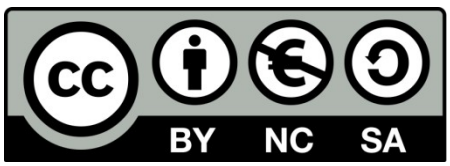
Curso básico de tecnologías XML

5. Generando formatos con ficheros XLS

INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Contenido



1. Introducción.

Hasta ahora se ha visto como construir documentos XML bien formados y con una estructura organizada. En definitiva, el propósito de este metalenguaje es éste. Las **hojas de estilo** (*style sheets*) son un conjunto de instrucciones, generalmente separadas en un archivo, que se asocian a los documentos, ocupándose de los aspectos del formato y de la presentación de los contenidos. **XSL** es el lenguaje de la familia de tecnologías de XML dedicado a dicha labor. En realidad este se divide en tres partes diferenciadas (XSL-FO, XSLT y Xpath) y se encarga de formatear el contenido para una obtener una presentación agradable y adecuada a cada una de las plataformas en las que se empleará la información que contenga el fichero XML.

2. Objetivos.

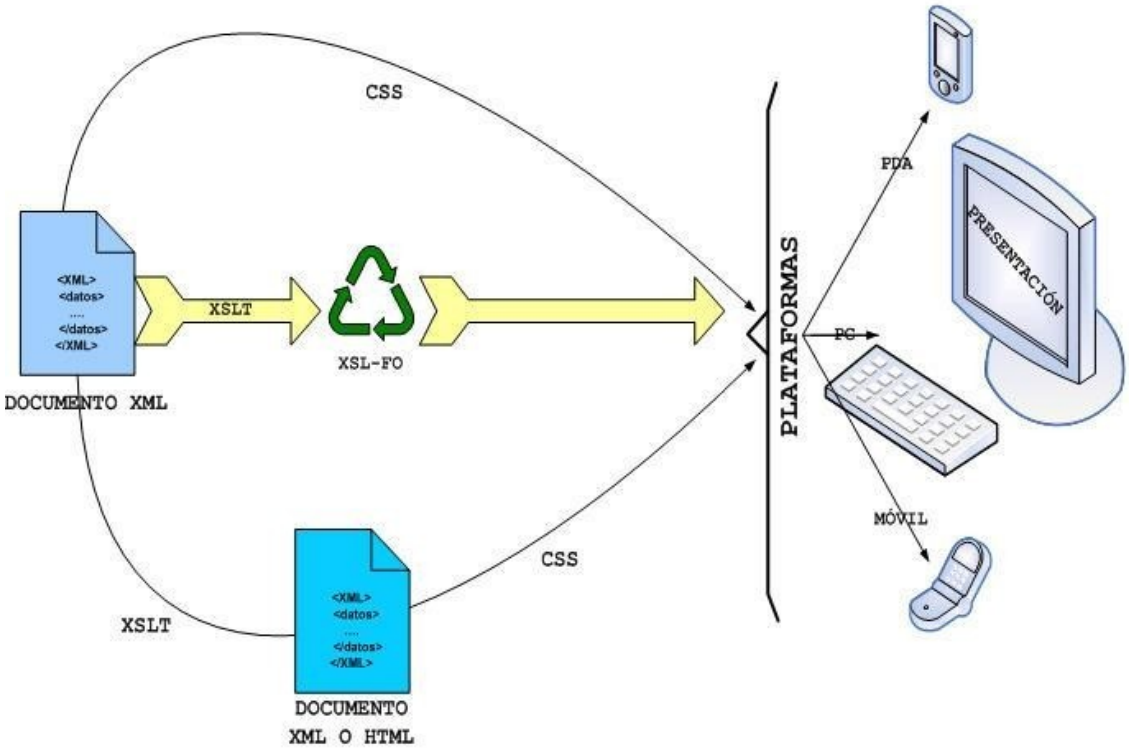
En esta unidad didáctica se aprenderán los siguientes conceptos:

1. Que es una hoja de estilo XSL.
2. Aprenderemos a construir hojas de estilo XSL (eXtensible Style Language).
3. Comprenderemos la importancia de separar la estructura de una información en un documento XML, y la presentación del mismo en una hoja XSL.
4. El alumno será capaz de dar formato a un documento XML mediante una hoja de estilo XSL y de generar un fichero HTML u otro fichero XML a partir de un fichero XML y una hoja de estilo.

3. XLS y XSLT.

En puntos anteriores hemos visto que una de los propósitos con los que nace el metalenguaje de marcas XML es el de **separar la información o el contenido de la presentación**. De esta forma, podremos conseguir muchas ventajas. Imaginemos una serie de documentos que pertenecen a una misma empresa u organigrama. Cada uno de ellos tiene una estructura bien definida, propia de los distintos documentos que se utilizan en dicha empresa. Por otro lado, lo normal es que una empresa mantenga una imagen corporativa y un estilo de presentación de sus documentos. Pues bien, gracias a XML y XSL es posible realizar distintos documentos con una estructura diferente, pero aplicarles el mismo estilo de presentación. Es fácil adivinar que el ahorro de tiempo que obtendremos realizando documentos centrándonos en la estructura sin preocuparnos de la presentación será considerable. La eficiencia que conseguimos de esta manera es difícilmente superable con otros métodos de estructuración y presentación del contenido.

Por otro lado, a veces podemos desear el fenómeno contrario. Es decir, tenemos un contenido ya estructurado y queremos presentarlo de distintas maneras. Esto nos proporciona una gran versatilidad; por ejemplo, con el auge de Internet en los móviles, PDAs, etc. existen muchos sitios Web cuyo contenido puede mostrarse en estas plataformas tan dispares gracias a las hojas de estilo XSL. El contenido como decíamos anteriormente, es el mismo en todos los casos, pero el estilo de presentación se puede adaptar para mostrarlo en la pequeña pantalla de un móvil o en un monitor convencional, pasando por tamaños intermedios como los PCs de bolsillo.



Antes de introducirnos en la sintaxis de XSL, continuemos describiendo su funcionamiento. Una de las características básicas de XSL es que debe crear las hojas en ficheros independientes. No es posible incluir el propio código en el documento XML. De algún modo, iría en contra de la filosofía que se persigue (que no es otra que la de independizar el contenido de la presentación). De este modo, la forma de asociar un documento XML a una hoja XSL es sencilla; una referencia en el documento XML bastará. Hay que tener la precaución de que XSL debe estar situado en la misma ubicación (URI) en la que se almacena el fichero de contenido o de datos. El proceso que se lleva a cabo se resume en los siguientes puntos:

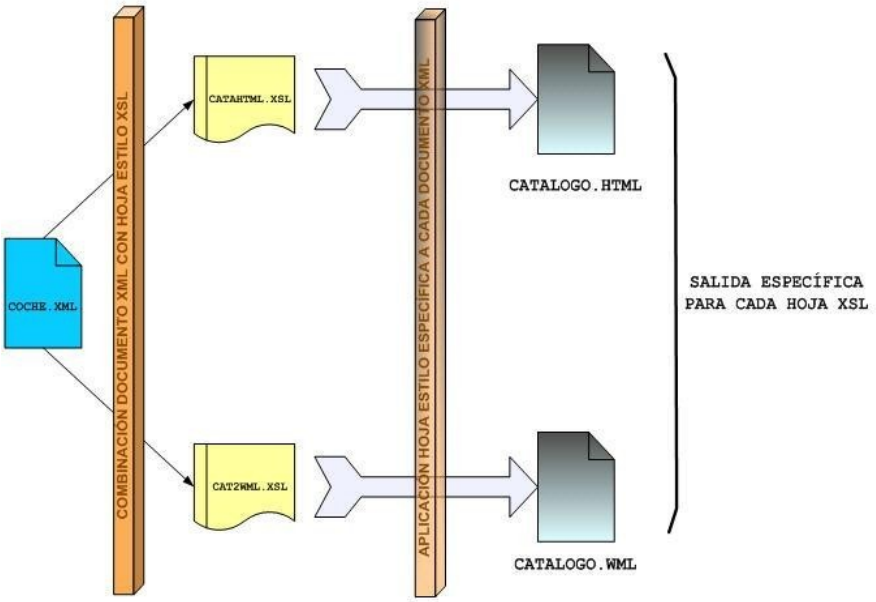
- **Se crea un documento XML** que contiene una estructura de información y una referencia a un fichero XSL.
- **Se comprueba la validez de ese documento XML** mediante los mecanismos vistos anteriormente (*Document type definition* y *XML Schema*).
- Si el fichero contrastado resulta ser un documento bien formado y válido, **se creará una imagen de la información del documento en memoria** y se ejecutará el fichero XSL.

- Suponiendo que la estructura de este fichero XSL es válida, s**e creará un documento HTML resultado de 'fusionar' la información contenida en el documento XML y la presentación contenida en el fichero XSL**.
- Esa salida HTML será introducida en una aplicación final capaz de interpretar dicho lenguaje.

En la mayor parte de las veces, dicha aplicación será un navegador. Hasta el momento no hemos hablado de las Hojas de Estilo en Cascada o CSS. Estas son hojas de estilo descritas en un lenguaje formal que funcionan de modo similar a las hojas XSL. Ambos tipos son recomendados por el consorcio W3 (W3C). En este curso, no abordaremos el uso de las CSS, pero podemos decir que funcionalmente, las hojas XSL aportan algunas características que las hacen más potentes. Por ejemplo, con XSL disponemos de instrucciones que nos permiten realizar un tratamiento de los conjuntos de elementos o de elementos individuales. De esta forma podemos realizar un tratamiento mucho más preciso, estableciendo criterios de filtrado, etc. Otra de las características interesantes de las hojas XSL es que pueden aplicarse tanto en el origen (por ejemplo, un servlet) como en el destino (en un navegador Web).

El lenguaje **XSL**, o **lenguaje de Hojas de Estilo Extensible** (*eXtensible Stylesheet Language* en ingles) se compone de una familia de lenguajes comprendidos en el estándar XML que permiten describir cómo la información contenida en un documento XML cualquiera, debe ser tratada para su presentación. Esta familia se compone de tres lenguajes:

- **XSLT** (Extensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación). Este lenguaje permite convertir un documento XML que mantiene una sintaxis a otra. Es importante reseñar que la transformación puede obtener documentos que no contienen código XML. Para ello se utilizan una serie de plantillas que en base a una serie de reglas, realizan la transformación. En la siguiente imagen veremos conceptualmente como XSLT es capaz de producir dos ficheros distintos a partir de una misma fuente (documento XML COCHE.XML) y dos hojas de estilo diferentes (CATAHTML.XSL y CATAWML.XSL).



- **XSL-FO** (lenguaje de hojas extensibles de formateo de objetos). Este lenguaje permite especificar el formato visual que marcará la presentación de un documento XML.
- **XPath**, (o XML Path Language): Más que un lenguaje, podemos considerarlo una sintaxis que nos permite acceder a distintas secciones de un documento XML. Con esta sintaxis se pueden seleccionar los elementos y atributos necesarios para realizar la transformación.

Como se ha mencionado anteriormente, XSLT está basado en una serie de reglas que especifican la forma en la que el documento XML debe ser procesado. Desde este punto de vista, podemos considerar XSLT como un lenguaje de tipo declarativo.

4. Creando hojas de estilo XSL.

4.1. Contenidos.

XSL se basa en plantillas, entendiendo plantilla como un conjunto de operaciones que nos permite la selección de una parte del contenido del documento XML. Digamos que todo el procesamiento que XSL realiza con un documento XML se basa en la posibilidad de direccionar y acceder de forma independiente a cada una de las partes que componen el documento. Si esto no es así, no tendremos demasiadas posibilidades de formatear el documento mediante las hojas de estilo. Una vez localizado un documento, mediante XML Path Lenguaje (XPath) seleccionamos las distintas partes de las que consta. XPath es un lenguaje algo sofisticado que en definitiva tiene la misión de indicar como debe procesar una hoja de estilo XSL el contenido de un documento XML.

La estructuración de dicho lenguaje se basa es una organización arborescente que facilita el análisis del mismo. Se crea un árbol de nodos, que comienza con un nodo raíz (que no necesariamente tiene que coincidir con el elemento raíz del documento XML), y del que cuelgan una serie de nodos correspondientes a los diferentes elementos de contenido, terminando en los nodos hoja que son los que contienen texto, comentarios, etc. Siempre que un elemento tiene atributos, se crea un nodo por cada atributo que tiene. Este 'cuelga' de el en la estructura arbórea, aunque no se considera que sea un hijo suyo.

4.2. Tipos de nodos.

No ahondaremos mucho en el lenguaje XPath, pero creemos necesario tener unas nociones previas para comprender bien como crear las hojas de estilo XSL. A continuación introduciremos los tipos de nodos que contempla dicho lenguaje.

- **NODOS RAÍZ**: Los identificamos con '/'. No tienen porque coincidir con los elementos raíz de un documento XML. Es más, se puede considerar que el nodo raíz del árbol contiene al elemento raíz.

- **NODOS ELEMENTO**: Se trata de cualquier elemento de un documento XML. Cualquier nodo elemento del árbol tiene un nodo padre, que es a su vez otro elemento. En el caso del elemento raíz, su padre es el nodo raíz. Los nodos además de tener padres, pueden tener hijos que son a su vez nodos elemento, nodos texto etc. Un nodo elemento puede contener información sobre su nombre, sus atributos, etc.
- **NODOS ATRIBUTO**: No tienen carácter de hijos del nodo del que cuelgan. Se consideran etiquetas adicionales. Cada nodo atributo consta de un nombre y un valor.
- **NODOS TEXTO**: Se considera texto, todos aquellos caracteres del documento que no están marcados por alguna etiqueta. Un nodo texto no puede tener hijos.
- **NODOS ESPACIO DE NOMBRE**: Un nodo elemento es padre de un nodo espacio de nombre, pero no a la inversa, es decir, un nodo espacio de nombre no es hijo de su nodo padre. Los elementos tienen nodos espacio de nombres con estas condiciones:
  - Tienen uno por cada atributo que empiece por **xmlns**:
  - Por cada atributo situado sobre un elemento previo cuyo nombre empiece por **xmlns**:
  - Por un atributo xmlns en el elemento o en algún antecesor siendo dicho atributo diferente a **xmlns=**””
- **NODOS COMENTARIO**: Contienen comentarios y a su contenido se puede acceder mediante la propiedad string-value.

5. Patrones de Ajuste.

Los patrones de ajuste son una serie de símbolos que se usan en el código de una plantilla para especificar a que elementos o registros del documento XML queremos aplicarles determinadas transformaciones. Se pueden considerar como los elementos básicos de las páginas de estilo. Para cada nodo del árbol XML podemos definir un patrón de ajuste que marcará como se va a ver afectado este nodo en el documento final.

Los patrones que veremos a continuación constituyen formas abreviadas de selección de nodos. Posteriormente se verán los ejes XML que realizan la misma operación con otra sintaxis. Los símbolos de descripción son los siguientes:

Patrón	Descripción
/	Se utiliza para indicar al procesador XSL que aplique una plantilla partiendo del nodo raíz. Nos permite referirnos también a los hijos directos de un elemento.
//	Se utiliza para referenciar a un nodo que es descendiente de otro sea cual sea la profundidad. También es válido para el nodo raíz.
.	Se utiliza para referenciar al nodo actual.
*	Se utiliza para referenciar a cualquiera de los contenidos de un nodo.
[]	Se utiliza para referenciar a un elemento por su nombre y permite efectuar selecciones.
@	Se utiliza para referenciar el contenido de un atributo.

Para utilizarlos, se debe aplicar el patrón en cuestión mediante el atributo **select**.

6. Uso de ejes.

6.1. Tipos de ejes.

Hasta el momento, hemos visto cómo se pueden seleccionar hijos, atributos, comentarios, etc. con expresiones abreviadas. Un eje forma parte de una expresión (o patrón) Xpath. Los ejes, dentro de una expresión Xpath, definen la dirección que se toma en la estructura jerárquica a partir del nodo de contexto, de manera que esta expresión Xpath, de la que forma parte el eje, permita seleccionar de manera flexible una colección de elementos. Existen varios tipos de ejes que enumeramos a continuación:

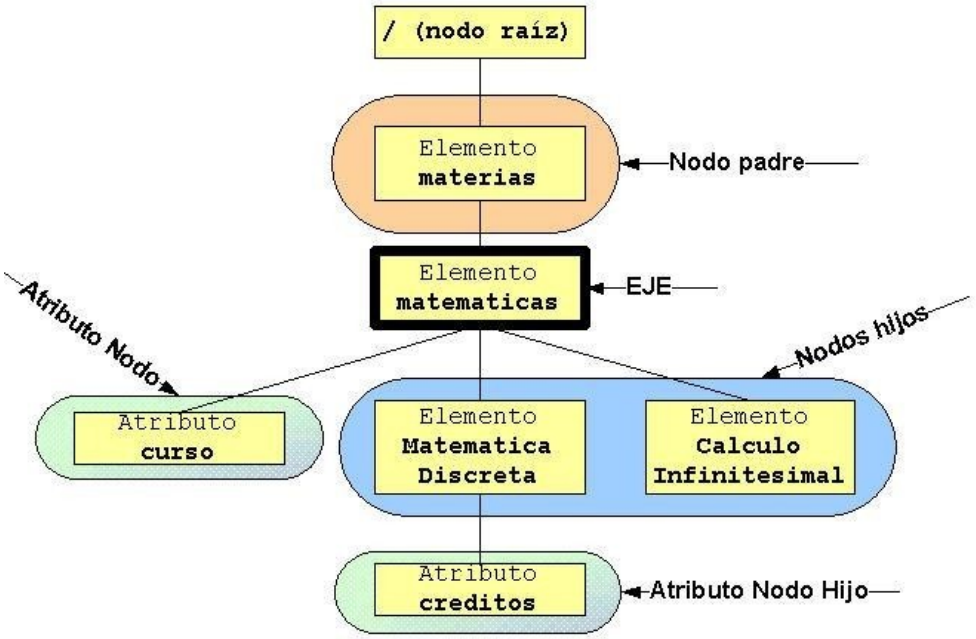
- Eje **ancestor**. El eje ancestor indica todos los antecesores del nodo o elemento de contexto, comenzando con el nodo padre y ascendiendo hacia el nodo raíz.
- Eje **ancestor-or-self**. El nodo ancestor-or-self indica el nodo o elemento de contexto y todos sus antecesores, incluyendo el nodo raíz.
- Eje **attribute**. El eje attribute indica los atributos del nodo o elemento de contexto. Recordamos que sólo los elementos tienen atributos. Este eje se puede abreviar con el signo (@).
- Eje **child**. El eje child indica el hijo del nodo de contexto. Si una expresión XPath no especifica un eje, se considera por defecto. Ya que sólo los nodos raíz o los nodos elementos tienen hijos, cualquier otro uso no selecciona nada.

- Eje **descendant**. El eje descendant indica todos los hijos del nodo de contexto, y todos sus hijos y así sucesivamente. No se incluyen los atributos y namespaces.
- Eje **descendant-or-self**. El eje descendant-or-self indica el nodo de contexto y todos sus descendientes. No se incluyen los nodos atributos y namespaces - el parent de un nodo attribute es un nodo elemento, pero los nodos attribute no son los hijos de sus padres.
- Eje **following**. El eje following indica todos los nodos que aparecen después del nodo de contexto, excepto cualquier nodo descendant, attribute y namespace.
- Eje **following-sibling**. El eje following-sibling indica todos los nodos que tienen el mismo padre que el nodo de contexto y aparecen después del nodo de contexto en el documento de origen.
- Eje **namespace**. El eje namespace indica todos los nodos que están en el ámbito del nodo de contexto. En este caso, el nodo de contexto debe tener un nodo elemento.
- Eje **parent**. El eje parent indica el único nodo que es el padre del nodo de contexto. Se puede abreviar con dos puntos (..).
- Eje **preceding**. El eje preceding indica todos los nodos que preceden al nodo de contexto en el documento excepto cualquier nodo ancestor, attribute y namespace.
- Eje **preceding-sibling**. El eje preceding-sibling indica todos los nodos que tienen el mismo padre que el nodo de contexto y aparecen antes que el nodo de contexto en el documento de origen.
- Eje **self**. El eje self indica el nodo de contexto si mismo. Se puede abreviar con un solo punto (.). Para afianzar más los conocimientos sobre ejes, se realizará a continuación un ejemplo más completo, partiendo del documento XML y mostrando el árbol de nodos, de forma que se vea gráficamente cuál son los nodos afectados por los ejes.

En el siguiente ejemplo, mostraremos como acceder al **padre**, al **atributo** y a los **hijos** del **nodo matematicas**.

```
<?xml version="1.0" encoding="UTF-8"?>
<materias>
    <matematicas curso="primero">
         <Matematica Discreta creditos="6">
               Se ocupa de la aritmética modular y entera además
               de ahondar en la teoría de grafos. Se puede
               considerar una primera aproximación al Álgebra
               Lineal.
         </Matematica Discreta>
         <Calculo Infinitesimal>
               Se trata de ahondar en el concepto de continuidad
               de funciones.
         </Calculo Infinitesimal>
    </matematicas>
</materias>
```

Ahora quizás se entienda mejor el concepto de eje. Considerando la siguiente figura, se muestran a continuación algunos ejemplos:



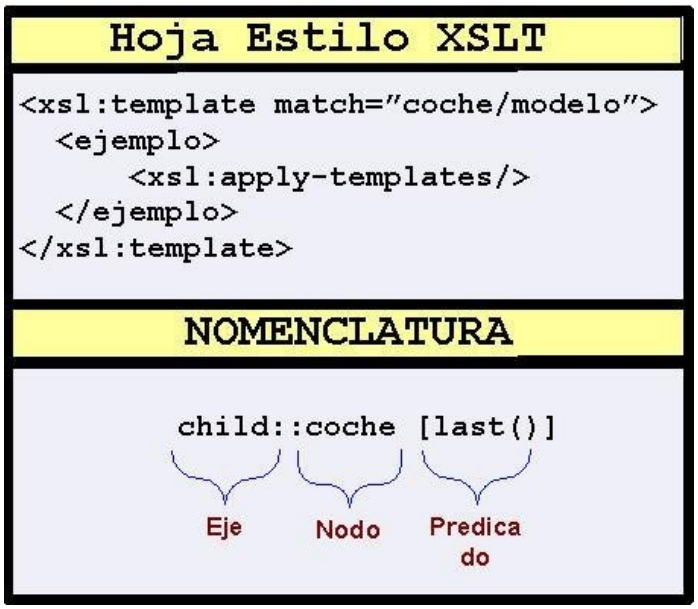
- **Child::matematicas**. Se refiere a los hijos del elemento matematicas. Según la figura, Indica los elementos Matematica Discreta y Calculo Infinitesimal (cuadro de color azul).
- **Ancestor::matematicas**: Se refiere a todos los antecesores del nodo matematicas. Según la figura, el nodo materias y el nodo raíz.
- **Attribute::matematicas**: Se refiere a los atributos del nodo matematicas, por lo que se refiere al atributo curso.

7. Predicados.

Una vez vistos los distintos tipos de ejes que admite XSL, veremos ahora como emplearlos y combinarlos con los predicados que dispone el lenguaje. XPath permite usar una serie de predicados acompañando a los ejes, cuya función es la de realizar una búsqueda más precisa del elemento que queremos.

La sintaxis que se emplea para el uso de ejes puede resultar un poco complicada al principio. Se trata de poner el tipo de eje, acompañado del símbolo '::', del nodo del árbol que vamos a testear y opcionalmente de un tipo de predicado. Estos predicados se separan del resto mediante unos corchetes y se sitúan al final de la ruta de localización.

A continuación mostraremos un esquema que indicará como funciona dicha sintaxis:



Es decir, cada vez que empleamos un eje, debemos aplicárselo al nodo que deseamos testear. En este ejemplo anterior, se ha aplicado el eje **Child** al **nodo coche**. Además se ha empleado el predicado **[last()]** cuya función es devolver el número de elementos hermanos del nodo activo. En el siguiente punto de la unidad didáctica se explica más a fondo el uso de xsl:template.

El significado de la expresión será: **Seleccionamos el último hijo del nodo coche**.

Los predicados pueden utilizar los operadores condicionales típicos (<=,>=,>,<,=,!=,and y or) así como los operadores aritméticos más comunes (+,-,div y mod).

Ahora veremos las funciones que se pueden emplear dentro del predicado:

- **last**(): Devuelve el número de elementos hermanos del nodo actual.
- **position**(): Devuelve un número que corresponde a la posición del nodo contexto entre sus hermanos.
- **count**(*ruta_localizacion*): Devuelve el número de nodos que existen para la ruta de localización señalada en el argumento.
- **id**(*id_nombre*): Selecciona el elemento que tiene el atributo ID reflejado en id_nombre.
- **local-name**(*ruta_localizacion*): Devuelve el nombre local de un elemento. Este nombre local, es el nombre del elemento incluidos los prefijos. Si se omite el argumento, dará como resultado el nombre del nodo actual o contexto.
- **namespace-uri**(*ruta_localizacion*): Devuelve el nombre del URI si el nodo forma parte de un espacio de nombre. Si no forma parte, devuelve una cadena vacía.
- **name**(*ruta_localizacion*): Devuelve el nombre de los nodos en un grupo de nodos, incluído el URI del espacio de nombre, en caso de que lo tenga.

Existen también una serie de funciones dedicadas a la creación y manipulación de cadenas:

- **string**(*objeto*):Convierte un objeto en una cadena de alguna de las siguientes formas:
  - Si el argumento es un nodo-grupo, se devolverá el primer valor de cadena que encuentra (este sería el primer nodo texto encontrado). En caso de omitir el argumento se realiza sobre el nodo actual.
  - Un número se convierte en cadena de acuerdo a las siguientes reglas:
  - Infinito positivo será “Infinity”.
  - Infinito negativo será “-Infinity”.
  - Los valores booleanos se convierten en “true” o “false”.
- **concat**(*cadena1,cadena2,…,cadenaN*): Concatena todas las cadenas introducidas en los argumentos y devuelve esa misma cadena.
- **contains**(*cadenaBase,subcadena*); Devuelve true o false, dependiendo de si subcadena está contenida en cadenaBase.
- **starts-with**(*cadenaBase,subcadena*): Devuelve true o false, dependiendo de si cadenaBase empieza con subcadena o no.
- **substring-before/substring-after**(*cadenaBase,subcadena*): Devuelve el fragmento de la cadena cadenaBase que se encuentre antes o después de la aparición de subcadena.
- **substring**(*cadena,posicionInicial,longitudSubcadena*): Devuelve la subcadena designada por cadena, partiendo desde la posicionInicial. La longitudSubcadena es un parámetro opcional.
- **string-length**(*cadena*): Esta función devuelve el número de caracteres que tiene una cadena en el argumento.
- **normalice-space**(*cadena*): Se encarga de eliminar los espacios en blanco en la cadena pasada como parámetro. Para ello, se eliminan los espacios en blanco iniciales y los sobrantes al final de la cadena. Además, cualquier secuencia de espacios en blanco en el interior de la cadena se reduce a un único espacio.

Y por último una serie de funciones booleanas y numéricas:

- **boolean**(*objeto*): Devuelve true o false en función de los siguientes criterios:
  - Si el objeto es un número, devolverá true sólo si el número es negativo o cero.
  - Si el objeto es un nodo-grupo, devolverá true si el grupo no está vacío.
  - Si el objeto es una cadena, la función devuelve true en caso de que la cadena tenga al menos un carácter.
- **not**(*booleano*): Devuelve true si el argumento es false y false si el argumento es true.
- **true**(): Devuelve siempre true.
- **false**(): Devuelve siempre false.

- **number**(*objeto*): Lo que hace la funión number es una conversión de lo que se le pasa como parametro a un tipo numérico, en concreto de tipo double. Es decir, le podemos pasar un string y si es un string que representa un numero lo convierte (por ejemplo, el string "12,34" lo convertiria a 12,34); si el string no fuera numerico, devolveria NaN, que significa Not a Number, pero no daría un error.
- **sum**(*nodo-grupo*): Devuelve la suma de todos los valores de cadena del grupo de nodos especificado.
- **ceiling**(*numero*): Esta función realiza un redondeo por exceso.
- **floor**(*numero*): Esta función realiza un redondeo por defecto.
- **round**(*numero*): Esta función devuelve el número entero más cercano al argumento.

Otra de las posibilidades que ofrece XSTL, está directamente relacionada con los comentarios (nodos comentario). XSTL está dotado de un mecanismo que nos permite usarlos de forma normalizada. Se trata de la función **comment**(). Es una función que nos permite dar un tratamiento especial a los comentarios.

Con la misma filosofía que la función **comment**(), tenemos otras dos:

- **Node**(): Devuelve completo cualquier tipo de nodo.
- **Text**(): Devuelve completo cualquier nodo de tipo texto

8. Creando hojas de estilo XSLT para generar ficheros HTML.

8.1. Introducción.

El objetivo para el que ha sido creado XSLT es transformar un documento XML. El documento de salida, será el que fichero que típicamente se muestre en un navegador Web. En ocasiones, se transforma a un fichero XHTML y en ocasiones a un fichero HTML. Para llevar a cabo este proceso de creación ya hemos adelantado que sea crea una estructura de nodos de tipo árbol con el código XML de origen. Cuando este árbol ha sido creado por el procesador de XSLT, el procesador centra su atención en las instrucciones que debe contener el documento XSLT. Ese conjunto de instrucciones se conocen como plantillas, que a su vez están formadas por reglas de plantilla.

Las reglas de plantilla se aplican a cada nodo o grupo de nodos que cumplen un determinado patrón. Para ello, se aplican las expresiones XPath ya comentadas. El número de reglas de plantilla que puede tener un documento XSLT es en principio ilimitado. Recordamos ahora la sintaxis que debemos emplear para usar una regla de plantilla:

```
<xsl:template match="grupo-nodos">
"Instrucciones y elementos que queremos aplicarle"
</xsl:template>
```

Para ello, se valen de un atributo **match** que se ocupa de indicar que elemento del documento de entrada está afectado por la plantilla.

Veamos un ejemplo de uso:

```
<xsl:template match="coche">
        <xsl:value-of select="@potencia"/>
</xsl:template>
```

Este código está extrayendo (mediante **value-of-select**) el valor del atributo potencia para el nodo coche. Por supuesto, si hay varios nodos coche en el árbol (circunstancia bastante probable) mostrará los distintos valores del atributo potencia para cada uno de ellos.

Estos elementos template se componen de una serie de instrucciones XSLT que transforman el contenido del documento XML (como la que hemos visto en el anterior ejemplo) y de texto que se transcribe literalmente a la salida del documento (se conocen con el nombre de elementos literales).

Si nos fijamos en el código XSLT, éste se expresa siempre mediante un **espacio de nombres** denominado '**xsl'**, circunstancia que no debe llevarnos a pensar que el código es XSL (de hecho, técnicamente podríamos usar otro prefijo, pero usamos xsl por mantener una uniformidad).

Recordamos que XSL es el conjunto de tecnologías que constituyen una hoja de estilo. (Estas a su vez se dividen en XPath para realizar la selección de los nodos y XSLT para realizar las transformaciones a dichos nodos). Además, nos puede servir para identificar literales si tenemos en cuenta la siguiente regla:

> *'Todo lo que no empiece por 'xsl:' se considera un elemento literal y se trascribe exactamente a la salida'.*

Esta es la filosofía de XSLT; seleccionar diversos elementos del árbol e intercalarlos con distintos literales (generalmente código HTML) para producir una salida correctamente formateada.

Para ello, el procesador de XSLT va buscando distintas instrucciones que se encarguen de extraer información del árbol de partida. También es posible crear nuevos elementos con sus respectivos atributos.

Con estas premisas, es fácil adivinar que las hojas de estilo están compuestas básicamente de instrucciones XSLT (a su vez creadas gracias al lenguaje XPath) y código HTML.

8.2. Plantillas raíz.

A veces podemos tener la sensación de que las reglas de plantilla se ejecutan de forma directa a como las podemos leer en el texto. Esto no es así exactamente. El procesador XSLT ejecuta sólo una regla de plantilla. El resto, deben ser llamadas de forma explicita desde esa primera regla. Esta primera regla recibe el nombre de **plantilla raíz** y tiene el siguiente aspecto:

```
<xsl:template match="coche">
        <xsl:value-of select="@potencia"/>
</xsl:template>
```

Esta plantilla raíz es la única regla de plantilla ejecutada de forma automática.

Veamos un ejemplo sencillo de plantilla, aplicado al nodo raíz del documento (/) en la que únicamente tenemos elementos literales:

```
<xsl:template match="/">
  <html>
    <head>
    </head>
    <body>
    </body>
  </html>
</xsl:template>
```

Cuando el procesador XSLT lee el documento de entrada, el primer nodo que se encuentra es el nodo raíz. Con esta regla se empareja al elemento raíz, y hace que el procesador XSLT emita el siguiente código:

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

Este texto, coincide con la estructura de un documento HTML bien formado.

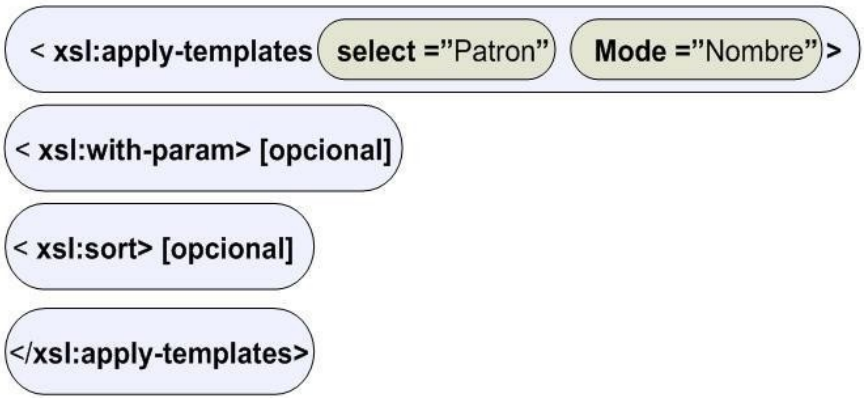
8.3. Aplicación de plantillas múltiples.

Un procesador XSLT ejecuta de forma automática sólo la regla de la plantilla raíz. Si tiene otras reglas de plantilla, hemos visto que deben ser llamadas de forma explícita. Existe para ello otro elemento, que se encarga de ir explorando el documento de manera recursiva y alcanzando los distintos hijos del nodo raíz. Este elemento se llama **xsl:apply-templates**. Funciona seleccionando un conjunto de nodos del documento de entrada e instruyendo al procesador para aplicar las plantillas apropiadas a cada uno. Se usa en conjunción con el atributo **select**.

El atributo select usa el mismo tipo de patrones que los que usa el atributo match para los elementos xsl:template.

Cuando el procesador de código trata un elemento, este nodo funciona del mismo modo que si se tratase de un árbol completo. Esta es una de las ventajas que tiene la estructura de árbol.

Cada parte del mismo es tratada como un todo. La sintaxis de uso es la siguiente:



Donde select utiliza una **expresión del lenguaje XPath** para indicar los nodos que van a ser procesados. Si el atributo no se establece, todos los nodos hijos del nodo actual son seleccionados.

Por otro lado, **mode** se utiliza en el caso de que existan diferentes maneras de procesar el mismo nodo. Con este nombre, distinguiríamos entre ellas.

El elemento opcional '**<'xsl:with-param'>'** establece el valor de un parámetro para pasarlo como 'argumento' a una plantilla.

El elemento opcional '**<'xsl:sort'>**' se utiliza para ordenar. Como clave de ordenación se puede usar cualquier atributo XPath.

El funcionamiento de xsl:apply-templates es el siguiente:

1. El procesador **encuentra en el patrón** o grupo de nodos.
2. El procesador **recorre el documento XSLT para ver cuál es la regla de plantilla más apropiada**.
3. **La regla xsl:apply-templates es sustituida por el resultado que produce esa regla de plantilla**.

Veamos ahora un código XML sobre el que realizaremos posteriormente, diversas selecciones de los diferentes nodos.

```
<?xml version="1.0" encoding="UTF-8”?>
<indice>
   <hijo> Este ejemplo nos servirá para los patrones que usamos a
          continuación
   </hijo>
   <hijo2> Cuerpo del documento
   </hijo2>
</indice>
```

Veamos algunos ejemplos de selección con patrones:

- Si queremos acceder a todos los elementos hijo:

```
<xsl:apply-templates select ="hijo"/>
```

- Si queremos acceder a todos los elementos hijo cuyo padre sea indice:

```
<xsl:apply-templates select ="indice/hijo"/>
```

- Si queremos acceder Acceso al nodo raíz del documento XML:

```
<xsl:apply-templates select ="/"/>
```

- Si queremos acceder a todos los elementos hijo que tengan como antecedente a indice:

```
 <xsl:apply-templates select ="indice//hijo"/>
```

- Si queremos acceder al primer elemento hijo que tenga como padre a indice:

```
<xsl:apply-templates select ="indice/hijo[1]"/>
```

- Si queremos acceder al ultimo elemento hijo que tenga como padre a indice:

```
<xsl:apply-templates select ="indice/hijo[position()=last ()]"/>
```

- Si queremos acceder a los elementos hijo que sean pares y que tengan como padre a indice:

```
<xsl:apply-templates select ="indice/hijo[position ()mod 2=0 ]"/>
```

- Si queremos acceder a todos los elementos hijo que tengan un atributo atb

```
<xsl:apply-templates select ="hijo[@atb]"/>
```

- Si queremos acceder a todos los elementos hijo que no tengan un atributo atb

```
<xsl:apply-templates select ="hijo[not(@atb)]"/>
```

- Si queremos acceder a todos los elementos hijo cuyo atributo atb tenga el valor X:

```
<xsl:apply-templates select ="hijo[@atb='X']"/>
```

- Si queremos acceder a todos los elementos indice que tengan un elemento hijo con valor X:

```
<xsl:apply-templates select ="indice[hijo='X']"/>
```

- Si queremos realizar la misma operación anterior, pero normalizando la búsqueda. De este modo se eliminarán los espacios en blanco:

```
<xsl:apply-templates select =

"test1[normalizespace(titulo)='XXX']"/>
```

- Si queremos acceder a todos los nodos hijo o hijo2:

```
<xsl:apply-templates select ="hijo|hijo2”/>
```

Veamos una tabla en la que se muestran las diferentes fases por las que pasará el documento XML.

CÓDIGO XML

```
<Codigo>
    <Modelo>berlina</Modelo>
    <Fabricante>SEAT</Fabricante>
</Codigo>
```

Hoja Estilo XSLT

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

<xsl:template match="/">
 <hijo1>
  <xsl:value-of select="//Modelo"/>
 </hijo1>
 <hijo2>
  <xsl:value-of select="//Fabricante"/>
 </hijo2>
</xsl:template>

</xsl:stylesheet>
```

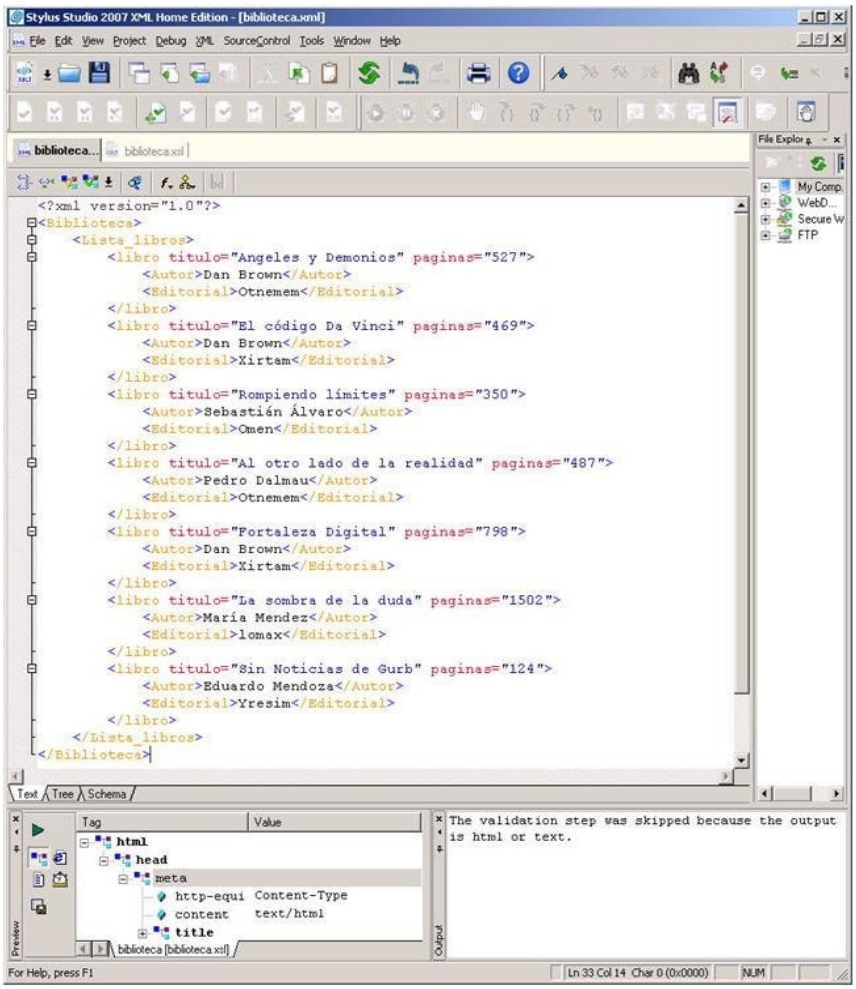
SALIDA

```
<HIJO1> Berlina </HIJO1>
<HIJO2> SEAT </HIJO2>
```

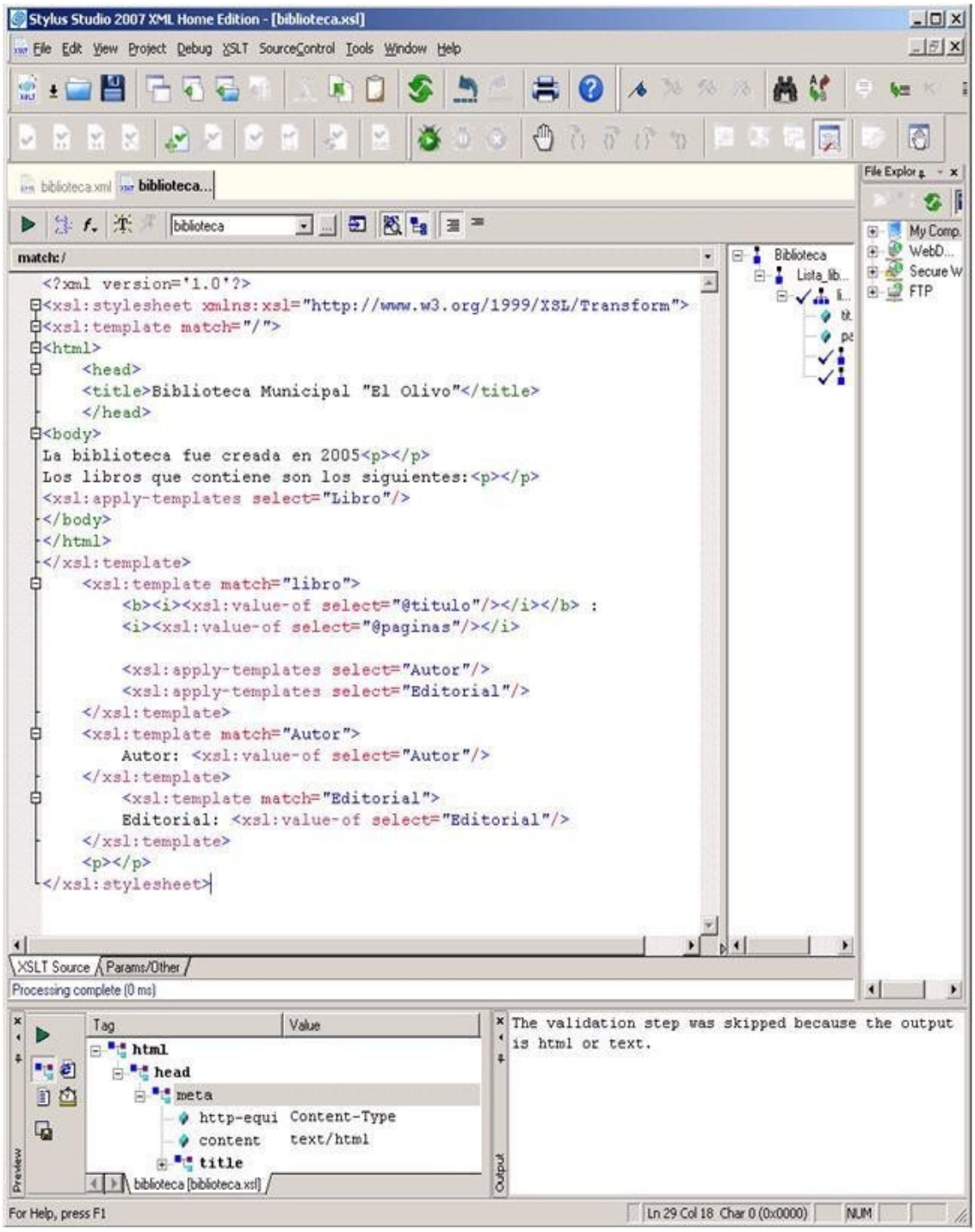
Vista HTML

Berlina
SEAT

A continuación mostraremos un ejemplo completo, partiendo del código XML hasta la salida generada una vez aplicado XSLT.

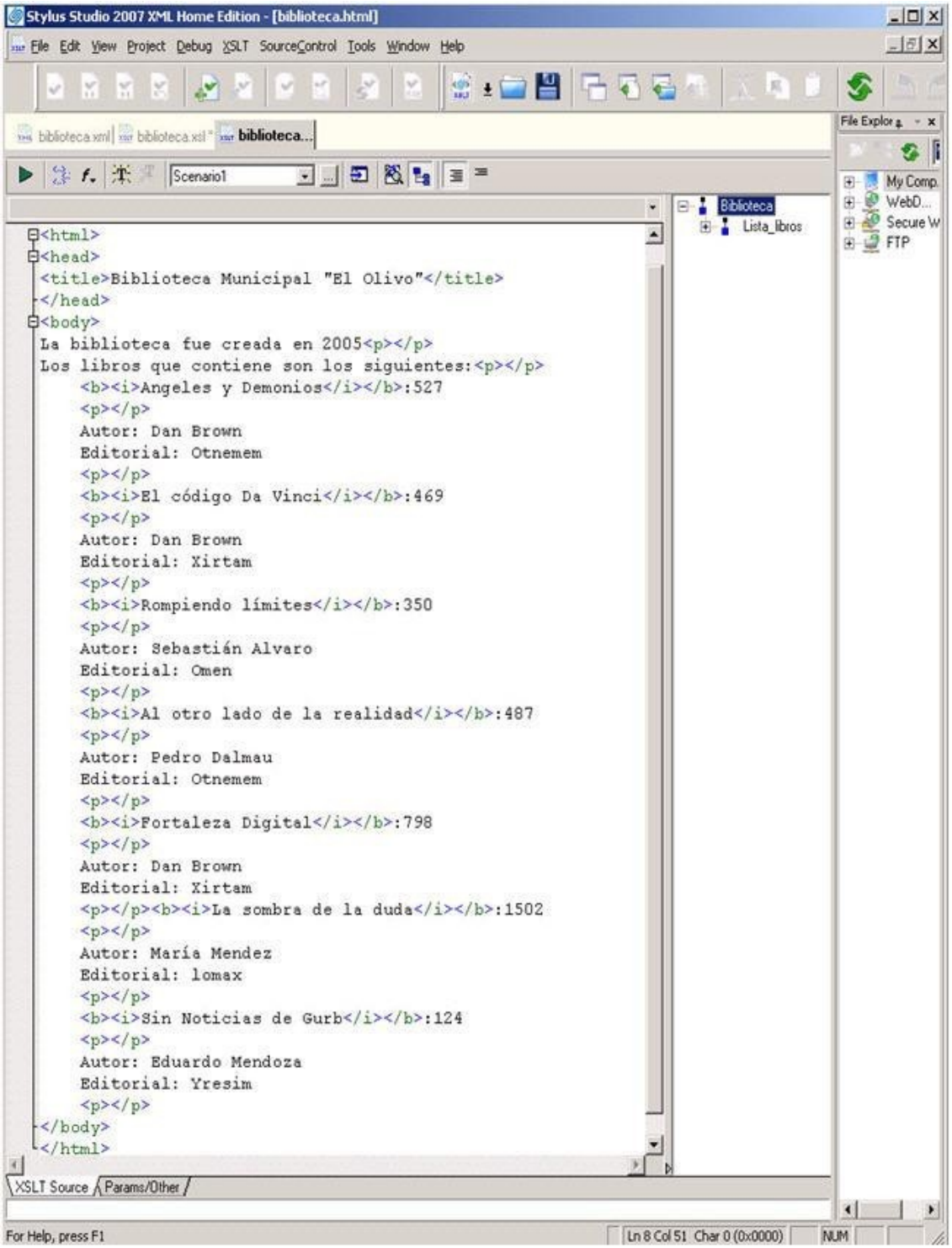


Una vez visto el código del documento XML, mostraremos el documento XSLT:

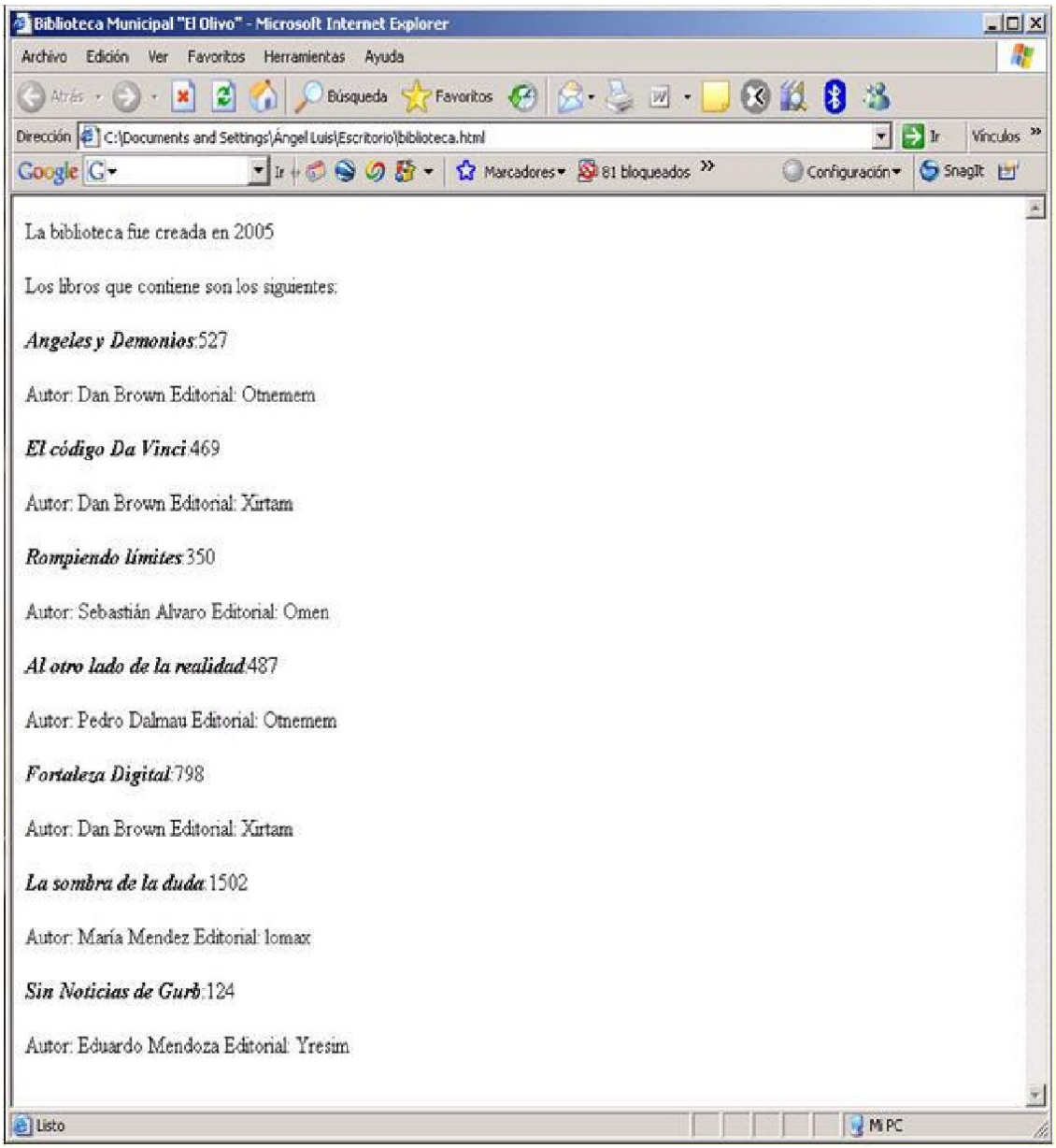

Stylus Studio 2007 XML Home Edition - [biblioteca.xsl]
File Edit View Project Debug XSLT SourceControl Tools Window Help
biblioteca.xml
biblioteca...
biblioteca
match:/
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
<title>Biblioteca Municipal "El Olivo"</title>
</head>
<body>
La biblioteca fue creada en 2005
Los libros que contiene son los siguientes:
<xsl:apply-templates select="Libro"/>
</body>
</html>
</xsl:template>
<xsl:template match="libro">
<xsl:value-of select="@titulo"/> :
<xsl:value-of select="@paginas"/>
<xsl:apply-templates select="Autor"/>
<xsl:apply-templates select="Editorial"/>
</xsl:template>
<xsl:template match="Autor">
Autor: <xsl:value-of select="Autor"/>
</xsl:template>
<xsl:template match="Editorial">
Editorial: <xsl:value-of select="Editorial"/>
</xsl:template>

</xsl:stylesheet>
Biblioteca
Lista_lib...
XSLT Source
Params/Other
Processing complete (0 ms)
Tag
Value
html
head
meta
http-equi Content-Type
content text/html
title
biblioteca [biblioteca.xsl]
The validation step was skipped because the output is html or text.
Preview
Output
For Help, press F1
Ln 29 Col 18 Char 0 (0x0000)
NUM

Finalmente, este código XSLT produciría el siguiente fichero de salida:



Que al ejecutarlo en un navegador Web como por ejemplo Internet Explorer, mostraría el siguiente resultado:



El proceso que puede parecer un poco largo, muestra unos resultados bastante aceptables. El funcionamiento es el siguiente:

1. La plantilla raíz invoca a la regla de plantilla asociada al nodo libro.
2. En el interior de esta plantilla se invoca a dos nuevas plantillas: una para el Autor y otra para la Editorial.
3. Hay que notar que la plantilla raíz llama a la plantilla libro tantas veces como nodos libro encuentre el documento XML original. En este caso, la invoca 7 veces (hay 7 libros en el documento original).
4. Finalmente se general el documento HTML con los elementos literales introducidos (código HTML) y los resultados de la invocación de las diversas plantillas.

8.4. Instrucciones.

XSTL permite utilizar diversos tipos de instrucción para extraer valores de los nodos elementos o nodos atributo. En anteriores ejemplos se han visto algunos de ellos, y es probable que el alumno no supiera el significado de ellos. A continuación, mostraremos una tabla con la sintaxis y la explicación de cada uno de ellos.

INSTRUCCIÓN	SINTAXIS	UTILIZACIÓN
Xsl:value-of	`<xsl:value-of select="cadena-seleccion"/>`	Permite extraer una cadena de cualquier lugar en el árbol de origen. En cadena-selección podemos incluir una expresión XPath para apuntar a un nodo.
Xsl:strip-space	`<xsl:strip-space="lista"/>`	Toma una lista de elementos con espacios en blanco, eliminándolos de la parte delantera y trasera y comprimiendo los espacios dobles.
Xsl:preserve-space	`<xsl:preserve-space elements="lista"/>`	Toma una lista de nombres de elemento separada por espacios en blanco y actúa manteniendo dichos espacios

Xsl:sort	`<xsl:sort select=cadena-expresion`
	`Lang={nmtoken}`
	`Data-type={"text"|"number"|nombre}`
	`Order={"ascending"|"descending"}`
	`Case-order={"upper-first"|"lower-first"}/>`
		Permite ordenar los nodos que llegan como resultado de aplicar expresiones de XPath. De este modo, el orden no tiene porque coincidir con el orden en el que aparecen los nodos en el documento.

Xsl:include	`<xsl:include href="uri-referencia"/>`	Con esta instrucción podemos incluir una hoja de estilo desde una referencia Web o desde un URI. Este elemento debe ser un hijo directo de *xsl:stylesheet*.
Xsl:import	`<xsl:import href="uri-referencia"/>`	Con esta instrucción permite importer una hoja de estilo externa. Este elemento debe ser un hijo directo de *xsl:stylesheet*.
Xsl:apply-imports	`<xsl:apply-imports/>`	Se emplea para indicar que la regla de plantilla del documento original reemplace a la del documento importado.

Xsl:apply-templates	`<xsl:apply-templates select="nodo-grupo" mode="nombre">`
	`...`
	`</xsl:apply-templates>`
		Comentado anteriormente.

Xsl:call-template	`<xsl:call-template named="nombre">`
	`...`
	`</xsl:call-template>`
		Se emplea para invocar una plantilla a través del atributo named coincidente. No modifica el nodo actual.

Xsl:copy	`<xsl:copy use-attribute-sets="nombres">`
	`...`
	`</xsl:copy>`
		Se emplea para copiar el nodo actual en el árbol resultante.

Xsl:for-each	<xsl:for-each select="nodo-grupo"> ... </xsl:for-each>	Realiza un bucle con tantas iteraciones como nodos haya en el grupo. Dentro del bucle podremos realizar diversas instrucciones.
Xsl:if	<xsl:if test="expresión-booleana" ... </xsl:if>	Si la expresión booleana tiene el valor *true* se ejecuta el contenido del elemento xsl:if.
Xsl:choose	<xsl:choose> [<xsl:when>|<xsl:otherwise] </xsl:choose>	Se utiliza para realizar una elección entre distintas alternativas. Las alternativas pueden ser elementos when u otherwise
Xsl:otherwise	<xsl:otherwise> ... </xsl:otherwise>	Funciona como una instrucción else en programación convencional. También funciona del mismo modo que xsl:when pero sin prueba booleana.
Xsl:when	<xsl:when test="expresión-booleana" ... </xsl:when>	Funciona como xsl:if. La diferencia es que se utiliza siempre dentro de una instrucción xsl:choose.
Xsl:variable	<xsl:variable name="nombre" select="expresión"> ... </xsl:variable>	Esta instrucción se encarga de crear una variable. Encerrando el nombre de la variable entre llaves y precediéndolo del carácter $, está variable queda accesible para el resto de instrucciones XSLT.
Xsl:param	<xsl:param name="nombre" select="expresión"> ... </xsl:param>	Esta instrucción crea una variable si y sólo si esa variable no ha sido creada todavía. En caso de que hubiera sido creada, funciona cono xsl:variable.
Xsl:with-param	<xsl:with-param name="nombre" select="expresion"> ... </xsl:with-param>	Gracias a esta instrucción podemos pasar una variable a una regla de plantilla de forma parecida a como pasamos un parámetro a una función en otros lenguajes de programación. Está permitida dentro de elementos *xsl:apply-templates* y *xsl:call-template*.
Xsl:decimal-format	<xsl:decimal-format name="nombre" decimal-separator="carácter" grouping-separador="carácter" infinity="cadena" minus-sign="carácter" NaN="cadena" percent="carácter" per-mile="carácter" zero- digit="carácter" pattern-separator="carácter"/>	Se utiliza para declarar un formato decimal. Los atributos son: -**decimal-separator:** especifica el carácter usado para la coma decimal. -**grouping-separator:** especifica el carácter usado para el punto de miles. -**pattern-separator:** especifica el carácter usado para separar subpatrones positivos y negativos en un patrón.
Xsl:template	<xsl:template match="nodo-grupo" name="nombre" priority="numero"	Comentado anteriormente.

	mode="nombre"> ... </xsl:template>	
Xsl:namespace-alias	<Xsl:namespace-alias stylesheet- prefix="prefijo"|"#default" result- prefix="prefijo"|"#default"/>	Permite crear un alias para un espacio de nombre. El atributo *stylesheet-prefix* es el alias que está creando, y *result-prefix* es el atributo real.
C R E A C I Ó N D E N O D O S		
Xsl:element	<xsl:element name={nombre} namespace = {uri-referencia} use-atribute-sets="nombres"> ... </xsl:element>	Con esta instrucción podemos crear un elemento nuevo con un nombre asociado.
Xsl:attribute	<xsl:attribute name={nombre} namespace={uri-referencia}> ... </xsl:atributte>	Con esta instrucción podemos crear un atributo con un nombre asociado.
Xsl:attribute-set	<xsl:attribute-set name={nombre} use-attribute-sets="nombres"> ...xsl:attribute elements.. </xsl:atributte-set>	Con esta instrucción podemos crear un conjunto de atributos.
Xsl:text	<xsl:text disable-output- escaping="yes"|"no"> ...#PCDATA.. </xsl:text>	Con esta instrucción podemos crear un nodo texto.
Xsl:processing-instruction	<xsl:processing-instruction name={nombre}> ...contenido... </xsl:processing-instruction>	Genera un nodo de instrucción de procesamiento en el resultado. El atributo name indica el nombre. El contenido del elemento proporciona el resto de la instrucción de procesamiento.
Xsl:number	<xsl:number level="single"|"multiple"|"any" count="patron" from="patron" value="numero-expresion" format={cadena} lang={nmtoken} letter- value={"alphabetic"|"tradicional"} grouping-separator={carácter} grouping-size={numero}/>	Se emplea para devolver un número formateado en el árbol resultado.
Xsl:comment	<xsl:comment> ... </xsl:comment>	Se emplea para crear un comentario XML.

8.5. Funciones de XSLT

Además, del conjunto de instrucciones que se ha visto en la anterior tabla, XSLT permite utilizar una serie de funciones dentro de esas instrucciones con el objetivo de refinar la búsqueda de elementos a transformar.

Las funciones más destacadas son:

- **document**(*uri[,nodo-grupo]*) Su utilización nos permite extraer información de un documento XML distinto al documento fuente. Esa información es analizada

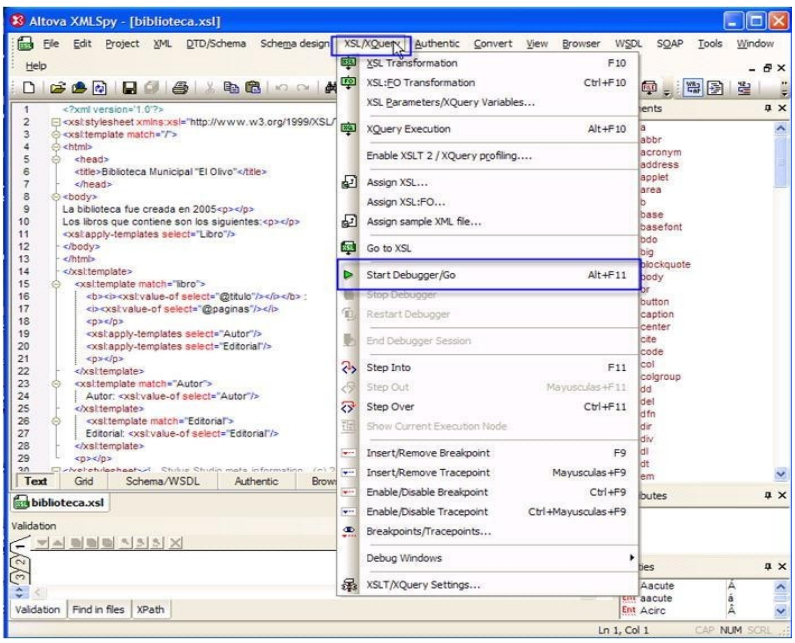
convenientemente y adaptada al tipo de datos del documento XML fuente. El valor de salida es un nodo-grupo.

- **format-number**(*numero, formatPattern [,decimalPattern]*) Esta función devuelve una cadena con el número formateado. Para ello, se encarga de coger un número y formatearlo de acuerdo a los argumentos segundo y tercero.
- **current**(). La sintaxis de esta función es sencilla ya que se invoca sin ningún tipo de parámetro. Se encarga de devolver un nodo-grupo que únicamente contiene en ese grupo el nodo actual.
- **system-property**(*nombreElemento*) Permite devolver algunas propiedades del elemento que introducimos como parámetro.
    - **Xsl:version** devuelve un número con la versión de XSLT que implementa el procesador.
    - **Xsl:vendor** devuelve el fabricante del procesador XSLT.
    - **Xsl:vendor-url** devuelve el URL de la página web del fabricante.

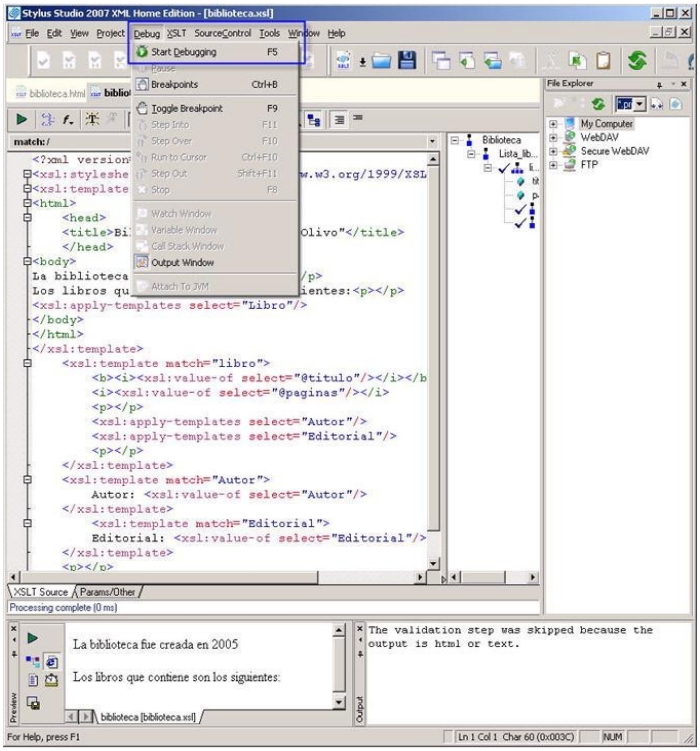
9. Herramientas para la depuración de hojas de estilo XSLT

Como en otros lenguajes de programación, existen aplicaciones que están provistas de depuradores de código XSLT con el fin de obtener hojas de estilo más optimizadas y más adaptadas a nuestras preferencias. Generalmente este tipo de herramientas de depuración suelen incluirse en los entornos de desarrollo que soportan el lenguaje XML. Como ejemplo, pueden servirnos las herramientas Altova XMLSpy 2007 y Stylus Studio 2007 XML vistas en el último libro de la unidad didáctica esquemas de XML. A continuación incluiremos dos imágenes del menú de depuración que ofrecen ambas herramientas:

- **Altova XMLSpy 2007:**



- **Stylus Studio 2007.**



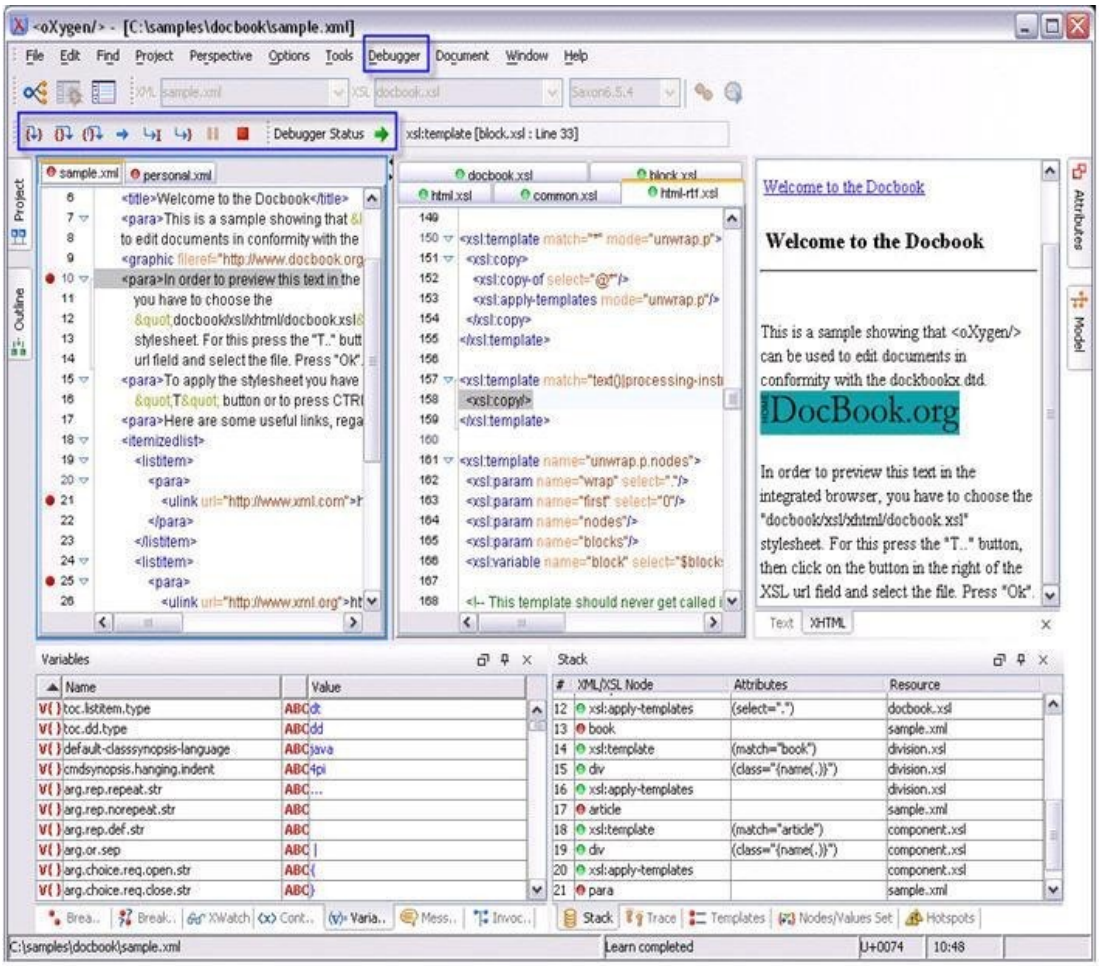
Además de estos dos entornos de programación XML, existen otras herramientas como por ejemplo:

- **Visual Studio de Microsoft:**

Poco queda por decir de la potente plataforma de desarrollo que ha creado Microsoft. Esta incluye soporte para el desarrollo de código XML a través de diversos paquetes. Así mismo, contiene un depurador de código que nos será de gran utilidad a la hora de optimizar nuestro código XSLT. Por último indicar que no se trata de una versión freeware y que es necesario adquirir una licencia para su uso.

- **Oxygen:**

A continuación incluimos una captura extraída de la página oficial de la herramienta (http://www.oxygenxml.com/) en la que podemos descargar una versión de evaluación por un periodo de 30 días.



En todas ellas, las opciones de depuración son las típicas de cualquier depurador; introducir puntos de ruptura, conocer el valor de determinadas variables (en este caso elementos o atributos), ejecutar parcialmente un segmento de código, etc.

10. Conclusión.

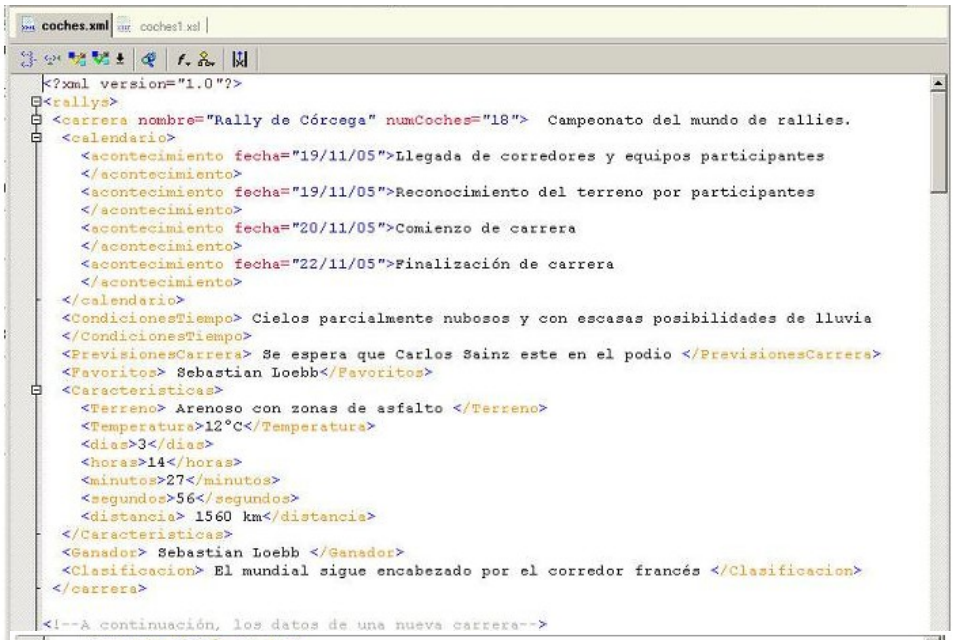
En esta unidad didáctica el alumno ha debido consolidar sus conocimientos sobre la función de las hojas de estilo y sobre las posibilidades que ofrecen. Gracias a eXtensible Stylesheet Language (XSL) y los paquetes que lo forman podremos realizar la presentación del documento XML de modo que se adapte a nuestras exigencias. Es importante que el alumno haya comprendido las posibilidades que proporciona el hecho de separar la información o contenido de la presentación. Para ello, se han incluido una serie de funciones e instrucciones propias de XSTL que nos facilitarán la tarea de transformación de elementos con el objetivo de obtener un documento final con la presentación que buscamos.

11. Ejercicio resuelto.

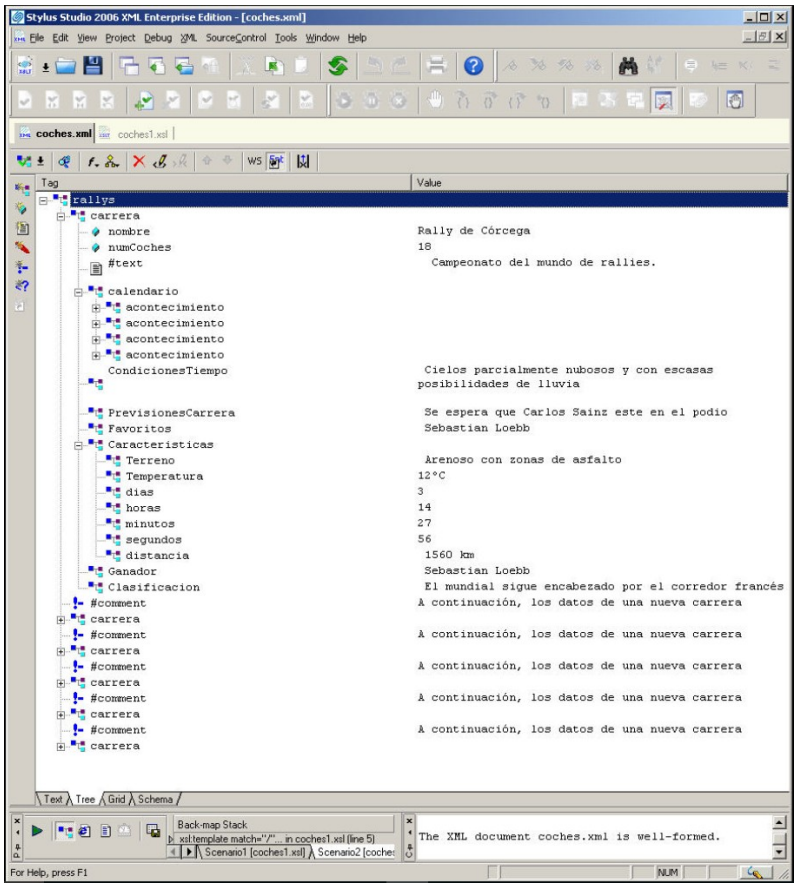
A continuación se va a mostrar un ejercicio completo de creación de una hoja XSLT para un documento XML. Esta hoja XSLT generará código HTML que veremos como es interpretado por un navegador Web. El dominio sobre el que se ha creado el ejercicio es el Campeonato del Mundo de Rallies. Se generará mediante la hoja de estilo mencionada, una tabla resumen de las distintas carreras que componen el mundial. Esta tabla contendrá datos relevantes como el ganador de la prueba, el tiempo total empleado, la temperatura media del evento y las condiciones del terreno entre otros datos.

11.1. Posible solución.

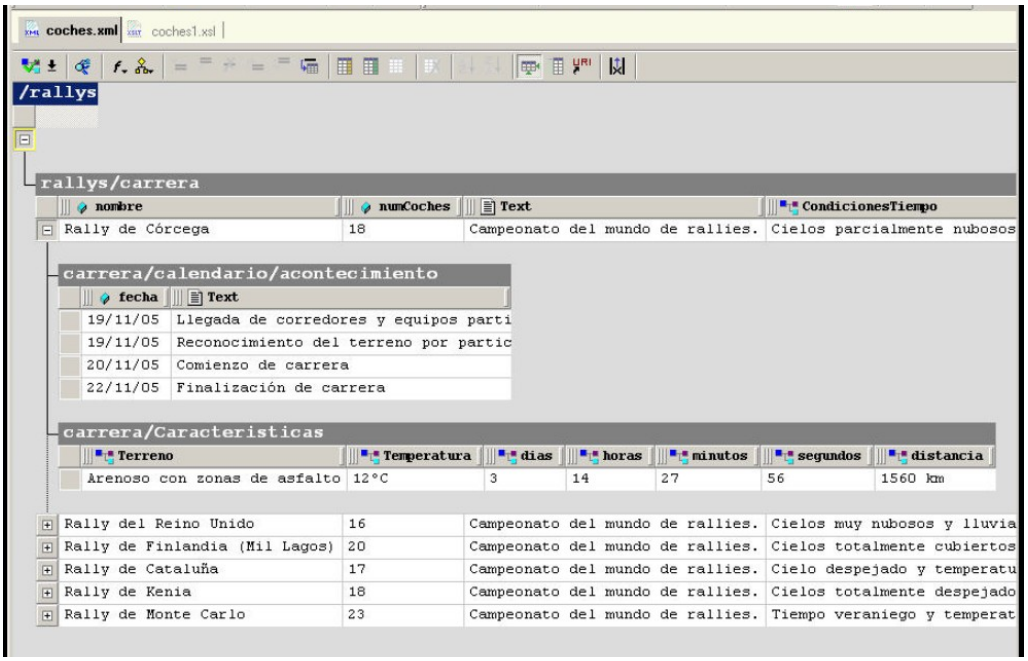
En primer lugar, mostraremos una serie de capturas sobre el documento XML que contiene la información mencionada del campeonato del mundo de rallies.



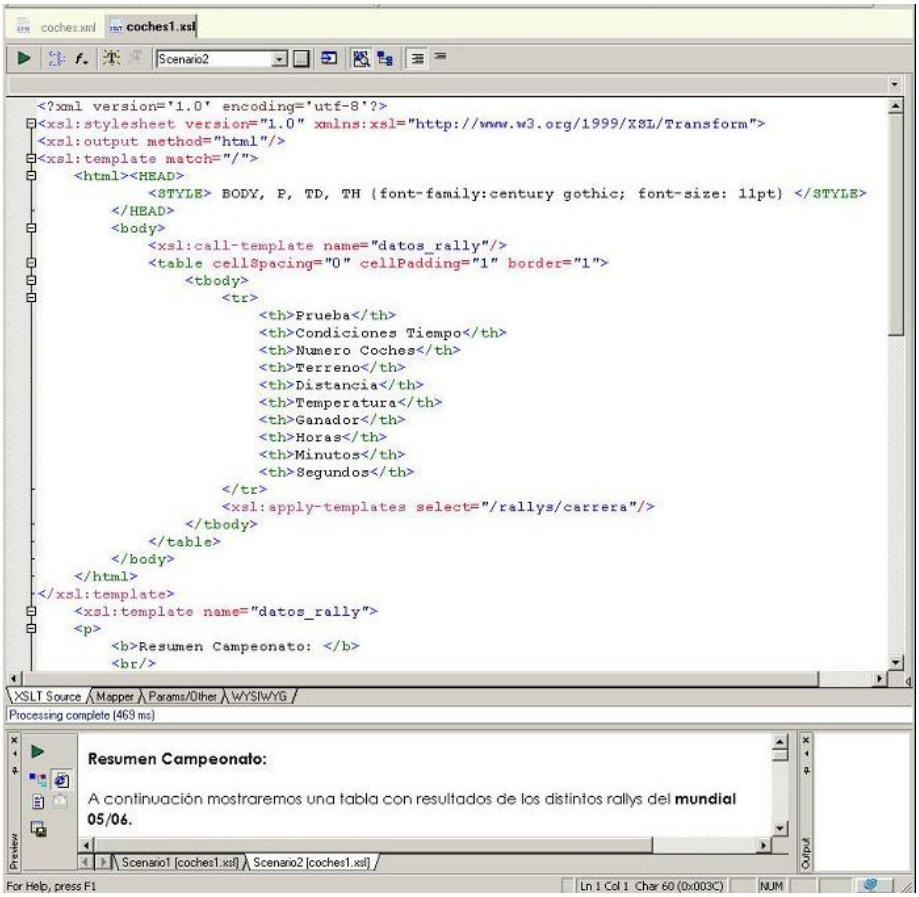
Una vez visto el documento XML en las anteriores imágenes, nos disponemos ahora a mostrar la jerarquía del código gracias a una funcionalidad que nos ofrece la herramienta Stylus Studio para ver el documento en forma de árbol.



La vista de la pestaña Grid de ésta herramienta también nos ofrece una visión en forma de tabla del código que compone el documento XML:



Una vez visto el documento XML (de las distintas formas que ofrece la herramienta), mostraremos a continuación el código XSLT que transformará la información contenida en el documento XML para darle una presentación adecuada. En este caso, crearemos una tabla con algunos campos que se han considerado importantes para el dominio que nos ocupa.



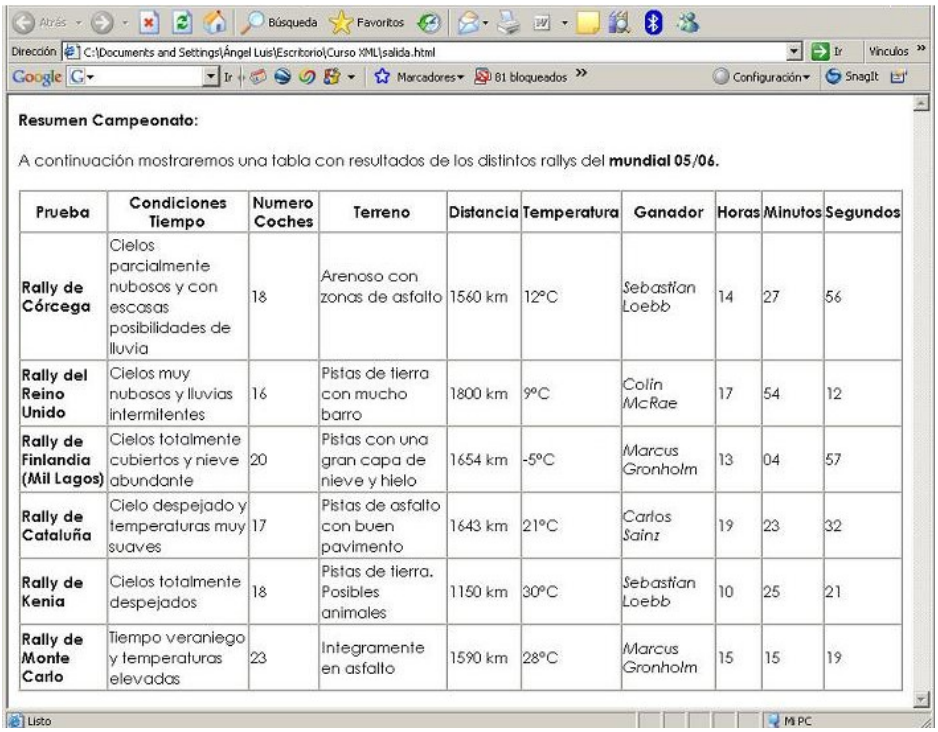
En esta imagen que se muestra a continuación, aparece una vista previa que proporciona la herramienta y que nos permite ver el resultado que hemos conseguido con la hoja de estilo XSL.

Resumen Campeonato:

A continuación mostraremos una tabla con resultados de los distintos rallys del **mundial 05/06.**

Prueba	Condiciones Tiempo	Numero Coches	Terreno	Distancia	Temperatura	Ganador	Horas	Minutos	Segundos
Rally de Córcega	Cielos parcialmente nubosos y con escasas posibilidades de lluvia	18	Arenoso con zonas de asfalto	1560 km	12ºC	*Sebastian Loebb*	14	27	56
Rally del Reino Unido	Cielos muy nubosos y lluvias intermitentes	16	Pistas de tierra con mucho barro	1800 km	9ºC	*Colin McRae*	17	54	12
Rally de Finlandia (Mil Lagos)	Cielos totalmente cubiertos y nieve abundante	20	Pistas con una gran capa de nieve y hielo	1654 km	-5ºC	*Marcus Gronholm*	13	04	57
Rally de Cataluña	Cielo despejado y temperaturas muy suaves	17	Pistas de asfalto con buen pavimento	1643 km	21ºC	*Carlos Saínz*	19	23	32
Rally de Kenia	Cielos totalmente despejados	18	Pistas de tierra. Posibles animales	1150 km	30ºC	*Sebastian Loebb*	10	25	21
Rally de Monte Carlo	Tiempo veraniego y temperaturas elevadas	23	Integramente en asfalto	1590 km	28ºC	*Marcus Gronholm*	15	15	19

A continuación mostraremos el resultado final, ejecutado en el navegador Web Internet Explorer 6.0.

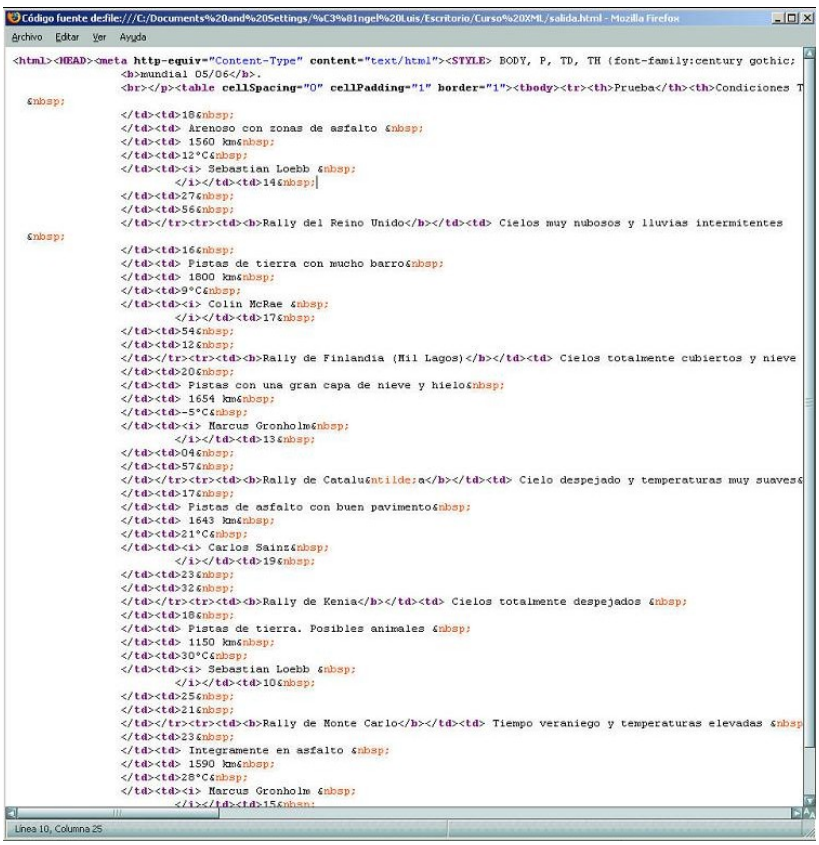


Resumen Campeonato:

A continuación mostraremos una tabla con resultados de los distintos rallys del **mundial 05/06.**

Prueba	Condiciones Tiempo	Numero Coches	Terreno	Distancia	Temperatura	Ganador	Horas	Minutos	Segundos
Rally de Córcega	Cielos parcialmente nubosos y con escasas posibilidades de lluvia	18	Arenoso con zonas de asfalto	1560 km	12ºC	*Sebastian Loebb*	14	27	56
Rally del Reino Unido	Cielos muy nubosos y lluvias intermitentes	16	Pistas de tierra con mucho barro	1800 km	9ºC	*Colin McRae*	17	54	12
Rally de Finlandia (Mil Lagos)	Cielos totalmente cubiertos y nieve abundante	20	Pistas con una gran capa de nieve y hielo	1654 km	-5ºC	*Marcus Gronholm*	13	04	57
Rally de Cataluña	Cielo despejado y temperaturas muy suaves	17	Pistas de asfalto con buen pavimento	1643 km	21ºC	*Carlos Sainz*	19	23	32
Rally de Kenia	Cielos totalmente despejados	18	Pistas de tierra. Posibles animales	1150 km	30ºC	*Sebastian Loebb*	10	25	21
Rally de Monte Carlo	Tiempo veraniego y temperaturas elevadas	23	Integramente en asfalto	1590 km	28ºC	*Marcus Gronholm*	15	15	19

Finalmente, se muestra el código HTML que ha sido generado tras la creación de la hoja de estilo, y que es el que se encuentra detrás de los resultados mostrados en la anterior imagen.



12. Ejercicio propuesto.

En el ejercicio resuelto hemos visto como producir una presentación en forma tabular de los datos contenidos en el fichero XML. Realizar ahora una hoja de estilo XSLT sobre el anterior código XML que produzca una presentación con los distintos rallies del campeonato del mundo en forma de lista. Para ello, formatear los nombres más importantes, resaltándolos con negrita y cursiva.