

Tema 5. “Conversión y adaptación de documentos XML”

Actualizado al 24/01/2023

—

5.1.	Técnicas de transformación de documentos XML	2
5.1.1.	XSLT	2
5.1.2.	XSL-FO.....	2
5.1.3.	XPath	2
5.2.	XPath	3
5.2.1.	Términos básicos.....	3
5.2.2.	Expresiones	7
5.3.	XSLT	9
5.3.1.	Introducción	9
5.3.2.	Vinculación de documentos	9
5.3.3.	Comenzando con el XSLT.....	10
5.3.4.	Incluir elementos literales.....	11
5.3.5.	Incluir valores del XML	12
5.3.6.	Iterar múltiples valores	13

5.1. Técnicas de transformación de documentos XML

5.1.1. XSLT

Permite transformar un documento XML a:

- Otro documento XML diferente.
- Documento HTML.
- Documento de texto plano.

Se verá en profundidad en el apartado 3 de este tema.

5.1.2. XSL-FO

FO proviene de “Formatting Objects” (objetos para dar formato). Con esta técnica es posible generar documentos con formato (como PDFs) partiendo de un documento XML que incorpora una información adicional sobre dicho formato.

Puede verse un ejemplo en este link:

https://es.wikipedia.org/wiki/XSL_Formatting_Objects

La generación del documento con formato la realiza un programa, tal como lo es “fop”:

<https://xmlgraphics.apache.org/fop/>

5.1.3. XPath

Es un lenguaje de consulta no procedimental (algo así como SQL), que permite construir expresiones que recorren y procesan un documento XML. Un ejemplo de expresión XPath podría ser:

<code>/nodoraiz/persona/nombre</code>

Que mostraría el contenido de todos los elementos “nombre” que haya dentro de elementos “persona” que haya dentro de la raíz “nodoraiz”.

Se verá más en detalle a continuación.

5.2. XPath

5.2.1. Términos básicos

Para poder practicar los términos básicos y las expresiones sobre XPath, te recomiendo que te instales un sistema gestor de base de datos basado en XML, como es el caso de BaseX.

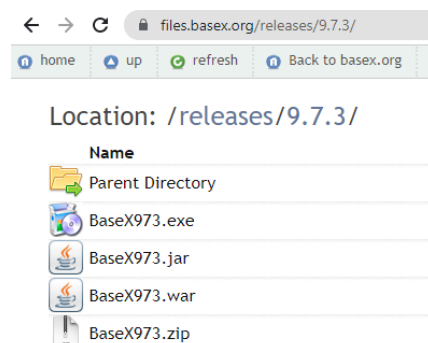
5.2.1.1. Instalación de BaseX

La web oficial de BaseX la encontramos aquí: <https://basex.org/>

La versión 10 requiere Java. Recomiendo instalar la última versión 9 que no lo requiere, que se puede encontrar en esta URL:

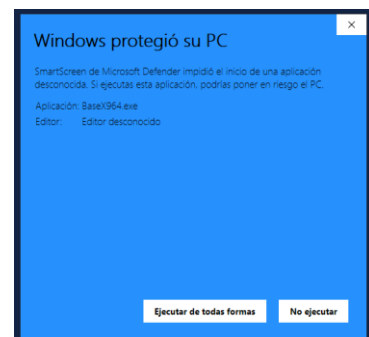
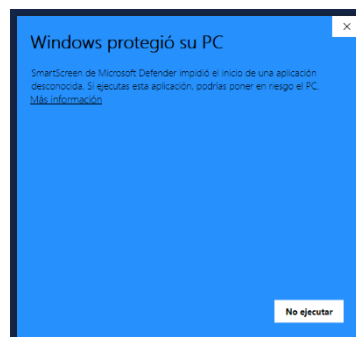
<https://files.basex.org/releases/9.7.3/>

Dentro de esta URL, debemos buscar el instalador de nuestro sistema operativo. En este ejemplo se usará el de Windows “BaseX973.exe”.

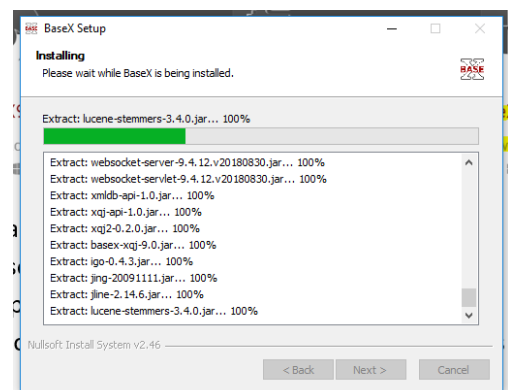


Una vez descargado, ejecutaremos el programa de instalación.

Puede que nos salte esta protección de sistema operativo, en tal caso aceptamos la instalación haciendo clic primero en el vínculo “Más información” y después en el botón “Ejecutar de todas formas”.



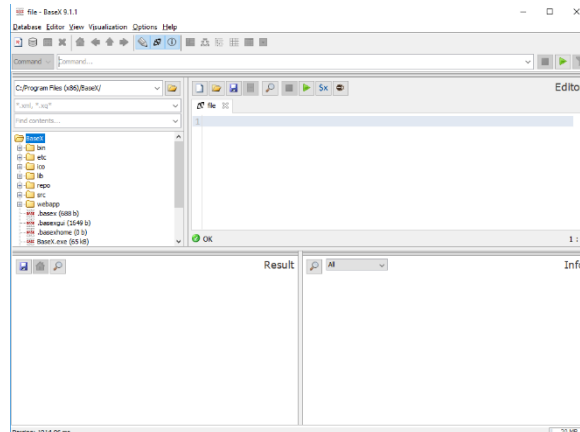
Después seguiremos los pasos que él nos sugiriera, dejando las opciones por defecto. En realidad, todas las opciones de configuración admitidas deben abordarse tras la instalación.



5.2.1.2. Primera ejecución

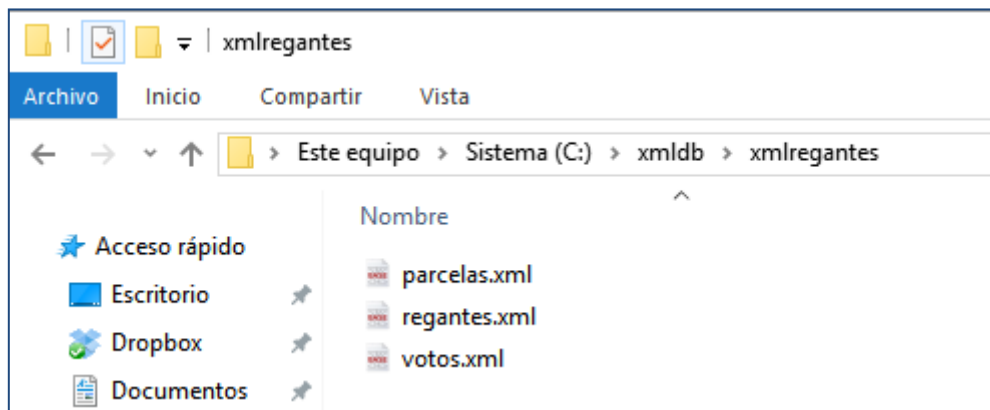
Una vez instalado, la interfaz mostrará cuatro secciones diferenciadas:

1. La parte superior izquierda muestra un explorador de archivos.
2. La parte superior derecha muestra un editor de texto.
3. La parte inferior izquierda muestra resultados de consultas.
4. La parte inferior derecha muestra información adicional sobre las consultas realizadas.



5.2.1.3. De XML a XMLDB

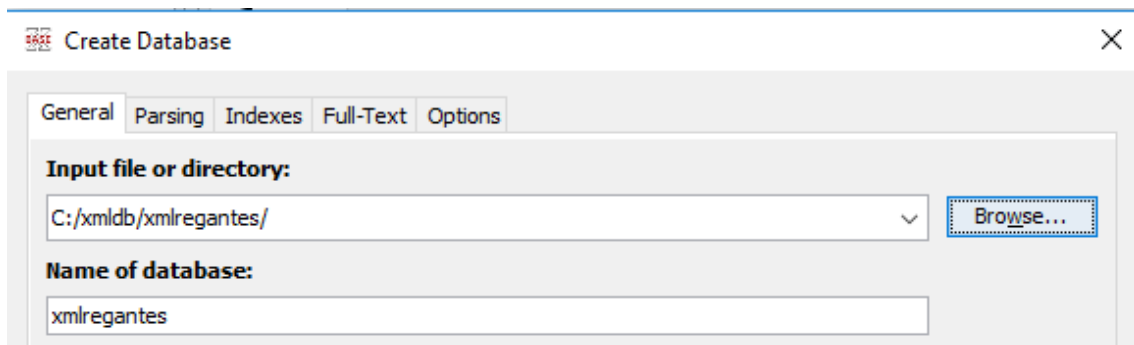
Queremos decir aquí que vamos a convertir archivos XML en una base de datos XML. Para ello, cogemos el archivo de recurso "xmlregantes.rar" y descomprimos su contenido en una carpeta. En nuestro ejemplo, hemos usado la carpeta "c:\xmlldb\", pero puede ser otra:



Los tres archivos XML contienen información sobre una comunidad de regantes, concretamente incorporan datos de 3 regantes, 5 parcelas y una tabla que indica el derecho a votos que tienen los regantes según las hectáreas totales que suman sus parcelas.

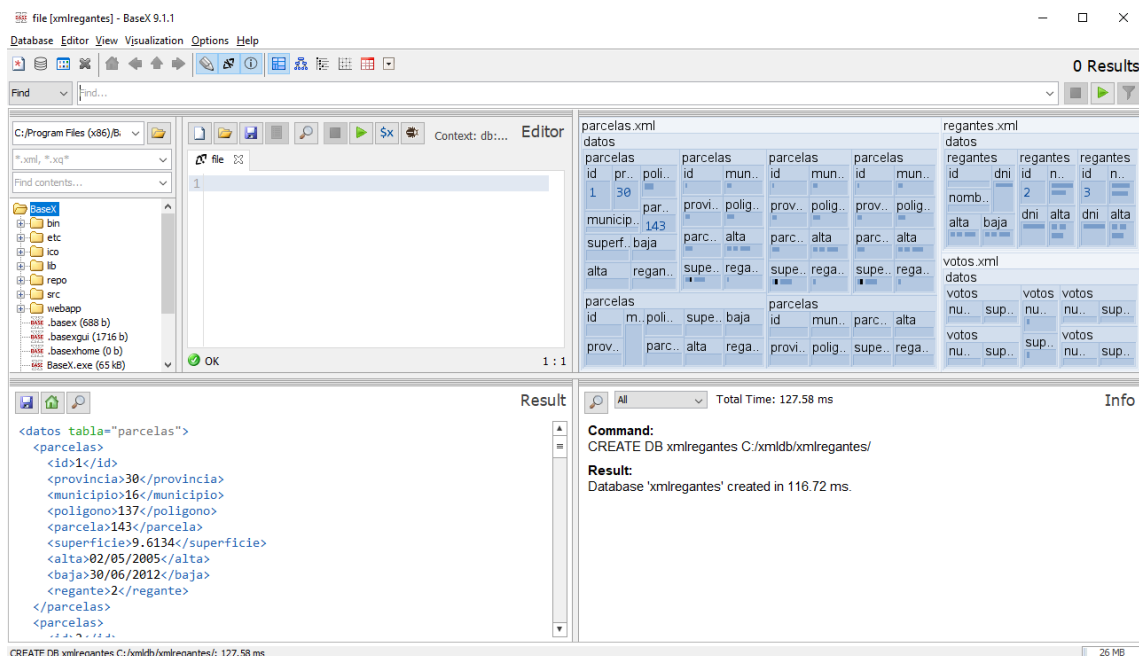
Sin entrar en profundidad en la semántica de esta “base de datos XML”, vamos a usar BaseX y XPath para acceder a sus registros. Para lograrlo, llega el momento de interpretar esta carpeta como una base de datos XML (XMLDB). ¿Qué le falta? No mucho. Lo más importante es una información adicional que ayudará al sistema gestor a acceder más rápidamente a la información: los índices.

Con el entorno BaseX abierto, vamos hasta Database > New. A continuación, buscamos la carpeta que hemos habilitado ("c:\xmlldb\xmlregantes") con el botón "Browse..." y confirmamos su apertura:



El nombre de la base de datos cambiará, para llamarse igual que la carpeta. Es buena praxis dejarlo así, si bien hay que asegurarse de haber nombrado a la carpeta en consonancia con su contenido. El resto de parámetros podemos dejarlo por defecto.

Ahora el entorno muestra una sección más: el mapa, en la parte superior derecha. Este mapa pretende resumir gráficamente la información contenida en los diferentes ficheros:



El editor de consultas permanece en la parte superior central. Si queremos desactivar el mapa para maximizar el área de la consulta, podemos hacerlo en el menú "Visualization > Map".

¿Dónde está la información que complementa a los ficheros XML? Por ejemplo, los índices de los que hablábamos. BaseX los ha ubicado en la carpeta "data", organizados por carpetas con los nombres de las diferentes bases de datos ya registradas. La carpeta "data" por defecto está dentro de su carpeta de instalación. Su contenido puede verse en esta captura:

xmlregantes

Archivo

Inicio

Compartir

Vista










←

→

⌵

⬆

> Este equipo > Disco local (C:) > Archivos de programa (x86) > BaseX > data > xmlregantes

Nombre	Fecha de modifica...	Tipo	Tamaño
 atv.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 atvl.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 atvr.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 inf.basex	21/01/2019 20:14	BaseX Configurati...	4 KB
 tbl.basex	21/01/2019 20:14	BaseX Configurati...	4 KB
 tbli.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 txt.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 txtl.basex	21/01/2019 20:14	BaseX Configurati...	1 KB
 txtr.basex	21/01/2019 20:14	BaseX Configurati...	1 KB

Con estos pasos hechos, ya estamos en disposición de acceder a la información de los ficheros XML mediante el sistema gestor.

5.2.1.4. Primeras consultas con BaseX

Podemos ver un primer ejemplo de sentencia aquí:

```
1 doc("c:/xmlldb/xmlregantes/votos.xml")
```

El resultado de la consulta será el contenido completo del archivo XML.


Para ejecutarla podemos usar tres métodos:

- El botón de la fecha verde, habilitado justo encima del editor.
- La combinación Ctrl+INTRO.
- Activar la opción del menú "Options > Realtime Execution" (las consultas se ejecutan conforme se van escribiendo).

Podemos realizar una consulta más elaborada con este ejemplo:

```
1 doc("c:/xmlldb/xmlregantes/regantes.xml")/datos/regantes/nombre
```

Que lo que hace es acceder, dentro del fichero XML, a su elemento "datos" (es el elemento raíz) y, dentro de éste, a sus elementos "regantes" y, dentro de éstos, a sus elementos "nombre". De modo que al final tenemos una lista con los nombres de los regantes, tal como ésta:


Result

```
<nombre>Mariano Iglesias</nombre>
<nombre>Sebastian Alonso</nombre>
<nombre>Amancio Gasset</nombre>
```

Hemos realizado un acceso a un fichero, haciendo referencia a su nombre completo. Sin embargo, es más interesante hacer referencia a un identificador de base de datos, buscando así en todo su contenido (en todos sus archivos XML). La forma de conseguirlo es mediante la instrucción `db:open()`, la cual requiere como único argumento el nombre de una base de datos registrada. En nuestro caso, la base de datos se llama "xmlregantes", así que podemos acceder a toda su información mediante la consulta:

```
1 db:open("xmlregantes")
```

En el panel de resultados vemos el contenido de todos los ficheros XML, uno a continuación de otro. En lo sucesivo usaremos este modo de acceso, pues nos permite ver la base de datos "como un todo" y, después, mediante varios mecanismos, accederemos a los datos que nos interesan, sin preocuparnos de en qué ficheros concretos residen esos datos.

5.2.2. Expresiones

5.2.2.1. Ruta de localización (path expressions)

Se usan para navegar a través de los elementos de una base de datos XML. Si queremos acceder a los elementos "datos" de "xmlregantes", podemos hacerlo mediante la sentencia:

```
1 db:open("xmlregantes")/datos
```

A efectos prácticos, esta sentencia provoca el mismo resultado que el anterior ejemplo, pues "datos" es el elemento raíz de todos los ficheros XML de esta base de datos. OJO: no siempre ha de ser así, ocurre ahora porque se adoptó esa consideración de diseño, pero cada archivo XML podría contener un elemento raíz diferente.

Si se desea acceder a elementos dentro de elementos "datos", podemos seguir escribiendo barras y los nombres de los "sub-elementos" a los que queremos acceder, como hicimos antes:

```
1 db:open("xmlregantes")/datos/regantes/nombre
```

Ahora lo hemos expresando con la función "db:open()", pero el resultado es el mismo: acceder a los nombres de todos los regantes. Eso sí, hemos ganado en que la función `db:open()` será la misma para cualquier tabla a consultar, pues representa a toda la base de datos y no sólo a los regantes. Por ejemplo, también podemos ver todas las parcelas (y sus sub-elementos) así:

```
1 db:open("xmlregantes")/datos/parcelas
```

Y no ha hecho falta cambiar la referencia a la base de datos, sólo la ruta a consultar.

5.2.2.2. Predicados (predicates)

Un predicado es una condición que deben cumplir los datos a mostrar. Hasta ahora, con las rutas podíamos imponer ciertas limitaciones, pero todas ellas referidas a su "ubicación" dentro del árbol. Esto es insuficiente, pues no nos permite diferenciar cosas que se encuentren en una misma "rama", particularmente a un mismo nivel.

Por ejemplo, si deseamos mostrar toda la información relativa al regante nº 2 (y sólo el 2), podemos escribir algo así:

```
1 db:open("xmlregantes")/datos/regantes[id=2]
```

Esto es, indicamos entre corchetes que queremos mostrar nodos de elementos "regantes" que deben estar dentro de elementos "datos" pero que deben contener elementos "id" cuyo valor ha de ser "2".

Las expresiones de comparación que admiten estos predicados pueden incorporar otros operadores, como por ejemplo los de "mayor que", "menor o igual que", etc.

```
1 db:open("xmlregantes")/datos/parcelas[superficie>8]
```

Este ejemplo muestra todos los datos de las parcelas cuya superficie es mayor de 8 hectáreas. En SQL hubiera sido algo así:

```
SELECT * FROM PARCELAS WHERE SUPERFICIE>8;
```

Si bien, al menos con lo que hemos visto hasta ahora, la forma de presentar los resultados difiere bastante, pues en XPath se asemejan más a "subfragmentos" de XML.

Algunos operadores de comparación soportados en los predicados:

- > Estrictamente mayor que
- < Estrictamente menor que
- >= Mayor o igual que
- <= Menor o igual que
- = Igual a
- != Diferente de

Los predicados también soportan estos operadores lógicos:

- and Operación "y" lógica, a semejanza de SQL.
- or Operación "o" lógica, a semejanza de SQL.
- not Negación, a semejanza de SQL.

Por supuesto, también es posible el uso de paréntesis para alterar la precedencia de los operadores anteriores.

En algunos casos se desea evaluar simplemente la existencia o no de un sub-elemento. Por ejemplo, si queremos sacar todos los regantes que poseen fecha de baja (sea la que sea):

```
1 db:open("xmlregantes")/datos/regantes[baja]
```

Lo que equivaldría a algo así en SQL:

```
SELECT * FROM REGANTES WHERE baja is not null;
```


5.2.2.3. Acceso a atributos

En todos estos ejemplos, también podemos hacer referencia a atributos (y sus valores) en lugar de a sub-elementos. La diferencia es que tendremos que anteponer la arroba (@) al identificador del atributo. En nuestra base de datos sólo el elemento "datos" tiene un atributo "tabla". Según lo dicho, otra forma de mostrar a todos los regantes podría haber sido:

```
1 db:open("xmlregantes")/datos[@tabla="regantes"]
```

A efectos prácticos, la única diferencia de hacerlo así es que, en este último ejemplo, aparecen también los elementos con la etiqueta "datos" (puesto que en realidad es una consulta de elementos "datos" la cual, por defecto, incluye también a los sub-elementos de ellos).

5.3. XSLT

5.3.1. Introducción

Vamos a ver un rápido ejemplo donde se apliquen los conceptos vistos en los contenidos de la plataforma EAD.

Recordemos que XSLT es un sub-lenguaje de XML (es XML en realidad) y que su cometido es transformar un documento XML en otro archivo de texto, bien sea:

- Un XML con una estructura diferente.
- Una página HTML.
- Un fichero de texto plano.



Para nuestro ejemplo, usaremos el fichero XML usado en la tutoría pasada, referido al detalle de una factura de una compañía telefónica. Vamos a “dar formato HTML” al contenido del archivo XML, tomando valores de elementos y atributos, para tabular el detalle de llamadas telefónicas que encontramos dentro de él.

5.3.2. Vinculación de documentos

El primer paso es tener claro que necesitamos:

- El archivo XML con los datos a “formatear”. Es el llamado “llamadas.xml”.
- El archivo XSLT que indicará las reglas de transformación. Lo creamos nosotros, por ejemplo, con Notepad++. Partiendo de un documento en blanco, lo llamaremos “llamadas.xsl” y lo ubicaremos en la misma ruta.
- Vincular ambos archivos. Ahora lo vemos.

Nombre

 llamadas.xml
 llamadas.xsl

La vinculación se realiza igual que en CSS (después de todo, XSLT es a XML lo que CSS es a HTML, esto es, un lenguaje de marcado que especifica el aspecto final de su contenido).

Quiero decir que hay que ir al archivo XML e indicar una línea que recordará al “<link...”:

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE factura>
3 <?xml-stylesheet type="text/xsl" href="llamadas.xsl"?>
4 <factura emision="2016-01-31" cliente="Albaladejo Segado, Mario">
```

La línea 3 es la que hemos insertado nueva. En el atributo “href” se indicará la URI al archivo XSLT, que actualmente está vacío. Pero bueno, al menos ya está enlazado para cuando sirva.

Por otra parte, hemos dotado a la raíz de un par de atributos (*emision* y *cliente*), para tener más juego a base de incorporar esos valores al documento formateado.

5.3.3. Comenzando con el XSLT

Vayamos dando forma ahora a “llamadas.xsl”. Lo primero, incluir el típico encabezado XML:

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

Y a continuación declarar un par de cosas a la vez:

- La etiqueta raíz, que será “stylesheet”
- El espacio de nombres “xsl”, que será el que usemos en todas las etiquetas (incluyendo la ya mencionada raíz “stylesheet”.

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

Siempre usaremos esos valores para versión y para la URI del espacio de nombres, pues son los estándares reconocidos. Más información en:

https://www.w3schools.com/xml/xsl_transformation.asp

Las reglas de transformación se agrupan en lo que se conoce como “plantillas” (templates). Es necesaria esta agrupación, así que debemos definir al menos una plantilla. Seguimos por ahí:

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

La plantilla se define como un elemento “template” dentro de la raíz. El atributo “match” es importante, pues es una forma de indicar a qué parte del XML (a transformar) se aplica la plantilla (es decir, qué parte del documento original XML se va a tener en cuenta para llevar a cabo la transformación definida en esa plantilla).

Esa “parte” se indicará mediante una expresión XPath, vistas anteriormente.

En nuestro caso, sólo usaremos una plantilla, que se referirá a todo el documento y, por eso, la expresión correcta que se refiere a todo el documento es “/”.

Más información en https://www.w3schools.com/xml/xsl_templates.asp

Ya tenemos el “esqueleto mínimo” necesario para poder comenzar.

5.3.4. Incluir elementos literales

Si deseamos generar un archivo HTML de salida, lo haremos “manualmente”. Esto es, generaremos las etiquetas propias del HTML “a mano”. Es lo primero que vamos a hacer.

Para ello, simplemente incluimos dichas etiquetas dentro de la plantilla. El resultado podría ser este:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <h2>Detalle de llamadas</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Id</th>
            <th>Tipo</th>
            <th>Fecha y hora</th>
            <th>Destino</th>
            <th>Tiempo</th>
          </tr>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Aunque no responde a ningún estándar HTML específico, el código insertado en el elemento “template” sería suficiente para mostrar una tabla únicamente con el encabezado:

Detalle de llamadas				
Id	Tipo	Fecha y hora	Destino	Tiempo

Por tanto, para incluir elementos literales basta con indicarlos tal cual. Así ha ocurrido con todas las etiquetas y valores HTML que hemos incorporado hasta el momento.

Como puede intuirse, sólo las etiquetas del espacio de nombres “xsl” son interpretadas durante la transformación. Eso permite utilizar (sin alias) cualquier etiqueta de HTML, sin miedo a ambigüedad (sin miedo a que sea interpretada como comando de transformación).

5.3.5. Incluir valores del XML

Es de lo que se trata. Para ello usaremos la etiqueta “xsl:value-of”. Debemos hacerlo en conjunción con su atributo más importante: “select”, al que debemos dar como valor la expresión XPath que accede al valor.

Imaginemos que queremos poner en el título <h2> el nombre del cliente y la fecha de la factura (los dos atributos nuevos que nos hemos “inventado” en esta nueva versión del XML de ejemplo). Antes que nada, recordar que las expresiones XPath para llegar hasta esos valores son:

```
/factura/@emision  
/factura/@cliente
```

Bien, apliquemos la teoría a la práctica. El código del archivo “llamadas.xml”, en la referencia al encabezado <h2>, tras agregar los valores variables quedaría:

```
1 <?xml version="1.0" encoding="UTF-8"?>  
2 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
3   <xsl:template match="/">  
4     <html>  
5       <body>  
6         <h2>Detalle de llamadas  
7           facturadas el <xsl:value-of select="/factura/@emision" />  
8           al cliente <xsl:value-of select="/factura/@cliente" />  
9         </h2>  
10        <table border="1">
```

Y tras estos cambios, abierto en Internet Explorer o en Edge, se mostraría así:



NOTA: El navegador Chrome no aplica transformaciones XSLT. Aduce falta de seguridad. Safari para Mac es otro navegador que puede hacer esta tarea.

✖ Unsafe attempt to load URL file:///C:/Users/jlcruz-cifp/Desktop/llamadas.xml:3 op/llamadas/llamadas.xml from frame with URL file:///C:/Users/jlcruz-cifp/Desktop/llamadas/llamadas.xml. 'file:' URLs are treated as unique security origins.

El anterior ejemplo va bien cuando la expresión XPath devuelve un valor y sólo uno. Debemos saber que:

- Si no devuelve valor alguno, simplemente no aparecerá nada en la salida, sin dar error.
- Si devuelve más de un valor, sólo el primero (en orden de aparición) se mostrará.
- Si la expresión XPath es errónea, es como si no existiera valor alguno. Tampoco se produce ningún error que impida que el documento se siga transformando.

5.3.6. Iterar múltiples valores

Acabamos de decir que, en caso de resultados con múltiples valores, se toma el primero y se descartan el resto.

Obviamente, este comportamiento puede cambiarse, pero para ello debemos usar una etiqueta diferente, una que nos da la posibilidad de generar un bucle para cada valor dentro del conjunto de valores resultantes.

Es lo que necesitamos para poder completar nuestra tabla, puesto que debemos adaptar la cantidad de filas de esta al número de llamadas concretas que haya en el XML.

El elemento `<xsl:for-each>` es el encargado de habilitar estos bucles. Al igual que “value-of”, posee el atributo “select” en el que se indicará la expresión XPath que puede devolver una cantidad variable de valores.

Sin embargo, a diferencia de “value-of”, el contenido interno al elemento “for-each” se re-interpretará repetidamente, como un bucle, para cada valor devuelto. En cada iteración del bucle, el contenido producido es volcado a la salida.

Manos a la obra, que esto se ve mejor con un ejemplo. Imaginemos que queremos producir estrictamente el código HTML para el esqueleto de la tabla vacía, pero contando con una fila (`<tr>`) para cada “llamada”. Nuestro XSLT quedaría así:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="/">
4      <html>
5        <body>
6          <h2>Detalle de llamadas
7            facturadas el <xsl:value-of select="/factura/@emision" />
8            al cliente <xsl:value-of select="/factura/@cliente" />
9          </h2>
10         <table border="1">
11           <tr bgcolor="#9acd32">
12             <th>Id</th>
13             <th>Tipo</th>
14             <th>Fecha y hora</th>
15             <th>Destino</th>
16             <th>Tiempo</th>
17           </tr>
18           <xsl:for-each select="/factura/llamadas/llamada">
19             <tr>
20               <td>1</td>
21               <td>2</td>
22               <td>3</td>
23               <td>4</td>
24               <td>5</td>
25             </tr>
26           </xsl:for-each>
27         </table>
28       </body>
29     </html>
30   </xsl:template>
31 </xsl:stylesheet>
```

Y la interpretación de Internet Explorer para su contenido se muestra ahora así:

Detalle de llamadas facturadas el 2016-01-31 al cliente Albaladejo Segado, Mario

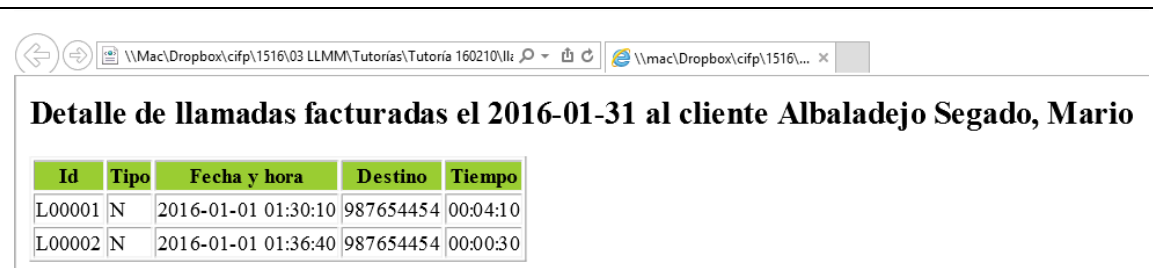
Id	Tipo	Fecha y hora	Destino	Tiempo
1	2	3	4	5
1	2	3	4	5

Vemos las dos filas de las dos llamadas, aunque con los valores literales del “1” al “5”.

Si completamos ahora el código del XSLT con los debidos elementos “value-of”:

```
<xsl:for-each select="/factura/llamadas/llamada">
  <tr>
    <td><xsl:value-of select="@id" /></td>
    <td><xsl:value-of select="@tipo" /></td>
    <td><xsl:value-of select="fechahora/@inicio" /></td>
    <td><xsl:value-of select="destino/@numero" /></td>
    <td><xsl:value-of select="tarificacion/@duracion" /></td>
  </tr>
</xsl:for-each>
```

El resultado es mucho más definitivo:



Id	Tipo	Fecha y hora	Destino	Tiempo
L00001	N	2016-01-01 01:30:10	987654454	00:04:10
L00002	N	2016-01-01 01:36:40	987654454	00:00:30

Es muy importante hacer ver que las expresiones XPath usadas en los “value-of” interiores al “for-each” eran relativas a la expresión de dicho “for-each” (la del atributo “select”).

Por supuesto, la expresión del “for-each” podría haber incluido predicados de filtrado. Esa sería la forma de seleccionar sólo un subconjunto de las llamadas.

Veamos un ejemplo en el que se muestra sólo la segunda llamada, junto con su efecto visual:

```
<xsl:for-each select="/factura/llamadas/llamada[@id='L00002']">
```

Id	Tipo	Fecha y hora	Destino	Tiempo
L00002	N	2016-01-01 01:36:40	987654454	00:00:30