Creando Documentos XML válidos. Ficheros DTD

Contenido

1. Introducción.

En el tema anterior se dispusieron las bases del funcionamiento de XML. En ese mismo tema distinguimos lo que se considera un documento XML **bien formado**. Los documentos bien formados, aseguran que las reglas de XML se cumplen y que no hay ninguna incoherencia al usar el lenguaje.

Sin embargo, aun los documentos XML bien formados darían problemas a la hora de elaborar documentos XML propios, ya que los documentos no tienen por qué mantener homogeneidad y sin embargo estar bien formados. Sin embargo, esa homogeneidad es vital para poder manipular documentos XML.

Por ejemplo, supongamos que disponemos de un servicio de Internet que permite informar sobre el tiempo que hace en nuestra zona. Supongamos también que entregamos esa información en XML. Para ello podríamos elaborar un XML con el tiempo de cada día y desde Internet cuando se solicite un día concreto, entregar el XML correspondiente.

Para que ese servicio sea útil, la estructura del XML de cada día debe ser igual, de otro modo los servicios que requieren nuestra información no funcionarían adecuadamente. Así si en nuestro XML indicamos la fecha y luego la temperatura, habrá que mantener ese orden siempre.

De ahí sale la idea de **validación**. Un documento XML es válido si ha pasado las reglas definidas en otro documento llamado documento de validación. De modo que, un documento XML que sea válido deberá indicar qué plantilla de reglas utiliza y deberá cumplirlas a rajatabla.

Así explicado parece que la validación supone un problema, pero en realidad es una ventaja; con la validación tenemos la seguridad de que los documentos cumplen unas reglas más concretas y de esa forma es fácil establecer un protocolo en las empresas para sus documentos. En definitiva, la validación permite establecer lenguajes propios de marcado para nuestros documentos.

Principales lenguajes de validación

Las técnicas más populares para definir validaciones son:

- **DTD**, *Document Type Definition*. Validación por documentos de definición de tipos. Se utilizaba en el lenguaje SGML y se permite su uso con XML. Es la más utilizada, pero tiene numerosas voces críticas porque su sintaxis no es XML.
- **XML Schema** o esquemas XML. Mucho más coherente con el lenguaje XML. Es la aconsejada actualmente, pero todavía no tiene una implantación al 100%
- **Relax NG**. Es una notación sencilla y fácil de aprender que está haciéndose muy popular. No tiene tantas posibilidades como XML Schema, pero tiene una sintaxis más sencilla. Además, admite añadir instrucciones de tipo XML Schema por lo que se convierte en una de las formas de validación más completas.
- **Schematron**. Permite establecer reglas que facilitan establecer las relaciones que han de cumplir los datos de un documento XML. No es tan bueno para establecer el resto de reglas de validación (orden de elementos, tipos de datos…).

2. Objetivos.

En esta unidad pretendemos que el alumno consiga los siguientes objetivos:

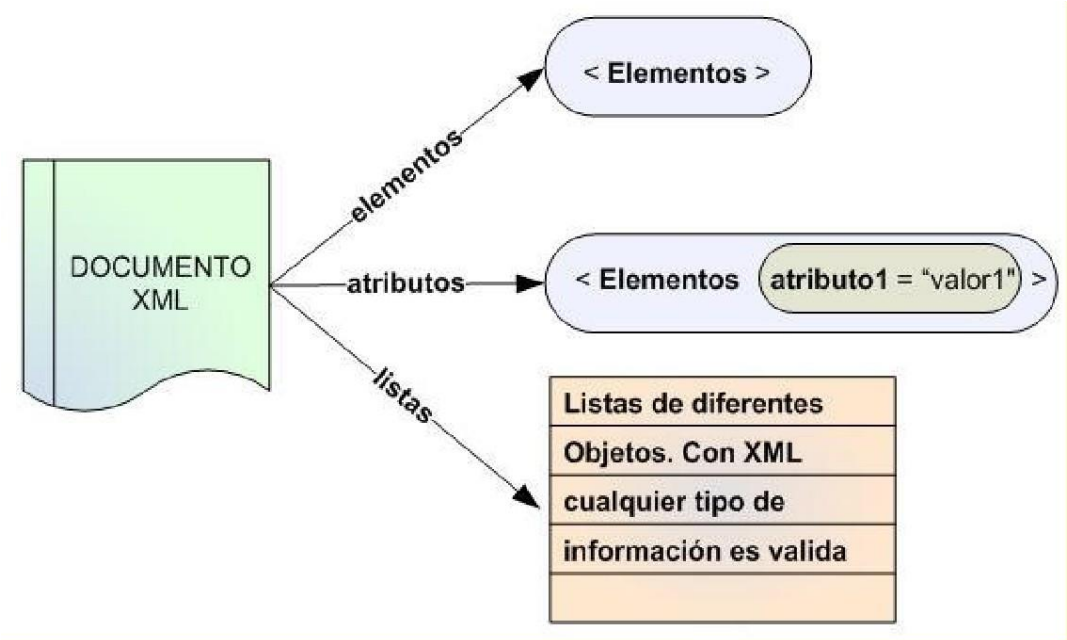
1. Conocer qué es una definición de tipo de documento, conocida como **DTD**.
2. Construir ficheros **DTD** (*Document Type Definition*), para contener la sintaxis y la estructura que determina la validez de un documento XML.
3. Se mostrará cómo podemos declarar elementos dentro de una **DTD** así como la forma de incluir atributos de distintos tipos.

Una vez terminada la unidad, el alumno será capaz de crear ficheros **DTD** (Document Type Definition) y de comprender la separación que establece XML entra la sintaxis de un documento y la estructura del mismo.

3. Especificación de elementos.

3.1. Introducción.

Dado el carácter de metalenguaje de XML, cuando creamos nuestro propio lenguaje, existe una regla básica que consiste en crear un diccionario que defina ese lenguaje. Para ello, usaremos los ficheros **DTD** (*Document Type Definition*). Una **DTD** consiste en una definición que especifica las restricciones que debe tener la estructura de un documento. Su función es la de delimitar todas y cada una de las partes de las que consta un documento XML:



La DTD que debe utilizar el procesador XML para validar el documento XML se indica mediante la etiqueta DOCTYPE. La DTD puede estar incluida en el propio documento (DTD interna), ser un documento externo (DTD externa) o combinarse ambas.

Si la DTD es interna, puede incluirse en el propio archivo XML con la siguiente sintaxis:

```
<!DOCTYPE elemento-raíz [ declaraciones ]>
```

La interpretación de esta DTD es:

- **!DOCTYPE note**: indica que note es el elemento raíz del documento
- **!ELEMENT note**: define el elemento "note" que a su vez está compuesto por los cuatro elementos: "to", "from", "heading" y "body".

En caso de que la DTD sea externa, la declaración de tipo de documento (DTD) estará en un fichero externo. La definición <!DOCTYPE> del archivo XML debe contener la referencia al archivo .dtd que contiene la declaración de tipo de documento. En este caso la sintaxis es la siguiente:

```
<!DOCTYPE elemento-raíz SYSTEM "fichero.dtd">
```

Siguiendo el ejemplo anterior, quedaría así:

Archivo XML con DTD externa:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Archivo note.dtd que es el que contiene la declaración de tipo de documento:

```
<?xml version="1.0"?>
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Posteriormente, se deben declarar los distintos elementos (y opcionalmente atributos) que podrá tener el documento XML que aplique esa DTD.

Una declaración de tipo elemento debe comenzar con la cadena <!ELEMENT

```
<!ELEMENT coche (modelo, potencia, cilindrada)>
```

Como vemos, se ha declarado un elemento de nombre 'coche' que a su vez puede

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note    (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

contener dentro los elementos hijo modelo, potencia y cilindrada. De la misma forma, estos elementos deberán estar definidos posteriormente en la DTD. Con esta definición, veamos ahora un ejemplo de documento XML válido y no válido:

```
<?xml version="1.0" enconding="UTF-8"?>
<coche>
     <modelo>BMW 530</modelo>
     <potencia>232C.V.</potencia>
     <cilindrada>3000c.c</cilindrada>
</coche>
```

El ejemplo siguiente no es válido porque hemos incluido un elemento hijo (combustible) en el mismo, que no estaba definido en la DTD.

```
<?xml version="1.0" enconding="UTF-8"?>
<coche>
     <modelo>BMW 530</modelo>
     <potencia>232C.V.</potencia>
     <cilindrada>3000c.c</cilindrada>
     <combustible>Gasolina Sin plomo 98 octanos<combustible>
</coche>
```

3.2. Variantes de la especificación de contenido.

La especificación de contenido para la definición de un elemento en una DTD admite algunas variantes que veremos a continuación.

3.2.1 ANY

Permite tener cualquier contenido. No es recomendable usarlo, porque puede causar errores de estructura en el documento XML.

```
<!ELEMENT cualquier_cosa ANY>
```

3.2.2. EMPTY

Permite que el elemento no tenga contenido. Su uso suele limitarse a los atributos, aunque se pueden encontrar excepciones.

```
<!ELEMENT salto_de_linea EMPTY>
```

3.2.3. #PCDATA

Ya sabemos que cualquier elemento puede aparecer dentro de un documento XML, siempre y cuando este haya sido definido correctamente. La especificación de #PCDATA en un elemento indica que podemos incluir cualquier tipo de carácter que no sea ya una etiqueta. Veamos un ejemplo:

```
(En la DTD)

<!ELEMENT GRUPO (#PCDATA)>
-----------------------------
(En el documento XML)
<GRUPO>Ciencias Biológicas</GRUPO>
<GRUPO>32M y 33M</GRUPO>
<GRUPO>52314</GRUPO>
```

Sin embargo, especificaciones como las que mostramos a continuación serán incorrectas por el hecho de incluir subelementos o elementos hijo, dentro de un elemento previamente declarado como #PCDATA:

```
<GRUPO>
      <CLASE>Materiales</CLASE>
      <CLASE>Estructuras</CLASE>
</GRUPO>
```

El **orden no es relevante** cuando declaramos varios elementos en una DTD. Veamos dos posibles declaraciones y un ejemplo sencillo para ambas.

```
(En la DTD)

<!ELEMENT GRUPO (#PCDATA)>
<!ELEMENT CLASE ANY>
```

```
(En la DTD)

<!ELEMENT CLASE ANY>
<!ELEMENT GRUPO (#PCDATA)>
```

```
(En el documento XML)

<CLASE>
     <GRUPO>Ciencias Físicas</GRUPO>
</CLASE>
```

3.2.4. Elementos hijos

A la hora de construir una DTD también está permitido declarar elementos hijos que nos pueden ayudar a tener una idea más exacta de la estructura de los datos. Para

ello, se declara el elemento hijo entre paréntesis seguido del elemento padre. A continuación podemos ver un ejemplo:

```
<!ELEMENT CLASE (GRUPO)>
<!ELEMENT GRUPO (#PCDATA)>
```

Como vemos en el ejemplo, el orden no es relevante, ya que hemos situado la declaración de GRUPO después de la declaración de CLASE. XML permite hacer referencias a elementos que aún no han sido declarados.

- **UNO O MÁS HIJOS.** Tal y como se ha definido el ejemplo anterior, sólo podría haber un elemento Grupo dentro de un elemento Clase. Esto nos limitaría notablemente las posibilidades de desarrollo de nuestra DTD. Afortunadamente, existe un mecanismo que nos permite especificar la introducción de más de un elemento hijo. Para ello, se introduce el símbolo **suma (+)**, que hará que podamos incluir cualquier cantidad de elementos grupo. La sintaxis es la siguiente:

  ```
  <!ELEMENT CLASE (GRUPO+)>
  <!ELEMENT GRUPO (#PCDATA)>
  ```

  Con esta sintaxis nos aseguramos de que Clase contendrá al menos un grupo, pudiendo tener un número ilimitado de grupos.

- **CERO O MÁS HIJOS**. Si nos ceñimos a lo que simboliza el ejemplo, es posible que una clase no tenga ningún grupo por diversas razones. Con la sintaxis vista anteriormente, hemos concluido que como mínimo deberá existir un elemento de tipo grupo. Si lo que deseamos es tener la posibilidad de que no exista ningún elemento, usaremos el **símbolo asterisco (*)**. Con este símbolo, podremos tener desde cero a un número ilimitado de elementos.

  ```
  <!ELEMENT CLASE (GRUPO*)>
  <!ELEMENT GRUPO (#PCDATA)>
  ```

- **CERO O UN HIJO**. Finalmente, las DTD nos proporcionan la posibilidad de declarar un tipo de elemento que puede no aparecer o en caso de aparecer, hacerlo únicamente una vez. Esto puede sernos muy útil por ejemplo en un caso en el que creamos diferentes hijos con características del padre. De esta forma, puede darse el caso de que una de las características no proceda en un determinado caso. Para ello se usará el **símbolo de interrogación (?)** detrás del nombre del elemento.

  Veamos un ejemplo:

```
<!ELEMENT COCHE(MODELO,CILINDRADA?,POTENCIA?,PRECIO?)>
<!ELEMENT MODELO(#PCDATA)>
<!ELEMENT CILINDRADA(#PCDATA)>
<!ELEMENT POTENCIA(#PCDATA)>
<!ELEMENT PRECIO(#PCDATA)>
```

Con este ejemplo, podemos describir las características (modelo, cilindrada, potencia y precio) de un coche con la posibilidad de omitir alguna de ellas. Por otro lado, si la incluimos, solo tiene sentido referenciarla una única vez, ya que si no, no tendría sentido. En el ejemplo planteado, no tiene sentido hablar de más de un precio o más de una cilindrada para un modelo concreto de coche, por lo que dichos elementos hijo llevarán el símbolo interrogante (?) para indicar esta circunstancia.

- **ELECCIÓN ENTRE DOS O MÁS HIJOS**. Hasta ahora hemos visto que un elemento puede contener varios hijos, así como distintos modificadores para indicar la cardinalidad de esos hijos. En algunas ocasiones, no tiene sentido que, en la estructura de un elemento, aparezcan todos los hijos, sino que lo correcto es elegir sobre las distintas posibilidades. Las DTD ofrecen un mecanismo para este tipo de casos en los que la declaración de un elemento permite la elección de un único hijo. Para ello se utiliza una barra vertical (|) como separación de dichos elementos hijo. Veamos un ejemplo para aclarar el concepto:

```
<!ELEMENT COCHE(Berlina | Coupé | Utilitario)>
```

3.2.5. Contenido Mixto

Hasta ahora hemos visto que una definición de elemento contenía una lista de elementos hijos con distintas cardinalidades o un conjunto de caracteres (tipo #PCDATA), pero nunca hemos visto ejemplos que incluyan ambos contenidos. Las DTD permiten esta posibilidad. Su uso suele ser bastante frecuente en las páginas que tienen un alto contenido de texto como por ejemplo un periódico. A continuación, se muestra un ejemplo de contenido mixto:

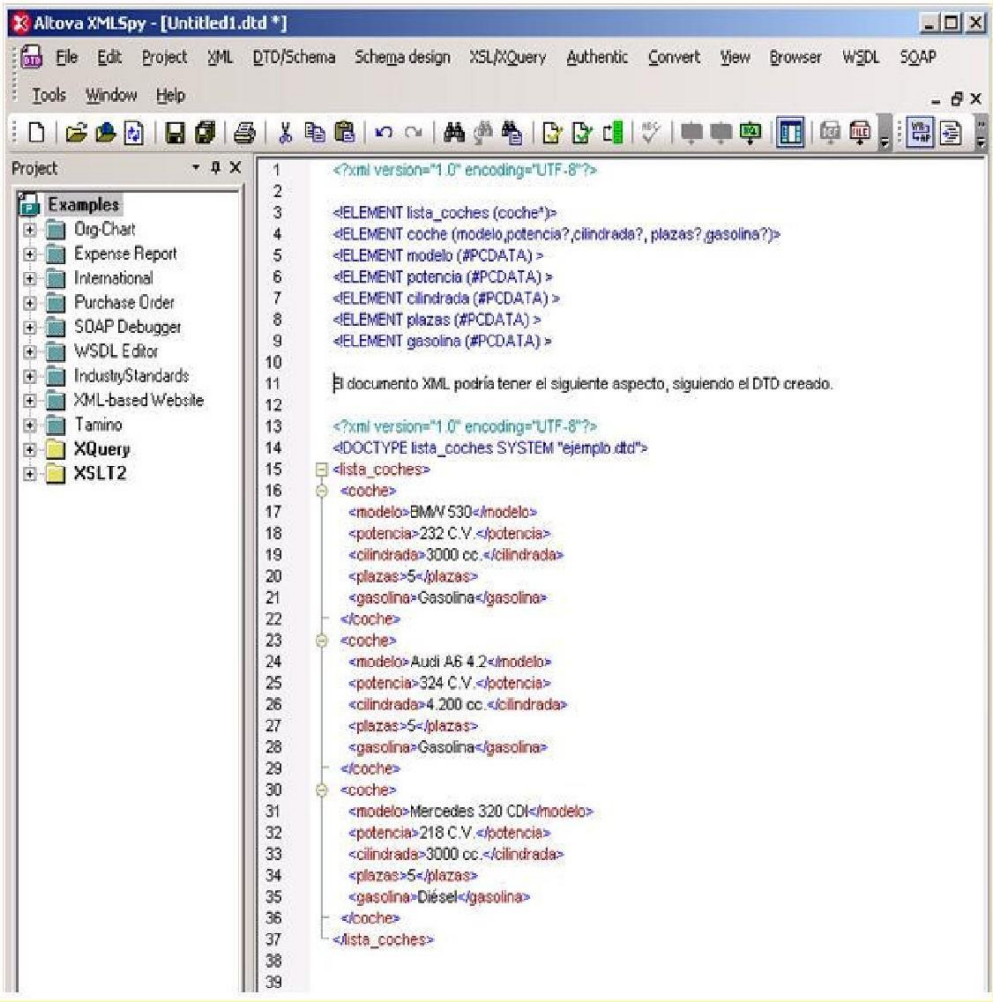
```
<!ELEMENT COCHE (#PCDATA | MODELO)*>
```

De esta forma, se ha definido un coche como un elemento que puede tener una descripción textual o un elemento hijo (modelo).

Cuando realizamos este tipo de definiciones que incluyen declaraciones de tipo mixto, no está permitido usar comas, signos de interrogación, asteriscos, etc.

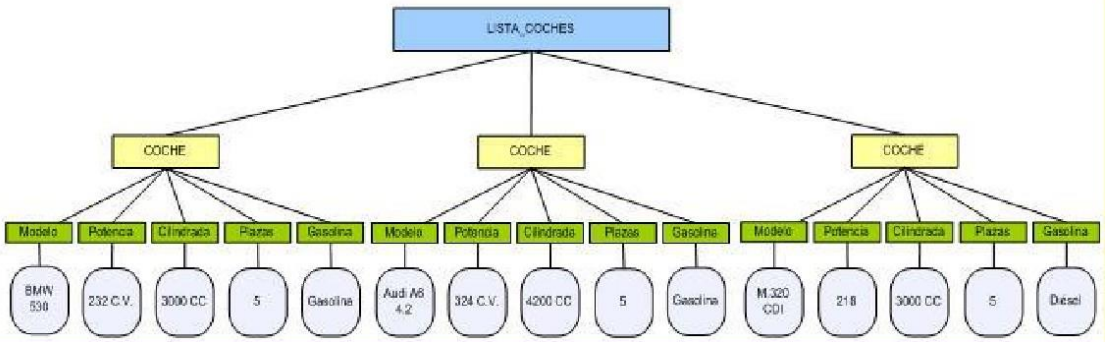
En el ejemplo que mostraremos a continuación, vamos a ver dos partes diferenciadas. Una primera parte que declara una DTD sencilla, para posteriormente incluir una

segunda parte en la que se muestra un posible documento XML que validado por dicha DTD. Veamos dicho ejemplo:



En la **DTD** del ejemplo vemos como se han utilizado algunas de las propiedades descritas anteriormente. Por ejemplo, lista_coches se ha declarado como un elemento que puede contener uno o más hijos (coches). A su vez, coche se ha declarado como un elemento que contendrá varios hijos (modelo y opcionalmente potencia, cilindrada, plazas y gasolina).

A continuación, mostraremos un gráfico que nos ayudará a entender la **estructura de árbol** que sigue un documento XML.



La descripción de la estructura es la siguiente:

- En **azul** vemos el elemento raíz. Es el elemento imprescindible para constituir un documento XML. Como vemos, su nombre es el que aparecía en el código anterior para especificar la DTD.
- En **amarillo** vemos tres elementos incluidos en ese elemento raíz.
- En **verde** aparecen los elementos hijos de cada uno de esos elementos.
- En **azul celeste** vemos los valores que toman dichos elementos hijos.

Como vemos en ambos ejemplos XML es un lenguaje con una estructura clara y bien definida.

4. Especificación de atributos.

4.1. Introducción.

Un atributo se puede incluir en cualquier elemento del documento XML. Estos además estarán definidos en la DTD. Recordamos que anteriormente hemos visto que la sintaxis para la especificación de atributos está constituida por un **par nombre_ de_ atributo = 'Valor'.** La función de un atributo es aportar información específica al elemento que lo incluye. En XML, un elemento puede contener una cantidad ilimitada de atributos.

Cuando queremos incluir uno o más atributos en una **DTD**, usaremos la palabra resevada **ATTLIST**. XML permite crear más de un **ATTLIST** por elemento y la sintaxis será como vemos a continuación:

```
<!ATTLIST nombreElemento nombreAtributo tipo
característica>

donde "tipo" puede ser:

    ·CDATA
    ·Enumerado
    ·ID
    ·IDREF
    ·IDREFS
    ·NMTOKEN
    ·NMTOKENS

y donde "característica" puede tomar los siguientes
valores:

    ·#IMPLIED
    ·#REQUIRED
    ·#FIXED valor (donde "valor" debe ir
     entrecomillado)
    ·valor por defecto (donde este "valor por defecto"
     debe ir entrecomillado)
```

En una DTD, cuando se declara más de un atributo para un elemento no es necesario escribir varias veces <!ATTLIST, pudiéndose escribir, por ejemplo:

```
<!ATTLIST f1 pais CDATA "España"
fecha_de_nacimiento CDATA #IMPLIED
equipo CDATA #REQUIRED>
```

4.2. Tipos de atributos.

Este tipo de atributos puede contener cualquier tipo de datos que deseemos. Su valor puede ser cualquiera a excepción de otro elemento.

- **ENUMERADO.** Después de **CDATA** es el tipo de atributo que más aparece. Se representa con una lista de valores separador por barras verticales. Se conoce como enumerado, porque su definición es similar a como declaramos los tipos enumerados en algunos lenguajes de programación. A continuación mostraremos un ejemplo:

```
<!ATTLIST coche color(Negro | Antracita | Rojo | Blanco) #REQUIRED >
```

  Este tipo de atributo enumerado indica que para el atributo color del elemento coche tenemos cuatro valores permitidos (Negro, Antracita, Rojo y Blanco). La lista de valores posibles para ese atributo debe estar encerrada entre paréntesis y no es necesario el uso de comillas para delimitar cada valor.

  No confundir el uso del operador "|" cuando se usa en la declaración de un elemento o en la declaración de un atributo. Como acabamos de ver, en la declaración de un atributo su uso indica la lista de valores posibles que puede tomar el atributo. Sin embargo, en la declaración de elementos, el operador "|" se usar para declarar elementos que contengan elementos opcionales, pero no indica el listado de valores que puede tomar.

  En el ejemplo de declaración de atributo:

```
<!ATTLIST coche color (negro | Antracita | Rojo |
Blanco) #REQUIRED>
```

  el tipo enumerado indica que el atributo "color" del elemento "coche" solo puede ser uno de los valores indicados en la lista.

  En el caso de declaración de elementos:

```
<!ELEMENT articulo (codigo | id)>
<!ELEMENT codigo (#PCDATA)>
<!ELEMENT id (#PCDATA)>
```

  el operador "|" indica que el elemento "articulo" puede contener un elemento "código" o un elemento "id" pero no ambos. Y los elementos "código" e "id" están a su vez definidos como #PCDATA por lo que podrán tomar cualquier valor. No se restringe los valores posibles, sino que se indica que puede elegir entre varios elementos opcionales.

- **ID**. Se utilizan para **crear un elemento único dentro de un documento XML**. Para ello, debemos asignar a ID un valor único, no repitiéndolo en el resto del documento. Con este atributo podremos representar el código de identificación de un producto, un DNI o un número de socio de cualquier organización.

  Las normas para los nombres de atributos de tipo ID son las mismas que explicamos en su momento para los elementos (empezar por una letra, no contener un espacio entre dos palabras, etc.). Veamos un ejemplo a continuación de su modo de uso:

```
<!ATTLIST coche matricula ID>
```

  Cuando declaremos en el documento XML un atributo de tipo matrícula lo haremos del siguiente modo:

```
<coche matricula="M2547MK">
```

  Una vez declarado este valor para el atributo matricula, no podrá aparecer en ninguna parte del documento XML, otro elemento que tenga este mismo valor de atributo.

- **IDREF**. Este tipo, permite a un atributo referenciar un ID desde cualquier parte del documento XML. Veamos a continuación un ejemplo más completo que agrupe los tres tipos de atributo vistos hasta el momento:

```
Declaración de la DTD

<!ELEMENT garaje(lista_coches?,bastidor*)>
<!ELEMENT lista_coches(coches*)>
<!ELEMENT coches(modelo,cilindrada?,potencia?)>
<!ATTLIST coches matricula ID>
<!ELEMENT modelo(BMW|AUDI|MERCEDES)>
<!ELEMENT cilindrada(#PCDATA)>
<!ELEMENT potencia(#PCDATA)>
<!ELEMENT version ANY>
<!ATTLIST version bastidor IDREF>
```

  En este ejemplo, el atributo bastidor declarado al final, tomará el valor que tome el atributo matricula, debido a que este ha sido declarado como ID.

- **IDREFS**. Su comportamiento es igual que el tipo **IDREF**, con la excepción de que ahora tomará el valor de una **lista de ID** en lugar de tomar el valor de un sólo **ID**.

  Imaginemos que tenemos una lista de coches en la que cada coche tiene un atributo matrícula. Si posteriormente declaramos un elemento con un atributo (por ejemplo, lista) de tipo **IDREFS**, el valor de ese atributo será una lista de todas las matrículas de los coches que tiene el documento XML separadas por

un espacio en blanco.

- **NMTOKEN**. Este tipo de atributo restringe el valor del atributo a un nombre valido en XML. Esto ya ocurría con el tipo **ID**, con la excepción de que ahora se permite que el valor de dicho atributo no sea único y pueda aparecer varias veces dentro de un documento XML.
- **NMTOKENS**. En este caso, un atributo que sea declarado con el tipo NMTOKENS, permitirá tener como valor una lista de nombres admitidos por las reglas de nombrado de XML.

Valor Predeterminado de un Atributo

Se pueden designar cuatro tipos de valor predeterminado para un atributo. Este tipo de valores realizan modificaciones sobre la estructura del documento XML. Es importante resaltar que los valores predeterminados son incompatibles entre sí. Es decir, no se pueden combinar en la declaración de un atributo ya que su significado suele ser contradictorio. Los cuatro tipos son:

- **#REQUIRED**. Si incluimos este modificador en la declaración del atributo, forzaremos a que siempre que aparezca el elemento sea necesario incluir el atributo. No tiene ninguna influencia sobre el valor del atributo. En el siguiente ejemplo estaríamos indicando que siempre que se declare el elemento coche, este deberá ir acompañado del atributo matrícula.

```
<!ATTLIST coche matricula #REQUIRED>
```

- **#IMPLIED.** Si incluimos este modificador en la declaración del atributo, indicaremos que el atributo es opcional. Podemos entenderlo como la situación opuesta al modificador #REQUIRED. En un caso se indica que el atributo debe existir siempre que se declare un elemento, mientras que en el otro caso indicamos que el atributo tiene un carácter opcional. En el siguiente ejemplo, el atributo cilindrada del elemento coche podrá aparecer o no en el documento XML.

```
<!ATTLIST coche cilindrada #IMPLIED>
```

- **#FIXED.**Si incluimos este modificador en la declaración del atributo, estaremos indicando que el atributo debe tener un valor fijo que es indicado a continuación del modificador, es decir, siempre que aparezca en el documento XML, deberá llevar el valor indicado en su declaración. En caso de que al utilizar un elemento no incluyamos el atributo, el parser de XML lo incluirá. En el ejemplo que vemos a continuación, el elemento coche tiene un atributo matricula cuyo valor queda fijado en la declaración a “M2547MK”.

```
<!ATTLIST coche matricula #FIXED "M2547MK">
```

El valor específico

Cuando usamos un valor específico el comportamiento del atributo es el siguiente:

- Si el atributo no está presente en la declaración, mantiene ese valor.
- Si el atributo está presente, asumirá el valor indicado.

5. Declarando entidades.

Otra de las propiedades importantes de XML es la capacidad de integración de datos procedentes de distintas fuentes. Hasta ahora hemos visto **documentos XML** que siempre contenían datos en un único fichero, pero las posibilidades son mucho más amplias. Con el nombre de entidades denominamos a las distintas unidades de almacenamiento que conforman un documento XML. Estas entidades pueden estar compuestas por cualquier tipo de dato. Desde los datos más simples como cadenas de texto hasta datos, pasando por registros de una base de dato, hasta los datos más complejos como algunos tipos de gráfico.

Cada unidad de almacenamiento con su declaración XML, la definición de una DTD y su elemento raíz (siempre imprescindible en cualquier documento XML) recibe el nombre de **document entity**.

Las entidades que podemos encontrar en un documento XML pueden ser de cuatro tipos:

- **Analizadas**: Anteriormente se ha visto como todo documento XML es 'parseado' por una aplicación, para comprobar su validez Este tipo de entidad, debe ser analizada por el parser de forma obligatoria.
- **No analizadas**: Si la entidad no tiene que ser analizada por el parser.
- **Externas**: La entidad se encuentra en un documento distinto al documento en el que aparece la referencia. Esas referencias son siempre con respecto a la situación del recurso en el que la declaración tiene lugar.
- **Internas**: La entidad se encuentra declarada en el mismo documento en el que aparece la referencia a esa entidad. En este tipo de entidades no hay un almacenamiento físico distinto al del documento XML. Este tipo de entidades son siempre analizadas.

Con el siguiente ejemplo quedará más claro cómo se declaran las entidades en XML:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE modelo[
3    <!ENTITY prestaciones "El modelo que veremos a continuación
4   tiene las siguientes características">
5    <!ELEMENT modelo (#PCDATA)>
6 ]>
7 <modelo>
8    &prestaciones; 100 C.V. y 2.000 C.C.
9 </modelo>
```

En este ejemplo, hemos declarado una entidad del tipo más sencillo (cadena de caracteres). Se trata de una entidad interna, ya que se incluye dentro del propio documento. El ejemplo funciona del siguiente modo:

- **Línea 1**: Cabecera indicando la versión de Xml empleada y la codificación del documento.
- **Línea 2**: Declaración de la DTD interna coches.
- **Líneas 3 y 4**: Declaración de la entidad prestaciones cuyo valor es la cadena de texto: "*El modelo que veremos a continuación tiene las siguientes características*".
- **Línea 6**: Declaración del elemento raíz del documento.
- **Línea 7**: En esta línea usamos por primera vez la entidad declarada previamente. La sintaxis como vemos es sencilla; símbolo *&* seguido del nombre de la entidad y finalizado con un punto y coma. Posteriormente introducimos más contenido para ese elemento. Esta entidad hace lo siguiente; cada vez que introducimos **&prestaciones**; el parser incluirá el texto "*El modelo que veremos a continuación tiene las siguientes características*".
- **Línea 8**: Etiqueta de cierre del elemento raíz.

La entidad del ejemplo es una entidad de tipo Interna, ya que su contenido está integrado en la declaración de la misma. En el caso de que la entidad fuese de tipo Externa, el ejemplo sería el siguiente:

```
1 <?xml version="1.0" enconding="UTF-8"?>
2 <!DOCTYPE coches [
3 <!ENTITY prestaciones SYSTEM "ejemplo.xml">
4  ]>
5
6 <coches>
7       &prestaciones; 100 C.V. y 2000 C.C.
8 </coches>
```

El contenido del fichero "ejemplo.xml" sería el siguiente:

```
1 <?xml version="1.0" enconding="UTF-8"?>
2
3 <características>
4       "El modelo que veremos a continuación tiene las siguientes
5      características"
6 </características>
```

Al margen de la clasificación de entidades comentada anteriormente, podemos distinguir también entre dos tipos de entidad en función de dónde se use dicha entidad:

- • **General**: Cuando una entidad es referenciada como parte del contenido de un documento XML. Las entidades generales pueden ser entidades analizadas o sin analizar.
- • **Paramétrica**: Cuando una entidad es referenciada en el DTD.

6. Declarando la localización de un fichero DTD en un documento XML.

Anteriormente hemos comentado que en una **DTD** se especifica que elementos y atributos son susceptibles de ser incluidos en el documento. En cuanto a su estructura, podemos establecer algún paralelismo con la estructura que sigue una gramática regular.

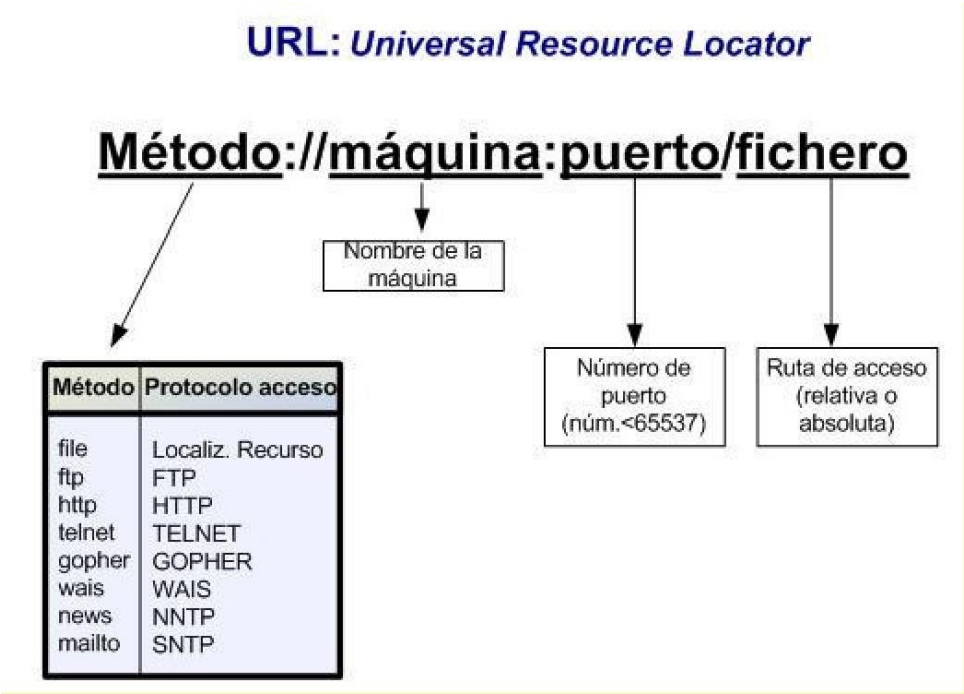
Concretamente, se deben escribir en la forma extendida de **Backus Naur** en un formato llamado *Extended Backus Naur Form* (EBNF).

A la hora de hacer uso de una **DTD** disponemos de dos alternativas; crearla en un documento independiente (DTD **externa**) o crearla dentro del mismo documento (DTD **interna**). Si se crea en un documento separado, se crea como un documento de texto (en ASCII).

Anteriormente hemos visto la sintaxis que se utiliza para declarar una **DTD externa**. En el caso de las DTD **internas**, el procedimiento es muy similar.

```
<!DOCTYPE nombre_documento
[declaraciones que forman la DTD]>
```

En caso de incluir la DTD en el propio documento XML, es muy importante que la DTD aparezca antes del primer elemento del documento. Además, el nombre que sigue a la palabra **DOCTYPE** en el documento de definición de tipos, debe ser el mismo nombre que el que tiene el elemento raíz. Las ventajas de no incluir la DTD en el propio documento son diversas, pero quizás la más reseñable es que **una DTD puede ser referenciada por una URL** (siglas correspondientes a *Localizador Uniforme de Recurso*).



De esto modo, podemos tener una DTD en cualquier máquina de Internet y referenciarla mediante la URL cuando lo necesitemos. Para usar la URL debemos incluir la palabra reservada **SYSTEM** o **PUBLIC** en la invocación de la DTD.

En el caso de usar **SYSTEM**, se debe indicar a continuación el identificador de sistema o URI para conocer el lugar donde se encuentra el **fichero .dtd**.

En el caso de utilizar **PUBLIC** indicaremos que, además del identificador de sistema, identificamos la DTD con un identificador público.

Es importante no confundir el concepto de DTD **externa** y DTD **interna**, ya que mientras que en el segundo caso, todo el contenido de la DTD se encuentra dentro del documento, en el primer caso, debe existir una referencia a la ubicación en la que se sitúa esa DTD externa. Veamos a continuación un ejemplo de cómo referenciar una DTD mediante una URL.

```
<?xml version="1.0" enconding="UTF-8" ?>
<!DOCTYPE Lista SYSTEM
"http://www.fi.upm.es/XMLCourse/DTDexternal">
```

La palabra Lista hace referencia al nombre del elemento raíz del documento.

7. Conclusión.

En esta sección hemos podido adentrarnos en el concepto de **validez** de un documento XML. Para ello, nos hemos valido de las DTD como iniciación a la validación de código XML. Hemos visto que su estructura es similar a la que tiene el documento, constando éstas de atributos (o listas de ellos) y elementos.

Se han visualizado cómo utilizar los distintos tipos de atributos que nos ofrece este metalenguaje. Además, hemos introducido el concepto de entidad en XML, comentando los distintos tipos existentes (general y paramétrica, analizada y no analizada, etc.). Por último, comentar las dos posibilidades que podemos asumir a la hora de ubicar una DTD que valide el documento (estas pueden declararse dentro y fuera del documento XML). Incorrección en las referencias, ejemplos, casos prácticos aportados y bibliografía.

8. Ejercicio resuelto.

Escribir una posible DTD para un documento relacionado con libros de texto. Teniendo en cuenta que:

- Debe tener un elemento referencia que contenga los siguientes elementos:
  - libro,
  - editorial (el cual no es obligatorio),
  - encuadernación (podrá ser un tipo enumerado con las alternativas que crea el alumno)
  - ISBN.
- El elemento referencia debe tener un atributo idioma (necesario que aparezca) que pueda tener los valores de español o inglés.

Además, el contenido de los elementos libro, editorial e ISBN debe ser texto. Por último, reseñar que el orden de los elementos si es importante.

8-1. Posible solución.

```
<!ELEMENT referencia (libro, editorial?, encuadernacion, ISBN)>
<!ATTLIST referencia idioma (español|ingles) #REQUIRED>
<!ELEMENT libro (#PCDATA)>
<!ELEMENT editorial (#PCDATA)>
<!ELEMENT encuadernacion (pastas_duras | bolsillo | edicion_coleccionista)>
<!ELEMENT ISBN (#PCDATA)>
```

9. Ejercicios propuesto.

Ejercicio 1.

El siguiente documento XML (***"perro.xml"***) está bien formado. Sin embargo, no es válido. Para que lo sea, realizar los cambios necesarios en dicho documento, pero sin modificar la DTD interna.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE perro [
  <!ELEMENT perro (edad, nombre)>
  <!ELEMENT edad (#PCDATA)>
  <!ELEMENT nombre (#PCDATA)>
]>

<perro>
 <nombre>Pancho</nombre>
 <edad>8</edad>
</perro>
```

Ejercicio 2.

El siguiente documento XML (***"ciudades.xml"***) está bien formado. Sin embargo, no es válido. Para que lo sea, realizar los cambios necesarios en la DTD interna de dicho documento.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ciudades [
  <!ELEMENT ciudades (ciudad)>
  <!ELEMENT ciudad (nombre, capital)>
  <!ELEMENT nombre (#PCDATA)>
  <!ELEMENT capital EMPTY>
]>

<ciudades>
  <ciudad>
    <nombre>Buenos Aires</nombre>
    <capital/>
  </ciudad>
  <ciudad>
```

```
    <nombre>Liverpool</nombre>
  </ciudad>
  <ciudad>
    <nombre>Osaka</nombre>
  </ciudad>
  <ciudad>
    <nombre>Oslo</nombre>
    <capital/>
  </ciudad>
</ciudades>
```

Ejercicio 3.

Crear una DTD que valide el siguiente fichero XML:

```
<?xml version  ="1.0"?>
<Biblioteca>
        <Lista_libros>
                <libro titulo="Ángeles y Demonios" paginas="527">
                        <Autor>Dam Brown</Autor>
                        <Editorial>Otnemem</Editorial>
                </libro>
                <libro titulo="El código Da Vinci" paginas="469">
                        <Autor>Dam Brown</Autor>
                        <Editorial>Xirtam</Editorial>
                </libro>
                <libro titulo="Rompiendo límites" paginas="350">
                        <Autor>Sebastían Álvaro</Autor>
                        <Editorial>Omen</Editorial>
                </libro>
                <libro titulo="Al otro lado de la realidad" paginas="487">
                        <Autor>Pedro Dalmau</Autor>
                        <Editorial>Otnemem</Editorial>
                </libro>
                <libro titulo="Fortaleza Digital" paginas="798">
                        <Autor>Dan Brown</Autor>
                        <Editorial>Xirtam</Editorial>
                </libro>
                <libro titulo="La sombra de la Duda" paginas="1502">
                        <Autor>María Mendez</Autor>
                        <Editorial>lomax</Editorial>
                </libro>
                <libro titulo="Sin Noticias de Gurb" paginas="124">
                        <Autor>Eduardo Mendoza</Autor>
                        <Editorial>Yresim</Editorial>
                </libro>
        </Lista_libros>
</Biblioteca>
```