élogos

ACCESO A DATOS EN JAVA

élogos

XML como almacenamiento de datos

Nacimiento del XML

XML es un lenguaje de Marcado, tal como lo es HTML. Los dos comparten un mismo origen, el SGML, el SGML (Standard Generalized Markup Language o Lenguaje estandarizado y generalizado de Marcado) es un estándar internacional (ISO 8879) que nació con la necesidad de publicar documentos extensos, originalmente manuales de mantenimiento de aeronaves, su inventor fue Charles F. Goldfarb en 1974. La principal característica del lenguaje SGML era y es la de crear elementos personalizados y desarrollar vocabularios que pueden ser utilizados por otras aplicaciones. En definitiva, lo que SGML permite es crear otros lenguajes de marcado personalizados. Esto es lo que Tim Berners-Lee hizo al crear el lenguaje HTML (Hipertext Markup Language), a partir de SGML creó su propio lenguaje de Marcado.

Esto nos lleva a la primera conclusión, que probablemente destierre alguna de las ideas que podamos tener acerca del lenguaje XML. El lenguaje XML no es una evolución de HTML ni es una versión extendida de éste. HTML y XML comparten el mismo origen, puesto que los dos están basados en SGML.

Cuando el W3C (World Wide Web Consortium), pensó que era hora de poner orden en el galimatías que representaba el sinfín de tecnologías derivadas de HTML, que los distintos fabricantes de software para la red habían realizado a partir de sus interpretaciones personales y objetivos empresariales de distinto signo, decide que va ha realizar un nuevo lenguaje que dé soporte a la red tanto en el presente como en el futuro. Para ello, una de las primeras premisas que se fijó en W3C fue la escalabilidad.

El W3C en esta ocasión jugaba con ventaja, ya que disponía de la suficiente información y experiencia basada en lo que actualmente representaba, para lo bueno y para lo malo, el lenguaje de marcado HTML. Es por esto que también se plantearon los problemas que presentaba el lenguaje HTML para darles una solución. Entre otros, y a grandes rasgos, el lenguaje HTML presentaba y presenta problemas como que es un lenguaje que está basado más en la definición de la presentación que en el contenido. HTML, además, presenta problemas significativos de internacionalización, posee una estructura caótica (más adelante veremos por qué, al definir la estructura de los documentos XML en comparación con HTML). La interpretación del HTML también es ambigua, dependiendo del software que se utilice. Además, el lenguaje HTML no es fácilmente procesable, y sólo posee un uso, el de las páginas Web.

NOTA: El W3C es el organismo universal que regula el funcionamiento desde el punto de vista técnico de la World Wide Web. La principal función del W3C es la de crear estándares aceptados universalmente, para que los estándares que se creen no favorezcan a ningún fabricante, u organismo en particular. Otra de sus obligaciones es la de mantener al día las tecnologías de la WWW para que éstas respondan a las necesidades crecientes de los usuarios de la WWW. Entre otros, el W3C controla los estándares de html y xml. El W3C fue constituido en 1994 con el objetivo del desarrollo de protocolos comunes para la evolución de Internet. El W3C es un consorcio de industrias internacionales y organismos, como el mit (EEUU), inria (Francia) y keio university (Japón). El W3C cuenta con el apoyo oficial de darpa (EEUU) y de la Comisión Europea.

Llegados a este punto, el W3C decide poner manos a la obra y comienza a plantearse un nuevo lenguaje, que ante todo, y como hemos apuntado anteriormente, fuera muy escalable. Para ello, en un primer momento el W3C pensó directamente en el lenguaje SGML, que poseía todas las características técnicas para garantizar los objetivos que el W3C perseguía con el nuevo lenguaje, pero SGML era un lenguaje bastante complicado, y precisamente lo que había hecho popular al lenguaje HTML era la sencillez de su uso. Para ello, se marcaron un objetivo de crear un nuevo lenguaje de marcado, que pudiese ser tan escalable como SGML, pero que conservase la sencillez como base del mismo para que éste se hiciese popular.

XML nace entonces (1996) como una parte de SGML, y es adoptado como estándar en Febrero de 1998 por el W3C.

XML son las siglas de eXtended Markup Language o lenguaje de marcado extendido.

XML

Aunque en su nombre se hace referencia a él, seguramente estamos confundidos acerca de si XML es un lenguaje de Marcado. A simple vista, podríamos decir que sí, ya que XML está basado en un lenguaje de Marcado SGML y además tiene un primo que es un conocido lenguaje de Marcado HTML.

XML debería ser también un lenguaje de Marcado, sin embargo, no lo es, XML es más un lenguaje de lenguajes de marcado, o más concretamente un meta-lenguaje de marcado. Para explicar esto en palabras llanas, podríamos decir que a partir de XML podemos crear nuestro propio lenguaje de Marcado. Además, aunque está basado en SGML está adaptado para su uso en Internet, y se ha simplificado de cara a su sencillez de uso y aprendizaje.

La diferencia con un lenguaje de marcado es que su principal función es la de la descripción del contenido de un documento, mientras que XML se encarga del formato de ese contenido. XML, además, puede ser ejecutado desde cualquier sistema operativo o plataforma. Describe información, por ello, es ideal para el intercambio de datos en la red o fuera de ella, por ejemplo entre aplicaciones, éste es el caso de la suite de Microsoft Office que utiliza XML para intercambiar datos entre las distintas aplicaciones, como por ejemplo entre Word, Excel y Access, así como la transformación de documentos entre distintos formatos, por ejemplo de RTF (Rich Text Format) a HTML.

> **NOTA:** Microsoft ha sido y es desde siempre un acérrimo defensor de XML, tanto es así que podemos encontrar mucha información acerca de XML en su Web site:
>
> www.microsoft.com.
>
> Además, el primer navegador capaz de soportar XML fue Exlorer 4 de Microsoft, que de esta manera se adelantaba a su más directo competidor, Netscape.

XML es suficientemente potente para ser usado no sólo para describir documentos, sino también para describir Meta datos (Conjunto de información que describe a otra).

Información

XML nació para la red, pero la red no constituye la única funcionalidad de XML. Borland utiliza en sus herramientas de programación DELPHI y C++ BUILDER, así como en KYLIX, XML como lenguaje de definición de tablas de bases de datos, además de la información contenida en ellas.

Los documentos XML son procesados mediante aplicaciones que analizan XML (analizadores o Parsers). Estos analizadores generan una salida basada en los contenidos del documento y en la marca que el documento XML utiliza para describir los contenidos. XML a diferencia de su primo el HTML, separa drásticamente el contenido de la visualización, aunque esto también es posible ahora con HTML, utilizando hojas de estilo (CSS Cascade Style Sheets, hojas de estilo en cascada).

XML versus HTML

XML no sustituirá a HTML, más bien XML complementará a HTML, permitiendo de esta manera poder realizar aplicaciones para la red más robustas y escalables. Esto no quiere decir que XML hipotéticamente podría sustituir a HTML en un futuro, incluso el W3C ha definido el modelo de objeto de documento (DOM Document Object Model) de HTML en el que están basados los navegadores de acuerdo al lenguaje XML (con estructura XML), pero la implantación de DOM HTML en los navegadores es opcional.

Pero, sin duda, el mayor freno que tiene la sustitución de HTML por XML está basado en el éxito que HTML tiene y ha tenido, que hace prácticamente imposible y económicamente inviable la migración de millones y millones de páginas WEB, que actualmente están en la red, además HTML seguirá siendo un lenguaje fácil y asequible para todas aquellas personas que desean publicar páginas web sencillas, y que no justifican la necesidad de aprender otro lenguaje más potente, como XML.

Soporte XML

En un futuro no muy lejano los navegadores soportarán nativamente XML, de hecho, a día de hoy Microsoft Explorer y Netscape lo soportan. Pero, la forma de interpretarlo en el lado del cliente puede ser distinta dependiendo del navegador, es por ello que hoy en día la utilización de XML en la red se restringe a su uso de lado del servidor, y a corto plazo es donde tiene un mayor éxito, y lo tendrá sin duda pero, no olvidemos que día a día se encuentran nuevos usos y funcionalidades para el lenguaje XML, por lo que seguramente en un futuro próximo podremos encontrar XML por todas partes.

En XML se separa, como se dijo anteriormente, la presentación del contenido, hablando en este caso de la creación de documentos para la red. Por ejemplo, de un mismo documento XML podríamos crear diferentes interpretaciones de visualización, como por ejemplo la ya conocida de HTML, WML (Wireless Markup Language, lenguaje de marcado, que utiliza la tecnología WAP para las telecomunicaciones móviles), RTF o PDF (formato utilizado para documentos por Adobe Acrobat), etc. y en todos ellos podríamos ver la misma información.

Además, XML está diseñado para ser utilizado con cualquier tipo de conjunto de caracteres y, por consiguiente, cualquier tipo de lenguaje o idioma. XML se presenta como el formato ideal para el intercambio de documentos, a través de la red o a través de cualquier otro método. Algunas de las utilidades que se ven beneficiadas por el uso de XML son, entre otras, las transacciones entre empresas en la red, las técnicas de extracción de datos, como DATAMINING o DATAMARK y, por supuesto, la creación de nuestros propios lenguajes de marcado para dotar a nuestras aplicaciones de herramientas que permitan el intercambio de datos entre distintas aplicaciones.

En XML las reglas de composición de un documento son estrictas, no pueden ser dejadas al libre albedrío tal y como sucede en HTML, más adelante ampliaremos esta característica.

Partes que forman un documento XML

Los documentos XML están formados por varias partes, que son:

El elemento

Es la parte fundamental de la estructura del documento XML. Se identifica mediante el nombre del elemento. Se corresponde con la etiqueta (tag) que podemos encontrar en HTML

HTML	**XML**
<HEAD>	<VEHICULO>

En el ejemplo anterior, el nombre del elemento se corresponde con VEHÍCULO.
Sin embargo, en HTML es opcional la etiqueta de cierre, y no pasa nada si se omite, pero en XML es completamente imprescindible la etiqueta de fin.

HTML	**XML**
<HEAD>	<VEHICULO> </VEHICULO>

El contenido del elemento

Es la parte del documento XML que contiene, valga la redundancia, el valor que representa el elemento.

<VEHICULO> **M-9999-ZZ** </VEHICULO>

El atributo

Forma parte del elemento, su función es la de aportar mayor información acerca del elemento.

<VEHICULO **MARCA=”OPEL”**> M-9999-ZZ </VEHICULO>

Un elemento puede contener más de un atributo, o ninguno.

<VEHICULO **MARCA=”OPEL” MODELO=”CORSA”**>
M-9999-ZZ
</VEHICULO>

Dentro del atributo encontramos dos partes, el nombre del atributo y el valor del atributo. En los ejemplos anteriores:

El nombre del atributo sería

MARCA
MODELO

Y el valor de los mismos

OPEL
CORSA

Reglas de composición de documentos XML

Las reglas de composición de documentos XML son muy sencillas, pero inflexibles, es decir, no puede saltarse ninguna de estas reglas.

Seguidamente, veremos cada una de ellas:

Estructura jerárquica de los elementos

Los elementos deben seguir una estructura jerárquica o en árbol, es decir, cada elemento debe estar contenido dentro de otro elemento.

La excepción a esta regla es el elemento raíz, del que parten todos los demás elementos.

```
<VEHICULO>

</VEHICULO>
```

Los elementos además deben estar anidados.

```
<VEHICULO>
        <PROPIETARIO>
                <NOMBRE>

                </NOMBRE>
        </PROPIETARIO>
</VEHICULO>
```

Orden de los elementos

Los elementos no pueden estar superpuestos

INCORRECTO

```
<VEHICULO>
        <PROPIETARIO>
</VEHICULO>
        </PROPIETARIO>
```

CORRECTO

```
<VEHICULO>
        <PROPIETARIO>
                </PROPIETARIO>
</VEHICULO>
```

Los elementos, obligatoriamente, deben estar "cerrados"

INCORRECTO

```
<VEHICULO>
  <PROPIETARIO>
</VEHICULO>
```

CORRECTO

```
<VEHICULO>
        <PROPIETARIO>
</PROPIETARIO>
</VEHICULO>
```

Sólo puede existir un elemento raíz del que parten los demás

INCORRECTO

```
<VEHICULO>

        </VEHICULO>
        <PROPIETARIO>

</PROPIETARIO>
```

CORRECTO

```
<VEHICULO>

            <PROPIETARIO>
            </PROPIETARIO>

    </VEHICULO>
```

Nombre de los elementos

El nombre de los elementos debe empezar siempre con una letra, y puede ser seguido por otros caracteres alfanuméricos. El nombre de un elemento nunca puede comenzar por XML, xml, xML o cualquier variación de éste.

XML es case sensitive

Debemos recordar siempre que XML es Case Sensitive (distingue entre mayúsculas y minúsculas), por lo que para el analizador el elemento <Elemento> no será igual que <elemento>. Veamos el siguiente ejemplo:

INCORRECTO

```
<Elemento>

        <elemento2>

        </elemento2>

</elemento>
```

CORRECTO

```
<elemento>

        <elemento2>

        </elemento2>

</elemento>
```

Uso de espacios y retorno de carro

El espacio y el retorno de carro sólo son tenidos en cuenta si aparecen en el valor de un atributo o, como en HTML si se indica su significado.

Marcado y datos

Las marcas se corresponden con las que comienzan por < y terminan con >. Evidentemente, los elementos que anteriormente hemos visto son marcas o "mark-up".
El texto que encontramos entre las marcas o etiquetas son los datos del documento propiamente dicho.

En las referencias de entidad se utiliza el carácter ampersand (&), y finalmente el punto y coma (;).

Prólogo

El prólogo es opcional, pero generalmente comienza especificando que se trata de un documento XML y la versión de XML que contiene. También, a veces, encontramos referencia a la codificación de caracteres con la que está realizada el documento.

```
<?xml version = "1.0"?>
        ó
<?xml version  = "1.0" encoding = "UTF-7"?>
```

En la segunda línea del prólogo, generalmente encontramos la referencia a los documentos de definición (DTD), que más adelante veremos.

```
<!DOCTYPE documento SYSTEM "documento.dtd">
```

Entidades predefinidas

Son aquellas que no son interpretadas como marcado por el analizador de XML. Es decir, si queremos utilizar alguno de estos caracteres no podemos hacerlo directamente, y, si lo hiciésemos, el analizador lo interpretaría como marca y no como valor. Son cinco:

CARÁCTER	ENTIDAD
&	&Amp;
>	>
<	<
'	'
"	"

Documentos XML bien formados (well formed)

Los documentos XML “bien formados” son aquellos que cumplen las reglas arriba descritas. Para que un documento sea un documento XML tiene obligatoriamente que ser un documento bien formado.

Además de las reglas arriba escritas existen caracteres que no pueden ser utilizados para formar parte del nombre de elementos o atributos. Tampoco se pueden utilizar acentos, ni otros elementos de puntuación.

Que un documento XML esté bien formado no significa que sea un documento válido, tal como veremos más adelante.

Un ejemplo xml

En el siguiente ejemplo se muestra un documento XML.

```
<?xml: version=”1.0”?>
<!—Ejemplo 1—>
<libro>
<titulo>Un titulo cualquiera</titulo>
        <capitulos>
                <capitulo>
                <titulo> Capitulo 1 </titulo>
                <párrafo número = 1>
                                Un párrafo
                        </párrafo>
                </capitulo>
                <capitulo>
            <titulo> Capitulo 2 </titulo>
                </capitulo>
        </capitulos>
</libro>
```

Comencemos a describir el documento por la primera línea.

```
<?xml: version=”1.0”?>
```

Todos los documentos XML deben comenzar con una línea como la anterior, donde especificamos que el documento es un documento XML y que se corresponde con la versión 1.0.
Esta línea es necesaria en los documentos XML y sirve para que los navegadores o analizadores XML detecten que el fichero que están leyendo es un documento XML, y que se corresponden con la versión 1.0 de las especificaciones propuestas por el consorcio w3c.

```
<!—Ejemplo 1—>
```

En el caso de la segunda línea, si conoce algo de HTML reconocerá fácilmente que se trata de un comentario, todos los comentarios en XML deben comenzar con <!— y finalizar con —>.

```
<!— Esto es un comentario XML —>
```

Si el texto que forma parte del comentario tiene más de una línea no supone ningún problema, siempre que empecemos y terminemos el comentario con los caracteres de comienzo y fin, respectivamente.

```
<!— Esto es un comentario XML que
        ocupa más de una línea —>
```

Se pueden utilizar los comentarios en cualquier parte del documento, excepto dentro de las etiquetas, declaraciones o dentro de otros comentarios.

```
<libro>
```

Es el primer elemento que encontramos en el documento XML, por esto se corresponde con el elemento raíz del documento, lo que quiere decir que todos los demás elementos estarán contenidos en el elemento raíz “<libro>”.

```
<titulo>Un título cualquiera</titulo>
```

Un nuevo elemento que parte directamente de la raíz del documento, el elemento <titulo> título tiene como valor “Un título cualquiera” y, finalmente, está cerrado “</titulo>”, lo que quiere decir que el elemento título no tiene ningún otro subelemento o elemento que dependa de él.

```
<capitulos>
```

Un nuevo elemento que parte directamente de la raíz del documento.

<capitulo>

Un elemento que depende del elemento <capitulos>

<titulo> Capítulo 1 </titulo>

<titulo> es un nuevo elemento, pero se diferencia del elemento <titulo>, que vimos anteriormente, en que éste no depende de la raíz, sino que en este caso depende del elemento <capitulo>, se trata de dos elementos totalmente diferentes. “Capítulo 1” es el valor del elemento y </titulo> se corresponde con el cierre del elemento.

```
<párrafo número = 1>
        Un párrafo
</párrafo>
```

<párrafo> es un nuevo elemento, que al igual que el anterior parte o está incluido dentro del elemento <capitulo>, también se diferencia del anterior en que este elemento posee un atributo “número” y se le asigna un valor a ese atributo “1”; por lo demás, es igual al anterior, el elemento tiene un valor “Un párrafo”, y finalmente encontramos una etiqueta de cierre </parrafo>.

</capitulo>

Cerramos el elemento <capitulo> con la etiqueta de cierre </capitulo>.

<capitulo>

Abrimos nuevamente un elemento <parrafo>, como en el caso del anterior elemento párrafo éste se encuentra dependiente de la raíz, y por tanto se encuentra al mismo nivel que el anterior, por lo que el analizador lo detectará como el segundo valor correspondiente al elemento <capitulo>. Este caso es completamente diferente del que veíamos anteriormente con <titulo>, ya que como veíamos antes el elemento título no se encontraba en el mismo nivel que el otro elemento <titulo>, que dependía del elemento raíz del documento.

<titulo> Capítulo 2 </titulo>

Segundo valor de la etiqueta <titulo> dependiente de <capitulo>.

```
<capitulo>
```

Abrimos un nuevo elemento <capitulo>.

```
</capitulos>
```

Cerramos el elemento <capitulos>

```
</libro>
```

Cerramos el elemento raíz. Al cerrar este elemento (el raíz) el documento se da por finalizado, a partir de este momento no podemos abrir o cerrar ningún otro elemento, sí podríamos, por supuesto, incluir comentarios; en caso de que abriésemos un nuevo elemento no nos daría como resultado un documento bien formado, y como consecuencia un error.

Este otro ejemplo sería incorrecto (no estaría bien formado).

```
<?xml: version=”1.0”?>
<!—Ejemplo 1—>
<libro>
<titulo>Un titulo cualquiera</titulo>
        <capitulos>
                <capitulo>
                <titulo> Capitulo 1 </titulo>
                <párrafo número = 1>
                                Un párrafo
                        </párrafo>
                </capitulo>
                <capitulo>
                        <titulo> Capitulo 2 </titulo>
                </capitulo>
        </capitulos>
</libro>
<autor>
        el autor del libro
</autor>
```

La incorrección de este ejemplo está fundamentada en que no existe elemento raíz. Para que el ejemplo estuviese bien formado debería escribirse de esta otra manera:

```
<?xml: version=”1.0”?>
<!—Ejemplo 1—>
<libro>
        <titulo>Un titulo cualquiera</titulo>
        <capitulos>
                <capitulo>
                <titulo> Capitulo 1 </titulo>
                <párrafo número = 1>
                                Un párrafo
                        </párrafo>
                </capitulo>
                <capitulo>
                <titulo> Capitulo 2 </titulo>
                </capitulo>
        </capitulos>
        <autor>
            el autor del libro
        </autor>
</libro>
```

Atributos

Se puede dar el caso de que nos decidamos por elementos que no tengan valor ni subelementos, pero que asignen distintos valores a atributos que pertenezcan al elemento.

```
<elemento atributo=”valor del atributo”> </elemento>
```

Aunque un elemento no contenga ningún valor, tal como hemos visto, tiene obligatoriamente que ser cerrado como cualquier otro elemento, pero en un caso como éste podemos abrir la etiqueta del elemento y cerrar el elemento dentro de la misma etiqueta de esta otra manera:

```
<elemento atributo=”valor del atributo”/>
```

Recordemos que el valor de los atributos siempre debe encerrase entre comillas simples (‘) o comillas dobles (“).

Esto es correcto:

```
<elemento atributo = “valor del atributo”/>
```

Esto no es correcto

```
<elemento atributo = valor del atributo />
```

Secciones CDATA

Hay ocasiones en las que el autor del documento XML debe incorporar texto como valor y que este texto contenga caracteres no permitidos, ya que podrían ser confundidos con marcado del documento XML tal como vimos en las entidades predefinidas (&,>,<,’,”). Para que el analizador XML tome el texto como “simple texto” y no intente analizarlo, para ello encerramos el texto en lo que llamamos una sección CDATA (Character Data).

En este ejemplo, el contenido de la etiqueta <etiqueta1> es código HTML, que como sabemos también utiliza los caracteres < y > para señalar las etiquetas o tags HTML.

Sin sección CDATA:

```
<etiqueta1>
        &lt;HTML&gt;
                &lt;HEAD&gt;
                        &lt;TITLE&gt;
                        Esta es la Cabecera del Documento HTML
                        &lt;/TITLE&gt;
                &lt;/HEAD&gt;
</etiqueta1>
```

Con Sección CDATA:

```
<etiqueta1>
                <![CDATA[
                   <HTML>
                                <HEAD>
                                        <TITLE>
                        Esta es la Cabecera del Documento HTML
                                        </TITLE>
                                </HEAD>
                ]]>
</etiqueta1>
```

Evidentemente, existe una excepción que se corresponde con el final de la sección CDATA]]>, al utilizarse estos caracteres para señalar el final de la sección CDATA no pueden ser utilizados dentro de la sección CDATA. Si los utilizáramos el analizador de XML los confundiría con el final de la sección.

Definición de documentos y documentos válidos

No es suficiente para que un documento sea un documento XML que esté bien formado, además el documento tiene que ser válido. Pero, ¿qué significa que un documento sea válido? Si tuviésemos que hacer una definición, diríamos que un documento XML es válido cuando está bien formado, y además cumple las definiciones propuestas en su definición de documento o DTD (Document Type Definition).

La DTD de un documento XML no es ni más ni menos que la definición de cada uno de los elementos y atributos que forman parte del documento XML. Como veremos más adelante esta DTD puede estar incluida (a nivel físico) en un fichero aparte con la extensión DTD o en el mismo fichero del XML. Realmente, lo que hacemos al crear la DTD es definir las etiquetas y atributos de nuestro lenguaje de marcado, especificando qué elementos forman parte de él, qué atributos, de qué tipo son, etc...

Si nos paramos a pensar, hay un paralelismo entre esto y el código HTML, ya que el HTML procede del SGML, pero tiene sus etiquetas específicas que han sido propuestas en una “DTD”, en la DTD del lenguaje HTML. Así las etiquetas <HTML> o <HEAD> o <B> están incluidas dentro de esa DTD del lenguaje HTML.
La DTD no sólo se limita a presentar o definir los elementos y atributos del documento XML, también puede definir reglas de combinación de éstos.
Evidentemente, podemos crear documentos XML sin DTD, y probablemente funcionen correctamente, pero en este caso, como dijimos anteriormente, no son documentos válidos, en este caso serían tan sólo documentos bien formados.

Pasamos ahora a ver un pequeño ejemplo:

```
<?xml versión = “1.0”?>

<!DOCTYPE midocumento [
        <!ELEMENT midocumento (elemento1, elemento2)>
        <!ELEMENT elemento1 (#PCDATA)>
        <!ELEMENT elemento2 (#PCDATA)>
  ]>

<midocumento>

        <elemento1> contenido del elemento1 </elemento1>
        <elemento2> contenido del elemento2 </elemento2>

</midocumento>
```

En este ejemplo, podemos ver un documento XML con su DTD definida incluida en el propio documento. La DTD define la estructura del documento “midocumento”, que coincide con el elemento raíz del documento:

```
<!DOCTYPE midocumento [
```

Después de declarar, el documento pasa a definir los elementos del mismo:

```
<!ELEMENT midocumento (elemento1, elemento2)>
```

Define el elemento mi documento, y señala que está formado por dos subelementos, elemento1 y elemento2.

```
<!ELEMENT elemento1 (#PCHAR)>
```

Define el elemento 1, y como no está formado por ningún otro subelemento, directamente señala el tipo de información que contendrá el elemento, en este caso caracteres (#PCHAR).

```
<!ELEMENT elemento2 (#PCHAR)>
```

Define el elemento2 tal como lo hemos hecho con el elemento1.

```
]>
```

Cierra la declaración de tipo de documento (DTD).

En el caso en el que la DTD no esté incluida en el propio documento XML, debemos hacer referencia a ella desde el documento XML. Quedaría de esta manera, tomando el ejemplo anterior:

```
<?xml versión = “1.0”?>

<!DOCTYPE midocumento SYSTEM “c:\midocumento.dtd”>

<midocumento>

        <elemento1> contenido del elemento1 </elemento1>
        <elemento2> contenido del elemento2 </elemento2>

</midocumento>
```

LA DTD externa tendría este aspecto (fichero c:\midocumento.dtd):

```
<!ELEMENT midocumento (elemento1, elemento2)>
<!ELEMENT elemento1 (#PCDATA)>
<!ELEMENT elemento2 (#PCDATA)>
```

Declaración de elementos en la dtd

Como los elementos son la base de los documentos XML, éstos deben estar definidos de manera correcta para que el documento XML pueda ser válido. Como hemos visto anteriormente en el ejemplo, los elementos se declaran con la etiqueta <!ELEMENT.
Al declarar los elementos debemos declarar también su contenido. Tal como vimos en el ejemplo anterior si el elemento contiene otros elementos o subelementos, éstos deben estar definidos en la declaración del elemento, de esta manera, definimos la estructura en árbol del documento XML.

```
<!ELEMENT midocumento (elemento1, elemento2)>
```

En el ejemplo, declaramos un elemento midocumento y definimos su contenido, en este caso el contenido de “midocumento” es otro elemento el “elemento1” y el “elemento2”.
Según esta declaración del elemento “midocumento”, el siguiente documento XML sería válido:

```
<midocumento>

    <elemento1> contenido del elemento1 </elemento1>
    <elemento2> contenido del elemento2 </elemento2>

</midocumento>
```

Sin embargo, este otro no lo sería:

```
<midocumento>

    <elemento1> contenido del elemento1 </elemento1>
    <elemento2> contenido del elemento2 </elemento2>
    <elemento3> contenido del elemento3 </elemento3>

</midocumento>
```

Para que este documento XML fuese válido tendría que tener una declaración como esta:

<!ELEMENT midocumento (elemento1, elemento2, elemento3)>

Al hacer la declaración del elemento “midocumento” no hemos definido los elementos “elemento1” y “elemento2”, sólo hemos dicho que el contenido de “midocumento” está formado por “elemento1” y “elemento2”, es por esto que tenemos que declarar también los elementos que forman parte de “midocumento”:

```
<!ELEMENT elemento1 (#PCDATA)>
<!ELEMENT elemento2 (#PCDATA)>
```

Esta vez el contenido de los elementos “elemento1” y “elemento2” no son elementos, en este caso el contenido es una cadena de texto que define el tipo (#PCDATA).

Al definir el contenido de un elemento este puede ser de varios tipos:

Empty

Es un tipo de contenido para el que el elemento puede no tener contenido, generalmente este tipo es utilizado para declarar atributos.

```
<!ELEMENT elementovacio EMPTY>
```

Any

Con este tipo, señalamos que nuestro elemento puede contener cualquier tipo de información, generalmente no se usa, ya que es conveniente poseer una perfecta definición de nuestro documento.

```
<!ELEMENT cualquiera ANY>
```

Mixed

El tipo mixed señala que el elemento puede contener caracteres de texto, pero también elementos.

```
<!ELEMENT elementoN  (#PCDATA | elementoN1)*>
```

Element

El elemento en cuestión sólo puede contener elementos:

```
<!ELEMENT midocumento (elemento1, elemento2)>
```

Modelos de contenido en la declaración de elementos

El modelo de contenido dentro de la declaración de elementos, va a definir la forma en la que debemos encontrar los elementos dentro de nuestro documento XML.
Con el modelo de contenido definimos el orden de los elementos, la obligatoriedad o no de ellos, elementos opcionales, etc....
Comenzamos con el más básico:

```
<!ELEMENT midocumento (elemento1)>
```

El ejemplo muestra la declaración del elemento “midocumento”, y señala que “midocumento” debe incluir el elemento “elemento1” en su interior.

```
<!ELEMENT midocumento (elemento1, elemento2)>
```

Esta otra declaración señala que el elemento “midocumento” está formado por elemento1, seguido de elemento2, lo que denota el orden de los elementos. El orden de los elementos se define por medio de la “,”.

```
(elemento1, elemento2, elemento3)
```

“elemento1”, seguido de “elemento2” y “elemento2” seguido de “elemento3”.

```
<!ELEMENT midocumento (elemento1 | elemento2)>
```

En este caso, hemos sustituido la “,” por “|”, con esta declaración estamos indicando opción. El elemento “midocumento” puede estar formado por “elemento1” o por “elemento2”.

También se pueden combinar:

```
<!ELEMENT midocumento (elementoN, (elemento1 | elemento2))>
```

El elemento “midocumento” está formado por el elemento “elementoN”, seguido por un “elemento1” o un “elemento2”. Las combinaciones también pueden ser agrupadas:

```
<!ELEMENT
    midocumento
    (elementoN, (elemento1|(elemento2,elemento2-1)))>
```

El elemento “midocumento” está formado por un elemento “elementoN”, seguido por un “elemento1” o, un “elemento2”, seguido por un “elemento2-1”. También, mediante el modelo de contenido podemos hacer referencia a la frecuencia con la que deben aparecer los elementos.

```
<!ELEMENT  midocumento (elemento1?)>
```

Esta declaración utiliza símbolo “?” para indicar que el elemento es opcional, el elemento “elemento1” se puede repetir 0 ó 1 veces.

```
<!ELEMENT  midocumento (elemento1+)>
```

Esta otra utiliza el símbolo “+” para indicar que el elemento es obligatorio, y que se puede repetir más de una vez. El elemento “elemento1” es obligatorio y puede ser repetido tantas veces como queramos.

```
<!ELEMENT  midocumento (elemento1*)>
```

El símbolo “*” indica que el elemento es opcional y repetible, el elemento “elemento1” puede existir una o más veces o no existir. Como en los ejemplos anteriores, podemos hacer distintas combinaciones con la frecuencia de los elementos, como en el siguiente ejemplo:

```
<!ELEMENT
midocumento
(elementoN?,(elemento1|(elemento2,elemento2-1))*)>
```

Esta declaración indica que “midocumento” puede estar formado o no por “elementoN”, seguido de 0 o “n” combinaciones de “elemento1” o “elemento2”, seguido de “elemento2-1”.

Declaración de atributos

La declaración de los atributos se diferencia de la declaración de los elementos, sobre todo, en que los atributos no pueden contener a su vez “sub-atributos”.

Como vimos en la unidad en la que presentábamos las diferentes partes de un documento XML, los atributos se usan para complementar o añadir información a un elemento. Generalmente, la información que se incluye en los atributos suele ser una información corta y que no está estructurada.

Otra de las reglas a la hora de declarar Atributos es que los atributos sólo pueden ser declarados una sola vez, y en el orden que queramos.

Las declaraciones de los atributos en la DTD se realizan con <!ATTLIST, seguido por el nombre del elemento al que pertenece el atributo, el nombre del atributo y, por último, el tipo del atributo y el valor por defecto del mismo.

Veamos un ejemplo:

```
<?xml version=”1.0”?>

<monitores>

    <monitor marca=”Fujitsu”> 23456723 </monitor>

    <monitor marca=”SONY”> 15556723 </monitor>

</monitores>
```

La DTD de este ejemplo será:

```
<!ELEMENT monitores (monitor)+>
<!ELEMENT monitor (#PCDATA)>
<!ATTLIST monitor marca CDATA #REQUIRED>
```

En el ejemplo, el atributo monitor puede contener texto, esto es lo que especificamos con CDATA, y además especificamos que no tiene valor por defecto y que es obligatorio especificar el valor del atributo.
De otra manera, podríamos haber incluido un valor por defecto en el atributo marca.

```
<!ATTLIST monitor marca SONY>
```

También podemos especificar varios valores alternativos de esta otra forma:

```
<!ATTLIST monitor marca (SONY | FUJITSU)>
```

Con esta declaración, estamos especificando que el valor del atributo marca del elemento monitor será o bien SONY o bien FUJITSU, pero no otro. Es posible añadir a la declaración anterior un valor por defecto:

```
<!ATTLIST monitor marca (SONY | FUJITSU) SONY>
```

En este caso como en el anterior, el valor del atributo “marca” que pertenece al elemento “monitor” puede ser o bien SONY o bien FUJITSU, y por defecto es SONY, si no especificamos el valor del atributo. Se puede dar el caso de que no especifiquemos el valor del atributo, pero no queramos que el atributo adopte el valor por defecto, para ello utilizamos #IMPLIED.

```
<!ATTLIST monitor marca CDATA #IMPLIED>
```

Tipos de atributos

Podemos definir atributos con diferentes tipos, para así disponer de un perfecta esquematización de la información que nuestro documento XML contiene. Los tipos de atributos son:

CDATA
NMTOKEN
ENUMERADOS
ID
IDREF

Los atributos CDATA más comunes, son aquellos que pueden contener cualquier cosa (exceptuando los caracteres y palabras reservadas de XML). Son los que se usan con mayor asiduidad dentro de los ficheros XML para contener valores de tipo alfanumérico.

```
<!ATTLIST monitor marca CDATA #REQUIRED>
```

Los atributos NMTOKEN son parecidos a los CDATA, pero sólo pueden contener letras, números, puntos, guiones, subrayados, y los dos puntos.

```
<|ATTLIST monitor marca NMTOKEN #REQUIRED>
```

Los atributos ENUMERADOS son aquellos en los que definimos los posibles valores que el atributo puede tener, tal como vimos en un ejemplo anterior.

```
<!ATTLIST monitor marca (SONY | FUJITSU) SONY>
```

```
<!ATTLIST monitor marca (SONY | FUJITSU | PHILIPS)>
```

Los atributos ID son aquellos que tienen como función servir de identificador de referencia, para que sea llamado desde otro elemento que posea un atributo de tipo IDREF y, por lo tanto, debe contener un valor único.

```
<!ATTLIST monitor marca ID #REQUIRED>
```

El atributo IDREF es el atributo con el que definimos que el valor del atributo es utilizado para localizar un elemento con un atributo de tipo IDREF.

```
<!ATTLIST monitor marca IDREF #REQUIRED>
```

Veamos un ejemplo del uso conjunto de atributos de tipo ID e IDREF:

```
<!ELEMENT monitor EMPTY>

<!ATTLIST monitor marca ID #REQUIRED>

<!ELEMENT repuesto EMPTY>

<!ATTLIST repuesto marca IDREF #REQUIRED>
```

En el ejemplo vemos cómo el atributo marca del elemento repuesto tiene un valor para hacer referencia al atributo marca del elemento monitor.

Declaración de entidades

Por medio de la declaración de entidades obtenemos la posibilidad de que el analizador XML no analice determinados contenidos, siguiendo las reglas sintácticas del lenguaje XML. La declaración de entidades se utiliza a menudo, por ejemplo, para declarar ficheros o direcciones de páginas web, no puede ser mas que una abreviatura utilizada para hacer referencia a determinado valor, al hacer referencia dentro del documento XML a la entidad, entonces el analizador la sustituye por el valor declarado en la DTD.
En el caso de que un documento haga referencia a un fichero de imágenes externo, la localización de la imagen puede ser declarada como entidad y hacer referencia sólo al nombre dado a la entidad, para que el analizador de XML sustituya el nombre de la entidad por el valor de ésta, en este caso la localización exacta del fichero de imágenes.

Las entidades en la DTD se declaran con <!ENTITY
Las entidades se agrupan en:

INTERNAS / EXTERNAS
ANALIZADAS / NO ANALIZADAS
GENERALES / PARÁMETRO

Entidades generales internas

Son abreviaturas que se usan dentro del documento XML, y son siempre analizadas, es decir, por medio de la declaración de la entidad declaramos una abreviatura y por medio del nombre de la entidad hacemos referencia al contenido de la entidad, el analizador entonces sustituye el nombre de la entidad o abreviatura con el valor de la entidad, y una vez sustituida por el analizador, lo analiza sintácticamente como cualquier otro texto.

Veamos un ejemplo completo:

```
<?xml version=”1.0”?>

<!DOCTYPE [

        <!ELEMENT monitores (monitor)*>

        <!ELEMENT monitor (#PCDATA)>

        <!ATTLIST monitor marca CDATA #REQUIRED>

        <!ENTITY VGA “Visual Graphics Adapter”>

        ]>

<monitores>

        <monitor marca=”Fujitsu”>
                23456723 &VGA
        </monitor>

</monitores>
```

Este ejemplo sería interpretado por el analizador como:

```
<?xml version=”1.0”?>

<!DOCTYPE [

        <!ELEMENT monitores (monitor)*>

        <!ELEMENT monitor (#PCDATA)>

        <!ATTLIST monitor marca CDATA #REQUIRED>

        ]>

<monitores>

        <monitor marca=”Fujitsu”>
                23456723  Visual Graphics Adapter
        </monitor>

</monitores>
```

Entidades generales externas analizadas

Las entidades externas generales analizadas funcionan de igual manera que las anteriores, con la excepción de que obtienen el valor de la entidad fuera del documento.

Si hacemos referencia desde nuestro documento XML a una entidad externa analizada, el analizador extraerá el contenido del fichero o link al que hacemos referencia por medio de la entidad, sustituirá al nombre de la entidad por el contenido del fichero externo y lo analizará.

```
<?xml version=”1.0”?>

<!DOCTYPE [

        <!ELEMENT monitores (monitor)*>

        <!ELEMENT monitor (#PCDATA)>

        <!ATTLIST monitor marca CDATA #REQUIRED>
```

```
        <!ENTITY MANUAL  SYSTEM  “c:\manual.doc”>

        ]>

<monitores>

        <monitor marca=”Fujitsu”>
                23456723  &MANUAL
        </monitor>

</monitores>
```

En la declaración de la entidad podemos observar la utilización de la palabra reservada SYSTEM, que es la que indica que el contenido de la entidad va ha ser obtenido fuera del documento.
Después de la palabra clave SYSTEM debemos hacer referencia al documento por medio de una URI (Universal Resource Identifier), que puede hacer referencia a un documento dentro de la máquina, o bien, entre otras, a través de la red.

```
<!ENTITY MANUAL  SYSTEM  “http//:www.miwebsite.com/manual.doc”>
```

Tal como hemos visto en los tipos anteriores de entidades, en este caso, después de ser sustituido el valor de la entidad, éste es analizado por el analizador XML de manera transparente, luego es ANALIZADA.

Entidades no analizadas

Imaginemos el caso en el que la declaración de la entidad haga referencia a un fichero externo, cuyo contenido sea una imagen o un sonido, en este caso, el analizador de XML intentará analizarlo de manera transparente, tal como hemos visto en los tipos anteriores, pero la diferencia es que si el tipo de recurso al que hace referencia la entidad no puede ser analizado, entonces el analizador no intentará analizarlo.
La forma de declarar entidades no analizadas es exactamente igual que para las analizadas.

```
<!ENTITY IMAGEN  SYSTEM  “http//:www.miwebsite.com/miimagen.gif”>
```

Entidades parámetro internas

Este tipo de entidades sólo se declaran para ser utilizadas exclusivamente dentro de la DTD, y no en el documento XML. Para hacer referencia a ellas no utilizamos el símbolo “&”, sino “%”, también utilizamos el símbolo “%” para declararlas.

```
<?xml version=”1.0”?>
<!DOCTYPE [
<!ELEMENT monitores (monitor)*>
<!ELEMENT monitor (#PCDATA)>
<!ATTLIST monitor marca CDATA #REQUIRED>
<!ENTITY % mielemento “<!ELEMENT elemento1 (#PCDATA)”>
%mielemento;
]>
```

En este ejemplo, hemos declarado una entidad de tipo parámetro llamada “mielemento”, que se corresponde con la declaración de un elemento, por ello, cada vez que tengamos que declarar un elemento como “elemento1”, bastará con hacer referencia a la entidad mielemento para que sea sustituida por la declaración del elemento.
Sin utilizar la entidad el ejemplo anterior quedaría de la siguiente manera:

```
<!DOCTYPE [
        <!ELEMENT monitores (monitor)*>
        <!ELEMENT monitor (#PCDATA)>
        <!ATTLIST monitor marca CDATA #REQUIRED>
        <!ELEMENT elemento1 (#PCDATA)”>
]>
```

Entidades parámetro externas

De igual manera que los otros tipos de entidades, las entidades de tipo parámetro pueden ser externas, y para hacer referencia a ellas utilizaremos la palabra clave SYSTEM, como en la declaración de las otras entidades externas.

```
<?xml version=”1.0”?>
    <!DOCTYPE [
        <!ELEMENT monitores (monitor)*>
        <!ELEMENT monitor (#PCDATA)>
        <!ATTLIST monitor marca CDATA #REQUIRED>
        <!ENTITY % mielemento SYSTEM “c:\elemento1.dat”>
        %mielemento;
        ]>
```

Lenguaje de estilo extendido XSL

XSL es un lenguaje para realizar hojas de estilo. El conjunto del lenguaje XSL está formado por tres partes:

- XLST (XSL Transformations): un lenguaje para transformar documentos XML.
- XPATH (XML Path Language): un lenguaje de expresiones para acceder a las diferentes partes de un documento XML.
- Un vocabulario XML para el formateo semántico de expresiones. (XSL Formating Objects)

Una hoja de estilo XSL especifica las reglas de presentación de un determinado documento XML.

XPATH

El lenguaje XPATH es una sintaxis, que nos ayuda a construir cadenas de texto para localizar determinados recursos dentro de documentos XML.

Usamos el lenguaje XPATH cuando queremos hacer referencia a determinado contenido de un elemento en un documento XML. La base de la utilización del lenguaje XPATH se centra en la estructura en árbol de los elementos de un documento XML, y cómo estos elementos mantienen entre sí una relación jerárquica. Cada elemento representa una estructura jerárquica de elementos por sí mismo.

XPATH no es ni más ni menos que un conjunto de reglas semánticas para representar la estructura jerárquica de un determinado elemento. XPATH puede recordarnos a la manera de moverse por los directorios que se utiliza en UNÍX o MSDOS.

Veamos un ejemplo:

```
Select=”monitor”
```

En este ejemplo se indica al analizador que seleccione el elemento, o elementos “monitor”, pero tenemos que tener en cuenta que sólo seleccionará el elemento “monitor” que dependa directamente del elemento en el que estamos situados en este mismo momento.

En el siguiente documento:

```
<?xml version=”1.0”?>
        <!DOCTYPE [
                <!ELEMENT monitores (monitor)*>
                <!ELEMENT monitor (#PCDATA)>
                <!ATTLIST monitor marca CDATA #REQUIRED>
                ]>
        <monitores>
                <monitor marca=”Fujitsu”> 23456723  </monitor>
        </monitores>
```

Si estamos situados en el elemento raíz “monitores”, entonces sí encontrará el elemento “monitor”, pero si estamos situados en el elemento “monitor” no encontrará ningún elemento “monitor”, ya que el elemento monitor no posee ningún subelemento.

Como señalamos anteriormente el XPATH es un lenguaje que guarda un gran parecido con el URI (Universal Resource Identifier), así si queremos hacer referencia a un elemento contenido dentro de otro elemento utilizaremos la “/”, tal como lo hacemos en URI.

```
<?xml version=”1.0”?>

        <!DOCTYPE [

                        <!ELEMENT perif (monitores,impresoras)>
                        <!ELEMENT monitores (monitor)*>
                        <!ELEMENT monitor (#PCDATA)>
                        <!ATTLIST monitor marca CDATA #REQUIRED>
                        <!ELEMENT impresoras (impresora)*>
                        <!ELEMENT impresora (#PCDATA)>
                        <!ATTLIST impresora marca CDATA #REQUIRED>
                ]>
```

```
<perif>
        <monitores>
                <monitor marca=”Fujitsu”>
                        23456723
                </monitor>
        </monitores>
        <impresoras>
                <impresora marca=”Epson”>
                        2345674423
                </impresora>
        </impresoras>
</perif>
```

Así en el ejemplo mostrado, si estamos situados en la raíz del documento XML y queremos referirnos al contenido de impresora lo haremos de la siguiente manera:

```
Select=”impresoras/impresora”
```

Pero, XPATH además nos brinda la posibilidad de referirnos a determinada repetición de un elemento concreto de un documento XML. Así, si el documento XML fuera como el que sigue:

```
<?xml version=”1.0”?>

        <!DOCTYPE [

                <!ELEMENT perif (monitores+,impresoras+)>
                <!ELEMENT monitores (monitor)*>
                <!ELEMENT monitor (#PCDATA)>
                <!ATTLIST monitor marca CDATA #REQUIRED>
                <!ELEMENT impresoras (impresora)*>
                <!ELEMENT impresora (#PCDATA)>
                <!ATTLIST impresora marca CDATA #REQUIRED>
        ]>
```

```
<perif>
        <monitores>
                <monitor marca=”Fujitsu”>
                        23456723
                </monitor>
        </monitores>
        <impresoras>
                <impresora marca=”Epson”>
                        2345674423
                </impresora>
        </impresoras>
        <impresoras>
                <impresora marca=”Nec”>
                        234474423
                </impresora>
                <impresora marca=”Hp”>
                        23456785674423
                </impresora>
        </impresoras>
        <impresoras>
                <impresora marca=”Epson”>
                        25674423
                </impresora>
                <impresora marca=”Nec”>
                        234114423
                </impresora>
                <impresora marca=”HP”>
                        234567113
                </impresora>
                <impresora marca=”Lexmark”>
                        5674423
                </impresora>
        </impresoras>
</perif>
```

Con esta sentencia XPATH

Select=”impresoras[3]/impresora[2]”

Estaríamos haciendo referencia al segundo elemento impresora contenido en el tercer elemento impresoras, y el contenido resultante sería: “234114423”.
Debemos tener en cuenta que XPATH no es un lenguaje para efectuar consultas (querys), XPATH es un lenguaje que permite al procesador de XSLT obtener los contenidos de los diferentes elementos de un documento XML.
Por último, no olvidemos nunca que nos movemos por una estructura jerárquica de elementos.
A continuación, definiremos los distintos patrones que pueden ser utilizados en XPATH:

/	Especifica el hijo inmediato, también puede ser utilizado para referirse al elemento raíz.
//	Selecciona a cualquier nivel dentro de la estructura de árbol.
.	Selecciona el nodo actual.
*	Selecciona todos los elementos del nodo actual.
@	Selecciona un atributo específico.
@*	Selecciona todos los atributos del nodo actual.

XSLT (XSL Transformations)

El lenguaje de estilo extendido conocido como XSLT (Extensible Stylesheet Language Transformations), es un lenguaje que nos permite definir un formato de presentación para un documento XML.

XSLT está diseñado para ser usado de manera independiente de XSL. De cualquier manera, XSLT no está diseñado para ser un lenguaje de transformación para cualquier propósito. Internet Explorer utiliza como lenguaje de transformación de XML, XSLT.

Un documento XSLT que defina una transformación de un documento XML, también está expresado en sí mismo como un documento XML bien formado. Un documento XSLT define las reglas para transformar un árbol origen (documento XML) en un árbol de resultado. La transformación se realiza asociando las distintas plantillas con sus correspondientes valores. La estructura resultante (árbol de resultado) puede ser

completamente diferente de la estructura de origen (árbol origen). Durante la construcción de la estructura resultante los elementos de la estructura origen pueden ser reordenados, filtrados, además de añadir otro tipo de estructuras arbitrarias en la construcción.

A la transformación expresada en el lenguaje XSLT es a lo que llamamos hoja de estilo.

Estructura de la hoja de estilo XSLT

El elemento stylesheet.

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

</xsl:stylesheet>
```

El elemento "stylesheet" es el elemento raíz de una hoja de estilo XSLT, es decir, todas las hojas de estilo XSLT deben comenzar con <xsl:stylesheet> y terminar con </xsl:stylesheet>. Dentro del elemento "stylesheet" también definimos mediante atributos la versión de XSL con la que estamos trabajando, así como la URI de lo que llamamos "namespace" (definición de las reglas semánticas y sintácticas redactadas por el consorcio a las que hace referencia el analizador de XSL). De todas maneras, es opcional hacer referencia al "namespace".

Al igual que cualquier documento XML todos los elementos que formen parte de la hoja de estilo deben partir de su elemento raíz, en este caso <xsl:stylesheet>.

También podremos encontrar el ejemplo anterior de esta otra forma:

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

</xsl:transform>
```

En este caso, los dos ejemplos son iguales y perfectamente válidos, ya que <xsl:stylesheet> tiene el mismo significado que <xsl:transform> para el analizador.

El elemento xsl:stylesheet puede contener cualquiera de los siguientes elementos:

- xsl:import
- xsl:include
- xsl:strip-space
- xsl:preserve-space
- xsl:output
- xsl:key
- xsl:decimal-format
- xsl:namespace-alias
- xsl:attribute-set
- xsl:variable
- xsl:param
- xsl:template

Estos elementos son también llamados elementos “top-level”. Veamos un ejemplo donde se declaran todos estos elementos:

```
<xsl:stylesheet version=”1.0”  xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
    <xsl:import href=”...”/>
    <xsl:include href=”...”/>
    <xsl:strip-space elements=”...”/>
    <xsl:preserve-space elements=”...”/>
    <xsl:output method=”...”/>
    <xsl:key name=”...” match=”...” use=”...”/>
    <xsl:decimal-format name=”...”/>
      <xsl:namespace-alias stylesheet-prefix=”...” result-prefix=”...”/>
    <xsl:attribute-set name=”...”>
```

```
    ...
  </xsl:attribute-set>
  <xsl:variable name="...">...</xsl:variable>
  <xsl:param name="...">...</xsl:param>
  <xsl:template match="...">
    ...
  </xsl:template>
  <xsl:template name="...">
    ...
</xsl:template>
</xsl:stylesheet>
```

El orden en el que los elementos "top_level" sean declarados no es significante, y es transparente para los analizadores.

Existe también una sintaxis simplificada para aquellas hojas de estilo que tengan como función extraer valores a partir del elemento raíz del documento XML. Para ello, utilizamos la etiqueta <xsl:value-of select>, tal como podemos ver en el ejemplo:

```
<xsl:version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns="http://www.w3.org/TR/xhtml1/strict">
<html>
  <head>
    <title>Informe Resumen de Gastos</title>
  </head>
 <body>
  <p>Total Amount:
  <xsl:value-of select="informe-gastos/total"/></p>
 </body>
</html>
```

En el ejemplo podemos observar, en primer lugar, cómo estamos ya insertando HTML dentro de la hoja de estilo, y cómo entre el código HTML podemos encontrar los elementos XSL, como el elemento <xsl:value-of select=”informe-gastos/total”/>, que es el encargado de extraer del documento XML el valor correspondiente al elemento “informe-gastos/total”. Por supuesto, hacemos referencia a determinado valor, utilizando las reglas semánticas del lenguaje XPATH.

El ejemplo anterior también podríamos haberlo realizado de esta otra forma:

```
<xsl:stylesheet version=”1.0”
xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”
xmlns=”http://www.w3.org/TR/xhtml1/strict”>

<xsl:template match=”/“>
    <html>
    <head>
        <title> Informe Resumen de Gastos </title>
    </head>
    <body>
        <p>Total Amount: <xsl:value-of select=”informe-gastos/total”/></p>
    </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

La principal diferencia entre los dos ejemplos se centra, como puede verse, en la existencia del segundo del elemento <xsl:template>, que tiene como función situarse en un determinado nodo de la estructura del documento XML, mediante el atributo “match”, en el ejemplo <xsl:template match=”/“>, el analizador se situará en el nodo raíz del documento XML, para a partir de ahí, mediante el elemento <xsl:value-of select= “....”/> extraer los valores de los elementos.

Veamos ahora un sencillo ejemplo completo:

El documento XML:

```
<?xml version=”1.0”?>
<?xml-stylesheet  href=”sitios.xsl” type=”text/xsl”?>
<sitiosweb>
 <sitioweb> www.disney.com </sitioweb>
 <sitioweb> www.sony.com </sitioweb>
 <sitioweb> www.fox.com </sitioweb>
 <sitioweb> www.dreamworks.com </sitioweb>
</sitiosweb>
```

El documento XSL:

```
<?xml version=”1.0”?>
<xsl:stylesheet xmlns:xsl=”http://www.w3.org/TR/WD-xsl”>
<xsl:template match=”/“>
<HTML>
<HEAD>
            <TITLE> Sitios de Cine </TITLE>
</HEAD>
<BODY>
 <xsl:for-each select=”sitiosweb/sitioweb”>
<b>
 <xsl:value-of />
</b><br/>
 </xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

Lo primero que tenemos que tener en cuenta es que siempre llamamos al fichero XML, y que es éste el que se encarga de realizar la llamada a la hoja de estilo, tal como vimos en el ejemplo anterior:

```
<?xml version=”1.0”?>

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

En el documento XML nos encontramos con varios elementos “sitioweb”, que se repiten, y que contienen distintos valores. Si lo que pretendemos es que la hoja de estilo muestre los distintos valores, hemos de realizar un bucle para que el navegador lea los distintos valores del elemento, tal como vemos:

```
 <xsl:for-each select=”sitiosweb/sitioweb”>
<b>
 <xsl:value-of />
</b><br/>
 </xsl:for-each>
```

En el ejemplo podemos ver cómo creamos un bucle con la sentencia “for-each” del elemento “sitiosweb/sitioweb”, recordemos el uso de XPATH en XSL. En la sentencia anterior habíamos hecho un “template match=/”, que nos situaba en la raíz del árbol del documento XML, por tanto, al realizar el bucle debemos situarnos en el nodo correcto del árbol.

Veamos ahora otro ejemplo:

El documento XML:

```
<?xml version=”1.0”?>
<?xml-stylesheet  href=”sitios2.xsl” type=”text/xsl”?>
<sitiosweb>
    <sitioweb href=”http://www.disney.com”
                titulo=”Disney”/>
    <sitioweb href=”http://www.sony.com”
                titulo=”Sony pictures”/>
    <sitioweb href=”http://www.fox.com”
                titulo=”Twenty Century Fox”/>
    <sitioweb href=”http://www.dreamworks.com”
                titulo=”Dreamworks”/>
</sitiosweb>
```

El documento XSL:

```
<?xml version=”1.0”?>
<xsl:stylesheet xmlns:xsl=”http://www.w3.org/TR/WD-xsl”>
<xsl:template match=”/“>
        <HTML>
        <HEAD>
        <TITLE>  </TITLE>
        </HEAD>

        <BODY>
         <xsl:for-each select=”/sitiosweb/sitioweb”>
        <A>
         <xsl:attribute name=”href”>
                    <xsl:value-of select=”./@href” />
         </xsl:attribute>
         <xsl:value-of select=”./@titulo” />
        </A><br/>
         </xsl:for-each>
        </BODY>
        </HTML>
         </xsl:template>
         </xsl:stylesheet
```

En este otro ejemplo hemos utilizado los atributos “href” y “titulo” para hacer referencia a los sitios web que se almacenan en nuestro documento XML. Tal como vemos en el ejemplo, para hacer referencia al valor de un atributo utilizamos la “@”.

```
<xsl:value-of select=”./@href” />
```

Recordemos siempre el uso del lenguaje XPATH en XSL “./”, que en este caso indica el nodo actual.

Elementos XSL

Veamos ahora los elementos más importantes del lenguaje XSL:

<xsl:stylesheet> </xsl:stylesheet>

Como hemos visto con anterioridad, es el elemento raíz de cualquier hoja de estilo XSL.

<xsl:template match=”....”> </xsl:template>

Tiene como función la definición de un patrón de búsqueda para un elemento determinado o conjunto de ellos.

<xsl:apply-template select=”.....”> </xsl:apply-template>

Hace que el procesador XML busque la plantilla adecuada para el nodo seleccionado.

<xsl:value-of select=”....”> </xsl:value-of>

Recoge el valor como texto del nodo seleccionado.

<xsl:attribute name=”...”> </xsl:attribute>

Crea un atributo dentro del documento XSL.

<xsl:element name=”....”> </xsl:element>

Crea un elemento dentro del documento XSL.

<xsl:for-each order-by=”....” select=”...”> </xsl:for-each>

Crea un bucle de repetición del nodo seleccionado. Tomemos como ejemplo el siguiente documento XML:

```
<clientes>
  <cliente>
    <nombre>...</nombre>
    <pedido>...</pedido>
    <pedido>...</pedido>
  </cliente>
  <cliente>
    <nombre>...</nombre>
    <pedido>...</pedido>
    <pedido>...</pedido>
  </cliente>
</clientes>
```

El siguiente documento XSL transformaría la información contenida en el documento XML en un documento HTML formateado, donde la información se incluiría en filas de una tabla.

```
<xsl:template match=”/“>
 <html>
  <head>
   <title>Clientes</title>
  </head>

  <body>
   <table>
        <tbody>
         <xsl:for-each select=”clientes/cliente”>
          <tr>
           <th>
                <xsl:apply-templates select=”nombre”/>
           </th>
           <xsl:for-each select=”pedido”>
                <td>
                 <xsl:apply-templates/>
                </td>
           </xsl:for-each>
          </tr>
         </xsl:for-each>
        </tbody>
   </table>
  </body>
 </html>
</xsl:template>
<xsl:if test=”.....”> </xsl:if>
```

Crea un condicional (la condición se sitúa en el valor del atributo “test”). El siguiente ejemplo pondría color a una tabla, fila sí, fila no, con el color amarillo.

```
<xsl:template match=”elemento1”>
 <tr>
 <xsl:if test=”position() mod 2 = 0”>
 <xsl:attribute
        name=”bgcolor”>yellow</xsl:attribute>
 </xsl:if>
 <xsl:apply-templates/>
 </tr>
</xsl:template>
<xsl:choose> </xsl:choose>
```

Crea un condicional del tipo “Case”, donde mediante la sentencia <xsl:when test=”....”> </xsl:when> establecemos las distintas opciones de la sentencia condicional.

```
<xsl:choose>
        <xsl:when test=”...”>  </xsl:when>
        <xsl:when test=”...”>  </xsl:when>
        <xsl:when test=”...”>  </xsl:when>
</xsl:choose>
```

También, dentro del elemento choose podemos encontrar el elemento <xsl:otherwise> </xsl:otherwise> para referirnos a las opciones que no coincidan con ninguna de las condiciones contenidas en el elemento "choose".

```
<xsl:choose>
        <xsl:when test="..."> </xsl:when>
        <xsl:when test="..."> </xsl:when>
        <xsl:when test="..."> </xsl:when>
        <xsl:otherwise> </xsl:otherwise>
</xsl:choose>
```

El siguiente ejemplo muestra cómo enumerar diferentes elementos en una lista ordenada, usando números, letras o números romanos, dependiendo de la profundidad a la que se encuentre el nodo.

```
<xsl:template match="listaordenada/elementosimple">
    <fo:list-item indent-start='2pi'>
     <fo:list-item-label>
     <xsl:variable name="level" select=
            "count(ancestor::listaordenada) mod 3"/>
     <xsl:choose>
            <xsl:when test='$level=1'>
                <xsl:number format="i"/>
            </xsl:when>
            <xsl:when test='$level=2'>
              <xsl:number format="a"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:number format="1"/>
            </xsl:otherwise>
     </xsl:choose>
     <xsl:text>. </xsl:text>
    </fo:list-item-label>
```

```
<fo:list-item-body>
        <xsl:apply-templates/>
</fo:list-item-body>
</fo:list-item>
</xsl:template>

<xsl:copy> </xsl:copy>
```

El elemento “xsl:copy” constituye una manera sencilla para copiar el nodo actual. Simplemente, haciendo una llamada al elemento, el nodo actual es copiado, así como el nombre del nodo actual, pero los atributos y subelementos del nodo actual no son copiados automáticamente.

```
<xsl:template match=”@*|elemento1()“>
<xsl:copy>
        <xsl:apply-templates select=”@*|elemento1()“/>
</xsl:copy>
</xsl:template>
<xsl:number/>
```

El elemento “xsl:number” se utiliza para insertar un valor numérico formateado dentro del árbol de resultados. El número que insertemos debe ser introducido como una expresión de cadena de caracteres, es el analizador el encargado de evaluar la expresión y de efectuar su conversión para, finalmente, ser incluido dentro del árbol de resultados.

```
<xsl:template match=”elemento1”>
 <xsl:for-each select=”elemento1”>
  <xsl:sort select=”.”/>
<p>
   <xsl:number value=”position()“ format=”1. “/>
   <xsl:value-of select=”.”/>
</p>
</xsl:for-each>
</xsl:template>
```

En el caso de que no especifiquemos una posición mediante el atributo value=”posición()”, entonces el resultado será introducido en la posición actual del nodo del árbol de resultados.
El elemento “xsl:number” tiene además otros atributos, como “level”, “count” y “from”.
El ejemplo siguiente numerará todos los elementos H4 con una etiqueta de texto dividida en tres partes:

```
<xsl:template match=”H4”>
  <fo:block>
    <xsl:number level=”any” from=”H1” count=”H2”/>
      <xsl:text>.</xsl:text>
      <xsl:number level=”any” from=”H2” count=”H3”/>
      <xsl:text>.</xsl:text>
      <xsl:number level=”any” from=”H3” count=”H4”/>
      <xsl:text> </xsl:text>
      <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

El siguiente ejemplo numerará los elementos “titulo”. Este ejemplo muestra cómo numerar los elementos de un documento, como si se tratase de una secuencia de capítulos, seguida de una secuencia de apéndices, donde tanto los capítulos como los apéndices contienen secciones que, a su vez, contienen subsecciones. Los capítulos estarán numerados de la forma 1, 2, 3, ..., los apéndices A, B, C, ... Las secciones en los capítulos serán numeradas de esta manera 1.1, 1.2, 1.3, ..., y las secciones en los apéndices serán numeradas A.1, a.2, a.3, ...

```
<xsl:template match=”titulo”>
  <fo:block>
    <xsl:number level=”multiple”
          count=”capitulo|seccion|subseccion”
          format=”1.1 “/>
```

```
            <xsl:apply-templates/>
          </fo:block>
        </xsl:template>
        <xsl:template match="apendice//titulo">
          <fo:block>
            <xsl:number level="multiple"
                    count="apendice|seccion|subseccion"
                    format="A.1 "/>
            <xsl:apply-templates/>
          </fo:block>
        </xsl:template>
<xsl:sort> </xsl:sort>
```

El elemento "xsl:sort" procesa los elementos pertenecientes al nodo actual de manera ordenada, esta ordenación puede ser ascendente o descendente. Supongamos, por ejemplo, el siguiente documento XML, conteniendo datos acerca de los empleados de una compañía:

```
<empleados>

    <empleado>

     <nombre>
       <nombredepila>Juan</nombredepila>
       <apellido>Perez</apellido>
     </nombre>
     ...
    </empleado>
  </empleados>
```

Para obtener como salida una lista de empleados ordenada por nombre:

```
<xsl:template match="empleados">
  <ul>
    <xsl:apply-templates select="empleado">
      <xsl:sort select="nombre/apellido"/>
      <xsl:sort select="nombre/nombredepila"/>
    </xsl:apply-templates>
  </ul>
</xsl:template>
<xsl:template match="empleado">
  <li>
    <xsl:value-of select="nombre/nombredepila"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="nombre/apellido"/>
  </li>
</xsl:template>
```

Funciones adicionales

XSLT dispone de una serie de funciones adicionales que, a continuación, señalamos:

- **document**(object, node-set?)

Permite acceder a otros documentos XML, que no sean el documento XML origen de la plantilla. Ejemplo:

generate-id

(document("foo.xml"))=generateid(document("foo.xml"))

- **key**(string, object)

Permite a los documentos XML, que contienen atributos del tipo ID, IDREF, IDREFS realizar referencias cruzadas.

<xsl:key name="idkey" match="div" use="@id"/>

- **format-number**(number, string, string?)

Realiza la conversión de un número a una cadena, utilizando un formato dado.

```
<xsl:decimal-format
        name                = qname
        decimal-separator   = char
        grouping-separator  = char
        infinity            = string
        minus-sign          = char
        NaN                 = string
        percent             = char
        per-mille           = char
        zero-digit          = char
        digit               = char
        pattern-separator = char />
```

- **current**()

Devuelve el nodo actual donde estamos situados. Este ejemplo:

<xsl:value-of select="current()"/>

Es igual a este otro:

<xsl:value-of select="."/>

- **unparsed-entity-uri**(string)

Devuelve el URI de una determinada entidad dentro del nodo actual.

generate-id(node-set?)

Devuelve una cadena de texto, que identifica el nodo actual de manera única.

- **system-property**(string)

Devuelve un valor coincidente con el valor de la propiedad del sistema a la que se hace referencia. Si la propiedad a la que se hace referencia no existe, entonces devuelve una cadena vacía.

Mensajes

```
<xsl:message
  terminate = “yes” | “no”>

        <!— Content: template —>

</xsl:message>
```

La instrucción “xsl:message” envía un mensaje para que sea interpretado por el analizador de XSLT. Generalmente, el procesador XSLT lanzará una pantalla de alerta al recibir la instrucción “xsl:message”.

```
<xsl:template name=”mensajepuntual”>
  <xsl:param name=”nombremensaje”/>
  <xsl:message>
    <xsl:value-of select=
                “$mensajes/mensaje[@name=$nombremensaje]“/>
  </xsl:message>
</xsl:template>
```