INTRODUCCIÓN A XML EN CASTELLANO

Versión 2.0 (26 Enero, 2000) - http://www.ibium.com/alf/xml/index.asp
Por Alfredo Reino Romero <alf@ibium.com>

Está disponible la presentación sobre XML impartida en el "IV Simpósium Internacional de Telemática" , de la Universidad de Colima (México) los dias 22 y 23 de Mayo de 2000.

```
Formato PDF (182 páginas, 1.57MB) http://www.ibium.com/alf/xml/colima2000.pdf
```

XML (*eXtensible Markup Language*) no es, como su nombre podría sugerir, un lenguaje de marcado. XML es un meta-lenguaje que nos permite definir lenguajes de marcado adecuados a usos determinados.

El HTML (*HyperText Markup Language*) se ha convertido en el lenguaje estándar (o *lingua franca*) del World Wide Web. En sus casi diez años de andadura, y tras una fase de desarrollo más o menos turbulento, se ha confirmado como un estándar aceptado y aprobado por la industria. HTML se puede considerar una aplicación de SGML (*Standard Generalised Markup Language*) Hay que desterrar ideas del tipo "XML es HTML mejorado" o "XML es HTML ampliable y personalizable."

Los fundamentos de XML son muy sencillos. En las siguientes páginas aprenderemos cómo crear documentos XML bien-formados, y que además sean válidos, es decir, que estén conformes a una Definición de Tipo de Documento (DTD) dada. Además veremos la creación y uso de hojas de estilo (XSL) para la presentación de los datos, así como diferentes aplicaciones actuales de XML.

ÍNDICE

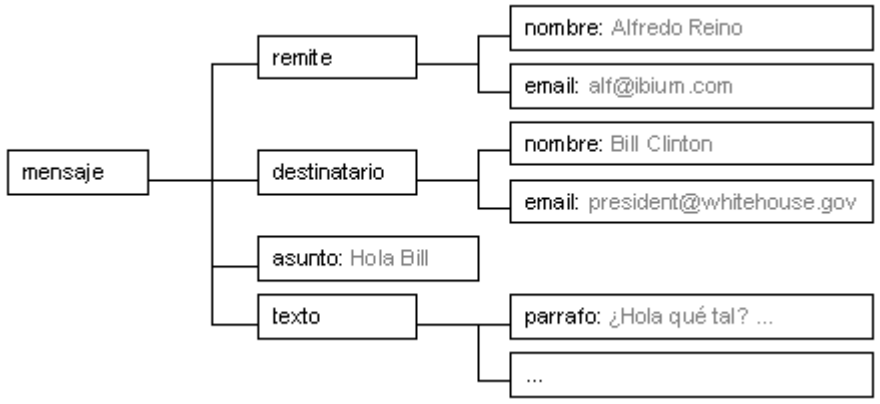
ESTRUCTURA DE UN DOCUMENTO XML

Aunque a primera vista, un documento XML puede parecer similar a HTML, hay una diferencia principal. Un documento XML contiene datos que se autodefinen, exclusivamente. Un documento HTML contiene datos mal definidos, mezclados con elementos de formato. En XML se separa el contenido de la presentación de forma total.

Una forma de entender rápidamente la estructura de un documento XML, es viendo un pequeño ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">
<mensaje>
 <remite>
  <nombre>Alfredo Reino</nombre>
  <email>alf@ibium.com</email>
 </remite>
 <destinatario>
  <nombre>Bill Clinton</nombre>
  <email>president@whitehouse.gov</email>
 </destinatario>
 <asunto>Hola Bill</asunto>
 <texto>
  <parrafo>¿Hola qué tal? Hace <enfasis>mucho</enfasis> que
  no escribes. A ver si llamas y quedamos para tomar algo.</parrafo>
 </texto>
</mensaje>
```

Este mismo documento puede ser visto de forma gráfica, para comprender mejor la estructura de un documento XML.



DOCUMENTOS XML BIEN-FORMADOS

Existen un número de diferencias entre la sintaxis de HTML y XML. Es útil, para aquellos que saben HTML y quieren usar XML, conocerlas perfectamente, para poder crear documentos XML bien-formados.

Estructura jerárquica de elementos

Los documentos XML deben seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente "incluida" en otra. Además, los elementos con contenido, deben estar correctamente "cerrados". En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda:

```
<LI>HTML <B>permite <I>esto</B></I>.
```

```
<LI>En XML la <B>estructura <I>es</I> jerárquica</B>.</LI>
```

Etiquetas vacías

HTML permite elementos sin contenido. Como veremos más adelante, XML tambien, pero la etiqueta debe ser de la siguiente forma: <elemento-sin-contenido/> En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda:

```
<LI>Esto es HTML<BR>en el que casi todo está permitido</LI>

<LI>En XML, somos<BR/> más restrictivos.</LI>
```

Un solo elemento raiz

Los documentos XML sólo permiten un elemento raiz, del que todos los demás sean parte. Es decir, la jerarquía de elementos de un documento XML bien-formado sólo puede tener un elemento inicial.

Valores de atributos

Los valores de atributos en XML, al contrario de HTML, siempre deben estar encerradas en comillas simples (') o dobles ("). En el siguiente ejemplo, la primera línea sería incorrecta en XML, no así la segunda:

```
<A HREF=http://www.disney.com/>

<A HREF="http://www.developer.com/">
```

Tipo de letra, espacios en blanco

El XML es sensible al tipo de letra utilizado, es decir, trata las mayúsculas y minúsculas como caracteres diferentes. Si un elemento de XML está definido como "ELEMENTO", no podemos usar "elemento", ni "Elemento", ni "eleMENto" para referirnos a él.

Existe unconjunto de caracteres denominados "espacios en blanco" que los procesadores XML tratan de forma diferente en el marcado XML. Estos caracteres son los "espacios" (Unicode/ASCII 32), tabuladores (Unicode/ASCII 9), retornos de carro (Unicode/ASCII 13) y los saltos de línea (Unicode/ASCII 10).

La especificación XML 1.0 permite el uso de esos "espacios en blanco" para hacer más legible el código, y en general son ignorados por los procesadores XML.

En otros casos, sin embargo, los "espacios en blanco" resultan muy significativos, por ejemplo, para separar las palabras en un texto, o separar lineas de párrafos diferentes.

Nombrando cosas

Al utilizar XML, es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen algunas características en común.

Según la especificación XML 1.0

> *Un nombre [empieza] con una letra o uno o más signos de puntuación, y [continúa] con letras, dígitos, guiones, rayas, dos puntos o puntos, denominados de forma global como caracteres de nombre. Los nombres que empiezan con la cadena "xml", se reservan para la estandarización de esta o de futuras versiones de esta especificación.*

Resumiendo, no se pueden crear nombres que empiecen con la cadena "xml", "xMl", "XML" o cualquier otra variante. Las letras y rayas se pueden usar en cualquier parte del nombre. También se pueden incluir dígitos, guiones y caracteres de punto, pero no se puede empezar por ninguno de ellos. El resto de caracteres, como algunos símbolos, y espacios en blanco, no se pueden usar.

Marcado y datos

Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan "marcas". Éstas son las partes del documento que el procesador XML espera entender. El resto del documento que se encuentra entre las marcas, son los datos que resultan entendibles por las personas.

Es sencillo reconocer las marcas en un documento XML. Son aquellas porciones que empiezan con "<" y acaban con ">", o bien, en el caso de las referencias de entidad, empiezan por "&" y acaban con ";".

EL PRÓLOGO

Aunque no es obligatorio, los documentos XML pueden empezar con unas línea que describen la versión de XML, el tipo de documento, y otras cosas.

La primera, o "declaración XML", define la versión de XML usada. Hasta ahora sólo hay una, la "1.0" Además, en la "declaración XML" especificamos la codificación del documento, que puede ser, por ejemplo, US-ASCII (7 bits) o UTF-8 (código Unicode del que el ASCII es un subconjunto), UCS-2, EUC-JP, Shift_JIS, Big5, ISO-8859-1 hasta ISO-8859-7. En general, y para uso con lenguajes europeos (incluyendo el juego de caracteres especiales del castellano, usamos UTF-7 o ISO-8859-1)

Además, se puede incluir una declaración de documento autónomo (*standalone*), que controla qué componentes de la DTD son necesarios para completar el procesamiento del documento.

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

La segunda, o "declaración de tipo de documento", define qué tipo de documento estamos creando para ser procesado correctamente. Es decir, definimos que Declaración de Tipo de Documento (DTD – *Document Type Definition*) valida y define los datos que contiene nuestro documento XML.

En ella se define el tipo de documento, y dónde encontrar la información sobre su Definición de Tipo de Documento, mediante un identificador público (PUBLIC) que hace referencia a dicha DTD, o mediante un Identificador Universal de Recursos (URI) precedido por la palabra SYSTEM.

Ejemplos:

```
<!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">

<!DOCTYPE HTML PUBLIC "-/ /W3C/ /DTD HTML 3.2 Final/ /EN">

<!DOCTYPE LABEL SYSTEM "http://www.empresa.com/dtds/label.dtd">
```

ELEMENTOS

Los elementos XML pueden tener contenido (más elementos, caracteres, o ambos a la

vez), o bien ser elementos vacíos.

Un elemento con contenido es, por ejemplo:

```
<nombre>Fulano Mengánez</nombre>

<aviso tipo="emergencia" gravedad="mortal">Que no cunda el pánico</aviso>
```

Siempre empieza con una <etiqueta> que puede contener atributos o no, y termina con una </etiqueta> que debe tener el mismo nombre. Al contrario que HTML, en XML siempre se debe "cerrar" un elemento.

Hay que tener en cuenta que el símbolo "<" siempre se interpreta como inicio de una etiqueta XML. Si no es el caso, el documento no estará bien-formado. Para usar ciertos símbolos se usanlas entidades predefinidas, que se explican más adelante.

Un elemento vacío, es el que no tiene contenido (claro). Por ejemplo:

```
<identificador DNI="23123244"/>

<linea-horizontal/>
```

Al no tener una etiqueta de "cierre" que delimite un contenido, se utiliza la forma <etiqueta/>, que puede contener atributos o no. La sintaxis de HTML permite etiquetas vacías tipo <hr> o <img src="..."> . En HTML reformulado para que sea un documento XML bien-formado, se debería usar <hr/> o

ATRIBUTOS

Como se ha mencionado antes, los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento.

Por ejemplo, un elemento "chiste" puede tener un atributo "tipo" y un atributo "calidad", con valores "vascos" y "bueno" respectivamente.

```
<chiste tipo="vascos" calidad="bueno">Esto es un dia que Patxi y Josu van paseando…
</chiste>
```

En una Definición de Tipo de Documento, se especifican los atributos que puede tener cada tipo de elemento, así como sus valores y tipos de valor posible.

Al igual que en otras cadenas literales de XML, los atributos pueden estar marcados entre comillas verticales (') o dobles ("). Cuando se usa uno para delimitar el valor del atributo, el otro tipo se puede usar dentro.

```
<verdura clase="zanahoria" longitud='15" y media'>

<cita texto="'Hola buenos dias', dijo él">
```

A veces, un elemento con contenido, puede modelarse como un elemento vacío con atributos. Un concepto se puede representar de muy diversas formas, pero una vez elegida una, es aconsejable fijarla en el DTD, y usar siempre la misma consistentemente dentro de un documento XML.

```
<gato><nombre>Micifú</nombre><raza>Persa</raza></gato>

<gato raza="Persa">Micifú</gato>
```

```
<gato raza="Persa" nombre="Micifú"/>
```

ENTIDADES PREDEFINIDAS

En XML 1.0, se definen cinco entidades para representar caracteres especiales y que no se interpreten como marcado por el procesador XML. Es decir, que así podemos usar el carácter "<" sin que se interprete como el comienzo de una etiqueta XML, por ejemplo.

Entidad	Caracter
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

SECCIONES CDATA

Existe otra construcción en XML que permite especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como marcado XML. La razón de esta construcción llamada CDATA (Character Data) es que a veces es necesario para los autores de documentos XML, poder leerlo facilmente sin tener que descifrar los códigos de entidades. Especialmente cuando son muchas.

Como ejemplo, el siguiente (primero usando entidades predefinidas, y luego con un bloque CDATA)

```
<parrafo>Lo siguiente es un ejemplo de HTML.</html>
<ejemplo>
&lt;HTML>
&lt;HEAD>&lt;TITLE>Rock &amp; Roll&lt;/TITLE>&lt;/HEAD>
</ejemplo>

<ejemplo>
<![CDATA[
<HTML>
<HEAD><TITLE>Rock & Roll</TITLE></HEAD>
]]>
</ejemplo>
```

Como hemos visto, dentro de una sección CDATA podemos poner cualquier cosa, que no será interpretada como algo que no es. Existe empero una excepción, y es la cadena "]] >" con la que termina el bloque CDATA. Esta cadena no puede utilizarse dentro de una sección CDATA.

COMENTARIOS

A veces es conveniente insertar comentarios en el documento XML, que sean ignorados por el procesado de la información y las reproducciones del documento. Los comentarios tienen el mismo formato que los comentarios de HTML. Es decir, comienzan por la cadena "<!--" y terminan con "-->".

```
<?xml version="1.0"?>
<!-- Aquí va el tipo de documento -->
<!DOCTYPE EJEMPLO [
<!-- Esto es un comentario -->
```

```
<!ELEMENTO EJEMPLO (#PCDATA)>
<!-- ¡Eso es todo por ahora! -->
]>

<EJEMPLO>texto texto texto bla bla bla
<!-- Otro comentario -->
</EJEMPLO>
<!-- Ya acabamos -->
```

Se pueden introducir comentarios en cualquier lugar de la instancia o del prólogo, pero nunca dentro de las declaraciones, etiquetas, u otros comentarios.

DOCUMENT TYPE DEFINITIONS (DTDs)

Crear una definición del tipo de documento (DTD) es como crear nuestro propio lenguaje de marcado, para una aplicación específica. Por ejemplo, podríamos crear un DTD que defina una tarjeta de visita. A partir de ese DTD, tendríamos una serie de elementos XML que nos permitirían definir tarjetas de visita.

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos.

Los documentos XML que se ajustan a su DTD, se denominan "válidos". El concepto de "validez" no tiene nada que ver con el de estar "bien-formado". Un documento "bien-fomado" simplemente respeta la estructura y sintaxis definidas por la especificación de XML. Un documento "bien-formado" puede además ser "válido" si cumple las reglas de una DTD determinada. También existen documentos XML sin una DTD asociada, en ese caso no son "válidos", pero tampoco "inválidos"... simplemente "bien-formados"... o no.

La DTD puede residir en un fichero externo, y quizá compartido por varios (puede que miles) de documentos. O bien, puede estar contenida en el propio documento XML, como parte de su declaración de tipo de documento.

Veamos un ejemplo:

```
<!DOCTYPE etiqueta[
<!ELEMENT etiqueta (nombre, calle, ciudad, pais, codigo)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT calle (#PCDATA)>
<!ELEMENT ciudad (#PCDATA)>
<!ELEMENT pais (#PCDATA)>
<!ELEMENT codigo (#PCDATA)>
]>

<etiqueta>
<nombre>Fulano Mengánez</nombre>
<calle>c/ Mayor, 27</calle>
<ciudad>Valderredible</ciudad>
<pais>España</pais>
<codigo>39343</codigo>
</etiqueta>
```

La declaración del tipo de documento comienza en la primera línea y termina con "]>". Las declaraciones DTD son las líneas que empiezan con "<!ELEMENT" y se denominan declaraciones de tipo elemento. También se pueden declarar atributos, entidades y anotaciones para una DTD.

En el ejemplo anterior, todas las declaraciones DTD que definen "etiqueta" residen dentro del documento. Sin embargo, la DTD se puede definir parcial o completamente en otro

lugar. Por ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE coche SYSTEM "http://www.sitio.com/dtd/coche.dtd">
<coche>
<modelo>...</modelo>
...
</coche>
```

DECLARACIONES TIPO ELEMENTO

Los elementos son la base de las marcas XML, y deben ajustarse a un tipo de documento declarado en una DTD para que el documento XML sea considerado válido.

Las declaraciones de tipo de elemento deben empezar con "<!ELEMENT" seguidas por el identificador genérico del elemento que se declara. A continuación tienen una especificación de contenido.

Por ejemplo:

```
<!ELEMENT receta (titulo, ingredientes, procedimiento)>
```

En este ejemplo, el elemento <receta> puede contener dentro elementos <titulo>, <ingredientes> y <procedimiento>, que, a su vez, estarán definidos también en la DTD y podrán contener más elementos.

Siguiendo la definición de elemento anterior, este ejemplo de documento XML sería válido:

```
<receta>
<titulo>...</titulo>
<ingredientes>...</ingredientes>
<procedimiento>...</procedimiento>
</receta>
```

Pero no este:

```
<receta>
<parrafo>Esto es un párrafo</parrafo>
<titulo>...</titulo>
<ingredientes>...</ingredientes>
<procedimiento>...</procedimiento>
</receta>
```

La especificación de contenido puede ser de cuatro tipos:

EMPTY

Puede no tener contenido. Suele usarse para los atributos.

```
<!ELEMENT salto-de-pagina EMPTY>
```

ANY

Puede tener cualquier contenido. No se suele utilizar, ya que es conveniente estructurar adecuadamente nuestros documentos XML.

```
<!ELEMENT batiburrillo ANY>
```

Mixed

Puede tener caracteres de tipo datos o una mezcla de caracteres y sub-elementos especificados en la especificación de contenido mixto.

```
<!ELEMENT enfasis (#PCDATA)>

<!ELEMENT parrafo (#PCDATA|enfasis)*>
```

Por ejemplo, el primer elemento definido en el ejemplo (<enfasis>) puede contener datos de carácter (#PCDATA). Y el segundo (<parrafo>) puede contener tanto datos de carácter (#PCDATA) como sub-elementos de tipo <enfasis>.

Element

Sólo puede contener sub-elementos especificados en la especificación de contenido.

```
<!ELEMENT mensaje (remite, destinatario, texto)>
```

Para declarar que un tipo de elemento tenga contenido de elementos se especifica un modelo de contenido en lugar de una especificación de contenido mixto o una de las claves ya descritas.

MODELOS DE CONTENIDO

Un modelo de contenido es un patrón que establece los sub-elementos aceptados, y el orden en que se acepta.

Un modelo sencillo puede tener un solo tipo de sub-elemento:

```
<!ELEMENT aviso (parrafo)>
```

Esto indica que <aviso> sólo puede contener un solo <parrafo>.

```
<!ELEMENT aviso (titulo, parrafo)>
```

La coma, en este caso, denota una secuencia. Es decir, el elemento <aviso> debe contener un <titulo> seguido de un <parrafo>.

```
<!ELEMENT aviso (parrafo | grafico)>
```

La barra vertical "|" indica una opción. Es decir, <aviso> puede contener o bien un <parrafo> o bien un <grafico>. El número de opciones no está limitado a dos, y se pueden agrupar usando paréntesis.

```
<!ELEMENT aviso (titulo, (parrafo | grafico))>
```

En este último caso, el <aviso> debe contener un <titulo> seguido de un <parrafo> o de un <grafico>.

Además, cada partícula de contenido puede llevar un indicador de frecuencia, que siguen directamente a un identificador general, una secuencia o una opción, y no pueden ir precedidos por espacios en blanco.

Indicadores de frecuencia	
?	Opcional (0 o 1 vez)
*	Opcional y repetible (0 o más veces)
+	Necesario y repetible (1 o más veces)

Para entender esto, vamos a ver un ejemplo.

```
<!ELEMENT aviso (titulo?, (parrafo+, grafico)*)>
```

En este caso, <aviso> puede tener <titulo>, o no (pero sólo uno), y puede tener cero o más conjuntos<parrafo><grafico>, <parrafo><parrafo><grafico>, etc.

DECLARACIONES DE LISTA DE ATRIBUTOS

Los atributos permiten añadir información adicional a los elementos de un documento. La principal diferencia entre los elementos y los atributos, es que los atributos no pueden contener sub-atributos. Se usan para añadir información corta, sencilla y desestructurada.

```
<mensaje prioridad="urgente">
<de>Alfredo Reino</de>
<a>Hans van Parijs</a>
<texto idioma="holandés">
Hallo Hans, hoe gaat het?
...
</texto>
</mensaje>
```

Otra diferencia entre los atributos y los elementos, es que cada uno de los atributos sólo se puede especificar una vez, y en cualquier orden.

En el ejemplo anterior, para declarar la lista de atributos de los elementos <mensaje> y <texto> haríamos lo siguiente:

```
<!ELEMENT mensaje (de, a, texto)>
<!ATTLIST mensaje prioridad (normal | urgente) normal>
<!ELEMENT texto(#PCDATA)>
<!ATTLIST texto idioma CDATA #REQUIRED>
```

Las declaraciones de los atributos empiezan con "<!ATTLIST", y a continuación del espacio en blanco viene el identificador del elemento al que se aplica el atributo. Despues viene el nombre del atributo, su tipo y su valor por defecto. En elejemplo anterior, el atributo "prioridad" puede estar en el elemento <mensaje> y puede tener el valor "normal" o "urgente", siendo "normal" el valor por defecto si no especificamos el atributo.

El atributo "idioma", pertenece al elemento texto, y puede contener datos de carácter (CDATA). Es más, la palabra #REQUIRED significa que no tiene valor por defecto, ya que es obligatorio especificar este atributo.

A menudo interesa que se pueda omitir un atributo, sin que se adopte automáticamente un valor por defecto. Para esto se usa la condición "#IMPLIED". Por ejemplo, en una supuesta DTD que defina la etiqueta <IMG> de HTML:

```
<!ATTLIST IMG URL CDATA #REQUIRED
```

```
ALT CDATE #IMPLIED>
```

Es decir, el atributo "URL" es obligatorio, mientras que el "ALT" es opcional (y si se omite, no toma ningún valor por defecto).

TIPOS DE ATRIBUTOS

Atributos CDATA y NMTOKEN

Los atributos CDATA (character data) son los más sencillos, y pueden contener casi cualquier cosa. Los atributos NMTOKEN (name token) son parecidos, pero sólo aceptan los caracteres válidos para nombrar cosas (letras, números, puntos, guiones, subrayados y los dos puntos).

```
<!ATTLIST mensaje fecha CDATA #REQUIRED>

<mensaje fecha="15 de Julio de 1999">

<!ATTLIST mensaje fecha NMTOKEN #REQUIRED>

<mensaje fecha="15-7-1999">
```

Atributos enumerados y notaciones

Los atributos enumerados son aquellos que sólo pueden contener un valor de entre un número reducido de opciones.

```
<!ATTLIST mensaje prioridad (normal | urgente) normal>
```

Existe otro tipo de atributo parecido, llamado de notación (NOTATION). Este tipo de atributo permite al autor declarar que su valor se ajusta a una notación declarada.

```
<!ATTLIST mensaje fecha NOTATION (ISO-DATE | EUROPEAN-DATE) #REQUIRED>
```

Para declarar las notaciones, se utiliza "<!NOTATION", con una definición externa de la notación. La definición externa puede ser pública o un identificador del sistema para la documentación de la notación, una especificación formal o un asistente de la aplicación que contenga objetos representados en la notación.

```
<!NOTATION HTML SYSTEM "http://www.w3.org/Markup">
```

<!NOTATION HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

Atributos ID e IDREF

El tipo ID permite que un atributo determinado tenga un nombre único que podrá ser referenciado por un atributo de otro elemento que sea de tipo IDREF. Por ejemplo, para implementar un sencillo sistema de hipervínculos en un documento:

```
<!ELEMENT enlace EMPTY>
<!ATTLIST enlace destino IDREF #REQUIRED>
<!ELEMENT capitulo (parrafo)*>
<!ATTLIST capitulo referencia ID #IMPLIED>
```

En este caso, una etiqueta <enlace destino="seccion-3"> haría referencia a un <capitulo referencia="seccion-3">, de forma que el procesador XML lo podría convertir en un hipervínculo, u otra cosa.

DECLARACIÓN DE ENTIDADES

XML hace referencia a objetos (ficheros, páginas web, imágenes, cualquier cosa) que no deben ser analizados sintácticamente según las reglas de XML, mediante el uso de entidades. Se declaran en la DTD mediante el uso de "<!ENTITY"

Una entidad puede no ser más que una abreviatura que se utiliza como una forma corta de algunos textos. Al usar una referencia a esta entidad, el analizador sintáctico reemplaza la referencia con su contenido. En otras ocasiones es una referencia a un objeto externo o local.

Las entidades pueden ser:

- Internas o Externas
- Analizadas o No analizadas
- Generales o Parámetro

Entidades generales internas

Son las más sencillas. Son básicamente abreviaturas definidas en la sección de la DTD del documento XML. Son siempre entidades analizadas, es decir, una vez reemplazada la referencia a la entidad por su contenido, pasa a ser parte del documento XML y como tal, es analizada por el procesador XML.

```
<!DOCTYPE texto[
<!ENTITY alf "Alien Life Form">
]>

<texto><titulo>Un día en la vida de un &alf;</titulo></texto>
```

Entidades generales externas analizadas

Las entidades externas obtienen su contenido en cualquier otro sitio del sistema, ya sea otro archivo del disco duro, una página web o un objeto de una base de datos. Se hace referencia al contenido de una entidad así mediante la palabra SYSTEM seguida de un URI (*Universal Resource Identifier*)

```
<!ENTITY intro SYSTEM "http://www.miservidor.com/intro.xml">
```

Entidades no analizadas

Evidentemente, si el contenido de la entidad es un archivo MPG o una imagen GIF o un fichero ejecutable EXE, el procesador XML no debería intentar interpretarlo como si fuera texto XML. Este tipo de entidades siempre son generales y externas.

```
<!ENTITY logo SYSTEM "http://www.miservidor.com/logo.gif">
```

Entidades parámetro internas y externas

Se denominan entidades parámetro a aquellas que sólo pueden usarse en la DTD, y no en el documento XML. Se pueden utilizar para agrupar ciertos elementos del DTD que se repitan mucho. Se diferencian las entidades parámetro de las generales, en que para hacer referencia a ellas, se usa el símbolo "%" en lugar de "&" tanto como para declararlas como para usarlas.

```
<!DOCTYPE texto[
<!ENTITY % elemento-alf "<!ELEMENT ALF (#PCDATA)>">
```

```
...
%elemento-alf;
]>
```

Tambien puede ser externa:

```
<!DOCTYPE texto[
<!ENTITY % elemento-alf SYSTEM "alf.ent">
...
%elemento-alf;
]>
```

EJEMPLO DE DTD

Un ejemplo de DTD que puede servir para resumir todo lo visto hasta ahora podría ser un DTD que nos defina un lenguaje de marcado para una base de datos de personas con direcciones e-mail.

El fichero LISTIN.DTD podría ser algo así:

```
<?xml encoding="UTF-8"?>
<!ELEMENT listin (persona)+>
<!ELEMENT persona (nombre, email*, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT relacion EMPTY>
<!ATTLIST relacion amigo-de IDREFS #IMPLIED
                   enemigo-de IDREFS #IMPLIED>
```

Basándonos en este DTD, podríamos escribir nuestro primer listín en XML de la siguiente manera:

```
<?xml version="1.0"?>
<!DOCTYPE listin SYSTEM "LISTIN.DTD">
<listin>
 <persona sexo="hombre" id="ricky">
   <nombre>Ricky Martin</nombre>
   <email>ricky@puerto-rico.com</email>
   <relacion amigo-de="laetitia">
 </persona>
 <persona sexo="mujer" id="laetitia">
   <nombre>Laetitia Casta</nombre>
   <email>castal@micasa.com</email>
 </persona>
</listin>
```

XML SCHEMAS

Un "*schema XML*" es algo similar a un DTD, es decir, que define qué elementos puede contener un documento XML, cómo están organizados, y que atributos y de qué tipo pueden tener sus elementos.

La ventaja de los *schemas* con respecto a los DTDs son:

- Usan sintaxis de XML, al contrario que los DTDs.
- Permiten especificar los tipos de datos.
- Son extensibles.

Por ejemplo, un *schema* nos permite definir el tipo del contenido de un elemento o de un atributo, y especificar si debe ser un número entero, o una cadena de texto, o una fecha, etc. Los DTDs no nos permiten hacer estas cosas.

Veamos un ejemplo de un documento XML, y su *schema* correspondiente:

```
<documento xmlns="x-schema:personaSchema.xml">
 <persona id="fulano">
  <nombre>Fulano Menganez</nombre>
 </persona>
</documento>
```

Como podemos ver en el documento XML anterior, se hace referencia a un espacio de nombres (*namespace*) llamado "x-schema:personaSchema.xml". Es decir, le estamos diciendo al analizador sintáctico XML (*parser*) que valide el documento contra el *schema* "personaSchema.xml".

El *schema* sería algo parecido a esto:

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <AttributeType name='id' dt:type='string' required='yes'/>
  <ElementType name='nombre' content='textOnly'/>
  <ElementType name='persona' content='mixed'>
    <attribute type='id'/>
    <element type='nombre'/>
  </ElementType>
  <ElementType name='documento' content='eltOnly'>
    <element type='persona'/>
  </ElementType>
</Schema>
```

El primer elemento del *schema* define dos espacios de nombre. El primero "*xml-data*" le dice al analizador que esto es un *schema* y no otro documento XML cualquiera. El segundo "*datatypes*" nos permite definir el tipo de elementos y atributos utilizando el prefijo "dt".

ElementType

Define el tipo y contenido de un elemento, incluyendo los sub-elementos que pueda contener.

AttributeType

Asigna un tipo y condiciones a un atributo.

attribute

Declara que un atributo previamente definido por **AttributeType** puede aparecer como atributo de un elemento determinado.

element

Declara que un elemento previamente definido por **ElementType** puede aparecer como contenido de otro elemento.

Tal como hemos visto, es necesario empezar el *schema* definiendo los elementos más profundamente anidados dentro de la estructura jerárquica de elementos del documento

XML. Es decir, tenemos que trabajar "de dentro hacia fuera".

Visto de otra manera, las declaraciones de tipo **ElementType** y **AttributeType** deben preceder a las declaraciones de contenido **element** y **attribute** correspondientes.

Consulta la referencia de schemas XML de Microsoft (http://msdn.microsoft.com/xml/reference/schema/start.asp) para más información.

EXTENDED STYLE LANGUAGE (XSL)

El XSL es un lenguaje que nos permite definir una presentación o formato para un documento XML. Un mismo documento XML puede tener varias hojas de estilo XSL que lo muestren en diferentes formatos (HTML, PDF, RTF, VRML, PostScript, sonido, etc.)

La aplicación de una hoja de estilo XSL a un documento XML puede ocurrir tanto en el origen (por ejemplo, un *servlet* que convierta de XML a HTML para que sea mostrado a un navegador conectado a un servidor de web), o en el mismo navegador (como es el caso del MS IE5, y en breve, Netscape 5).

Básicamente, XSL es un lenguaje que define una transformación entre un documento XML de entrada, y otro documento XML de salida.

Una hoja de estilo XSL es una serie de reglas que determinan cómo va a ocurrir la transformación. Cada regla se compone de un patrón (*pattern*) y una acción o plantilla (*template*).

De este modo, cada regla afecta a uno o varios elementos del documento XML. El efecto de las reglas es recursivo, para que un elemento situado dentro de otro elemento pueda ser también transformado. La hoja de estilo tiene una *regla raiz* que, además de ser procesada, llama a las reglas adecuadas para los elementos hijos.

Vamos a ver un ejemplo de todo esto:

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

  <capitulo>
   <titulo>Capítulo 1</titulo>
   <parrafo>....</parrafo>
   <parrafo>....</parrafo>
  </capitulo>

  <capitulo>
   <titulo>Capítulo 2</titulo>
   ...
  </capitulo>

 </capitulos>
</libro>
```

Queremos convertir este documento XML en HTML bien-formado, de la siguiente manera:

```
<HTML>
<HEAD>
 <TITLE>Un título cualquiera</TITLE>
</HEAD>
<BODY>
 <H1>Un título cualquiera</H1>
```

```
 <HR>
 <H2>Capítulo 1</H2>
 <P>...</P>
 <P>...</P>

 <HR>
 <H2>Capítulo 2</H2>
 <P>...</P>

</BODY>
</HTML>
```

La hoja de estilo XSL necesaria sería algo parecido a lo siguiente:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="libro">
 <HTML>
  <HEAD>
  <TITLE><xsl:process select="titulo"/></TITLE>
  </HEAD>
  <BODY>
  <H1><xsl:process select="titulo"/></H1>
  <xsl:process select="capitulos"/>
  </BODY>
 </HTML>
</xsl:template>

<xsl:template match="capitulos">
 <xsl:process select="capitulo">
</xsl:template>

<xsl:template match="capitulo">
 <HR/>
 <H2><xsl:process select="titulo"/></H2>
 <xsl:process select="parrafo"/>
</xsl:template>

<xsl:template match="parrafo">
 <P><xsl:process-children/></P>
</xsl:template>
</xsl:stylesheet>
```

XML DOCUMENT OBJECT MODEL Y VISUALBASIC

Vamos a ver a continuación como se maneja el **DOM** (*Document Object Model*) implementado por el **Microsoft XML Parser**, a través de Visual Basic. Aunque los detalles puedan cambiar, la forma de trabajar va aser similiar para otros *parsers* en otras plataformas de desarrollo (VC++, Java, etc.)

El *parser* o analizador de XML es el programa que lee el documento XML y hace que los datos que contiene estén accesibles de una manera u otra. Además, el *parser* puede tener otras funcionalidades, como validar el documento contra una DTD o un *schema*.

El *parser* de Microsoft (*msxml.dll*), que se distribuye con el IE5, hace eso mismo, haciendo que los datos estén disponibles a través de una serie de objetos que el programador puede utilizar.

La especificación "DOM Level 1" en la que esta basada esta implementación se puede encontrar en http://www.w3.org/TR/REC-DOM-Level-1

Este parser podemos utilizarlo tanto en programas VBScript que se ejecuten en un servidor web (ASP, *Active Server Pages*) o en un programa Visual Basic.

Por ejemplo, para crear una instancia del *parser* en un ASP, hacemos lo siguiente:

```
Set objParser = Server.CreateObject("Microsoft.XMLDOM")
```

Para utilizarlo en Visual Basic, tenemos que añadir el objeto COM **Microsoft XML, version 2.0**, en las referencias del proyecto, y luego crear una instancia del objeto:

```
Dim objParser As MSXML.DOMDocument
Set objParser = New MSXML.DOMDocument
```

Para cargar un documento XML, usamos el método *.Load* del objeto, especificando la ruta del documento, o bien mediante un URL que indique dónde se encuentra en la red.

```
If objParser.Load("c:\temp\xml\documento.xml") Then
' Ha funcionado
Else
' Ha ocurrido un error
End If

' Ahora destruimos el objeto parser
Set objParser = nothing
```

El método *.Load* puede fallar porque el nombre o ubicación del documento sea erronea, o bien porque el documento es inválido (no cumple las condiciones impuestas en su DTD o *schema*)

La validación del documento se puede suprimir si hacemos lo siguiente antes de cargar el documento:

```
objParser.validateOnParse = False
```

Una vez cargado, analizado y validado el documento, podemos empezar a utilizar la información que contiene. Para eso utilizamos la interface **IXMLDOMNode** del parser, que nos permite acceder a los diferentes nodos (elementos, atributos, texto, etc.) del documento.

Este interface nos provee de ciertos métodos y propiedades para acceder con comodidad a los contenidos del documento, tanto para lectura como para escritura.

`nodeType`

Nos da información sobre el tipo de un nodo. Este parser soporta 13 tipos diferentes de nodo. (*Están marcados en negrita los que más se usan*)

Constantes de tipos de nodo
NODE_ATTRIBUTE
NODE_CDATA_SECTION
NODE_COMMENT
NODE_DOCUMENT
NODE_DOCUMENT_FRAGMENT
NODE_DOCUMENT_TYPE
NODE_ELEMENT

NODE_ENTITY
NODE_ENTITY_REFERENCE
NODE_INVALID
NODE_NOTATION
NODE_PROCESSING_INSTRUCTION
NODE_TEXT

`nodeName`

Si el tipo de nodo es adecuado, nodeName nos devuelve el nombre del nodo. Por ejemplo un nodo tipo NODE_ELEMENT contendría en **nodeName** el nombre de un elemento de un documento XML.

`nodeValue`

Nos devuelve el valor que contiene ese nodo, si es aplicable.

`childNodes`

Contiene una colección de nodos "hijos" del nodo que estamos considerando. Esta colección puede ser iterada por una construcción **for each** de Visual Basic.

`hasChildNodes`

Propiedad *booleana* que nos dice si un nodo tiene "hijos".

`firstChild / lastChild`

Contienen referencias al primer y último "hijos" de un nodo.

`parentNode`

Nos devuelve una referencia al "padre" del nodo.

`nextSibling / previousSibling`

Nos devuelve una referencia al siguiente/anterior "hermano" del nodo, es decir, al siguiente/anterior nodo con el mismo "padre".

`attributes`

Nos devuelve una colección de los nodos tipo NODE_ATTRIBUTE del nodo.

Vamos a ver un ejemplo de un sencillo programa que selecciona todos los nodos tipo texto (NODE_TEXT) de un documento XML de una forma recursiva.

```
Public Sub CargaDoc()
Dim oParser As MSXML.DOMDocument
Set oParser = New MSXML.DOMDocument
If oParser.Load("test.xml") Then
  MuestraNodos oParser.childNodes
Else
  MsgBox "Ha ocurrido un error."
End If
End Sub
```

```
Public Sub MuestraNodos(ByRef Nodos As MSXML.IXMLDOMNodeList)
Dim oNodo As MSXML.IXMLDOMNode
For Each oNodo In Nodos
   If oNodo.nodeType = NODE_TEXT Then
      Debug.Print oNodo.parentNode.nodeName & "=" & oNodo.nodeValue
   End If
   If oNodo.hasChildNodes Then
      MuestraNodos oNodo.childNodes
   End If
Next oNodo
End Sub
```

Si el fichero test.xml que usa este programa fuera el siguiente:

```
<?xml version="1.0"?>
<documento>
  <titulo>Un documento cualquiera</titulo>
  <fecha><dia>1</dia><mes>12</mes><año>1999</año></fecha>
</documento>
```

El resultado sería:

```
titulo=Un documento cualquiera
dia=1
mes=12
año=1999
```

Para más información sobre el parser MSXML, visita el MSDN Online XML Developer Center (http://msdn.microsoft.com/xml)

XML DOCUMENT OBJECT MODEL Y JAVA

Como ya hemos podido suponer, el empleo de tecnología XML, al ser un estándar internacional y público, no nos ata a una plataforma o sistema de desarrollo concreto. Lo mismo se puede usar Perl bajo UNIX para generar documentos XML a partir de una base de datos, como usar Python en Windows NT para servir documentos HTML4 a navegadores web a partir de un documento XML.

Dicho esto, hay que decir que JAVA se posiciona como una opción interesante a la hora de desarrollar aplicaciones usando XML.

Por ejemplo, a partir de fuentes de datos en XML, podemos escribir un *servlet* que analice sintácticamente el XML, y que genere un árbol DOM (*Document Object Model*). Una vez generado el árbol DOM, podemos ir extrayendo la información que contiene e ir generando un documento HTML de acuerdo con ciertas reglas de formato, de modo que pueda ser visualizado por un navegador web.

Vamos a suponer que nuestro *servlet* carga ese fichero XML en un objeto *string* de Java, que nos vamos a disponer a analizar.

Lo primero que debemos hacer es crear un analizador sintáctico XML. Usamos el objeto *com.ibm.xml.Parser* para ello.

```
Parser parser = new Parser("xslparse.err");
```

El fichero "xslparse.err" será el registro de cualquier error que ocurra mientras procesamos el documento XML, que debe ser convertido a un *InputStream*.

```
ByteArrayInputStream bais = new ByteArrayInputStream(xmlString.getBytes());
```

Ahora le decimos al parser que ignore las declaraciones de tipo de documento, los comentarios, etc.

```
parser.setWarningNoXMLDecl(false);
parser.setWarningNoDoctypeDecl(false);
parser.setKeepComment(false);
parser.setProcessNamespace(false);
```

Analizamos el documento, y cerramos el *InputStream*.

```
doc = parser.readStream;
bais.close();
```

Ahora contenemos el árbol DOM en el objeto "doc", y lo que tenemos que hacer es movernos por el árbol DOM e ir sacando los datos que contiene.

Vamos a ver los métodos que tenemos para navegar el DOM (definidos en *org.w3c.dom.Node*)

`getDocumentElement()`

Devuelve el elemento raiz

`getFirstChild()`

Devuelve el nodo que es el primer "hijo" de este nodo.

`getNextSibling()`

Devuelve el nodo que es el siguiente "hermano" de este nodo.

`getLastChild()`

Devuelve el nodo que es el último "hijo" de este nodo.

`getPreviousSibling()`

Devuelve el nodo que es el último "hermano" de este nodo.

`getAttribute(java.lang.String attrName)`

Devuelve un objeto string que representa el valor de un atributo.

Existen más métodos para manipular el DOM. La especificación "DOM Level 1" (http://www.w3.org/TR/REC-DOM-Level-1) aporta más información sobre métodos del DOM.

LENGUAJE DE ENLACE XML (XLINK)

XLink es una aplicación XML que intenta superar las limitaciones que tienen los enlaces de hipertexto en HTML. Es una especificación que todavía está en desarrollo, y que todavía no tiene "rodaje" en el mundo real.

Los enlaces en HTML tienen una serie de limitaciones, que los hacen bastante pobres. Por

ejemplo, sólo tienen dos extremos, la terminación origen y la destino, y son unidireccionales.

Si yo creo un enlace en mi página web personal que me lleve hasta la página principal de Coca-Cola, estoy creando un "vínculo" entre esta página y la mia.

```
<a href="http://www.coca-cola.com/">Coca-Cola</a>
```

Este vínculo es unidireccional, porque un visitante cualquiera que entre en la página de Coca-Cola, no tiene forma de saber que mi página (así como otros cientos de miles) enlaza con ella.

El objetivo de XLink es crear enlaces entre recursos, de una forma de la que HTML no es capaz.

Por ejemplo, un estudiante podría hacer anotaciones a los apuntes que un profesor tiene disponibles en la red, por el método de insertar un enlace desde los apuntes (a los que no tiene acceso de escritura o modificación) y su propia página de anotaciones y comentarios.

O bien, podremos vincular dos páginas cualquiera, por ejemplo enlazando el texto de una noticia en un diario on-line, con el texto de la noticia equivalente en el diario competidor. Y no hay porqué quedarse en enlaces con dos extremos. Los enlaces extendidos permiten mucho más que eso.

En el momento actual, la tecnología para poder mantener una base de datos de enlaces mundial, no está desarrollada, pero se puede hacer a nivel local.

Por ejemplo, nuestra empresa quiere comprar ciertos productos de un suministrador. Los trabajadores de la empresa podrán visitar la página web del proveedor y hacer enlaces externos a comentarios sobre las especificaciones de tal producto. Cuando otro compañero de la empresa visite la página, el servidor de enlaces de nuestra empresa le mostrará la página junto con los enlaces externos creados, y mostrar nuestros comentarios como si fueran parte del documento original.

ENLACES DE INTERÉS

Especificación oficial XML v1.0 (http://www.w3.org/TR/REC-xml)
Especificación oficial HTML v4.0 (http://www.w3.org/TR/REC-html40)
Información oficial de XSL (XSLT/XPath) (http://www.w3.org/Style/XSL/)
The World Wide Web Consortium (http://www.w3c.org/)
XML en castellano (http://html.programacion.net/xml/principal.htm)
The XML Industry Portal (http://www.xml.org/)
XML.COM (http://www.xml.com/)
XML Software Guide (http://wdvl.internet.com/Software/XML/)
MSDN Online XML Developers Center (http://msdn.microsoft.com/xml/)
EarthWeb/IT Knowledge
"XML" de Natanya Pitts