

Tema 3: Definición de tipos de documento y esquemas 3

.1 La validación por DTD

En el tema anterior hemos visto cómo crear un documento XML basado en la gramática especificada por el W3C. Si se respetan esas normas, el documento XML estará **bien formado**. Sin embargo, si el documento respeta la gramática descrita en una *Document Type Definition* (DTD) o en un XML Schema o en un documento Relax NG, el documento será **válido**.

.1.1 Cómo vincular una DTD a los datos XML

Antes de ver cómo se construye una DTD debemos saber que la DTD va asociada a un documento XML utilizando la Declaración de Tipo de Documento DOCTYPE.

Esta declaración debe estar después de la declaración de XML y antes de la aparición de cualquier elemento del documento.

La Declaración DOCTYPE va seguida del nombre del elemento raíz del documento y, a continuación:

- a. Si el conjunto de declaraciones que forman la DTD se encuentran en propio documento XML:

```
<!DOCTYPE nombreelementoraiz [ Declaraciones ]>
```

- b. Si el conjunto de declaraciones que forman la DTD se encuentran en un archivo externo:

```
<!DOCTYPE nombreelementoraiz SYSTEM "nombrefichero.dtd">
```

para indicar que la DTD se encuentra en nuestro sistema , o bien:

```
<!DOCTYPE nombreelementoraiz PUBLIC "URI">
```

para indicar que la DTD es pública.

La opción **b** es la más conveniente puesto que resulta más flexible y reutilizable, la opción **a** solo se recomienda en casos de vocabularios muy simples.

.1.2 Estructura de una DTD

Los contenidos que pueden incluirse en un documento XML están definidos en la DTD utilizando cuatro posibles términos: ELEMENT, ATTLIST, ENTITY y NOTATION.

ELEMENT para declarar elementos.

ATTLIST para declarar atributos.

ENTITY para declarar contenidos reutilizables.

NOTATION para declarar formatos aplicables a contenidos externos (no XML) y dirigidos a las aplicaciones que deben manipular esos contenidos.

.1.2.1 Elementos

Los elementos se declaran en la DTD utilizando la etiqueta :



<!ELEMENT nombrelemento especificaciones de contenido ...>

Los elementos se clasifican, según el carácter de sus contenidos, en cuatro tipos:

1. De tipo **contenido de elemento (element)**: los que solo tienen permitido contener a otros elementos como contenido.

Ejemplo: El elemento A contiene el elemento B seguido del elemento C (en ese orden)

<!ELEMENT A (B,C)>

2. De tipo **contenido de texto**: los que solo pueden contener datos textuales (cadenas de caracteres). Permite que el elemento quede vacío.

Ejemplo: El elemento A contiene datos textuales.

<!ELEMENT A (#PCDATA)>

3. De tipo **mixto (mixed)**: los que pueden contener cualquier cosa (datos textuales, elementos, ambos o nada)

Ejemplo: El elemento A contiene cualquier cosa (datos textuales, elementos, nada ...)

<!ELEMENT A ANY>

4. De tipo **elemento vacío (empty)**: los que carecen de cualquier tipo de contenido. Implica que ese elemento tendrá la forma <etiqueta /> dentro del documento.

Ejemplo: El elemento A no contiene nada.

<!ELEMENT A EMPTY>

Todos los elementos, sean del tipo que sean, pueden tener atributos asociados.

Cuando se declara un elemento, el modelo de contenido de dicho elemento puede considerarse un **grupo**, es decir, puede estar formado por un conjunto de nombres de **elementos hijos, operadores y la palabra clave #PCDATA**.

Un grupo se define utilizando los paréntesis ()

<!ELEMENT nombrelemento (elementoshijos unidos por operadores)>

Ejemplo:

<!ELEMENT datos_personales (nombre, apellido_1, apellido_2, dirección, teléfono)>

Esta definición implica que el elemento `<datos_personales>` debe contener los elementos hijos `<nombre>`, `<apellido_1>`, `<apellido_2>`, `<dirección>`, `<teléfono>` en ese orden dentro del documento XML. El **operador de orden coma (,)** es el que establece que el orden de aparición de los elementos debe ser el especificado en la definición.

Existe otro **operador de orden, la barra vertical (|)**, que significa que el elemento contiene solo uno de los elementos definidos a uno y otro lado del |.

Ambos operadores se pueden combinar utilizando los paréntesis (). Por ejemplo:

```
<!ELEMENT datos_personales (nombre, dirección, (teléfono|mail ))>
```

Esto implica que todos los elementos deberán aparecer en el orden establecido hasta llegar a los últimos dos, que constituyen una opción, podrá aparecer uno o el otro. Por ejemplo:

```
<datosPersonales>
  <nombre>Juan</nombre>
  <dirección>Calle JJJ</dirección>
  <mail>correo@mimail.com</mail>
</datosPersonales>
```

```
<datosPersonales>
  <nombre>Juan</nombre>
  <dirección>Calle JJJ</dirección>
  <telefono>985674321</telefono>
</datosPersonales>
```

Sin embargo, los elementos que contienen los datos textuales (**#PCDATA**), nada (**EMPTY**) o cualquier cosa (**ANY**) son elementos **terminales**.

#PCDATA (significa Parsed Character DATA) el elemento contiene únicamente datos textuales.

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

EMPTY prohíbe a un elemento tener un elemento hijo o datos textuales.

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

ANY permite a un elemento contener cero o varios elementos hijos declarados o datos textuales. Esto permite mezclar todos los tipos de elementos declarados.

```
<!ELEMENT control ANY>
```

Operadores de cardinalidad o frecuencia.

Determinan el número de veces que un elemento puede aparecer:

Operador **?** : indica que el elemento es opcional, es decir, puede aparecer o no, pero de aparecer puede hacerlo solo una vez. (0 o 1 veces)

Operador **+** : indica que el elemento debe aparecer por lo menos una vez pero puede hacerlo n veces. (1 o n veces)

Operador ***** : indica que el elemento en cuestión también es opcional pero de aparecer lo puede hacer n veces. (0 o n veces)

Si no se usa ningún cardinal, el cardinal por defecto es 1.

Ejemplo:

```
<!ELEMENT datosPersonales (nombre+, dirección?, (teléfono | mail)*)>
```

Significa que el elemento <datosPersonales> contiene los elementos <nombre>, que debe aparecer por lo menos una vez pero puede aparecer n veces; <dirección> que puede aparecer o no pero de hacerlo solo puede hacerlo una vez; y **la opción <teléfono> o <mail> que puede repetirse ninguna o n veces pues están ambos afectados por el operador ***. El documento resultante podría tener la forma:

```
<datosPersonales>
  <nombre> ....</nombre>
  <nombre> ....</nombre>
  <nombre> ....</nombre>
  <mail>...</mail>
  <mail>...</mail>
  <teléfono>...</teléfono>
  <mail>...</mail>
</datos personales>
```

Analicemos otros ejemplos:

```
<!ELEMENT ele1 (a , (b | c))>
```

El elemento ele1 tiene dos hijos, el primero será siempre a y el segundo una elección entre b o c.

```
<!ELEMENT ele1 (a, b?, c)>
```

El elemento ele1 tiene dos o tres hijos en una secuencia estricta donde b es opcional.

```
<!ELEMENT ele1 ((a, b)|(c | d))>
```

El elemento ele1 puede tener uno o dos hijos, si la elección es (a, b) entonces tendrá dos hijos a y b que aparecerán siempre en ese orden entre sí; si la elección se decanta por la pareja (c | d) aparecen c o d pero nunca ambos.

```
<!ELEMENT ele1 (a?, ((b ,c ) | d), e?)>
```

El primer elemento ahora puede ser a o no, pues es opcional, el segundo puede ser la pareja b, c o ser d, el tercero sería e pero también es opcional.

```
<!ELEMENT ele1 ((a, b)+ | (c | d))>
```

Aquí ele1 puede consistir en una lista repetida de pares a y b o tener una sola ocurrencia que podrá ser c o d

```
<!ELEMENT ele1 (a, (b, c)*, d+)>
```

Aquí ele1 tendrá un elemento a seguido por cero o n pares de elementos b y c, seguidos de uno o n elementos d.

Otros ejemplos:

El elemento raíz XXX puede contener un elemento AAA seguido de uno o más elementos BBB. El elemento AAA puede contener un elemento CCC y varios elementos DDD. El elemento BBB tiene que contener, exactamente, un elemento CCC y un elemento DDD:

```
<!ELEMENT XXX (AAA? , BBB+)>
<!ELEMENT AAA (CCC? , DDD*)>
<!ELEMENT BBB (CCC , DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Los siguientes son documentos válidos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
  <XXX>
    <AAA>
      <CCC/><DDD/>
    </AAA>
    <BBB>
      <CCC/><DDD/>
    </BBB>
  </XXX>
```

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
  <XXX>
    <AAA/>
    <BBB>
      <CCC/><DDD/>
    </BBB>
  </XXX>
```

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
  <XXX>
    <BBB>
      <CCC/><DDD/>
    </BBB>
  </XXX>
```

El elemento raíz XXX debe contener un elemento AAA seguido de un elemento BBB. El elemento AAA tiene que contener un elemento CCC seguido de un elemento DDD. El elemento BBB tiene que contener bien un elemento CCC o bien un elemento DDD:

```
<!ELEMENT XXX (AAA , BBB)>
<!ELEMENT AAA (CCC , DDD)>
<!ELEMENT BBB (CCC | DDD)>
<!ELEMENT CCC (#PCDATA)>
<!ELEMENT DDD (#PCDATA)>
```

Los siguientes son documentos válidos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <CCC/>
  </BBB>
</XXX>
```

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
<XXX>
  <AAA>
    <CCC/> <DDD/>
  </AAA>
  <BBB>
    <DDD/>
  </BBB>
</XXX>
```

El elemento raíz XXX debe contener al menos un elemento AAA seguido de al menos un elemento BBB. El elemento AAA puede contener o bien BBB o CCC. Por otro lado el elemento BBB puede contener cualquier combinación de texto y elementos CCC.:

```
<!ELEMENT XXX (AAA+ , BBB+)>
<!ELEMENT AAA (BBB | CCC )>
<!ELEMENT BBB (#PCDATA | CCC )*>
<!ELEMENT CCC (#PCDATA)>
```

El siguiente es un documento válido:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
  <XXX>
    <AAA>
      <CCC>Exactamente un elemento.</CCC>
    </AAA>
    <AAA>
      <BBB>
        <CCC/>
        <CCC/>
        <CCC/>
      </BBB>
    </AAA>
    <BBB/>
    <BBB>
      Esta es <CCC/> una combinación <CCC/> de <CCC> elementos CCC </CCC> y texto CCC/>.
    </BBB>
    <BBB>
      Sólo texto.
    </BBB>
  </XXX>
```

.1.2.2 Entidades

Las entidades permiten definir las constantes de los documentos XML. Son útiles cuando hay gran cantidad de información que se repite, ya que una vez declaradas, podemos usarlas tantas veces como queramos. En función de si sus contenidos son XML o no hay dos tipos **entidades generales**: las entidades procesadas y las no procesadas.

- Las **procesadas** son aquellas cuyo contenido es un contenido XML y deben ser procesadas por el parser, el valor de una entidad procesada es conocido como *texto de reemplazo*. Una vez efectuado el reemplazo de la referencia a la entidad por su contenido, este contenido pasa a ser parte del documento y es analizado por el procesador como el resto del documento.
- Las **no procesadas** son aquellas cuyo contenido no es texto XML, puede ser un archivo en formato JPG o GIF, el procesador no analizará estos contenidos.
- Otro tipo de entidades son las **predefinidas** que ya hemos visto en el tema anterior. XML reserva algunos caracteres para su funcionamiento interno, los signos < y >, el &, las comillas simples y las comillas dobles; estos caracteres no deben incluirse de modo literal, excepto dentro de secciones CDATA, pero pueden incluirse en los documentos como entidades predefinidas. También es posible hacer referencia a cualquier carácter utilizando su notación numérica (**&#numero;**). Por ejemplo, el carácter > es **>**;

✚ La forma general de declarar una entidad en la DTD es:

```
<!ENTITY nombre_entidad "contenido de reemplazo" >
```

✚ Y la forma general de llamar a una entidad dentro del documento XML es:



```
<elemento> ..... &nombreentidad; ..... </elemento>
```

Ejemplo:

En la DTD se define:

```
<!ENTITY situacion "Calle de Strasburgo" >
```

En el documento XML se utiliza:

```
<dirección>
  <numero>172</numero>
  <calle>&situacion;</calle>
  <población>Buenos Aires</población>
</dirección>
```

- Otra posibilidad es que **el contenido de reemplazo se encuentre en un archivo externo**, sería posible para entidades generales que podrían ser utilizadas por muchos documentos.

- En el caso de las **entidades procesadas** (solo contienen texto) la sintaxis sería:

La entidad puede ser una entidad de sistema, con la siguiente sintaxis:

```
<!ENTITY nombre_entidad SYSTEM "URI" >
```

o puede ser una entidad pública, con la siguiente sintaxis:

```
<!ENTITY nombre_entidad PUBLIC "identificador publico formal" "URI" >
```

- Y en el caso de las **entidades no procesadas**, aquellas que hacen referencia a un fichero que no es de texto (por ejemplo, una imagen) la entidad no se sustituye por el contenido del archivo, solo se utiliza para saber donde está ubicado el archivo, de que tipo es y, con ayuda de la notación, con qué aplicación se puede tratar ese archivo.

La entidad puede ser una entidad de sistema, con la siguiente sintaxis:

```
<!ENTITY nombre_entidad SYSTEM "URI" NDATA tipo>
```

NDATA tipo: viene de Notation Data y significa que la entidad tiene asociada una notación (NOTATION) para ese *tipo* de archivo.

O puede ser una entidad pública, con la siguiente sintaxis:

```
<!ENTITY nombre_entidad PUBLIC "identificador publico formal" "URI" NDATA tipo>
```


Por ejemplo: `<!ENTITY companyLogo PUBLIC "-//W3C//GIF logo//EN" "http://www.w3.org/logo.gif" NDATA gif>`

- Existen otro tipo de entidades que son las denominadas **entidades parámetro**. Siguen la misma sintaxis que las generales, pero llevan el carácter "%" antes del nombre de la entidad. La diferencia entre entidades generales y paramétricas es que las entidades paramétricas se sustituyen por su valor en todo el documento XML incluida la propia DTD mientras que las generales no se sustituyen en la DTD solo en el documento XML.

Su forma general puede ser:

✓ `<!ENTITY %nombre_entidad "contenido de reemplazo" >`

✓ `<!ENTITY % nombre_entidad SYSTEM "URI" >`

Por ejemplo:

`<!ENTITY % fechaInicioFin SYSTEM "fechainiciofin.ent" >`

✓ `<!ENTITY % nombre_entidad SYSTEM "URI" NDATA tipo >`

Ejemplo para la primera sintaxis de una entidad paramétrica. Definimos una entidad en la DTD:

`<!ENTITY % otros_valores "edad CDATA #IMPLIED peso CDATA #IMPLIED altura CDATA #REQUIRED">`

Y la utilizamos también desde la DTD (La DTD debe definirse en un archivo externo):

`<!ATTLIST persona
nombre CDATA #REQUIRED
% otros_valores; >`

Esto sería equivalente a haber escrito:

`<!ATTLIST persona
nombre CDATA #REQUIRED
edad CDATA #IMPLIED
peso CDATA #IMPLIED
altura CDATA #REQUIRED >`

.1.2.3 Notaciones

Las notaciones permiten identificar el formato de **datos no XML** que queramos incluir en un documento XML. Esto permite que el parser XML o una aplicación cliente pueda procesar correctamente esos datos. Para llevar a cabo esa identificación, se puede especificar la aplicación encargada de tratarlos o, un identificador que permita buscar un recurso que sea capaz de interpretar los datos. Será el parser XML el encargado de remitir la información a dicha aplicación para su procesamiento.

Las formas generales de declarar una notación en la DTD son:

```
<!NOTATION nombre_notacion PUBLIC PublicID>
```

```
<!NOTATION nombre_notacion PUBLIC PublicID SystemID>
```

```
<!NOTATION nombre_notacion SYSTEM SystemLiteral>
```

Solamente veremos el último tipo, que corresponde a notaciones del sistema.

Por ejemplo, definimos notaciones en la DTD indicando la aplicación que procesa los datos:

```
<!NOTATION jpg SYSTEM "visor_de_jpg.exe" >
<!NOTATION gif SYSTEM "visor_de_gif.exe" >
```

O definimos notaciones en la DTD indicando el tipo MIME (especificaciones para intercambio de audio, video etc. a través de internet) para que la aplicación busque un recurso para procesar los datos:

```
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION jpg SYSTEM "image/jpeg">
<!NOTATION png SYSTEM "image/png">
```

3.1.2.4 Atributos

Los elementos pueden tener cero, uno o varios atributos. La especificación de atributos sólo puede aparecer dentro de la etiqueta de apertura y en los elementos vacíos. La declaración en la DTD comienza con **ATTLIST** seguido del nombre del elemento al que pertenece el atributo y después le sigue la definición individual de cada atributo.

```
<!ATTLIST nombreelemento
  nombreatributo definición
  nombreatributo definición
  nombreatributo definición
  ... >
```

➤ Dentro de las **definiciones** podemos consignar algunos **indicadores** que afectan a los valores que puede o debe tomar ese atributo. Se han definido tres:

#REQUIRED: el atributo debe aparecer en todas las instancias del elemento, es decir, siempre que aparezca el elemento que tenga asociado un atributo de este tipo. Es el indicador por defecto.

#IMPLIED: la aparición de este atributo es opcional.

#FIXED: este indicador hace que el valor del atributo se tome siempre como valor fijo, aparezca el atributo en la etiqueta o no. Si el atributo no aparece en la etiqueta, el procesador asume el que le hayamos asignado en esta declaración. Si aparece el atributo con un valor diferente a este se producirá un error de validación.

Otra cosa diferente es la definición de un **valor por defecto**. En ese caso, si no definimos un valor para el atributo, el procesador tomará el que le hemos asignado como su valor por defecto. Por ejemplo, si declaramos:

```
<!ATTLIST curso color CDATA "azul">
```

El valor "azul" es un valor por defecto. Por tanto, serán válidos los elementos:

```
<curso color="azul">...</curso>
```

Y también:

```
<curso>...</curso>
```

En el último caso, el parser interpretará, igual que en el primer caso, el valor "azul" para ese atributo.

- Dentro de las **definiciones**, después del nombre del atributo y antes del indicador de requerido, opcional o fijo, incluimos el **tipo** de atributo.

Los tipos de atributo pueden ser:

CDATA: el valor del atributo será del tipo cadena de caracteres (que no contenga símbolos >, <, &, ' , ") Por ejemplo:

Dada la siguiente línea en la DTD: `<!ATTLIST domicilio ciudad CDATA #IMPLIED>`

El siguiente elemento sería válido: `<domicilio ciudad="Buenos Aires"> ...contenido...</domicilio>`

Enumeraciones: permite establecer un número limitado de valores posibles para un determinado atributo. Los valores aparecen separados por una barra vertical (|), no van entrecomillados y son sensibles al uso de mayúsculas o minúsculas. No debe usarse en combinación con #REQUIRED, #IMPLIED, #FIXED

Dada la siguiente línea en la DTD: `<!ATTLIST semáforo color (rojo| amarillo| verde) "rojo">`

El atributo color puede incluir uno y solo uno de estos valores permitidos. El valor por defecto de este atributo se ha establecido como "rojo".

ID: el valor del atributo debe ser único en todo el documento. Ningún elemento puede tener más de un atributo ID. Un atributo ID solo puede ser #IMPLIED o #REQUIRED pero nunca #FIXED puesto que identifica a un elemento que puede aparecer más de una vez en el documento pero nunca debe hacerlo con el mismo valor para su atributo ID. El valor de un ID no puede comenzar por un número solamente por una letra o el símbolo de subrayado.

Dada la siguiente línea en la DTD: `<!ATTLIST empleado numero_identificador ID #REQUIRED>`

El siguiente elemento sería válido: `<empleado numero_identificador="AA_125">El Hombre Araña</empleado>`

IDREF: el valor del atributo es el valor del ID de un elemento relacionado que se encuentre en el mismo documento XML

Dada la siguiente línea en la DTD: `<!ATTLIST fechaIngreso registroInterno IDREF #REQUIRED>`



El siguiente elemento sería válido: `<fechaIngreso registroInterno="AA_125">23/2/1967</fechaIngreso>`

IDREFS: el valor del atributo es una lista de los valores de los ID de los elementos que queremos relacionar separados por un espacio en blanco. Solo se pueden relacionar IDs presentes en el mismo documento.

Si añadimos las siguientes líneas a la DTD del ejemplo anterior:

```
<!ELEMENT equipo EMPTY>
<!ATTLIST equipo miembros IDREFS #REQUIRED>
```

El siguiente elemento sería válido: `<equipo miembros="AA_125 AB_320 AA_210 AA_012" />`

Los tipos de atributo ID, IDREF e IDREFS son una herramienta necesaria para expresar relaciones del estilo de las que encontramos en las bases de datos relacionales y resultan particularmente útiles si estamos utilizando XML como formato de intercambio de datos.

NMTOKEN: el valor del atributo debe ser una cadena compuesta por los caracteres permitidos para identificadores XML (letras, números, subrayado, guión, dos puntos, punto) pero, a diferencia de estos identificadores, estas cadenas pueden comenzar por cualquiera de estos caracteres.

NMTOKENS: el valor del atributo podrá contener un número "n" de valores del tipo NMTOKEN separados entre sí por un espacio en blanco.

Si añadimos las siguientes líneas a la DTD del ejemplo anterior:

```
<!ELEMENT categoría (#PCDATA)>
<!ATTLIST categoría tipo NMTOKENS #REQUIRED>
```

El siguiente elemento sería válido: `<categoría tipo="director jefe_compras 1:contratoFijo" />`

ENTITY y ENTITIES: el valor del atributo es una entidad o una lista de "n" entidades.

Dadas las siguientes líneas en la DTD:

```
<!ENTITY graficoVentas_1 SYSTEM "grafico_1.gif" NDATA gif >
<!ATTLIST resultadoVentas grafico ENTITY #IMPLIED>
```

NDATA gif : significa que la entidad tiene asociado un tipo de notación gif

El siguiente elemento sería válido: `<resultadoVentas grafico="graficoVentas_1"> </resultadoVentas>`

NOTATION: el valor del atributo es alguna notación definida en la DTD.

Dadas las siguientes líneas en la DTD:

```
<!NOTATION gif SYSTEM "image/gif" >
<!NOTATION jpg SYSTEM "image/jpeg" >
<!ATTLIST foto tipo NOTATION ( gif | jpg ) #IMPLIED>
```



El siguiente elemento sería válido: <foto tipo="jpg" ></foto>

Otros ejemplos:

✚ Dada la siguiente DTD:

```
<!ELEMENT attributes (#PCDATA)>
<!ATTLIST attributes      aaa CDATA #REQUIRED
                        bbb CDATA #IMPLIED>
```

❖ Los siguientes son documentos válidos:

1) El elemento contiene datos textuales y atributos:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
  <attributes aaa="#d1" bbb="*~*">
    Text
  </attributes>
```

2) El orden de los atributos es indiferente:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
  <attributes bbb="$25" aaa="13%">
    Texto
  </attributes>
```

3) El atributo bbb puede omitirse ya que es #IMPLIED:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
  <attributes aaa="#d1" />
```

✚ Dado la siguiente DTD:

Los atributos bbb y ccc siempre tienen que estar presentes, el atributo aaa es opcional:

```
<!ELEMENT attributes (#PCDATA)>
  <!ATTLIST attributes      aaa CDATA #IMPLIED
                        bbb NMTOKEN #REQUIRED
                        ccc NMTOKENS #REQUIRED>
```

❖ Los siguientes son documentos válidos:


1) Todos los atributos obligatorios están presentes y sus valores son del tipo correcto:

```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
  <attributes aaa="#d1" bbb="a1:12" ccc=" 3.4 div -4"/>
```

2) Todos los atributos obligatorios están presentes y sus valores son del tipo correcto:



```
<!DOCTYPE attributes SYSTEM "tutorial.dtd">
<attributes bbb="a1:12" ccc="3.4 div -4"/>
```

 Dada la siguiente DTD:

Los atributos id, code y X determinan de manera inequívoca su elemento:

```
<!ELEMENT XXX (AAA+ , BBB+ , CCC+)>

<!ELEMENT AAA (#PCDATA)>
<!ELEMENT BBB (#PCDATA)>
<!ELEMENT CCC (#PCDATA)>

<!ATTLIST AAA id ID #REQUIRED>
<!ATTLIST BBB code ID #IMPLIED list NMTOKEN #IMPLIED>
<!ATTLIST CCC X ID #REQUIRED Y NMTOKEN #IMPLIED>
```

❖ Los siguientes son documentos válidos:

1) Todos los valores ID son únicos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">

<XXX>
  <AAA id="a1"/>
  <AAA id="a2"/>
  <AAA id="a3"/>
  <BBB code="QWQ-123-14-6" list="14:5"/>
  <CCC X="zero" Y="16" />
</XXX>
```

2) Los atributos list e Y son del tipo NMTOKEN no ID. Estos pueden tener, por lo tanto, el mismo valor que los atributos ID o tener el mismo valor en varios elementos:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">

<XXX>
  <AAA id="L12"/>
  <BBB code="QW" list="L12"/>
```

```
<CCC X="x-0" Y="QW" />
```

```
<CCC X="x-1" Y="QW" />
```

```
</XXX>
```

✚ Dada la siguiente DTD:

Los atributos id y mark determinan inequívocamente su elemento. Los atributos ref hacen referencia a estos elementos:

```
<!ELEMENT XXX (AAA+ , BBB+, CCC+, DDD+)>
```

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

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

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

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

```
<!ATTLIST AAA mark ID #REQUIRED>
```

```
<!ATTLIST BBB id ID #REQUIRED>
```

```
<!ATTLIST CCC ref IDREF #REQUIRED>
```

```
<!ATTLIST DDD ref IDREFS #REQUIRED>
```

❖ Los siguientes son documentos válidos:

- 1) Todos los valores ID son únicos y los valores IDREF e IDREFS apuntan a elementos con ID relevante:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">
```

```
<XXX>
```

```
<AAA mark="a1"/>
```

```
<AAA mark="a2"/>
```

```
<AAA mark="a3"/>
```

```
<BBB id="b001" />
```

```
<CCC ref="a3" />
```

```
<DDD ref="a1 b001 a2"/>
```

```
</XXX>
```

✚ Dada la siguiente DTD:

```
<!ELEMENT album (#PCDATA)>
<!NOTATION jpg SYSTEM "visor_de_jpg.exe">
<!NOTATION gif SYSTEM "visor_de_gif.exe">
<!ATTLIST album fotos ENTITIES #IMPLIED>
<!ENTITY playa SYSTEM "imagenes/vacaciones/playa.gif" NDATA gif>
<!ENTITY brasil SYSTEM "http://www.brasil.com/copacabana.gif" NDATA gif>
<!ENTITY candela SYSTEM "candela/escuela/primer_dia.jpg" NDATA jpg>
```

❖ El siguiente es un documento válido:

```
<!DOCTYPE XXX SYSTEM "tutorial.dtd">

<XXX>

  <album fotos="playa brasil candela"> ....</album>

</XXX>
```

3.1.2.5 Secciones condicionales

En ocasiones puede resultar útil ocultar algunas partes de la declaración de la DTD. Las palabras clave INCLUDE e IGNORE permitirán respectivamente incluir o ignorar secciones de declaraciones dentro de una DTD. La siguiente tabla muestra la sintaxis utilizada para realizar este tipo de operación:

Palabras	Sintaxis	Ejemplo
INCLUDE	<![INCLUDE[declaraciones visibles]]>	<![INCLUDE[<!ELEMENT pseudo #PCDATA]]>
IGNORE	<![IGNORE[declaraciones a ocultar]]>	<![IGNORE[<!ELEMENT contraseña #PCDATA]]>

Al utilizar INCLUDE, el contenido de la declaración es visible en la DTD. Sin embargo, la palabra clave IGNORE permite ocultar e ignorar algunos elementos de la DTD. El contenido ignorado o incluido aparece entre corchetes justo después de las palabras clave IGNORE o INCLUDE. El siguiente es un ejemplo de su utilización:

```
<!ENTITY % prueba "INCLUDE" >
<!ENTITY % final "IGNORE" >
< [ % prueba; [
<!ELEMENT libro ( comentarios*, titulo, cuerpo, anexos? ) > ]]>
< [ % final; [
<!ELEMENT libro ( titulo, cuerpo, anexos? ) > ]]>
```