

UNIDAD 7

DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS EN XML

1. INTRODUCCIÓN

Los lenguajes para el almacenamiento y transmisión de información son lenguajes que te indican cómo debes crear ficheros (generalmente de texto) que guardarán información de utilidad y que serán usados por distintas aplicaciones para almacenar y/o compartir información.

Estos lenguajes indican cómo debe estructurarse la información en ese fichero y cómo debe ser esa información. De esta forma cuando dos aplicaciones diferentes tienen que usar un fichero creado a partir de un lenguaje determinado, dichas aplicaciones saben perfectamente qué se van a encontrar en cada momento cuando están leyendo ese fichero.

Dentro de los principales lenguajes para el almacenamiento y transmisión de información nos encontramos con:

- Lenguajes de marcas creados a partir de **XML (eXtensible Markup Language)**. XML es un metalenguaje que permite crear lenguajes de marcas que se usan para elaborar ficheros para almacenar datos de forma legible.
- **JSON** es un formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript. La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX.
- **YAML** es un formato de serialización de datos legible por humanos inspirado en lenguajes como XML, C, Python o Perl. La sintaxis es relativamente sencilla y fue diseñada teniendo en cuenta que fuera muy legible pero que a la vez fuese fácilmente mapeable a los tipos de datos más comunes en la mayoría de los lenguajes de alto nivel.

Ejemplo de un fichero creado usando un lenguaje de marcas generado a partir de XML:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <breakfast_menu>
3    <food>
4      <name>Belgian Waffles</name>
5      <price>$5.95</price>
6      <description>Two of our famous Belgian Waffles with plenty
7        of real maple syrup</description>
8      <calories>650</calories>
9    </food>
10   <food>
11     <name>Strawberry Belgian Waffles</name>
12     <price>$7.95</price>
13     <description>Light Belgian waffles covered with
14       strawberries and whipped cream</description>
15     <calories>900</calories>
16   </food>
17 </breakfast_menu>
```

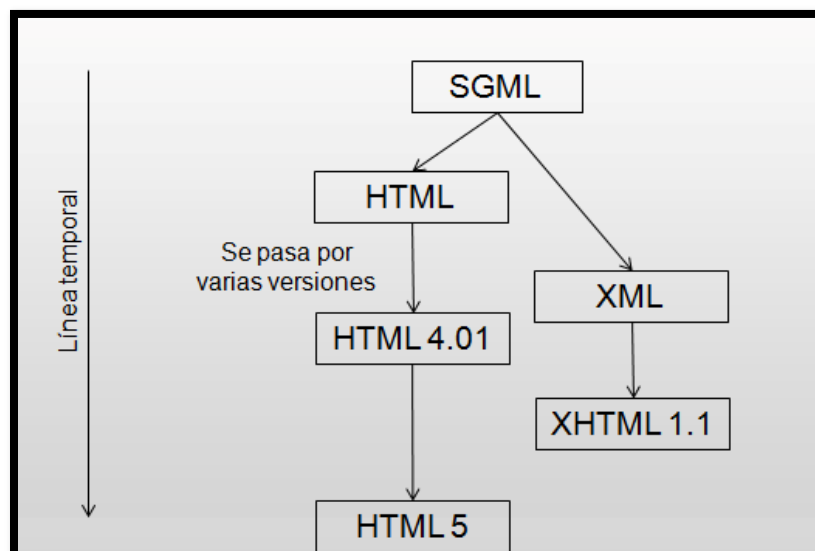
Ejemplo de un fichero creado usando JSON:

```
{ "Fruteria":  
  [  
    { "Fruta":  
      [  
        { "Nombre": "Manzana", "Cantidad": 10 },  
        { "Nombre": "Pera", "Cantidad": 20 },  
        { "Nombre": "Naranja", "Cantidad": 30 }  
      ]  
    },  
    { "Verdura":  
      [  
        { "Nombre": "Lechuga", "Cantidad": 80 },  
        { "Nombre": "Tomate", "Cantidad": 15 },  
        { "Nombre": "Pepino", "Cantidad": 50 }  
      ]  
    }  
  ]  
}
```

Ejemplo de un fichero creado usando YAML:

```
cdrom:  
  item:  
    title: cd 1  
    description: Descripción de cd 1  
  item:  
    title: cd 2  
    description: Descripción de cd 2  
dvd:  
  title: dvd 1  
  description: Descripción de dvd 1
```

¿Qué lenguajes de marcas conoces?



Si XHTML es un lenguaje de marcas, ¿por qué no puedo usarlo para el almacenamiento y transmisión de información?

- Una diferencia importante de un lenguaje de marcas generado a partir de XML con el lenguaje XHTML es que este último tiene un conjunto de etiquetas **predefinidas** que conforman el lenguaje, en cambio XML permite a los usuarios definir su propio lenguaje con sus propias etiquetas.
- Las etiquetas definidas en XHTML están orientadas a la presentación de la información (<p>, <table>, , etc.) y no al almacenamiento de la información.

2. XML

XML es un metalenguaje que permite generar lenguajes de marcas gracias a los cuales podemos crear ficheros para almacenar datos de forma legible.

La información se almacena en un fichero de texto plano y por tanto puede ser compartida entre aplicaciones independientemente del sistema operativo sobre el que se ejecuten dichas aplicaciones.

Pero... ¿qué es un lenguaje de marcas?

- Los lenguajes de marcas son aquellos que combinan la información que contiene un documento con marcas o etiquetas relativas a la estructura del texto o a la forma de representarlo.
- El lenguaje de marcas es el que especifica **cuáles serán las etiquetas posibles (y sus atributos), dónde deben colocarse y el significado que tendrá cada una de ellas.**

XML es un conjunto de reglas o normas que debemos seguir para crear un lenguaje de marcas. Gracias a XML podemos:

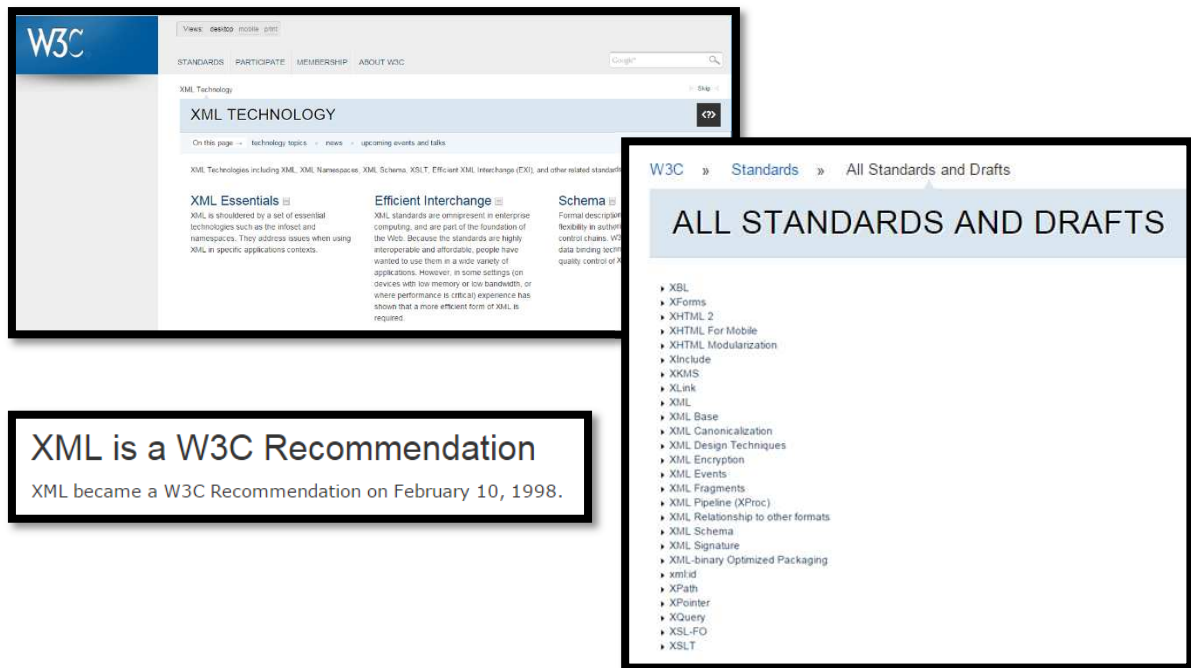
- Definir las etiquetas y sus atributos
- Definir las relaciones entre esas etiquetas

Existen numerosos lenguajes de marcas creados a partir de XML que se emplean en distintos ámbitos (de forma estándar) y que son definidos y mantenidos por distintas organizaciones, como por ejemplo el W3C. Algunos de estos lenguajes son:

- XHTML (usado para presentar información en la web).
- MathML (usado para intercambiar información en ámbito matemático).
- Open eBook (usado en el formato ePub por libros electrónicos).
- SVG (usado para representar gráficos vectoriales).
- Open Document (usado para almacenar y presentar información en un paquete ofimático).
- RSS (usando para difundir información actualizada a usuarios suscritos a una web).

En los siguientes temas vamos a crear nuestros propios lenguajes de marcas basados en XML definiendo las etiquetas que necesitemos (y las reglas que se deben cumplir para usar esas etiquetas) adaptadas a los problemas determinados que debamos resolver.

XML es mantenido por el W3C y en torno a él giran muchas otras tecnologías:

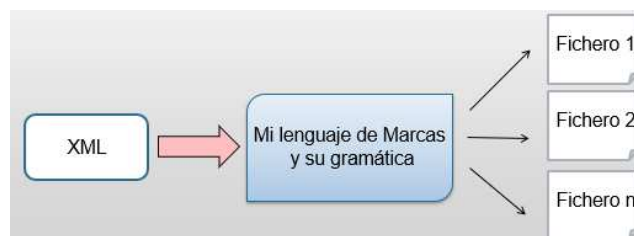


XML is a W3C Recommendation

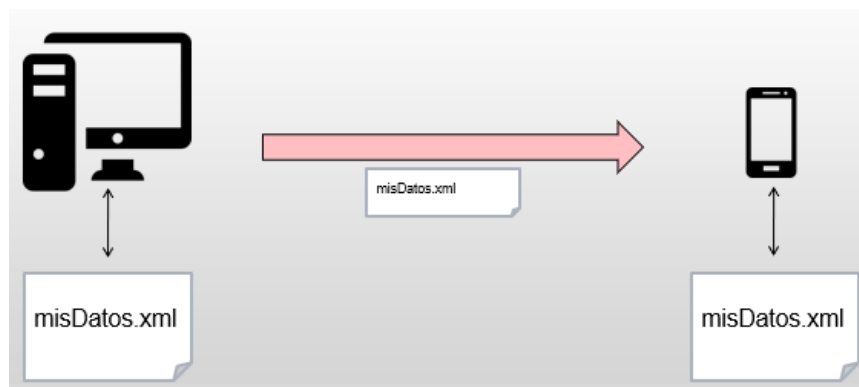
XML became a W3C Recommendation on February 10, 1998.

Así que **recuerda**:

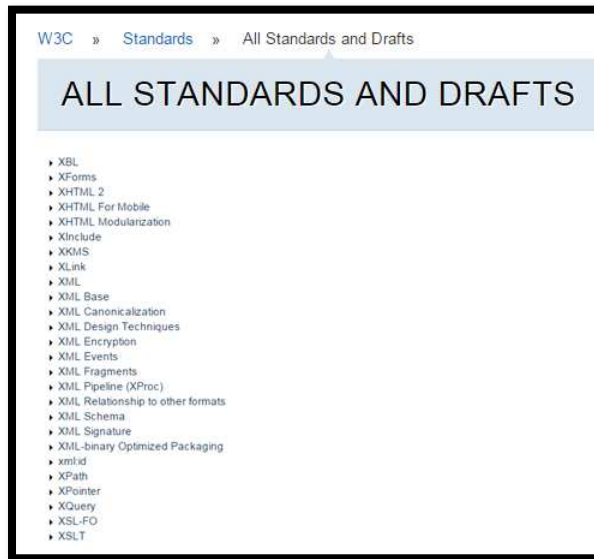
- XML permite generar lenguajes de marcas gracias a los cuales vamos a poder crear ficheros que van a almacenar información útil para nosotros.
- El lenguaje de marcas generado a partir de XML es el que especifica **cuáles serán las etiquetas posibles (y sus atributos)**, dónde deben colocarse y el significado que tendrá cada una de ellas.



- De este modo, si dos aplicaciones conocen el lenguaje que hemos creado, ambas aplicaciones pueden usar un mismo fichero para intercambiar información a través de Internet u otros medios.



- XML es una tecnología muy sencilla, pero tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande.



- Además, tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas de distinto tipo (Windows, Linux, Mac, Android, etc.) para compartir la información de una manera segura, fiable y fácil.

¡¡MUY IMPORTANTE!!

A los lenguajes de marcas generados a partir de XML (si no tienen nombre propio, como MathML, XHTML, etc.) se les suele denominar lenguajes XML.

Esta costumbre está ampliamente extendida y en clase la usaremos, pero debemos tener claro que XML es el metalenguaje que ha creado nuestro lenguaje de marcas.

El término correcto es **APLICACIÓN**.

Aplicación no significa un programa que utilice XML.

Aplicación significa el uso de XML para un dominio específico.

**Aplicación XML
=
Lenguaje de marcado
=
Vocabulario**

XML:

- **No es un lenguaje de marcas y no define una serie de etiquetas.** Es un metalenguaje que permite crear lenguajes de marcas.
- **No es un lenguaje de programación.** No se pueden ejecutar instrucciones a partir de un documento XML.
- **No es un protocolo** de comunicación.
- **No es un sistema gestor de bases de datos.** Aunque pueden existir bases de datos nativas XML.

3. HERRAMIENTAS

Una vez que hemos decidido cómo será nuestro lenguaje necesitamos una serie de herramientas:

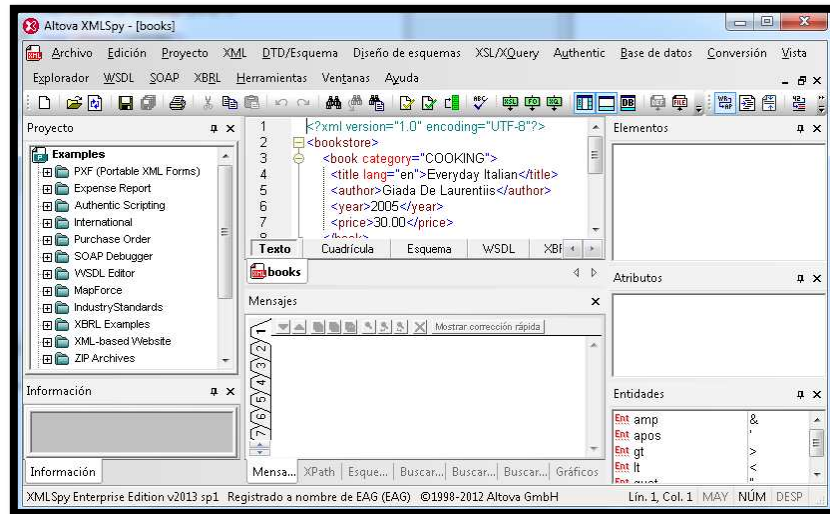
- **Editores:** Nos permiten crear (y editar) los ficheros que vamos a generar usando nuestro lenguaje basado en XML. Puesto que esta tecnología se basa en el uso de fichero de texto plano, cualquier editor de texto es válido: NotePad++, SublimeText, Bluefish, etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

- **Procesadores y analizadores básicos:** Permiten visualizar el fichero que hemos generado y detectar errores básicos. La herramienta más sencilla es cualquier navegador web: Chrome, Mozilla Firefox, Opera, etc.

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

- **Procesadores y analizadores avanzados (incluyen editores):** XML Spy, XML Copy Editor, Editix, etc.



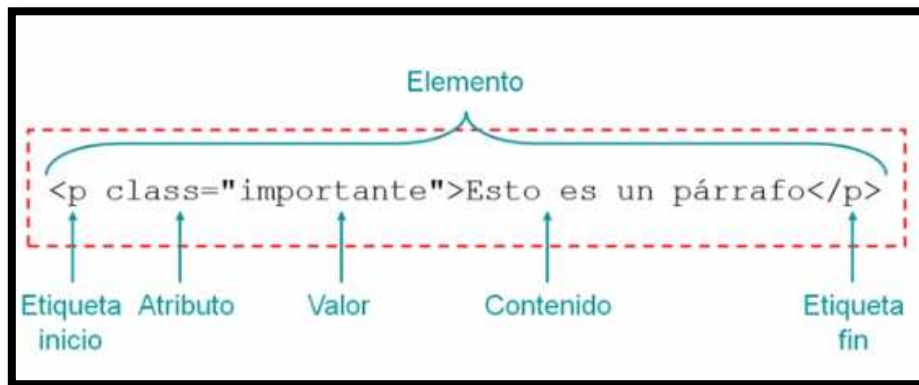
4. ESTRUCTURA Y SINTAXIS DE XML

XML proporciona una serie de normas básicas y una sintaxis para crear lenguajes de marcas para distintos propósitos.

A continuación, para explicar esas normas y esa sintaxis se presentan varios ficheros creados a partir de lenguajes sencillos generados con XML.

4.1. Etiquetas, elementos y atributos.

- **Etiquetas (tag):** Una etiqueta (tag) es un texto que va entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de inicio (como <nombre>) y etiquetas de fin (como </nombre>).
- **Elementos:** Los elementos representan estructuras mediante las que se organizará el contenido del documento. Constan de la etiqueta de inicio, la etiqueta de fin y de todo aquello que se encuentra entre ambas. Algunos elementos no tienen contenido. Se les denomina elementos vacíos y no deben llevar etiqueta de fin.
- **Atributos:** Un atributo es un par nombre-valor que se encuentra dentro de la etiqueta de inicio de un elemento e indican las propiedades que pueden llevar asociadas los elementos.



```
2 <bookstore>
3   <book category="COOKING">
4     <title lang="en">Everyday Italian</title>
5     <author>Giada De Laurentiis</author>
6     <year>2005</year>
7     <price>30.00</price>
8   </book>
9   <book category="CHILDREN">
10    <title lang="en">Harry Potter</title>
11    <author>J K. Rowling</author>
12    <year>2005</year>
13    <price>29.99</price>
14  </book>
15 </bookstore>
```

- **Reglas a la hora de crear elementos (etiquetas y atributos):**

XML Naming Rules

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Nombre de los elementos

– Primer carácter:

[A-Z] | "_" | [a-z]

– Resto:

[A-Z] | "_" | [a-z]

"-" | "." | [0-9]

Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each words except the first

Ejemplo:

- Suponer que hemos creado un lenguaje, "lenguaje_a", que permite almacenar los datos un alumno usando las siguientes etiquetas y atributos:

```
<alumno sexo="varon" fechaNacimiento="5/6/1990">  
  <nombre>Pablo</nombre>  
  <apellido>Pérez</apellido>  
  <telefono tipo="movil">91555555</telefono>  
  <direccion>Ronda de Segovia 111</direccion>  
</alumno>
```

- Suponer que hemos creado otro lenguaje, "lenguaje_b", que también permite almacenar los datos un alumno usando las siguientes etiquetas y atributos:

```
<alumno id="532">  
  <nombre>Pablo</nombre>  
  <apellido>Pérez</apellido>  
  <fechaNacimiento>  
    <dia>5</dia>  
    <mes>6</mes>  
    <año>1990</año>  
  </fechaNacimiento>  
  <sexo>varón</sexo>  
  <telefono tipo="movil">91555555</telefono>  
  <direccion>Ronda de Segovia 111</direccion>  
</alumno>
```

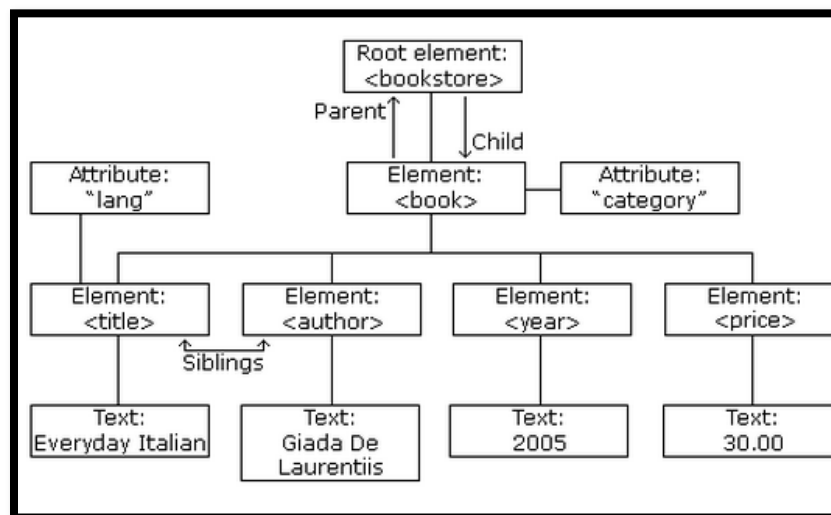
¿Cuál es mejor? ¿Qué recomienda el W3C?

Recomendación:

No usar atributos en exceso y dejarlo exclusivamente para la representación de metadatos. Como, por ejemplo, cuando usamos el atributo "id" o "name" en XHTML.

4.2. Árbol XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <bookstore>
3   <book category="COOKING">
4     <title lang="en">Everyday Italian</title>
5     <author>Giada De Laurentiis</author>
6     <year>2005</year>
7     <price>30.00</price>
8   </book>
9   <book category="CHILDREN">
10    <title lang="en">Harry Potter</title>
11    <author>J K. Rowling</author>
12    <year>2005</year>
13    <price>29.99</price>
14  </book>
15 </bookstore>
```



4.3. Sintaxis XML

- Todas las etiquetas deben cerrarse siempre (tengan o no contenido):

```
2 <!-- BIEN -->
3 <eConContenido>Aquí va el contenido</eConContenido>
4 <eSinContenido />
5 <p>Aquí va el contenido</p>
6 
7 <br />
8
9 <!-- MAL -->
10 <eConContenido>Aquí va el contenido
11 <eSinContenido>
12 <p>Aquí va el contenido
13 
14 <br>
```

- Etiquetas con contenido (no vacías):

```
3 <eConContenido>Aquí va el contenido</eConContenido>
4 <nombre>Juan Alberto</nombre>
5 <p>Soy un párrafo</p>
```

- Etiquetas sin contenido (vacías). Se pueden representar de dos formas:

```
8 <eSinContenido></eSinContenido>
9 <eSinContenido />
10
11 <estudiante></estudiante>
12 <estudiante />
13
14 <input type="button"></input>
15 <input type="button" />
```

- Se distingue entre mayúsculas y minúsculas (Case Sensitive):

```
3 <etiquetaBuena>Contenido</etiquetaBuena>
4 <EtiquetaMala>Contenido</etiquetamala>
```

- La anidación de etiquetas se debe hacer de forma correcta:

```
2 <!-- BIEN -->
3 <etiquetaExterior>
4   Contenido 1
5   <etiquetaInterior>
6     Contenido 2
7   </etiquetaInterior>
8 </etiquetaExterior>
9
10 <!-- MAL -->
11 <etiquetaExterior>
12   Contenido 1
13   <etiquetaInterior>
14     Contenido 2
15   </etiquetaExterior>
16 </etiquetaInterior>
```

- Siempre debe existir una etiqueta root que contendrá a las demás etiquetas:

```
3 <etiquetaPrincipal>
4   <etiquetaExterior>
5     <etiquetaInterior>
6       Contenido
7     </etiquetaInterior>
8   </etiquetaExterior>
9   <etiquetaExterior>
10     <etiquetaInterior>
11       Otro contenido
12     </etiquetaInterior>
13   </etiquetaExterior>
14 </etiquetaPrincipal>
```

- Un atributo siempre está formado por un par nombre-valor. Ese valor siempre irá entre comillas simples o dobles (da igual):

```
2 <!-- BIEN -->
3 <etiquetaExterior atributoE="valor">
4     Contenido 1
5     <etiquetaInterior atributoI="valor">
6         Contenido 2
7     </etiquetaInterior>
8 </etiquetaExterior>
9
10 <!-- MAL -->
11 <etiquetaExterior atributoE=valor>
12     Contenido 1
13     <etiquetaInterior atributoI=>
14         Contenido 2
15     </etiquetaInterior>
16 </etiquetaExterior>
```

- Si dentro de un atributo quieres usar una comilla simple o doble debes hacer uso de estas entidades:

" y '

- En XML existen unos atributos reservados y que empiezan por el prefijo xml:

```
- xml:lang
    • Idioma del elemento
- xml:space: default | preserve
    • Procesamiento normal o se tienen que conservar los espacios en blanco
- xml:id
    • Identificador único del elemento en todo el documento
```

Es decir que los atributos de nuestro lenguaje no pueden empezar por ese prefijo.

- Cuidado al usar caracteres especiales reservados del lenguaje XML:

There are 5 predefined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

```
2 <!-- BIEN -->
3 <unaEtiqueta>Aquí dentro uso el &lt; o el &gt;</unaEtiqueta>
4 <!-- MAL -->
5 <unaEtiqueta>Aquí dentro uso el < y el ></unaEtiqueta>
```

- Comentarios:

```
2 <!-- Esto es un comentario -->
3 <etiquetaExterior atributoE="valor">
4     Contenido 1
5     <etiquetaInterior atributoI="valor">
6         Contenido 2
7     </etiquetaInterior>
8 </etiquetaExterior>
```

- Espacios en blanco: Un fichero XML **SÍ** respeta los espacios en blanco.
- Secciones CDATA (Character DATA): Permite introducir "contenido" que no será interpretado por el procesador de XML.

```
4 <etiquetaPrincipal>
5     <![CDATA[
6         <html>
7             <head>
8                 <title>Rock & Roll</title>
9             </head>
10            </html>
11        ]]>
12    <unaEtiqueta>Aquí dentro uso el &lt; o el &gt;</unaEtiqueta>
13 </etiquetaPrincipal>
```

5. GRAMÁTICA

Cuando se crea un lenguaje de marcas a partir de XML, además de respetar la sintaxis y normas explicadas en el punto anterior, es necesario definirle una gramática a ese lenguaje. Esa gramática define el marcado permitido (etiquetas y atributos) y cómo debe usarse el marcado de ese lenguaje para crear ficheros.

Todo documento que se genere usando ese lenguaje de marcas tiene que cumplir esa gramática.

En XML para definir esas gramáticas se hace uso de otras tecnologías:

- ✓ DTD (Definición de Tipo de Documento).
- ✓ XML Schema (evolución de la DTD).
- ✓ RELAX NG.

6. PARTES DE UN DOCUMENTO

a) Prólogo

Aunque no es obligatorio es muy recomendable su uso y siempre aparece al principio del fichero. Un prólogo está formado por:

- **Una declaración XML.** Es la sentencia que declara al documento como un documento basado en XML. Además, esa declaración debe indicar su versión y la codificación empleada en el fichero.

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

- **Una declaración de tipo de documento.** Es una sentencia que enlaza el documento con sus gramáticas.
- **Instrucciones de procesamiento.** Es una sentencia que indica, por ejemplo, el estilo a aplicar o las hojas de transformación que se van a usar sobre el fichero.

b) Cuerpo

A diferencia del prólogo, el cuerpo no es opcional. El cuerpo debe contener un solo elemento raíz y los datos que almacene dicho fichero XML.

```
2 <note>
3   <date>2008-01-10</date>
4   <to>Tove</to>
5   <from>Jani</from>
6   <heading>Reminder</heading>
7   <body>Don't forget me this weekend!</body>
8 </note>
```

7. DOCUMENTOS XML BIEN FORMADOS

Un documento puede ser correcto a dos niveles:

- Documento bien formado.
- Documento válido.

Los documentos denominados como "bien formados" son aquellos que cumplen con todas las **definiciones básicas de formato** y pueden analizarse correctamente por cualquier analizador de XML.

¡Cuidado! Un documento "bien formado" no es lo mismo que un "documento válido".

Para que un documento esté bien formado debe cumplir las indicaciones dadas en los puntos anteriores:

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas.
- Los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML solo permiten un elemento raíz del que cuelgan todos los demás.
- Los valores de los atributos en XML siempre deben estar encerrados entre comillas simples (o dobles).
- El XML es sensible a mayúsculas y minúsculas.
- Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML y esto se debe tener siempre presente.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas y son partes del documento que el procesador XML espera entender. El resto del documento (el que va entre marcas) son los datos entendibles por las personas (contenido).

8. VALIDACIÓN DE DOCUMENTOS XML

Como ya se ha indicado, que un documento esté "bien formado" solamente se refiere a su estructura sintáctica básica, es decir, que se componga de elementos, etiquetas, atributos y comentarios, y que estos estén escritos como especifica XML.

Cada lenguaje creado a partir de XML necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento. Es decir, debe especificar su gramática.

Un determinado documento **será válido** en un determinado lenguaje si está "bien formado" y cumple la gramática definida para el mismo.

Un documento XML válido es aquel que, además de estar bien formado, está acorde con la especificación de un DTD o de un XSD.

Cuando se válida un documento XML se comprueban los siguientes aspectos:

- Sólo se pueden utilizar elementos y atributos definidos en el DTD o en el XSD.
- Los elementos deben estar en el orden definido en el DTD o en el XSD. Además, se comprueba que los atributos obligatorios estén presentes dentro del documento.
- Si se declaran valores que deben ser únicos, por ejemplo, atributos ID, debe asegurarse que en todo el documento XML se cumple esta condición.
- Si se definió un tipo concreto para los atributos o los elementos o si se definió una serie de restricciones, se deberá comprobar que se cumplan dichos tipos y dichas restricciones.

En la actualidad existen muchas herramientas que nos permiten comprobar si un documento XML está bien formado y es válido.

- a) Herramientas vía web:
- ✓ <http://www.validome.org/>
 - ✓ <http://www.xmlvalidation.com/>
- b) Aplicaciones software: La creación manual de documentos XML e incluso su manipulación pueden introducir todo tipo de errores, tipográficos, sintácticos y de contenido. Como ya sabes existen editores de XML que facilitan la tarea de crear documentos válidos y bien formados, ya que pueden advertir de los errores básicos cometidos e incluso escribir automáticamente la sintaxis más sencilla necesaria. Alguno de los editores más importantes:
- ✓ XMLSpy de Altova
 - ✓ EditiX de JAPISoft SARL
 - ✓ XML Pro de Vervet Logic
 - ✓ Liquid XML Studio de Liquid Technologies
 - ✓ <oxigen/> XML Editor
 - ✓ Turbo XML de TIBCOXML Notepad de Microsoft
 - ✓ XMLwriter de Wattle Software

¡Cuidado!

- Un documento "bien formado" no es lo mismo que un "documento válido".
- Un documento puede estar bien formado, pero puede ser no válido.
- En cambio, un documento válido siempre va a ser un documento bien formado.

9. ESTILOS EN XML

Aunque no es la práctica habitual, se pueden aplicar estilos a los ficheros XML (no olvides que XML está enfocado al intercambio de información y no a la presentación de la misma).

Existen dos tecnologías para para mejorar la presentación del fichero de cara a un usuario final:

- ✓ CSS: Ampliamente conocido y soportando por multitud de herramientas.
- ✓ XSLT: Más complejo y potente que CSS. XSLT es un estándar de la organización W3C que permite transformar documentos XML en otros e incluso permite dar un estilo a dicho documento.

9.1. XML Y CSS

Suponer que queremos aplicar un estilo al fichero "cd_catalog.xml":

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <CATALOG>
3   <CD>
4     <TITLE>Empire Burlesque</TITLE>
5     <ARTIST>Bob Dylan</ARTIST>
6     <COUNTRY>USA</COUNTRY>
7     <COMPANY>Columbia</COMPANY>
8     <PRICE>10.90</PRICE>
9     <YEAR>1985</YEAR>
10  </CD>
11  <CD>
12    <TITLE>Hide your heart</TITLE>
13    <ARTIST>Bonnie Tyler</ARTIST>
14    <COUNTRY>UK</COUNTRY>
15    <COMPANY>CBS Records</COMPANY>
16    <PRICE>9.90</PRICE>
17    <YEAR>1988</YEAR>
18  </CD>
```



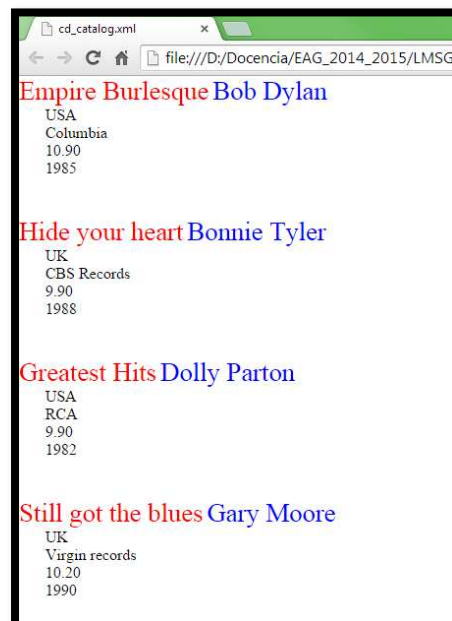
Para ello en primer lugar debemos indicarle en el "prólogo" que se va a aplicar un estilo CSS y dónde se encuentra ese estilo. Para ello se hace uso de `<?xml-stylesheet...>`:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/css" href="css/micss.css"?>
3 <CATALOG>
4   <CD>
5     <TITLE>Empire Burlesque</TITLE>
6     <ARTIST>Bob Dylan</ARTIST>
7     <COUNTRY>USA</COUNTRY>
8     <COMPANY>Columbia</COMPANY>
9     <PRICE>10.90</PRICE>
10    <YEAR>1985</YEAR>
11  </CD>
12  <CD>
```

Después se crea el fichero CSS:

```
1 @charset "UTF-8"
2 CATALOG
3 {
4     background-color: #ffffff;
5     width: 100%;
6 }
7 CD
8 {
9     display: block;
10    margin-bottom: 30pt;
11    margin-left: 0;
12 }
13 TITLE
14 {
15     color: #FF0000;
16     font-size: 20pt;
17 }
```

Y finalmente se visualiza:



10. DTD

Cuando se comparte un documento creado a partir de un lenguaje basado en XML es necesario que ambas entidades conozcan la estructura de dicho fichero: qué etiquetas tienen, en qué orden pueden aparecer, qué atributos tiene cada etiqueta, etc. Es decir, es necesario conocer la gramática de dicho lenguaje.

Existen diversas tecnologías para definir la gramática de un lenguaje basado en XML. Las dos más importantes:

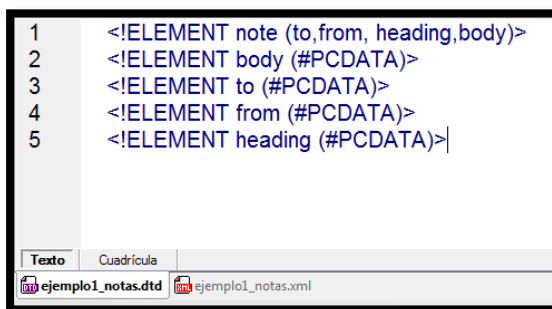
- ✓ DTD
- ✓ XML Schema

Una definición de tipo de documento o DTD (*Document Type Definition*) es una descripción de la estructura y sintaxis de un documento XML.

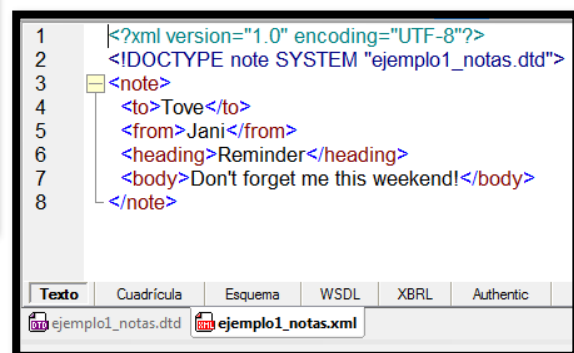
Su función básica es disponer de una estructura común y mantener la consistencia entre todos los documentos que utilicen la misma DTD.

Un fichero DTD **NO** es un fichero XML:

```
1 <!ELEMENT note (to,from, heading,body)>
2 <!ELEMENT body (#PCDATA)>
3 <!ELEMENT to (#PCDATA)>
4 <!ELEMENT from (#PCDATA)>
5 <!ELEMENT heading (#PCDATA)>
```



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE note SYSTEM "ejemplo1_notas.dtd">
3 <note>
4   <to>Tove</to>
5   <from>Jani</from>
6   <heading>Reminder</heading>
7   <body>Don't forget me this weekend!</body>
8 </note>
```

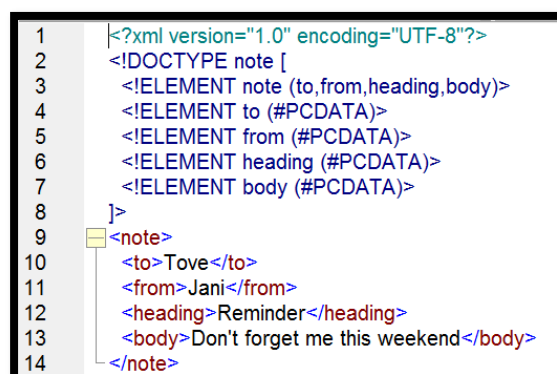


Un DTD describe:

- ✓ **Elementos:** indican qué etiquetas son permitidas, el contenido de dichas etiquetas y sus atributos.
- ✓ **Estructura:** indica el orden en que van las etiquetas en el documento.
- ✓ **Anidamiento:** indica qué etiquetas van dentro de otras.

Ejemplo de DTD dentro del propio documento XML:

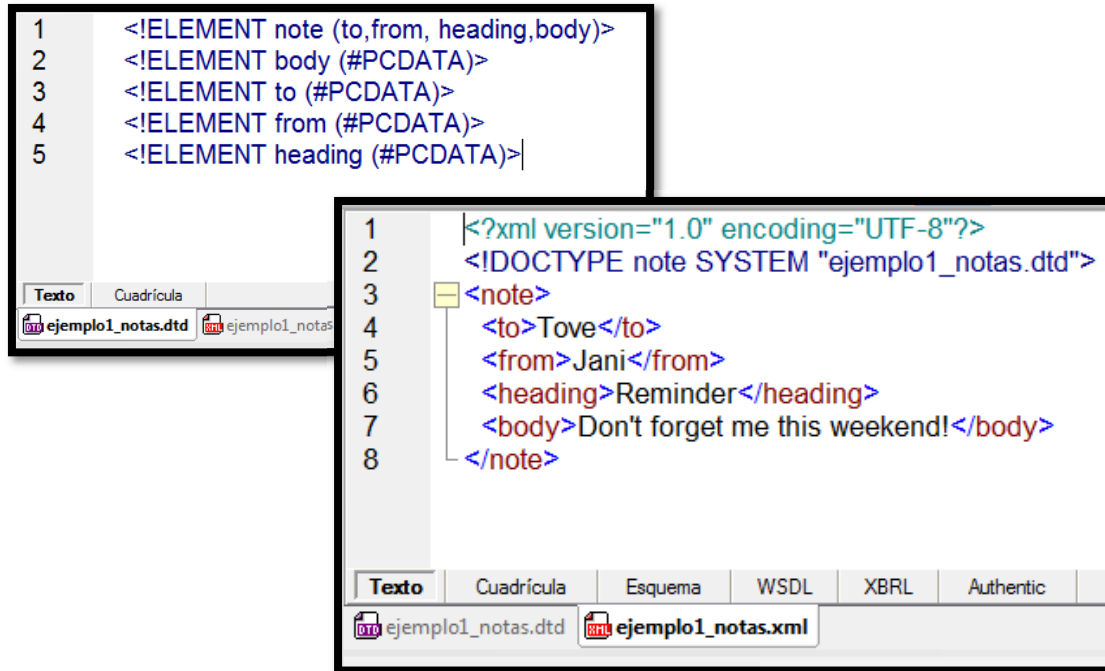
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE note [
3   <!ELEMENT note (to,from,heading,body)>
4   <!ELEMENT to (#PCDATA)>
5   <!ELEMENT from (#PCDATA)>
6   <!ELEMENT heading (#PCDATA)>
7   <!ELEMENT body (#PCDATA)>
8 ]>
9 <note>
10   <to>Tove</to>
11   <from>Jani</from>
12   <heading>Reminder</heading>
13   <body>Don't forget me this weekend!</body>
14 </note>
```



Nota: Los comentarios en DTD son:

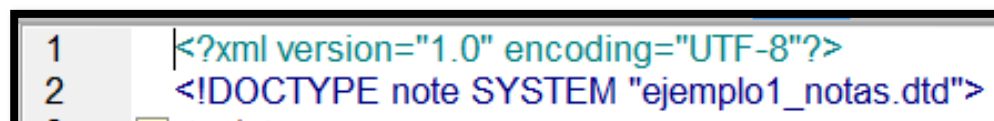
```
<!-- Soy un comentario -->
```

Ejemplo de DTD fuera del propio documento XML:



Observa que el archivo DTD externo **no incluye** la declaración del tipo de documento `<!DOCTYPE ...>` ni el prólogo `<?xml... ?>`. Eso solo debe aparecer en el documento XML.

✓ Ejemplo de DTD fuera del propio documento XML:



Carácter del DTD

- Uso **privado**: Se identifica con la palabra **SYSTEM**. Son DTD definidos por nosotros y que no han sido definidos con carácter público por ninguna organización de estándares.
- Uso **público**. Se identifica con la palabra **PUBLIC**. Son DTD definidos con carácter público por una organización de estándares como el w3c.

10.1. Bloques para construir un DTD

Antes de crear un documento XML se deberá analizar qué elementos, etiquetas y atributos se van a necesitar para el lenguaje que se quiere crear.

Después se crea la gramática sabiendo que un DTD puede estar formado por los siguientes bloques:

- ✓ Element
- ✓ Attlist
- ✓ Entity
- ✓ Notation

✓ Element

Con el DTD podemos definir los elementos y etiquetas que constituyen nuestros documentos XML.

Recuerda que un elemento puede tener contenido o puede no tenerlo. Y recuerda que ese contenido pueden ser otros elementos, o texto, o ambas cosas.

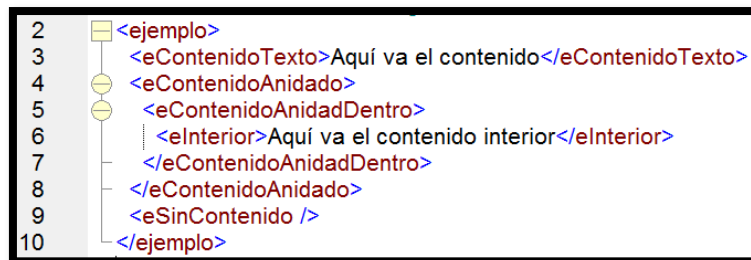


Diagrama de un elemento XML anidado con su estructura de árbol y código DTD:

```
2  <ejemplo>
3    <eContenidoTexto>Aquí va el contenido</eContenidoTexto>
4    <eContenidoAnidado>
5      <eContenidoAnidadDentro>
6        <eInterior>Aquí va el contenido interior</eInterior>
7      </eContenidoAnidadDentro>
8    </eContenidoAnidado>
9    <eSinContenido />
10 </ejemplo>
```

✓ Attlist

Asimismo, un DTD permite definir los atributos que pueden tener los elementos y etiquetas de nuestros documentos XML. Recuerda que los atributos son una manera de incorporar características o propiedades a los mismos.

Los elementos y etiquetas pueden tener cero, uno o varios atributos.

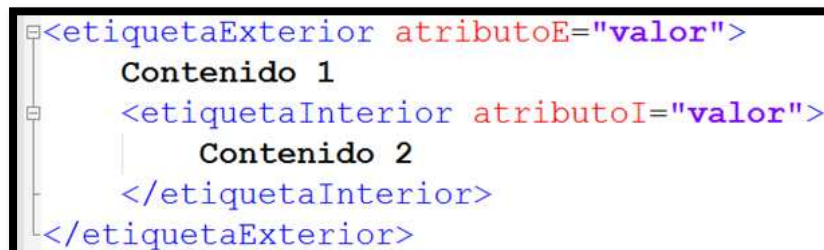


Diagrama de un elemento XML con atributos y su estructura de árbol:

```
<etiquetaExterior atributoE="valor">
  Contenido 1
  <etiquetaInterior atributoI="valor">
    Contenido 2
  </etiquetaInterior>
</etiquetaExterior>
```

✓ **Entity**

Un DTD también permite definir nuestras propias entidades. Una entidad asocia un nombre con un fragmento de contenido.

Las entidades ofrecen una serie de ventajas: facilita la escritura y la lectura, disminuye las posibilidades de error, clarifica la estructura y facilita el mantenimiento.

Las entidades pueden ser:

- Internas o externas.
- Analizadas (*parsed*) o no analizadas (*unparsed*).
- Generales o paramétricas.
- De carácter.

✓ **Notation**

Permite indicar el "formato de los datos" que NO estén en formato XML.

10.2. Elementos en un DTD

La declaración de los elementos se hace usando la siguiente sintaxis:

`<!ELEMENT nombre_elemento contenido >`

Donde contenido indica qué contendrá ese elemento. Puede ser:

- (elemento1, elemento2, ...): Una lista de elementos hijos.
- (#PCDATA): Texto para elementos con contenido de tipo texto.
- EMPTY: Vacío para elementos sin contenido.
- ANY: Cualquier contenido existente para elementos que pueden tener cualquier contenido ya definido.
- Mixto: Elementos y texto.

Nota: Recuerda que el nombre del elemento **SIEMPRE** debe ser válido:

- Primer carácter: letra, `_`, `:`
- Resto: letra, `_`, `:`, dígitos, `.` o `-`

✓ **Declaración de un elemento con hijos:**

```
<!ELEMENT note (to,from,heading,body) >
```

```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend</body>  
</note>
```

Observa que se usan los paréntesis '(' ')' y que cada hijo va separado por comas ','

✓ **Declaración de un elemento con hijos - Secuencia.**

Cada hijo debe aparecer una vez, de forma obligatoria y en el orden en que se indica:

```
<!ELEMENT note (to,from,heading,body) >
```

✓ **Declaración de un elemento con hijos - Selección.**

Puede aparecer un hijo u otro |:

```
<!ELEMENT note (to,from,header,( message | body) ) >
```

✓ **Declaración de un elemento con hijos - Número de ocurrencias.**

a) Cada hijo debe aparecer una sola vez:

```
<!ELEMENT note (message) >
```

b) Cada hijo debe aparecer al menos una vez, pero puede aparecer más veces +:

```
<!ELEMENT note (message+) >
```

- c) Cada hijo debe aparecer cero o más veces *:

```
<!ELEMENT note (message*) >
```

- d) Cada hijo puede aparecer una vez o ninguna ?:

```
<!ELEMENT note (message?) >
```

✓ Declaración de un elemento con hijos - Número de ocurrencias.

Ejemplo

```
<!ELEMENT algo (a, b?, c+, d*)>  
  
<algo>  
<a /><c /><c /><c /><d /><d /><d />  
</algo>
```

Ejemplo

```
<!ELEMENT algo ((a | b), c?)>  
  
<algo>  
<a /><b />  
<c />  
</algo>
```

Ejemplo

```
<!ELEMENT algo ((a | b)+, c?)>  
  
<algo>  
<a /><b /><a /><a /><b /><c />  
</algo>
```

Ejemplo

```
<!ELEMENT algo ((a, b)+, (c | d)+)>  
  
<algo>  
<a /><b /><a /><c /><d />  
</algo>
```


- ✓ **Declaración de un elemento cuyo contenido es solo texto:**

```
<!ELEMENT body (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
```

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Observa que se usan los paréntesis '(' ')' y que dentro se usa la palabra reservada #PCDATA

¿Qué ocurre en este ejemplo?

```
1 <!ELEMENT padre (nombre, hijo+)>
2 <!ELEMENT nombre (#PCDATA)>
3 <!ELEMENT hijo (nombre)>
4 <!ELEMENT nombre (#PCDATA)>
```

```
3 [ ] <padre>
4 |   <nombre>Juan</nombre>
5 |   [ ] <hijo>
6 |   |   <nombre>José</nombre>
7 |   |   </hijo>
8 |   </padre>
```

El documento está bien formado, pero no es válido.

- ✓ **Declaración de un elemento vacío (sin contenido):**

```
...
<!ELEMENT persona (nombre, (soltero|casado))>
...
<!ELEMENT soltero EMPTY>
<!ELEMENT casado EMPTY>
...
```

```
<personas>
  <persona>
    <nombre>Sergio</nombre>
    <casado />
  </persona>
  <persona>
    <nombre>Cristiano</nombre>
    <soltero />
  </persona>
</personas>
```

✓ **Declaración de un elemento de cualquier tipo:**

ANY permite indicar que cualquiera de los elementos ya conocidos serían válidos para dicho elemento:

<!ELEMENT other ANY>

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE note [
3    <!ELEMENT note (to,from,heading,body,other)>
4    <!ELEMENT to (#PCDATA)>
5    <!ELEMENT from (#PCDATA)>
6    <!ELEMENT heading (#PCDATA)>
7    <!ELEMENT body (#PCDATA)>
8    <!ELEMENT other ANY>
9  ]>
```

```
10  <note>
11    <to>Tove</to>
12    <from>Jani</from>
13    <heading>Reminder</heading>
14    <body>Don't forget me this weekend</body>
15    <other>
16      <to>John</to>
17      <heading>Reminder - Copy</heading>
18      <body>Don't forget me this weekend</body>
19    </other>
20  </note>
```

✓ **Declaración de un elemento mixto.**

Un elemento mixto es un elemento que puede contener texto y otras etiquetas anidadas. Se declara usando la siguiente notación:

<!ELEMENT elementoMixto (#PCDATA | unSubElemento | otroSubElemento) *>

```
1  <elementoMixto>
2      Texto
3      <unSubElemento>Un texto</unSubElemento>
4      <otroSubElemento>Otro texto</otroSubElemento>
5  </elementoMixto>
```

Nota: #PCDATA siempre debe ser lo primero que aparezca en la lista. Cada elemento se separa por | y todos los elementos están afectados por el *

10.3. Atributos en un DTD

En la declaración de atributos se usa la palabra reservada: ATTLIST

```
<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorAtributo>
```

```
<!ATTLIST nombreElemento
    nombreAtributo1 tipoAtributo1 valorAtributo1
    nombreAtributo2 tipoAtributo2 valorAtributo2>
```

Ejemplos:

```
<!ATTLIST persona dni ID #REQUIRED>
<!--<!ATTLIST direccion provincia CDATA #IMPLIED>
<!ATTLIST telefono tipo (fijo|movil) "fijo">
```

```
<persona dni="W30123456">
  ..
  <direccion provincia="Madrid">
    <calle>Melancolia</calle>
    ...
  </direccion>
  <telefono tipo="fijo">958456732</telefono>
  ...
</persona>
```

Ejemplos con errores:

```
<!ATTLIST persona dni ID #REQUIRED>

<!ATTLIST direccion provincia CDATA #IMPLIED>

<!ATTLIST telefono tipo ("fijo" | "movil") "fijo">
```

```
<persona dni="30123456W">
  ..
  <direccion provincia="Madrid">
    <calle>Melancolia</calle>
    ...
  </direccion>
  <telefono tipo="fijo">958456732</telefono>
  ...
</persona>
```

✓ **Tipos de atributos:**

<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorAtributo>

Type	Description
CDATA	The value is character data
(en1 en2 ..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

- Cuando se usa ID, el identificador debe tener un valor válido.

Primer carácter: letra, '_', ':', '@'
Resto: letra, '_', ':', '@', dígitos, '.', '-'

- Cuando se usa IDREF o IDREFS, los identificadores a los que se apunta deben existir en el documento.
- NMTOKEN y CDATA son muy parecidos. La diferencia es que cuando se usa NMTOKEN, el atributo sólo contiene letras, dígitos, y los caracteres punto ".", guión "-", subrayado "_" y dos puntos ":"

✓ **Valores de atributos:**

<!ATTLIST nombreElemento nombreAtributo tipoAtributo valorAtributo>

The **attribute-value** can be one of the following:

Value	Explanation
<i>value</i>	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED <i>value</i>	The attribute value is fixed

Nota: Siempre hay que especificar un valor de atributo, o bien por defecto, o bien si es obligatorio, o bien si es opcional o bien si es fijo.

Otros ejemplos:

DTD:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />

DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />

DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
<sender company="Microsoft" />

Invalid XML:
<sender company="W3Schools" />

DTD:
<!ATTLIST payment type (check|cash) "cash">

XML example:
<payment type="check" />
or
<payment type="cash" />

```
1      <?xml version="1.0" encoding="UTF-8"?>
2      <!DOCTYPE directorio [
3          <!ELEMENT directorio (persona)+ >
4          <!ELEMENT persona (#PCDATA) >
5          <!ATTLIST persona id ID #REQUIRED
6              | madre IDREF #IMPLIED
7              | padre IDREF #IMPLIED>
8      ]>
9      <directorio>
10         <persona id="p1">Pedro</persona>
11         <persona id="p2">Marisa</persona>
12         <persona id="p3" madre="p2" padre="p1">Carmen</persona>
13     </directorio>
```

10.4. Entidades en un DTD

Un usuario puede definir entidades propias para que, después de analizar el documento XML, se sustituyan por el valor indicado. Es como una especie de definición preestablecida.

Declaración:

```
<!ENTITY nombreEntidad "valorEntidad">
```

Uso dentro del XML:

```
&nombreEntidad;
```

Nota: Recuerda que existen distintos tipos de entidades, pero las entidades más usadas en nuestros ejemplos serán entidades internas, analizadas y generales.

✓ Entidades internas, analizadas y generales:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

```
<book>
  <title>...</title>
  ...
  <author>&writer;</author>
  ...
  <copy>&copyright;</copy>
</book>
```

✓ **Otros tipos:**

a) Externas:

```
<!ENTITY writer SYSTEM "http://www.w3schools.com/entities.dtd">  
<!ENTITY copyright SYSTEM "http://www.w3schools.com/entities.dtd">
```

b) Paramétricas (usan % en lugar de &):

```
<!ENTITY % fontstyle "TT | I | B | BIG | SMALL">  
<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; | %special; | %formctrl;">
```

La diferencia entre entidades generales y paramétricas es que las entidades paramétricas se sustituyen por su valor en todo el documento (incluso en el propio DTD) mientras que las generales no se sustituyen en el DTD.

c) Entidades de carácter:

```
<!ENTITY lt "&#38;#60;">  
<!ENTITY gt "&#62;">  
<!ENTITY amp "&#38;#38;">  
<!ENTITY apos "&#39;">  
<!ENTITY quot "&#34;">
```

Recuerda que XML ya define estas entidades porque representan a los caracteres especiales reservados:

Entity References
<
>
&
"
'

11. ESPACIOS DE NOMBRES

En muchas ocasiones los documentos XML creados con un determinado lenguaje (lenguaje A) se suelen combinar con otros documentos XML creados con otro lenguaje (lenguaje B) para crear un único documento.

Esta es una de las ventajas de XML y gracias a ello el desarrollador puede reutilizar código existente que ya ha sido depurado y probado.

Por ejemplo, podemos usar XHTML y MathML para crear un documento que permita almacenar formulas matemáticas y contenido web. También podemos combinar XHTML y SVG para crear un documento que permita almacenar gráficos vectoriales y contenido web.

```
1 <svg width="500" height="200">
2   <circle cx="50" cy="50" r="40"
3         stroke="green" stroke-width="4"
4         fill="yellow" />
5 </svg>
```

```
2 <html>
3   <head>
4     <title>Ejemplo</title>
5   </head>
6   <body>
7     <h1>Un ejemplo</h1>
8     <p>Aquí va un ejemplo</p>
9     <div id="figuras">
10
11   </div>
12 </body>
13 </html>
```

```
2 <html>
3   <head>
4     <title>Ejemplo</title>
5   </head>
6   <body>
7     <h1>Un ejemplo</h1>
8     <p>Aquí va un ejemplo</p>
9     <div id="figuras">
10       <svg width="500" height="200">
11         <circle cx="50" cy="50" r="40"
12               stroke="green" stroke-width="4"
13               fill="yellow" />
14       </svg>
15     </div>
16 </body>
17 </html>
```



El problema que surge en estos casos es la posible colisión que se puede producir en el nombre de los elementos. Es decir, lenguaje A puede definir una etiqueta que tenga el mismo nombre que otra etiqueta del lenguaje B, pero que tenga un significado completamente diferente.

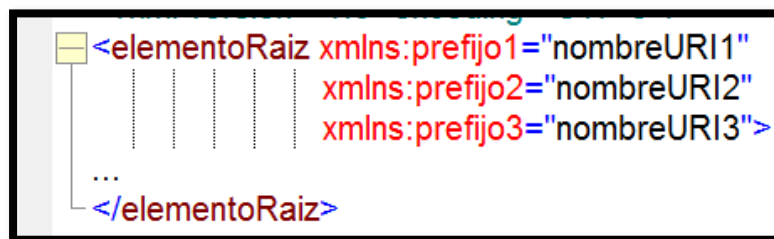
Supongamos que tenemos un documento XML con libros y otro documento XML con discos y queremos mezclar ambos documentos. En este caso habrá elementos que no se llamen igual, pero otros que sí (título, fecha, etc.). Para esos elementos, la gramática del primer documento entraría en conflicto con la del segundo documento. Puede ser que la fecha en libros solo almacene el número del año en que se publicó y que la fecha en discos almacena el día, mes y año usando tres etiquetas.

Para evitar este problema, XML proporciona un mecanismo denominado "espacio de nombres". Este mecanismo asocia un prefijo a los elementos del lenguaje A y otro prefijo a los elementos del lenguaje B. Cada prefijo tendrá asociada una referencia global y única para que no entren en conflicto entre sí.

Por tanto, los espacios de nombres proporcionan un método simple para cualificar los nombres de elementos y atributos usados por distintos lenguajes XML en un mismo documento.

El espacio de nombres se declara usando el atributo de XML "xmlns:" en el nodo raíz del documento (generalmente se hace en el nodo raíz, pero puede hacerse en cualquier elemento).

Tiene la siguiente sintaxis:



El diagrama muestra un elemento raíz XML con tres atributos xmlns: que declaran tres espacios de nombres diferentes. Los atributos están escritos en rojo y azul. El elemento raíz está etiquetado como 'elementoRaiz' en azul. Hay tres líneas de puntos entre los atributos, indicando que pueden haber más. El elemento se cierra con una etiqueta de cierre en azul.

```
<elementoRaiz xmlns:prefijo1="nombreURI1"
               xmlns:prefijo2="nombreURI2"
               xmlns:prefijo3="nombreURI3">
  ...
</elementoRaiz>
```

Donde:

- **prefijo:** será un prefijo que permita identificar si los datos a analizar son de un documento u otro.
- **nombreURI:** el valor del atributo nombreURI debe ser una URI que apunte al espacio de nombre definido para ese lenguaje.

Habrá que declarar tantos prefijos como lenguajes tengamos.

En dicha declaración se asocia una URI (que define el espacio de nombres de cada lenguaje) a cada prefijo.

Muy importante: El nombre URI no necesita ser una URI real, pero la práctica lo más habitual es utilizar en la ubicación URI de dicho espacio de nombres un DTD real.

11.1. Espacios de nombres por defecto

Para indicar un espacio de nombres por defecto se elimina el prefijo en el atributo *xmlns:* de uno de los lenguajes. Se indica así que un elemento (y todos sus descendientes) sin prefijo van a pertenecer al espacio de nombres que no tiene asignado prefijo.

Se consigue con código más limpio.

```
<?xml version="1.0"?>
<cliente xmlns='http://es.wikipedia.org/wiki/Espacio_de_nombres_XML/cliente'
  xmlns:ped='http://es.wikipedia.org/wiki/Espacio_de_nombres_XML/pedido'>
  <numero_ID>1232654</numero_ID>
  <nombre>Nombre</nombre>
  <telefono>99999999</telefono>
  <ped:pedido>
    <ped:numero_ID>6523213</ped:numero_ID>
    <ped:articulo>Caja de herramientas</ped:articulo>
    <ped:precio>187,90</ped:precio>
  </ped:pedido>
</cliente>
```

Nombre	URI	Descripción
XSLT	http://www.w3.org/1999/XSL/Transform	XSL - Transformations
XSL-FO	http://www.w3.org/1999/XSL/Format	XSL - Formatting
XML Schema	http://www.w3.org/2001/XMLSchema	
SVG	http://www.w3.org/2000/svg	Scalable Vector Graphics
RDF	http://www.w3.org/1999/02/22-rdf-syntax-ns	Resource Description Format
SMIL 2.0	http://www.w3.org/2001/SMIL20/	Synchronized Multimedia Integration Lang.
VoiceXML 2.0	http://www.w3.org/2001/vxml	Synthesized voice
XLink	http://www.w3.org/1999/xlink	XML Linking Language
XForms	http://www.w3.org/2002/xforms	Next generation of HTML forms.
XHTML	http://www.w3.org/1999/xhtml	
MathML	http://www.w3.org/1998/Math/MathML	Mathematical markup language