convertirlo a distintos dialectos XML para la transferencia de datos en B2B.
- Convertirlo a texto plano.
- Etc.

El uso de transformaciones de documentos se da en dos grandes escenarios: en la conversión de datos entre aplicaciones XML y en el proceso de presentación de documentos XML. En el primer caso hay que recalcar que para este intercambio no es posible limitarse a la simple transformación de datos XML a otros datos XML, ya que también se deben combinar datos XML con otros que no lo sean y en consecuencia hay que dotar al XML de una tecnología básica que permita las conversiones correspondientes. El segundo problema surge de la propia arquitectura de la Web y de los propios requisitos de XML y en consecuencia hay que poder transformar los datos para poder presentarlos.

Resolver la presentación de un documento XML es inevitablemente complejo, no solo por los temas de formato sino también por la necesidad de contar con técnicas que hagan de puente con las posibilidades específicas de cada periférico (impresoras, pantallas, terminales de voz, etc.) a la hora que las personas puedan acceder a los contenidos de los documentos. Esta adaptación es imprescindible, ya que a modo de ejemplo hay que estar preparados tanto para que ésta pueda ser objeto de impresión, como de publicación en la Web, siendo obvio que tanto una como otra, presentan sus particulares características.

Desde los inicios del XML se fue consciente de la necesidad de disponer de herramientas que ejercieran un control detallado de la presentación en la Web y para ello se recurrió a los CSS como una tecnología ya existente que se aplicaba con éxito a los documentos HTML y que se adaptó al XML sin grandes dificultades. La utilización de los CSS para mostrar los documentos XML ya la hemos tratado en un tema

anterior.

A pesar de su buen funcionamiento, CSS es incapaz de ejecutar operaciones lógicas, tales como construcciones "if-then", que son imprescindibles para adaptar un documento de un tipo de presentación a otra y desafortunadamente afecta a la posibilidad de cambiar un documento XML a otro similar. Ante la perspectiva que los CSS podían ser insuficientes para la tarea de adaptación a los distintos periféricos, se tuvo que buscar más allá, especialmente con la aparición de los Esquemas y la creciente complejidad de las hojas de estilo.

Los ambiciosos objetivos de XML motivaron que W3C empezara a tratar de superar las posibilidades de CSS e impulsó **XSL** (Extensible Stylesheet Language) como un metalenguaje para expresar hojas de estilo. La idea consiste en utilizar una clase de documentos XML que definen una hoja de estilo y que exprese la forma como un contenido estructurado en un documento XML puede presentarse en distintos entornos. En otras palabras, una forma de hacer que el contenido XML pueda asociarse a un estilo a un diseño y a una paginación, que generalice lo que CSS consigue en la ventana de un navegador.

Objetivos

- Entender qué es una hoja de estilo XSLT y por qué es necesaria.
 - Entender como funcionan las hojas de estilo junto con el motor XSL.
 - Saber utilizar los templates para crear hojas de estilo sencillas.
 - Utilizar los elementos de la gramática xsl y saber crear hojas de estilo XSLT sencillos.
-

Contenidos.

1. ¿Qué es XSL?
2. ¿Cómo funcionan las hojas de estilo XSLT?
3. Plantillas
4. Elementos XSLT
5. Ejercicios

¿Qué es XSL?

El lenguaje de hoja de estilo extensible o ampliado, tal y como su nombre indica, es un lenguaje basado en XML que se usa para crear hojas de estilo. Un motor de hojas de estilo usa hojas de estilo para transformar documentos XML en otros tipos de documentos y para formatear la salida. En estas hojas de estilo se define el diseño del documento de salida y desde dónde se obtienen los datos en el documento de entrada. Es decir, "los datos que se utilizan están en un sitio del documento de entrada y deben tener una distribución determinada en el documento de salida". En terminología XSL, el documento de entrada se denomina **árbol de origen** y el documento de salida se denomina **árbol de resultado**.

Bajo XSL existen en realidad dos lenguajes completamente diferentes:

- Un lenguaje de transformación, denominado **XSLT**
- Un lenguaje utilizado para describir la visualización de los documentos XML, que se denomina **XSL Formatting Objects**.

Por su puesto, los dos lenguajes se pueden usar juntos, de manera que XSLT transforma los datos y XSL Formatting Objects se encarga de la visualización de los datos, de manera similar las hojas de estilo en cascada.

Observar que además del XML bien formado, XSLT también puede producir HTML, o incluso texto regular. De hecho, uno de los usos más promocionados de XSLT es la transformación de XML en documentos HTML para su visualización en un navegador.

¿Cómo funcionan las hojas de estilo XSLT?

Las hojas de estilo XSLT se construyen sobre estructuras denominadas **plantillas**. Una plantilla especifica qué es lo que hay que buscar en el árbol de origen y qué es lo que hay que colocar en el árbol de resultado.

XSLT está escrito en XML, lo que significa que existen elementos y atributos XSLT especiales que se utilizan para crear nuestras hojas de estilo. Un ejemplo sencillo:

```
<xsl:template match="first">
  ¡Se encontró la primera etiqueta!
</xsl:template>
```

Las plantillas (**templates**) están definidas usando el elemento XSLT **<xsl:template>**. En esta plantilla existen dos partes importantes: el atributo **match** y los **contenidos** de la plantilla. EL atributo match especifica un modelo en el árbol de origen; esta plantilla se aplicará para todos los nodos del árbol que coincida en el modelo del patrón. En este caso se aplica a todos los elementos denominados first.

El contenido de una plantilla puede ser, y de hecho suele ser, mucho más complejo que simplemente producir la salida de texto. Existen varios elementos XSLT que se pueden insertar en la plantilla para realizar las distintas acciones. Por ejemplo existe un elemento **<xsl:value-of>**, que puede tomar información del árbol de origen y añadirlo en el árbol de resultado. Los siguiente también funcionará como en nuestra plantilla anterior, pero, en lugar de producir la salida anterior agregará los contenidos de todos los elementos first.

```
<xsl:template match="first">
  <xsl:value of select="."/>
</xsl:template>
```

Si un documento XML contiene los elementos **<first>Andrea</first>** y **<first>John</first>** nuestra plantilla producirá las salidas 'Andrea' y 'John'.

Una hoja de estilo XSL se puede asociar con un documento XML usando el mismo tipo de instrucción de procesamiento de hoja de estilo que se utilizó para CSS. Por ejemplo:

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

Un ejemplo.

Aunque es pronto para entender todos los elementos de la hoja de estilo vamos a ver un ejemplo que nos dará una idea general de lo que puede hacer XSL. En puntos posteriores veremos que significa cada uno de los elementos que utilizaremos en el ejemplo.

Primero necesitamos un documento XML que posteriormente transformaremos:

```
<?xml version="1.0" encoding="UTF-8"?>
<order>
  <salesperson>John Doe</salesperson>
  <item>Production-Class Widget</item>
  <quantity>16</quantity>
  <date>
    <month>1</month>
```

```
<day>11</day>
<year>2000</year>
</date>
<customer>Sally Finkelstein</customer>
</order>
```

Puedes descargar el fichero [aquí](#).

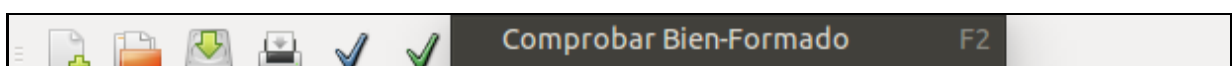
Ahora veamos la hoja de estilo:

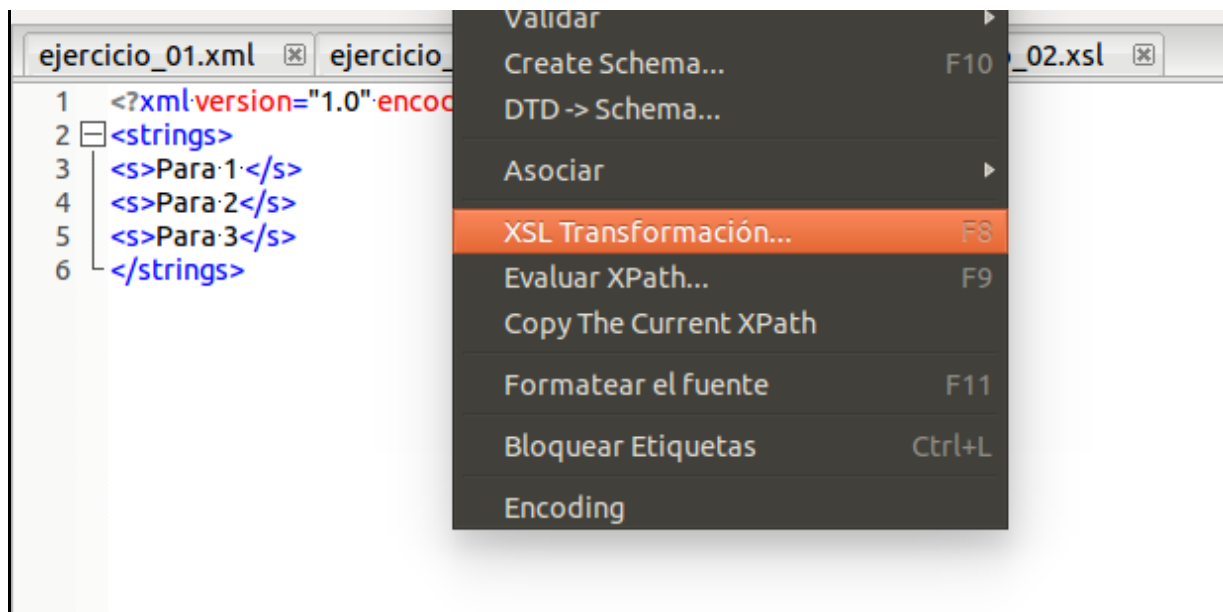
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <order>
      <date>
        <xsl:value-of select="/order/date/year"/>/
        <xsl:value-of select="/order/date/month"/>/
        <xsl:value-of select="/order/date/day"/>
      </date>
      <customer>Company A</customer>
      <item>
        <xsl:apply-templates select="/order/item"/>
        <quantity>
          <xsl:value-of select="/order/quantity"/>
        </quantity>
      </item>
    </order>
  </xsl:template>
  <xsl:template match="item">
    <part-number>
      <xsl:choose>
        <xsl:when test=". = 'Production-Class Widget'">E16-25A</xsl:when>
        <xsl:when test=". = 'Economy-Class Widget'">E16-25B</xsl:when>
        <xsl:otherwise>00</xsl:otherwise>
      </xsl:choose>
    </part-number>
    <description>
      <xsl:value-of select="."/>
    </description>
  </xsl:template>
</xsl:stylesheet>
```

Puedes descargar el fichero [aquí](#).

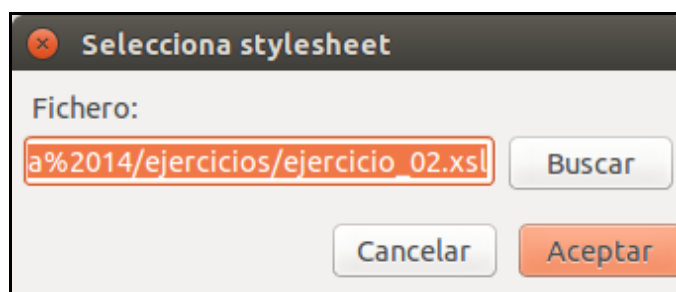
Utilizando XML Editor podemos aplicar al XML inicial la hoja de estilo anterior y observar el resultado. Para ello abrimos el ejemplo XML y el contenido de la hoja de estilo:

- En la parte superior, en la opción XML seleccionamos la opción XSL transformación:

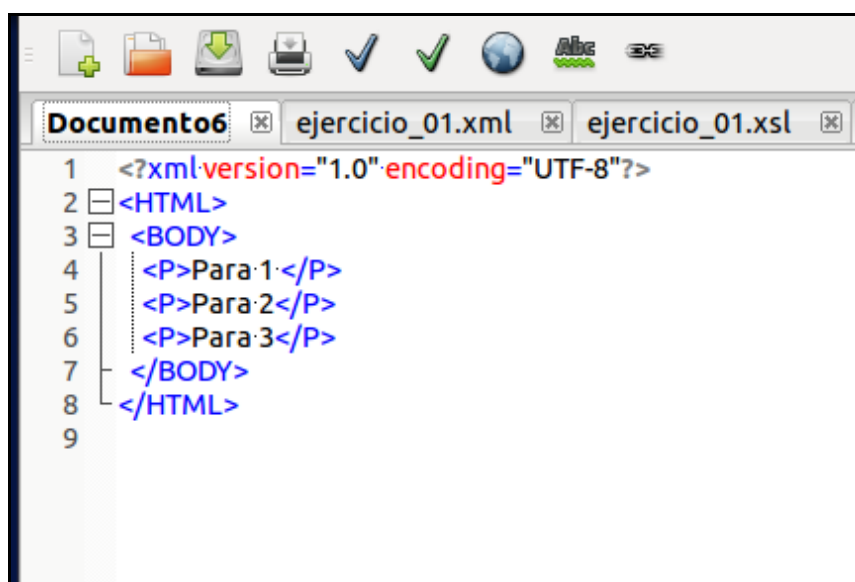




- Le indicamos el fichero xsl que determina el fichero de salida:



Podemos ver como hemos transformado el documento XML original en otro documento XML bien formado que nos puede ser útil por diversas razones como por ejemplo cumplir con un DTD para que el XML pueda ser procesado por una aplicación de la empresa o mostrado en un navegador aplicando un CSS concreto.



XSLT no es un lenguaje de programación imperativo (como los que normalmente utilizamos). Es un lenguaje de programación declarativo. Con XSLT no necesitamos especificar al ordenador cómo queremos que se hagan las cosas, simplemente indicamos qué resultado se pretende. Esto lo hacemos utilizando plantillas, que especifican las condiciones que se cumplen en los procesos y la salida que se produce. El modo en que se realiza la tarea depende del totalmente del procesador.

Veamos el siguiente ejemplo sencillo:

```
<?xml version="1.0" encoding="UTF-8"?>
<strings>
  <s>Para 1 </s>
  <s>Para 2</s>
  <s>Para 3</s>
</strings>
```

Queremos obtener un código HTML asociado a este XML equivalente al siguiente:

```
<HTML>
<BODY>
<P> Para 1 </P>
<P> Para 2 </P>
<P> Para 3 </P>
</BODY>
</HTML>
```

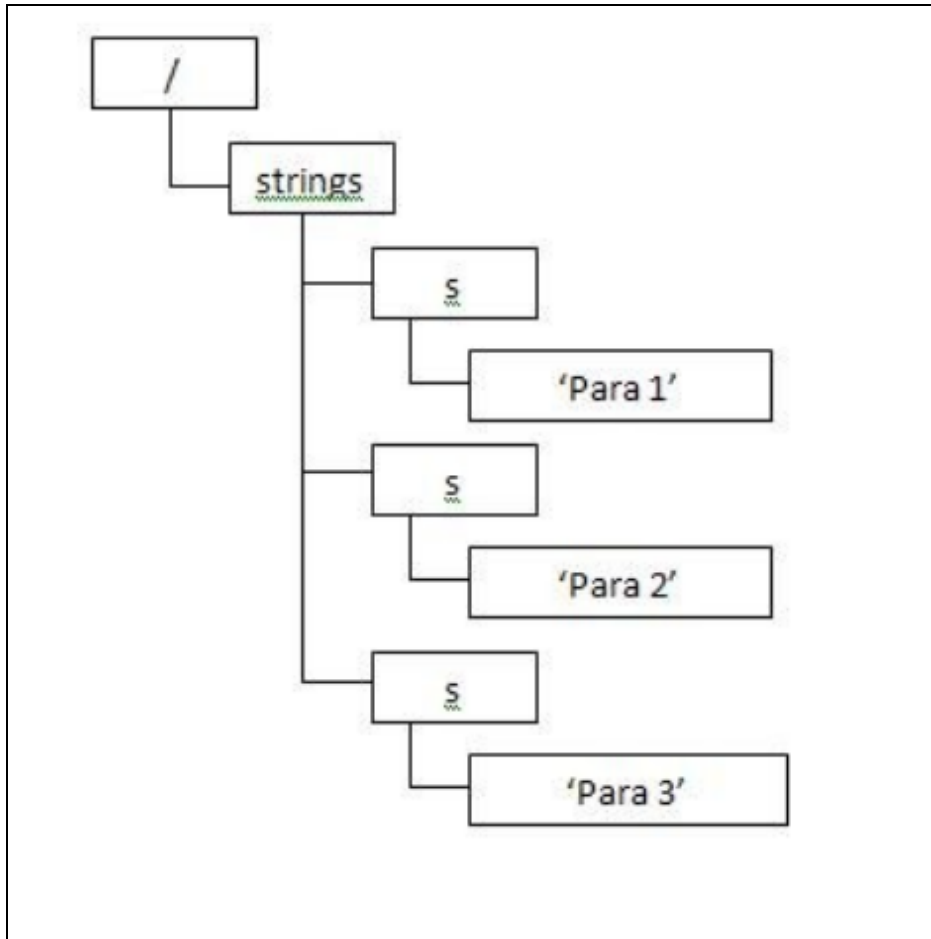
Para ello utilizamos el siguiente código XSL:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <BODY>
        <xsl:for-each select="/strings/s">
          <P>
            <xsl:value-of select="."/>
          </P>
        </xsl:for-each>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Podemos ver que la hoja de estilo se parece mucho a la salida, con algunos elementos XSLT combinados en el código para especificar dónde se coloca el contenido, estos son los elementos con el prefijo xsl:. El motor XSL hará lo siguiente:

- Buscará la raíz del documento en el árbol de origen (es decir, la raíz virtual de la jerarquía del documento).
- Comparará la raíz del documento con el contenido de la plantilla en nuestra hoja de estilo.
- Producirá la salida de los elementos HTML de la salida
- Procesará los elementos XSLT de la plantilla. En nuestro caso el <xsl:for-each...>

Este elemento funciona como una plantilla dentro de otra plantilla. Se utiliza para cada elemento <s> que sea hijo del elemento raíz <strings>. Por cada uno se genera un elemento <P> seguido del contenido del elemento <s> para ello se utiliza el elemento XSLT <xsl:value-of...> Como podemos apreciar el motor XSLT hace la mayor parte del trabajo por nosotros. Para el motor XSLT nuestro XML tendrá la siguiente apariencia de árbol:



Recordemos del tema anterior (XPATH) que los árboles formados a partir de un XML siempre tienen un nodo raíz virtual (/) del cuál cuelga el nodo raíz de nuestro documento.

Plantillas

Ya conocemos conceptualmente la idea propuesta por las plantillas. Realmente las podríamos considerar el alma de XSLT. Las hojas de estilo son simplemente una colección de estas plantillas, que se aplican a los documentos de entrada para crear el documento de salida.

Veamos de nuevo la hoja de estilo del punto anterior:

```

1  <?xml:version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3  <xsl:template match="/">
4  <HTML>
5  <BODY>
6  <xsl:for-each select="/strings/s">
7  <P>
8  <xsl:value-of select="."/>
9  </P>
10 </xsl:for-each>
11 </BODY>
12 </HTML>
13 </xsl:template>
14 </xsl:stylesheet>
15

```

Esta hoja de estilo tiene una única plantilla, **xsl:stylesheet**, pero las hojas de estilo pueden tener tantas plantillas como consideremos necesarias. Existen dos partes importantes:

- La sección árbol de origen a la que se aplica la plantilla.
- La salida que se insertará en el árbol de resultado.

La sección árbol de origen contra la que se está realizando la comparación se especifica en el atributo **match**. En este caso hemos especificado **match="/"**, que significa que la plantilla se compara con la raíz del documento.

Todo lo que está dentro de la plantilla es lo que será la salida del árbol de resultado. Todos los elementos XML normales o texto que aparecen del elemento **<xsl:template>** terminará en el documento de salida tal cual está. Todos los elementos con el prefijo '**xsl:**' indican al procesador que deberá hacerse algo con ellos.

EL orden de las operaciones de XSLT.

Si una hoja de estilo tiene más de una plantilla, entonces, ¿Cómo sabemos cual actuará primero? En realidad el procesador XSL inicia el proceso a partir de la comparación de la raíz del documento con la plantilla en la hoja de estilo que mejor se adapte; en la mayoría de los casos habrá una sola.

Si no se especifica una plantilla en nuestro documento, XSLT suministra una predeterminada que simplemente aplica cualquier otra plantilla que exista. La plantilla predeterminada se define de la siguiente manera:

```

<xsl:template match="*/">
  <xsl:apply-templates/>

```

</xsl:template>

Esto realiza la comprobación con todos los elementos del document, o la raíz del documento y llama a <xsl:apply-templates> para procesar a los hijos. La recomendación es tener siempre una plantilla con el atributo match="/". Esta plantilla será la que elegirá el procesador XSL al iniciar el proceso del árbol de entrada ya que se corresponde con la raíz del documento. A partir de esta controlaremos el proceso de creación del árbol de resultado.

¿Cómo afectan las plantillas al nodo de contexto?

Hay que tener en cuenta que en XSLT lo que la plantilla utilice en su atributo match se convierte en el nodo de contexto de esa plantilla. Esto significa que todas las expresiones XPATH dentro de la plantilla son relativas a ese nodo. Si tomamos el siguiente ejemplo:

```
<xsl:template match="/order/item">  
  <xsl:value-of select="part-number" />  
</xsl:template>
```

Esta plantilla se aplica a cada elemento **<ítem>** que sea hijo de **<order>** y que este sea hijo de la raíz (**/**) del documento. El elemento **<ítem>** se convierte en el nodo de contexto dentro de la plantilla.

Por lo tanto el elemento <xsl:value-of> está seleccionando solo aquellos elementos <part-number> que sean hijos del elemento <ítem> ya seleccionado para esta plantilla.

Elementos XSLT

Vamos a listar todos los elementos disponibles en XSLT. Simplemente presentaremos los más comunes que probablemente podamos encontrar.

Stylesheet

El elemento `<xsl:stylesheet>` es el elemento raíz de casi todas las hojas de estilo XSLT y se utiliza de la siguiente manera:

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

Este elemento siempre lo aplicamos igual mientras no cambie la versión o el espacio de nombres.

Template

Tal y como ya hemos visto el elemento `<xsl:template>` es el elemento utilizado para definir las plantillas. Este elemento es muy simple:

```
<xsl:template match="Xpath_expresion" name="template_name" priority="number"  
mode="mode_name">
```

El atributo `match` es para indicar el modelo XPath que se comparará con el árbol de origen. Lo que especificamos es el tipo de nodo que se selecciona, no la ruta de acceso a un nodo particular.

Por ejemplo la expresión XPath `"/name/first"` significa:

- ve a la raíz del documento.
- Luego al elemento hijo llamado 'name'
- y después al hijo de este llamado 'first'.

Pero si incluimos esta expresión en el elemento `template` como **`match="/name/first"`**, significa buscar la coincidencia de todo elemento `<first>` que sea hijo de un elemento `<name>` y que a su vez sea hijo del nodo raíz del documento.

El motor XSLT puede aplicar dos plantillas sobre un mismo nodo siempre elige la más específica. Por ejemplo si tenemos las siguientes plantillas definidas:

```
<xsl:template match="name">  
<xsl:template match="name[.='John']">
```

Un elemento `<name>` con un valor diferente a 'John' utilizará la primera plantilla. Mientras que un elemento `<name>` con valor 'John' utilizará la segunda por ser más específica.

En caso de poder aplicar dos plantillas se utiliza el atributo `priority` para obligar a elegir una sobre la otra.

También podemos crear plantillas con nombre utilizando el atributo `name`. Esto nos permitirá llamar a una plantilla desde cualquier parte del documento XSLT.

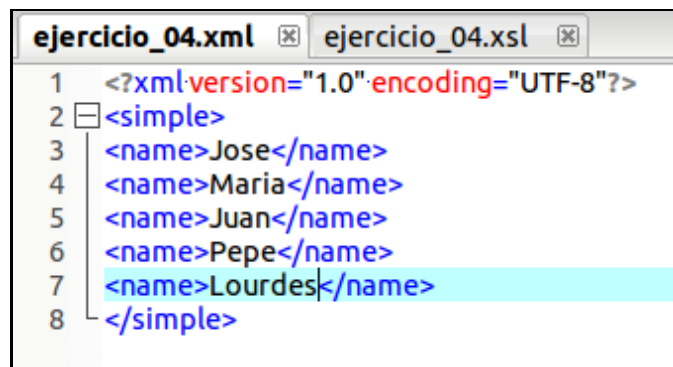
apply-templates

El elemento `<xsl:apply-templates>` se utiliza desde dentro de una plantilla para llamar a otras.

`<xsl:apply-templates select="expresión XPath">`

Si se especifica el atributo **select**, entonces se evaluará la expresión y el resultado se utilizará como nodo de contexto, que será utilizado por otras plantillas. Es opcional, si no es específica, se asume el nodo contexto vigente.

Para ver el funcionamiento de este elemento vamos a realizar una práctica. Para ello utilizaremos el siguiente XML:



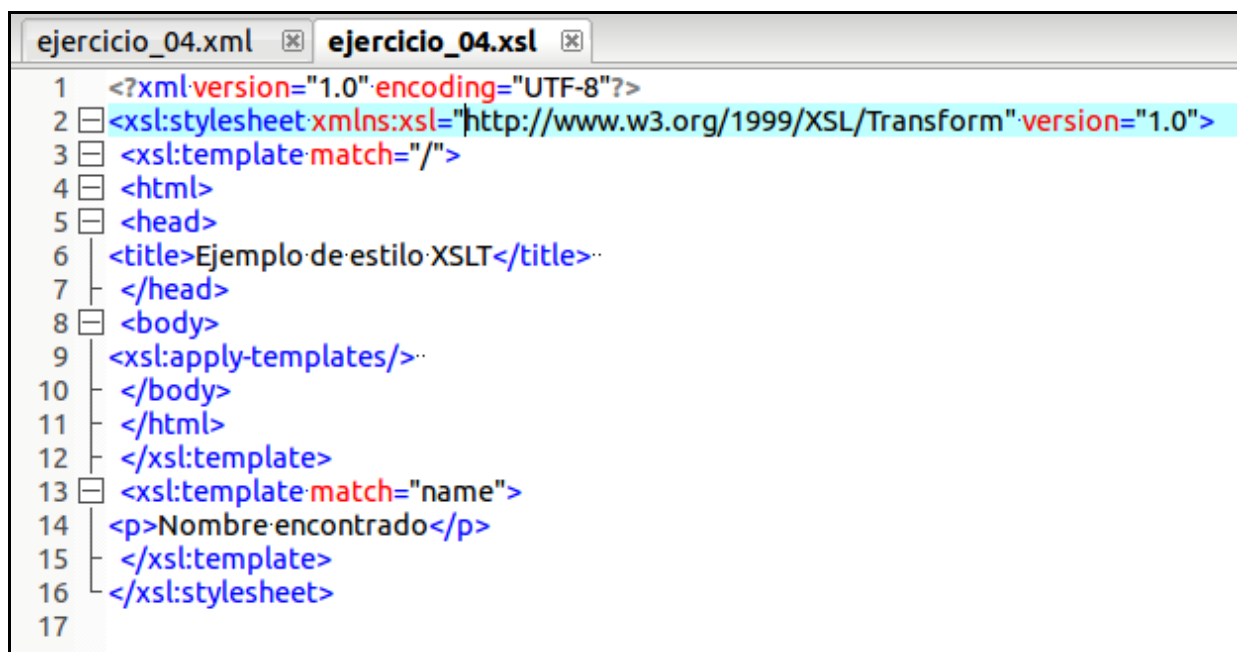
```

1 <?xml:version="1.0" encoding="UTF-8"?>
2 <simple>
3   <name>Jose</name>
4   <name>Maria</name>
5   <name>Juan</name>
6   <name>Pepe</name>
7   <name>Lourdes</name>
8 </simple>

```

A continuación creamos la hoja de estilo para este código XML.

Primero utilizamos una plantilla para crear los elementos principales HTML. Luego llamamos a `<xsl:apply-templates>` para que otra plantilla se ocupe de los elementos `<name>`. Puedes descargar el fichero [aquí](#):

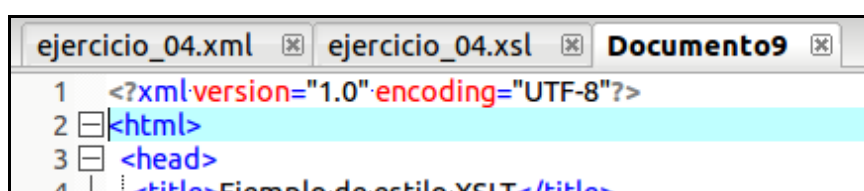


```

1 <?xml:version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3   <xsl:template match="/">
4     <html>
5       <head>
6         <title>Ejemplo de estilo XSLT</title>
7       </head>
8       <body>
9         <xsl:apply-templates/>
10      </body>
11    </html>
12  </xsl:template>
13  <xsl:template match="name">
14    <p>Nombre encontrado</p>
15  </xsl:template>
16 </xsl:stylesheet>
17

```

Ahora si aplicamos la plantilla a nuestro documento XML obtendremos el siguiente código HTML:



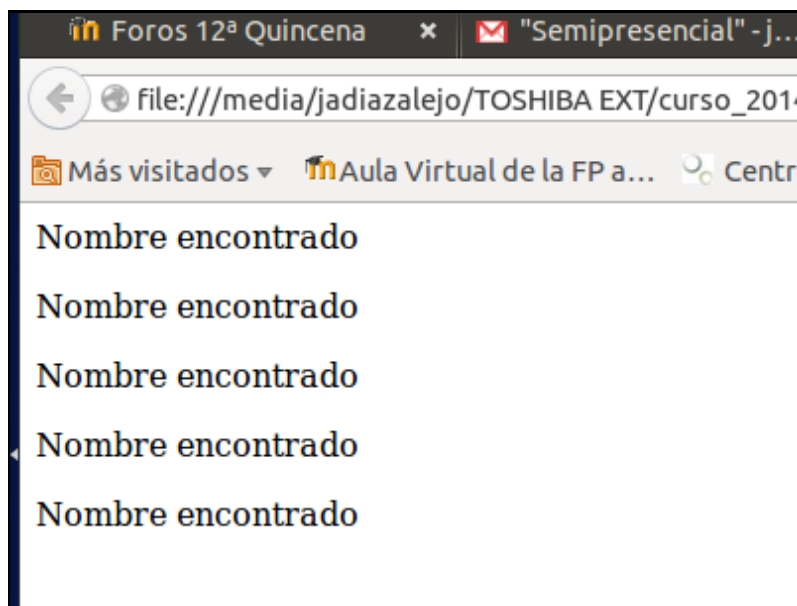
```

1 <?xml:version="1.0" encoding="UTF-8"?>
2 <html>
3   <head>
4     <title>Ejemplo de estilo XSLT</title>

```

```
5 | </head>
6 | <body>
7 | <p>Nombre encontrado</p>
8 | <p>Nombre encontrado</p>
9 | <p>Nombre encontrado</p>
10 | <p>Nombre encontrado</p>
11 | <p>Nombre encontrado</p>
12 | </body>
13 | </html>
14 |
```

Este resultado como se puede apreciar es una página WEB completa., que podemos ver en cualquier navegador.



Vamos a intentar explicar cómo ha sucedido todo:

- Después que el procesador XSL ha cargado el documento XML y la hoja de estilo XSLT, genera una instancia de la plantilla que coincide con la raíz del documento. Esta plantilla genera la salida de las etiquetas iniciales <HTML>, <HEAD>, <TITLE> y <BODY>.
- Luego se encuentra con el elemento <xsl:apply-templates>.
- A continuación el elemento sigue leyendo el árbol de entrada y verifica si puede ejecutar más plantillas. Esto ocurre cada vez que lee un elemento <name> ya que coincide con el match de una plantilla. Esta segunda plantilla simplemente genera un párrafo en HTML indicando que se ha encontrado un nuevo elemento <name> (<P>Nombre encontrado</P>).
- Una vez se ha leído el árbol completo el procesador XSL vuelve a la primera plantilla, al punto donde se había quedado y continua su ejecución. Insertando el cierre de las etiquetas HTML que quedaban.

value-of

El ejemplo anterior ha estado bien para comprender el funcionamiento de las plantillas. Pero es el momento de hacer algo más productivo con la información de nuestro árbol de entrada. Para ello vamos a utilizar el elemento <xsl:value-of>.

La sintaxis es la siguiente:

<xsl:value-of select="expresión XPath" disable-output-scoping="yes|no"/>

Este elemento busca el nodo de contexto para el valor especificado en la expresión XPath del atributo match y lo inserta en el árbol de resultado.

Por ejemplo:

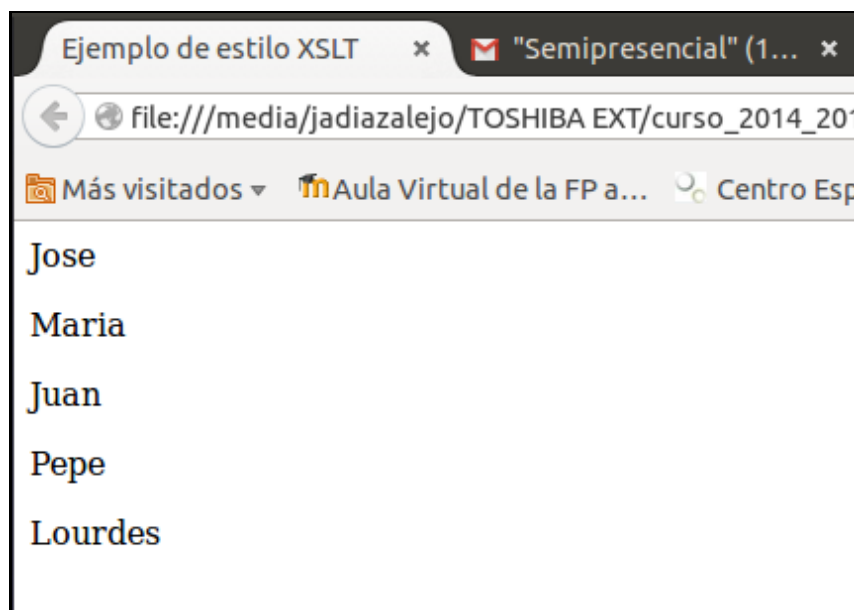
<xsl:value-of select="."/> Inserta PCDATA desde el nodo de contexto en la salida.

<xsl:value-of select="customer/@id"> Inserta el texto del atributo id del elemento <customer>.

El atributo disable-output-scoping es opcional su valor por defecto es no. Sirve para indicar al procesador que debe hacer con las secuencias de escape que podemos encontrar en los documentos XML. Como por ejemplo "&" o "<". Si el valor del atributo es no (por defecto) copiará la secuencia de escape en el árbol de salida. Si el valor es yes escribirá en la salida el carácter asociado a la secuencia de escape, "&" y "<" en nuestro ejemplo.

Ejercicio resuelto

Modifica el ejemplo anterior para obtener el siguiente resultado:



output

Anteriormente habíamos mencionado que XSLT puede generar salida XML, HTML o incluso formato de texto simple. El elemento **<xsl:output>** nos permite especificar el método que usaremos.

Si se incluye en nuestra hoja de estilo debe ser un hijo directo de <xsl:stylesheet>. En XSLT, los elementos que son hijos directos del elemento <xsl:stylesheet> se denominan **elementos de nivel superior**.

La siguiente es la sintaxis de <xsl:output>:

<xsl:output method="xml|html|text" version="version" encoding="encoding" standalone="yes|no" indent="yes|no"/>

El atributo **method** especifica el tipo de salida que se producirá. Si no se especifica el elemento `<xsl:output>` y el elemento raíz del árbol de salida es `<HTML>` o `<html>` entonces el método de salida predeterminado será `html`. En caso contrario será `xml`. Los atributos **version**, **encoding** y **standalone** se pueden utilizar cuando el método de salida es `xml`. Los valores especificados se utilizarán para crear la declaración XML del árbol de resultado.

El atributo **indent** tiene como valor por defecto `no`. Esto hace que la salida se escriba como una secuencia de elementos sin ningún tipo de orden, por lo tanto la lectura de este documento final no será nada clara para una persona. Sin embargo si utilizamos el valor **yes** el árbol de salida se escribirá en un fichero de forma legible para una persona. Cada elemento en una línea y los elementos hijos se irán indentando de forma que se vea a simple vista quién es hijo de quién.

El valor del parámetro debemos considerarlo desde el punto de vista de cómo vamos a utilizar el fichero de salida. Si lo va a tratar automáticamente una aplicación no necesitamos que esté indentado ya que el programa lo va a entender igual. Si es para leerlo una persona lo mejor es que esté indentado si no será difícil de comprender.

element

En ejemplos anteriores ya hemos visto como insertar elementos directamente en el árbol de resultado. Pero, ¿Qué ocurre si no sabemos de antemano el nombre de estos elementos? El elemento `<xsl:element>` nos permite crear elementos de manera dinámica:

```
<xsl:element name="element name" use-attribute-sets="attribute set names" />
```

El atributo **name** indica el nombre del element. Por tanto:

```
<xsl:element name="blah">My text</xsl:element>
```

Generará el siguiente resultado: `<blah>My text</blah>`. Pero esto no es muy dinámico (ya lo sabíamos hacer sin utilizar este nuevo elemento). Veamos ahora la siguiente plantilla:

```
<xsl:template match="name">
  <xsl:element name="{ . }">My text </xsl:element>
</xsl:template>
```

Esta plantilla aplicada al XML del ejercicio que vamos implementando generará en el árbol de salida los siguientes elementos:

```
<Juan>My Text </Juan>, <David> My Text </David> ...
```

attribute

Similarmente a los que sucede con `<xsl:element>`, `<xsl:attribute>` se puede usar para añadir dinámicamente atributos a un elemento.

```
<name><xsl:attribute name="id">213</xsl:attribute>Carlota</name>
```

La línea anterior producirá el siguiente resultado: `"<name id=213>Carlota</name>"`

```
<name><xsl:attribute name={ . }>213</xsl:attribute>Carlota</name>
```

La línea anterior hará lo mismo solo que el nombre del atributo será el texto del nodo de contexto.

Hay que tener en cuenta que el elemento `<xsl:attribute>` debe ir siempre detrás de la declaración del elemento al que queremos añadir el atributo.

text

El elemento `<xsl:text>` inserta algo de texto (PCDATA) en el árbol de resultado:

```
<xsl:text disable-output-escaping="yes|no">
```

En general este elemento no es necesario ya que en las plantillas podemos escribir directamente el texto que queremos que se escriba en el árbol de resultado.

Pero existen dos razones para poder, la primera es que mantiene todo el espaciado. Y la segunda que podemos habilitar/deshabilitar la salida de caracteres de escape tal y como vimos para el elemento `<xsl:value-of>`.

La siguiente línea:

```
<xsl:value-of="Uno"/> <xsl:value-of="Dos">
```

Esperaríamos que generara una salida como: "Uno Dos". Pero si lo probamos en una plantilla veremos que la salida que produce es "UnoDos". Ya que el procesador XSL elimina los espacios en blanco.

En este caso será útil `<xsl:text>`:

```
<xsl:value-of="Uno"/><xsl:text> </xsl:text> <xsl:value-of="Dos">
```

Dentro del elemento `<xsl:text>` solo podemos insertar PCDATA, no puede contener otros elementos XSLT.

If y choose (procesamiento condicional)

Todos los lenguajes de programación nos deben permitir realizar elecciones, ya que en caso contrario su utilidad sería muy escasa. XSLT no es una excepción a esa regla, y nos brinda un par de elementos para realizar elecciones: `<xsl:if>` y `<xsl:choose>`:

```
<xsl:if test="expresión booleana">
```

```
<xsl:choose>
```

```
  <xsl:when test="expresión booleana">
```

```
  <xsl:when test="expresión booleana">
```

```
  <xsl:otherwise>
```

```
</xsl:choose>
```

Para ambos elementos, la expresión booleana es simplemente una expresión XPath que se convierte a un valor booleano.

```
<xsl:if test="name"> Se encuentra el elemento name. </xsl:if>
```

Para el ejemplo anterior si existe el elemento `<name>` que sea hijo del elemento de contexto se insertará el

texto "Se encuentra el elemento name". Si no existe el elemento, no se insertará nada.

```
<xsl:choose>
  <xsl:when test="number[. > 2000]>Un número grande </xsl:when>
  <xsl:when test="number[. > 1000]>Un número mediano </xsl:when>
  <xsl:otherwise>Un numero pequeño </xsl:otherwise>
</xsl:choose>
```

En un elemento `<xsl:choose>` tan pronto como una de las expresiones de comprobación se evalúa como cierta, el control del proceso sale del elemento `<xsl:choose>`. Con esto, se quiere dejar claro, que aunque dos expresiones se puedan evaluar como ciertas solo se ejecutará una, la primera.

Veamos un ejemplo. Supongamos que tenemos una base de datos que es capaz de devolvernos la información en formato XML como la que sigue:

```
<?xml version="1.0" encoding="UTF-8"?>
<empleado FullSecurity="1">
  <nombre>Juan Garcia</nombre>
  <departamento>Ventas</departamento>
  <telefono>(96) 555 55 55<extension>5555</extension></telefono>
  <salario>24000</salario>
  <area>3</area>
</empleado>
```

- El atributo **FullSecurity** está determinado por el identificador del usuario que pide la información.
- "1" indica miembro de recursos humanos que tiene acceso a la información de salarios y un "0" indica que no tiene acceso a esta información. La empresa está subdividida en diversas ubicaciones y area nos indica la ubicación del empleado.
- Queremos generar un documento HTML a partir del documento XML anterior donde se muestre la información del empleado. Para ello vamos a crear tres plantillas. La primera generará la mayor parte del contenido HTML. Luego utilizaremos dos más para los casos especiales del teléfono y del área.

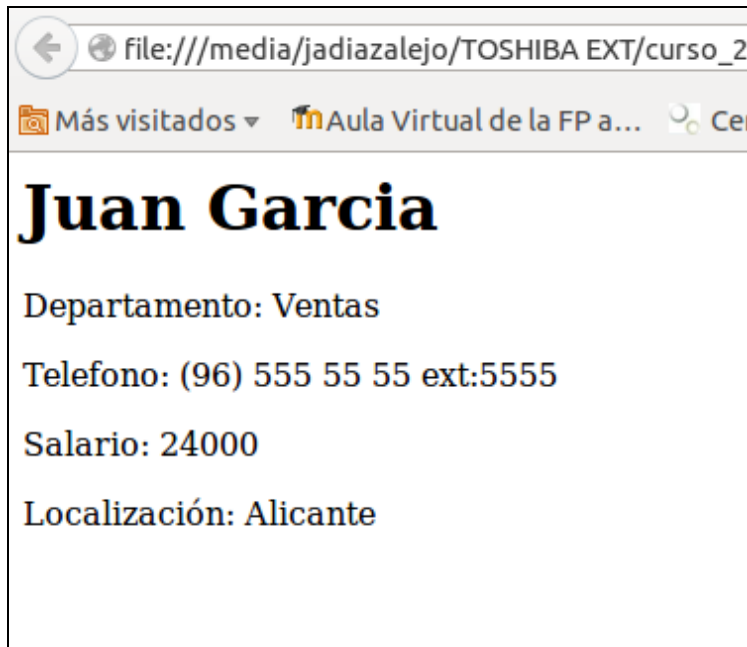
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:value-of select="/empleado/nombre"/>
        </TITLE>
      </HEAD>
      <BODY>
        <h1>
          <xsl:value-of select="/empleado/nombre"/>
        </h1>
        <p>Departamento: <xsl:value-of select="empleado/departamento"/></p>
        <p>
          <xsl:apply-templates select="empleado/telefono"/>
        </p>
        <xsl:if test="number(/empleado/@FullSecurity)">
          <p>Salario: <xsl:value-of select="empleado/salario"/></p>
        </xsl:if>
      </BODY>
    </HTML>
  </template>
  <xsl:template match="empleado/telefono">
    <p><xsl:value-of select="."/></p>
  </template>
  <xsl:template match="empleado/area">
    <p><xsl:value-of select="."/></p>
  </template>
</xsl:stylesheet>
```

```

        <p>Localización: <xsl:apply-templates select="empleado/area"/></p>
    </BODY>
</HTML>
</xsl:template>
<xsl:template match="telefono">
    Telefono: <xsl:value-of select="text()"/> ext:<xsl:value-of select="extension"/>
</xsl:template>
<xsl:template match="area">
    <xsl:choose>
        <xsl:when test="number(.) = 1">Valencia</xsl:when>
        <xsl:when test="number(.) = 2">Castellón</xsl:when>
        <xsl:when test="number(.) = 3">Alicante</xsl:when>
        <xsl:otherwise>Localización desconocida</xsl:otherwise>
    </xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

EL resultado de aplicar esta hoja de estilo al XML inicial sería el siguiente:



Como podemos ver en nuestro caso podemos ver el salario. Probar a cambiar el valor del parámetro FullSecurity y ver que el Salario aparece o desaparece en función de ese valor. Igual que si cambiamos el valor del elemento <area> debe cambiar la localización.

for-each.

En muchos casos es necesario realizar un procesamiento específico para varios nodos del árbol de origen. Por ejemplo cuando vimos el elemento <xsl:aply-templates> anteriormente procesamos varios nodos <name>.

Hasta el momento lo hemos solucionado creando una nueva plantilla para ese procesamiento. Pero con el elemento <xsl:for-each> existe un modo alternativo. El contenido de <xsl:for-each> forma una plantilla dentro de otra plantilla.

```

<xsl:for-each select="name">
    Es un nombre de element
</xsl:for-each>

```

Se genera una instancia de esta plantilla para cada elemento <name> que sea hijo del nodo de contexto. En este caso lo único que hace la plantilla es producir una salida de texto. Retomemos el XML de un ejemplo anterior:

```

<?xml version="1.0" ?>
<simple>
  <name>Juan</name>
  <name>David</name>
  <name>Andrea</name>
  <name>Isabel</name>
  <name>Claudia</name>
  <name>Carla</name>
  <name>Susana</name>
  <name>Marcos</name>
  <name>Carolina</name>
  <name>Angela</name>
</simple>

```

Podemos utilizar el elemento <xsl:for-each> para generar una página XML con todos los nombres:

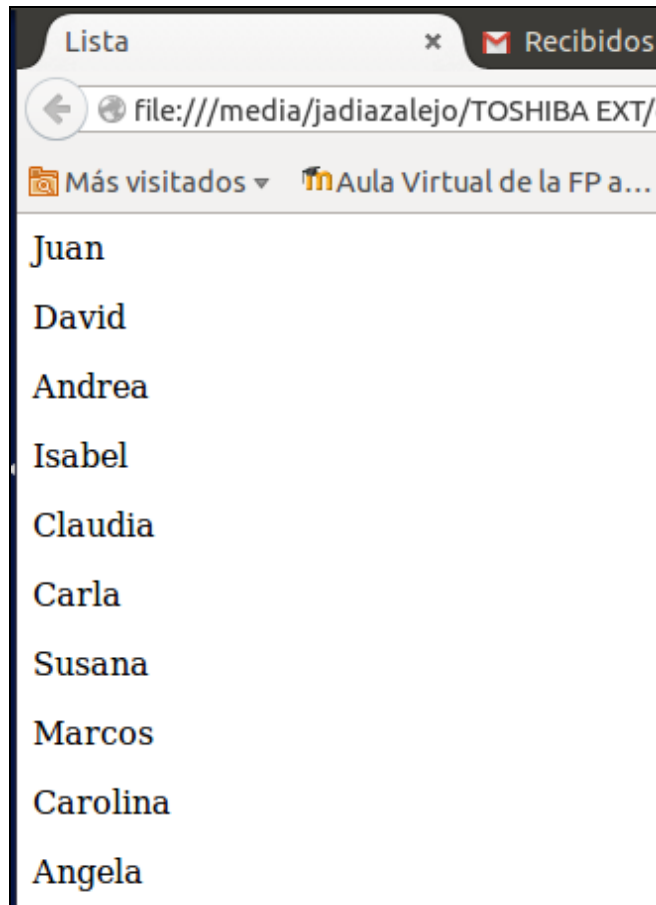


```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3  <xsl:template match="/">
4  <HTML>
5  <HEAD>
6  <TITLE>Lista</TITLE>
7  </HEAD>
8  <BODY>
9  <xsl:for-each select="/simple/name">
10 <P><xsl:value-of select="."/></P>
11 </xsl:for-each>
12 </BODY>
13 </HTML>
14 </xsl:template>
15 </xsl:stylesheet>

```

El resultado de aplicar este XSLT es el siguiente:



Que es el mismo que conseguimos con otro XSLT. Por tanto y como en todo lenguaje de programación en XSLT tampoco hay una forma única de hacer las cosas y cada uno puede crear diferentes XSLT que generen un mismo resultado.

copy-of

En muchos casos el árbol de resultado será similar al árbol de entrada. Quizá existan incluso secciones largas que sean exactamente iguales. El elemento `<xsl:copy-of>` nos permite tomar secciones del árbol de origen y copiarlas al árbol de resultado. Es mucho más fácil que crear todos los elementos/atributos manualmente.

La sintaxis:

```
<xsl:copy-of select="expresión XPath"/>
```

Para ver cómo funciona vamos a retomar el XML de empleado que vimos para el procesamiento condicional. Pero ahora en vez de generar código HTML imaginemos que queremos generar otro documento XML que en función del atributo de seguridad tenga o no tenga el elemento `<salario>`.

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="number(/empleado/@FullSecurity)">
      <xsl:copy-of select="/" />
    </xsl:when>
  </xsl:choose>
</xsl:template>
</xsl:stylesheet>
```

```

</xsl:when>
<xsl:otherwise>
<xsl:element name="empleado">
  <xsl:attribute name="FullSecurity">
    <xsl:value-of select="/empleado/@FullSecurity"/>
  </xsl:attribute>
  <xsl:for-each select="empleado/*[not(self::salario)]">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</xsl:element>
</xsl:otherwise>
</xsl:choose>
</xsl:template>
</xsl:stylesheet>

```

Como podemos ver la plantilla anterior se compone principalmente de dos partes. La primera:

```

<xsl:when test="number(/empleado/@FullSecurity)">
  <xsl:copy-of select=""/>
</xsl:when>

```

Donde cuando el nivel de seguridad del usuario es el indicado copiamos el XML del árbol de entrada tal cual en el árbol de resultado.

La segunda:

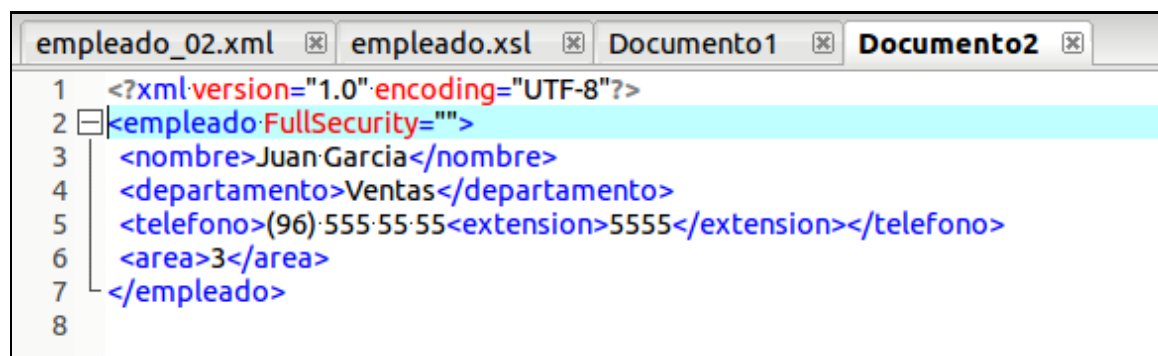
```

<xsl:otherwise>
<xsl:element name="empleado">
  <xsl:attribute name="FullSecurity">
    <xsl:value-of select="/empleado/@FullSecurity"/>
  </xsl:attribute>
  <xsl:for-each select="empleado/*[not(self::salario)]">
    <xsl:copy-of select="."/>
  </xsl:for-each>
</xsl:element>
</xsl:otherwise>

```

Donde se reconstruye de nuevo el árbol pero sin el elemento <salario>. Para ello creamos un elemento empleado, le agregamos el parámetro FullSecurity y a continuación copiamos todos los hijos de <empleado> excepto el hijo <salario>.

Si tiene un número de seguridad:



Para probarlo podemos borrar el número de seguridad y transformar de nuevo el fichero, obtendremos un fichero xml sin el salario.

copy

El elemento **<xsl:copy>** a diferencia del anterior simplemente copia el nodo de contexto. Los hijos y atributos del nodo de contexto no se copian al árbol de resultado.

Por ejemplo si tenemos el siguiente XML:

Y utilizamos la siguiente plantilla:

```
<xsl:template match="name">
  <xsl:copy/>
</xsl:template>
```

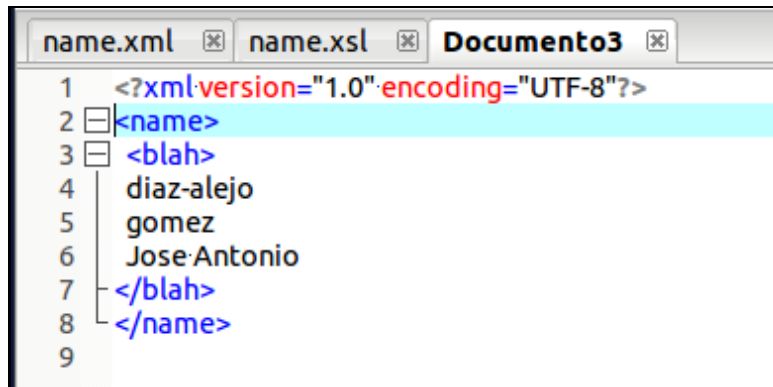
La salida sera:

```
<name/>
```

Pero si utilizamos la siguiente plantilla:

```
<xsl:template match="name">
  <xsl:copy>
    <blah><xsl:value-of select="."></blah>
  </xsl:copy>
</xsl:template>
```

El resultado será:



(*) recordar que cuando utilizamos `<xsl:value-of select=".">` se obtiene el contenido del nodo de contexto más el contenido de todos los nodos descendientes.

Sort

En XSLT la clasificación se realiza añadiendo uno o más elementos `<xsl:sort>` a un elemento `<xsl:apply-templates>` o a un elemento `<xsl:for-each>`:

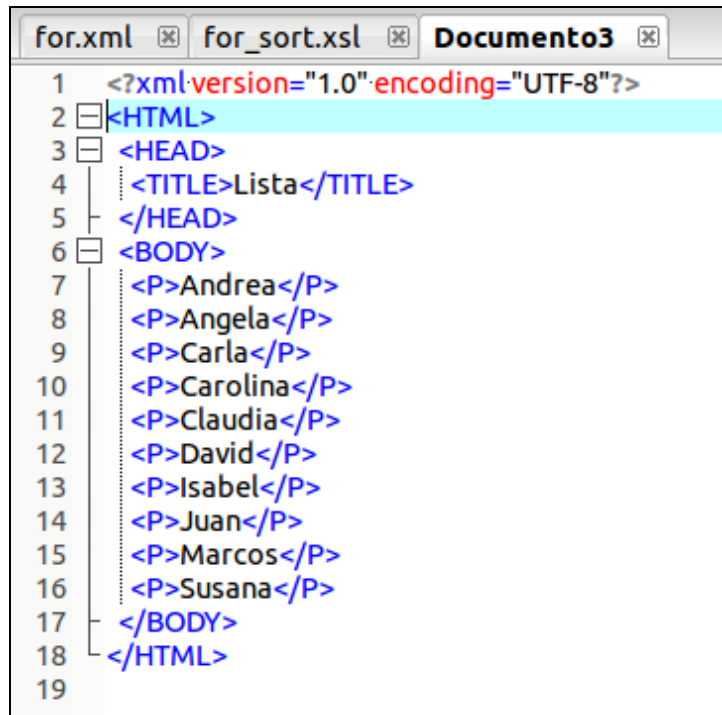
`<xsl:sort select="expresión XPath" lang="lang-code" data-type="text|number" order="ascending|descending" case-order="upper-first|lower-first"/>`

El atributo `select` elige el elemento/atributo por el cual se quiere la clasificación. Si se añade más de un hijo `<xsl:sort>` entonces se clasifica primero por el primer hijo `sort` luego por el segundo y así sucesivamente.

Para probar retomamos nuestro ejemplo para el elemento `<xsl:for-each>` y lo retocamos:

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>Lista</TITLE>
      </HEAD>
      <BODY>
        <xsl:for-each select="/simple/name">
          <xsl:sort select="."/>
          <P><xsl:value-of select="."/></P>
        </xsl:for-each>
      </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

Ahora la salida de nuestro XSLT será la siguiente:



Donde podemos apreciar como el procesador ha ordenado los elementos `<name>` antes de procesarlos.

Variables, constantes y plantillas con nombre

Todos los que hemos trabajado con lenguajes de programación estamos familiarizados con las variables y las constantes. Por ejemplo en Java podemos definir las así:

```
int intAge = 30;
final int csngPI = 3.14;
```

En el código, nosotros siempre podemos cambiar el valor de la variable `intAge` pero nunca el de la constante ya que como su propio nombre indica, su valor no varía durante toda la ejecución del programa.

En XSLT **solo podemos crear constantes** simples con el elemento `<xsl:variable>`.

Por ejemplo:

```
<xsl:variable name="csngPI">3.14</xsl:variable>
```

Ahora podemos utilizar esta constante en cualquier lugar (normalmente dentro de la plantilla donde ha sido definida) empleando un signo `$` seguido de la variable:

```
<math pi="($csngPI)"/>
<xsl:value-of select="$csngPI"/>
```

Pero `<xsl:variable>` también puede contener marcas XML e incluso elementos XSLT, por ejemplo:

```
<xsl:variable name="space">
  <xsl:text> </xsl:text>
</xsl:variable>

<xsl:variable name="name">
  <name>
```



```

    <xsl:value-of select="/name/first">
    <xsl:value-of select="$space">
    <xsl:value-of select="/name/last">
  </name>
</xsl:variable>

<!-- esto obtiene el valor de la variable $name, incluyendo cualquier marca XML
<xsl:copy-of select="$name"/>

```

Solo hay que tener en cuenta que una constante no puede hacer referencia a sí misma ni tampoco se permiten referencias circulares. Un motor XSL debe detectar estas situaciones para evitar los bloqueos pero podemos encontrar alguno que no lo detecte y nuestra aplicación se meta en un bucle sin fin.

Como sintaxis alternativa <xsl:variable> puede tener el atributo select para dar valor a la constante desde una expresión XPath.

Por ejemplo:

```
<xsl:variable name="name" select="/persona/nombre"/>
```

- Hay que hacer hincapié en que con lo que estamos trabajando son constantes y por lo tanto el valor de las mismas no puede alterarse una vez han sido declaradas.
- Otra cosa a tener en cuenta es el ámbito de las variables y constantes. Es decir dónde puedo declararlas y dónde puedo utilizarlas.

El elemento <xsl:variable> puede declararse bien como hijo de el elemento <xsl:stylesheet> con lo cual podrá ser utilizado en todas las plantillas de esa hoja de estilo. Bien como hijo de <xsl:template> con lo cual su ámbito será solo local dentro de esta plantilla.

```

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method = "text" />
  <xsl:variable name="age">30</xsl:variable>
  <xsl:template match="/">
    <xsl:variable name = "name">Fred</xsl:variable>
    <xsl:value-of select="concat($name, ' tiene ', $age, ' años')"/>
  </xsl:template>
</xsl:stylesheet>

```

En el ejemplo anterior hemos definido una constante global "age" y una contante local "name".

Por último vamos a ver las plantillas con nombre. Para ello utilizaremos el atributo name del elemento <xsl:template>. Y el elemento <xsl:call-template> para llamar a una plantilla con nombre.

Lo mejor es ver un ejemplo:

```

<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:for-each select="name">
      <xsl:call-template name="bold"/>
    </xsl:for-each>
  </xsl:template>
  <xsl:template name="bold">

```

```
<B><xsl:value-of select="."/></B>  
</xsl:template>  
</xsl:stylesheet>
```

Ejercicios

Partiendo del siguiente XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<horario>
  <dia>
    <numdia>1</numdia>
    <tarea prioridad="media">
      <hora-ini>12</hora-ini>
      <hora-fin>14</hora-fin>
      <nombre>Tutorías</nombre>
    </tarea>
  </dia>
  <dia>
    <numdia>2</numdia>
    <tarea prioridad="alta">
      <hora-ini>12</hora-ini>
      <hora-fin>14</hora-fin>
      <nombre>Autómatas</nombre>
    </tarea>
  </dia>
  <dia>
    <numdia>4</numdia>
    <tarea prioridad="alta">
      <hora-ini>9</hora-ini>
      <hora-fin>11</hora-fin>
      <nombre>Procesadores de lenguajes</nombre>
    </tarea>
    <tarea prioridad="alta">
      <hora-ini>16</hora-ini>
      <hora-fin>17</hora-fin>
      <nombre>EDI</nombre>
    </tarea>
  </dia>
  <dia>
    <numdia>3</numdia>
    <tarea prioridad="alta">
      <hora-ini>9</hora-ini>
      <hora-fin>11</hora-fin>
      <nombre>Procesadores de lenguajes</nombre>
    </tarea>
  </dia>
  <dia>
    <numdia>5</numdia>
    <tarea prioridad="baja"><hora-ini>17</hora-ini>
    <hora-fin>18</hora-fin>
```

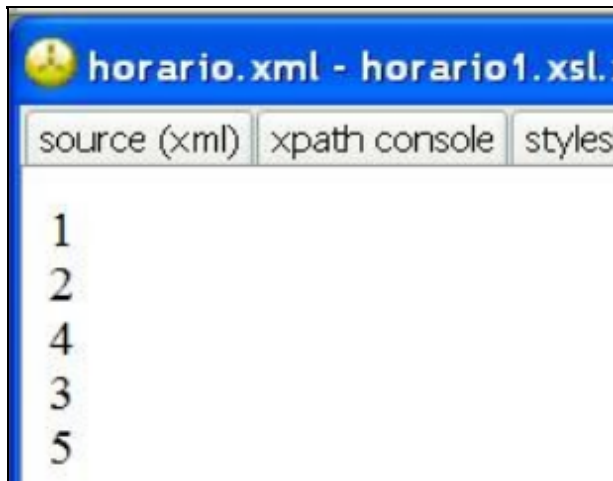
24XML – Lenguaje de presentación de datos

Tema 6. Transformación de documentos: XSLT

```
<nombre>Ver la tele</nombre>  
</tarea>  
</dia>  
</horario>
```

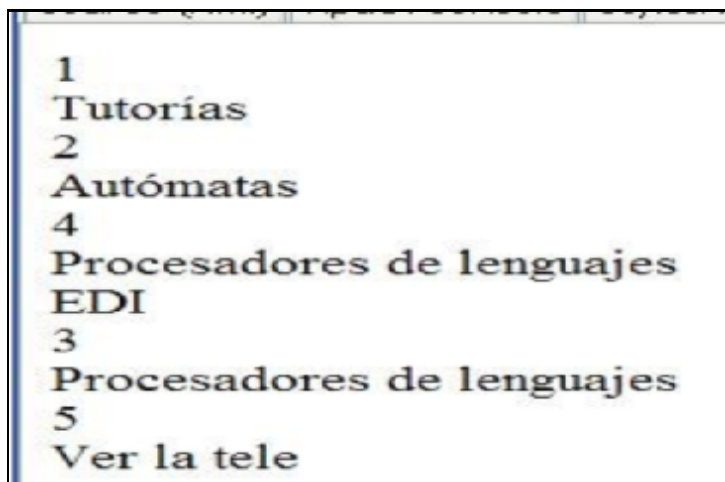
Ejercicio 1

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:



Ejercicio 2

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:



Ejercicio 3

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:

Tutorías -- media
Autómatas -- alta
Procesadores de lenguajes -- alta
EDI -- alta
Procesadores de lenguajes -- alta
Ver la tele -- baja

Ejercicio 4

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:

Día: 4
Tarea: Procesadores de lenguajes
Tarea: EDI
Día: 3
Tarea: Procesadores de lenguajes
Día: 5
Tarea: Ver la tele

Ejercicio 5

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:

Día 1
<ul style="list-style-type: none">• Tutorías- Prioridad:media De 12 a 14
Día 2
<ul style="list-style-type: none">• Autómatas- Prioridad:alta De 12 a 14
Día 4
<ul style="list-style-type: none">• Procesadores de lenguajes- Prioridad:alta De 9 a 11• EDI- Prioridad:alta De 16 a 17
Día 3
<ul style="list-style-type: none">• Procesadores de lenguajes- Prioridad:alta De 9 a 11
Día 5
<ul style="list-style-type: none">• Ver la tele- Prioridad:baja De 17 a 18 <hr/>

Ejercicio 6

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:

- 1 día - Hijos: 2
 - 1 numdia - Hijos: 0
 - 2 tarea - Hijos: 3
 - 1 hora-ini - Hijos: 0
 - 2 hora-fin - Hijos: 0
 - 3 nombre - Hijos: 0
 - 2 día - Hijos: 2
 - 1 numdia - Hijos: 0
 - 2 tarea - Hijos: 3
 - 1 hora-ini - Hijos: 0
 - 2 hora-fin - Hijos: 0
 - 3 nombre - Hijos: 0
 - 3 día - Hijos: 3
 - 1 numdia - Hijos: 0
 - 2 tarea - Hijos: 3
 - 1 hora-ini - Hijos: 0
 - 2 hora-fin - Hijos: 0
 - 3 nombre - Hijos: 0
 - 3 tarea - Hijos: 3
 - 1 hora-ini - Hijos: 0
- De 17 a 18

Ejercicio 7

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página HTML:

Día: 4
Tarea: EDI

Ejercicio 8

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página XML:

```
<?xml version="1.0" encoding="windows-1252"?>
<horario>
  <Lunes>
    < tarea prioridad="media">
      < hora-ini>12</ hora-ini>
      < hora-fin>14</ hora-fin>
      < nombre>Tutorías</ nombre>
    </ tarea>
  </ Lunes>
  < Martes>
    < tarea prioridad="alta">
      < hora-ini>12</ hora-ini>
      < hora-fin>14</ hora-fin>
      < nombre>Autómatas</ nombre>
    </ tarea>
  </ Martes>
</ horario>
```

Tarea obligatoria

A partir del XML inicial realiza la transformación necesaria para obtener la siguiente página XML:

```
<?xml version="1.0" encoding=
<horario>
<Lunes>
<num_tareas>1</num_tareas>
</Lunes>
<Martes>
<num_tareas>1</num_tareas>
</Martes>
<Miércoles>
<num_tareas>1</num_tareas>
</Miércoles>
<Jueves>
<num_tareas>2</num_tareas>
</Jueves>
<Viernes>
<num_tareas>1</num_tareas>
</Viernes>
</horario>
```