4. Documentos XML

4.1. Declaración XML

La especificación XML, es una línea que define una sintaxis para identificar que se trata de un documento XML; lo hacemos por medio de una etiqueta especial llamada declaración XML, que sirve para indicar que un documento es XML.

```
<? xml  version = "1.0" ?>
```

La declaración XML **es opcional**, aunque se considera recomendable, ya que tiene algunos atributos que pueden ayudar a los programas a entender características del documento: código de caracteres que se utiliza, la versión de XML que se ha usado , etc.

Si la declaración está presente se debe tener en cuenta que:

- La declaración debe estar en la primera línea del documento y debe comenzar en el primer carácter del documento.
- Debe tener obligatoriamente el atributo versión, que actualmente sólo puede ser 1.0 o 1.1.

Por lo tanto, la siguiente declaración es errónea porque deja una línea en blanco, deja espacios ante la declaración y no define el atributo versión:

```

<? xml?>
```

Como la declaración **es opcional** podemos encontrar documentos XML sin esta declaración. Si esto ocurre los programas que lean el documento deben suponer que se trata de un documento XML versión 1.0.

Atributos de la declaración XML

La declaración XML permite definir tres atributos (Tabla 4.1).

Tabla Atributos de la declaración XML

atributo	objetivo	Obligatorio?
version	Define qué versión de XML se ha utilizado para crear el documento.	sí
encoding	Define el código de caracteres que utiliza el documento.	no
standalone	Sirve para indicar si el documento no depende de otros documentos.	no

- **Atributo " versión "**

Sirve para especificar qué versión de XML debe interpretar el documento. Aunque la definición XML es opcional, si se especifica, este **atributo** se convierte en **obligatorio**.

Por lo tanto, si se carga el documento siguiente en un programa le estamos dando instrucciones que debe tratar el documento según las especificaciones de la versión 1.1.

```
<? xml  version="1.1"  ?>
<nombre > Pedro García </nombre >
```

Si no se especifica la definición o bien el valor es incorrecto los programas pueden ignorar lo que haya y suponer que la versión es la 1.0. Por lo tanto, los dos documentos siguientes deben ser interpretados como 1.0:

```
<? xml  version="1.1"  ?>
<nombre > Pedro García </nombre >
<nombre > Pedro García </nombre >
```

- **Atributo “encoding”**

Permite definir con qué código de caracteres se han codificado los datos.

Por defecto todos los programas que lean XML deben poder interpretar los códigos de caracteres que estén en UTF-8 y UTF-16 e incluso detectar en cuál de las dos codificaciones está hecho el documento. Esto implica que si el documento está creado con alguna de estas codificaciones no sería necesario especificar el atributo encoding. Sin

embargo, por motivos de legibilidad del documento menudo se especifica igualmente.

> Como el estándar ASCII es un subconjunto de UTF-8 los documentos que estén en ASCII tampoco requieren ninguna declaración.

En cambio, si el documento sigue alguna otra codificación, se debe especificar explícitamente. Para un documento que utiliza el código de caracteres ISO-8859-15 es obligatorio tener definido este atributo indicando el código de caracteres que se ha utilizado:

```
<? xml  version="1.0"  encoding="ISO-8859-15" ?>
```

Se pueden utilizar las abreviaciones de códigos de caracteres de la Tabla 4.2.

Tabla 4.2 abreviaciones de códigos de caracteres aceptadas por el XML

tipo	Códigos de caracteres
Unicode	UTF-8, UTF-16, ISO-10646-UCS-2, e ISO-10646-UCS-4
ASCII y variantes	ISO-8859-1, ISO-8859-2, ..., ISO-8859-n
JIS X-0208-1997	ISO-2022-JP, Shitf_JIS, EUC-JP

Para otros códigos de caracteres se recomienda utilizar los nombres que define la IANA (http://www.Iana.Org/assignments/character-sets).
Si a pesar de ello el código de caracteres que se utiliza no está en la lista se hará comenzar el nombre con x-.

- **Atributo " standalone "**

Los documentos de esquemas pueden hacer que algunas de las etiquetas se traten de manera diferente según el programa que los use. Por ejemplo, definiendo atributos por defecto en algunas etiquetas; no será necesario que sean específicamente declaradas en el documento XML.
En la definición del vocabulario se puede definir que el elemento <persona> tenga un atributo versión que por defecto es home. Al crear el documento se podría definir la

etiqueta sin especificar el atributo porque su valor está implícito.

```
<persona > Juan </persona >
```

Si un programa trata este documento debe tener en cuenta este valor, y por lo tanto tendrá que leer el documento que declara los atributos por defecto.

Con el atributo standalone se está definiendo si el documento XML se puede entender por sí solo o bien necesita que sea interpretado con la ayuda de algún otro documento, por lo que sólo tiene dos valores posibles (Tabla 4.3).

Tabla 4.3 Valores posibles del atributo standalone

valor	significado
yes	El documento es completo, y por lo tanto no necesita otros documentos para ser interpretado.
no	El documento no se puede entender por sí solo y debemos leer de otros para poder entender.

Si el atributo no se especifica se considera que el valor del atributo standalone es no.

4.2. Corrección

Los programas de ordenador no tienen la flexibilidad que tienen los documentos XML y necesitan poder trabajar con datos muy pautados, es por ello que se intentan hacer reglas para evitar que haya partes del documento que puedan ser malinterpretadas. Como la mayoría de las veces los documentos XML deberán ser leídos por programas, es muy importante tener alguna manera de comprobar que este documento XML esté bien formado, que cumpla las reglas de definición de XML.

> Comprobar la corrección de un documento XML es comprobar que no se incumple ninguna de las reglas de definición de XML.

Se considera que un documento es correcto o bien formado si cumple con las reglas básicas de creación de un documento XML. En general podemos definir estas reglas como:

- Sólo puede haber un elemento raíz.
- Todas las etiquetas que se abren deben cerrarse.
- Las etiquetas deben estar imbricadas correctamente.
- Los nombres de las etiquetas deben ser correctos.
- Los valores de los atributos deben estar entre comillas.

1. Sólo puede haber un elemento raíz

En el ejemplo siguiente:

```
<persona >
  <nombre > Pedro </nombre >
  <apellido > García </apellido >
</persona >
```

La etiqueta que sale en primer lugar, <persona> y que termina al final, </persona> define lo que se denomina elemento raíz. Que un elemento sea raíz significa que no tiene ningún elemento que lo contenga.

En este otro ejemplo, en cambio, hay dos elementos raíz, <persona>y <perro>:

```
<persona >
  <nombre > Pedro García </nombre >
</persona >
<perro >
  <nombre > Rufy </nombre >
</perro >
```

Se puede ver que tanto <persona> como <perro> no tienen ningún elemento que los contenga y, por tanto, ambos son elementos raíz. Esto hace que **no** sea un documento XML **bien formado**.

También si hay **dos elementos raíz** con las mismas etiquetas, Por ejemplo, en el documento siguiente tenemos dos elementos raíz <persona>. Por este motivo, este documento tampoco está bien formado.

```
<persona >
  <nombre > Pedro </nombre >
  <apellido > Pérez </apellido >
</persona >
<persona >
  <nombre > María </nombre >
  <apellido > García </apellido >
</persona >
```

Si se quisiera convertir el documento anterior en un documento bien formado se podría poner un elemento nuevo que englobara los dos elementos raíz. Por ejemplo, se añade un elemento <persones>y así se crea un documento XML bien formado:

```
<personas >
  <persona >
    <nombre > Pedro </nombre >
    <apellido > Pérez </apellido >
  </persona >
  <persona >
    <nombre > María </nombre >
    <apellido > García </apellido >
  </persona >
</personas >
```

2. Todas las etiquetas que se abren deben cerrarse

A pesar de lo que ocurre en otros lenguajes de marcas, por ejemplo, HTML, en el XML todas las etiquetas que se abran deben ser cerradas obligatoriamente. Por lo tanto, este sería un ejemplo correcto:

```
<nombre > Pedro </nombre >
```

Mientras que este no sería correcto:

```
<nombre > Pedro
```

En algunos momentos puede ser necesario reflejar algún dato que no requiera ningún valor. Por ejemplo, si tenemos una lista de alumnos, nos podría interesar marcar cuál de ellos es el delegado. Para definirlo no nos haría falta ningún dato, ya que simplemente el alumno que tenga la etiqueta <delegado>es el delegado.

Por lo tanto, para definir que Federico Gil es el delegado podríamos estar tentados de hacer esto:

```
<alumnos >
  <alumno >
    <nombre > Pedro García </nombre >
  </alumno >
  <alumno >
    <nombre > Federico Gil </nombre >
    <delegado >
  </alumno >
  <alumno >
    <nombre > María Pérez </nombre >
  </alumno >
</alumnos >
```

Es **incorrecto**. Si abrimos una etiqueta la tenemos que cerrar; por lo tanto, deberíamos hacer:

```
...
<alumno >
  <nombre > Federico Gil </nombre >
  <delegado > </delegado >
</alumno >
...
```

O bien utilizar la versión de definición de etiquetas vacías terminando la etiqueta con "/>":

```
...
<alumno >
   <nombre > Federico Gil </nombre >
   <delegado />
</alumno >
...
```

3. Las etiquetas deben estar organizadas correctamente

Para mantener la jerarquía, XML define que:

- Las etiquetas no se pueden cerrar hasta que se haya cerrado alguna etiqueta que forma parte del contenido.
- Es decir, no se permite que un elemento tenga una parte de su contenido en un elemento y otra parte de su contenido en otro elemento; esto rompería la **jerarquía** de datos.

4. Cierre de las etiquetas desordenado

Algunos lenguajes de marcas permiten abrir y cerrar las etiquetas en cualquier orden. Por ejemplo, las versiones anteriores a la 4.0 de HTML permitían abrir y cerrar etiquetas en el orden que quisiéramos siempre que acabaran cerrándose todas. Pero a partir de la versión 4.0 ya no se acepta, a pesar de que la mayoría de los navegadores sí lo permiten.

En XML siempre que se abra una etiqueta dentro de un elemento, ésta debe ser cerrada antes de terminar el elemento. El ejemplo sería correcto porque el elemento <persona> en su contenido abre la etiqueta <nombre> pero antes de terminar cierra </nombre>:

```
<persona > <nombre > Pedro </nombre > </persona >
```

En cambio, si las etiquetas no se cierran en el orden correcto, aunque se cumpla la regla que se deben cerrar todas las etiquetas, no tenemos un documento XML bien formado.

```
<persona > <nombre > Pedro </persona > </nombre >
```

En el ejemplo que acabamos de ver se está creando un bucle, ya que <persona>contiene una parte de <nombre >tiempo que <nom>contiene una parte de <persona>. Esto rompe con la idea de jerarquía de los datos que define el XML y, por tanto, no es correcto.

Para facilitar la escritura de documentos XML y minimizar los posibles errores por culpa de cierres desordenados, los documentos XML a menudo se representan sangrados.

> **sangría**
> La sangría consiste en escribir las etiquetas de manera sangrado en función de su nivel dentro de la jerarquía de elementos. A medida que se van creando nuevos elementos dentro de otros se van añadiendo sangrías.

En el XML la sangría se hace en función del nivel en que se encuentran los datos. En general cada uno de los hijos hace que se incremente la sangría. Por ejemplo:

```
<alumno >
   <persona >
      <nombre >
      </nombre >
   </persona >
</alumno >
```

Utilizando la sangría es más fácil determinar en qué lugar se ha producido un error de cierre de etiquetas y arreglarlo. Como podemos ver en el ejemplo siguiente, si las etiquetas de cierre no están en la misma columna que las de apertura es que hay algo mal.

```
<instituto >
   <clase >
      <alumno > Pedro </alumno >
      <alumno > Juan </alumno >
   </instituto >
</clase >
```

En general la sangría aporta dos ventajas al XML:

1. Evita que se cometan errores a la hora de cerrar las etiquetas.
2. Hace que sea más fácil ver de un vistazo la estructura del documento.

5. Los nombres de las etiquetas y los atributos deben ser correctos

El XML da mucha libertad a la hora de definir los nombres de los elementos y los atributos, pero no todos los nombres son aceptados.

En general tendremos que:

- No se pueden poner nombres que no empiecen por un carácter
- No pueden contener espacios en su interior.
- Las mayúsculas y las minúsculas son caracteres diferentes para XML.
- Los nombres que empiecen por *xml* están reservados.

Por lo tanto, sería incorrecto escribir una etiqueta en minúsculas y después cerrarla en mayúsculas:

```
<persona > Pedro </Persona >
```

No se pueden poner etiquetas ni atributos que empiecen por *xml* porque están reservados por el estándar.
En el ejemplo siguiente hay un error en el nombre del atributo xml persona:

```
<persona  xmlpersona="si" > Pedro </persona >
```

De hecho, ya hay algunos atributos que comienzan con xml que se aceptan porque están definidos en el estándar y tienen un objetivo claro: xml:lang, xml:space, xmlns...

6. Los valores de los atributos deben estar entre comillas

En el apartado de definición de los atributos ya se comentó que los valores de los atributos siempre deben ir entre comillas, pero podemos resumir lo más importante.

- No importa cuales sean las comillas que usamos, dobles o simples, siempre que se utilicen las mismas en abrir que al cerrar.

```
<clase  nombre="Lenguajes de marcas" >
   <alumno  delegado='si' > Pedro García </alumno >
   <alumno > Federico Ruiz </alumno >
</clase >
```

- Aunque el contenido de los atributos sea un valor numérico debe especificarse entre comillas:

```
<alumno  nota="10" > María Prados </nombre >
```

- Podemos especificar tantos atributos como sea necesario en una etiqueta y el orden en que se especifican no tiene importancia:

```
<alumno  nombre="Pedro"  apellido="García"  nota="6"  />
```

7. procesadores XML

El objetivo principal de todas las reglas que definen cómo se ha de escribir un documento XML es definir una manera de escribir documentos XML que puedan ser leídos e interpretados por un programa de ordenador. Los p**rogramas encargados de detectar si un documento XML está bien formado** denominan genéricamente procesadores de XML, aunque a menudo se utiliza el nombre en inglés, *parsers*.

Un procesador de XML es un programa que permite leer un documento XML y acceder a su estructura y a los datos que contiene. Esto lo hace leyendo todo el documento y determinando cuáles de los caracteres que encuentra son parte de la estructura y cuáles son realmente datos.

4.3. Estructura de los documentos XML

Una de las cosas que se consiguen los procesadores es forzar a que los documentos XML cumplan con que la información que contienen, se organice de **manera jerárquica**.

Por ejemplo, tenemos el documento XML siguiente:

```
<? xml versión n="1.0" encoding="UTF-8"?>
<clase >
  <profesor >
    <nombre > Mario </nombre >
    <apellido > Palacios </apellido >
  </profesor >
  <alumnos >
    <alumno >
      <nombre > Filomeno </nombre >
      <apellido > García </apellido >
    </alumno >
    <alumno >
      <nombre > Federico </nombre >
      <apellido > Gil </apellido >
    </alumno >
    <alumno >
      <nombre > Manuel </nombre >
      <apellido > Pérez </apellido >
      <delegado />
    </alumno >
  </alumnos >
</clase >
```

Gracias a que las etiquetas tienen nombres claros se puede deducir que en este documento XML están definiendo estructuradamente una clase con un profesor y tres alumnos. Hay un elemento raíz llamado <clase>que engloba todo el documento, y como contenido tiene dos elementos más, <profesor> y <alumnos>.

Los elementos que forman parte del contenido de un nodo se llaman **hijos**.

Por lo tanto, <**clase**>tiene dos elementos hijos:

<profesor>

<alumnos>

Asimismo, el elemento <**profesor**> que es uno de los hijos de <clase> también tiene dos hijos:

<nombre>

<apellido>.

Por equivalencia con las relaciones humanas a los hijos de los hijos de un elemento se denominan **nietos**. Por lo tanto, <nombre>y <apellido> son hijos de <profesor>y nietos de <clase>:

```
clase -> profesor -> nombre
```

Los elementos <nombre>y <apellido> no tienen nodos hijos, sino que contienen los datos del documento. Es el caso de las dos cadenas de texto Manuel y Pérez. Los nodos finales se denominan **hojas** y generalmente serán siempre nodos de datos.

Este hijo de <clase>, pero tiene un nivel más. Ahora se tiene una estructura más larga:

```
clase -> alumnos -> alumno -> nombre
```

Todos los elementos que cuelgan de un elemento se denominan genéricamente **descendientes**. Por lo tanto, <alumnos>, <alumno> y <nombre> son descendientes de <clase>.

Como puede ver, es relativamente sencillo interpretar cuáles son los datos que contiene un documento XML, gracias a que

- Se usan nombres de etiqueta que tienen sentido para un posible lector,
- Se puede deducir cuál es la estructura de los datos y las relaciones que tendrán estos datos entre sí.

El hecho de seguir este sistema, que es flexible en determinados momentos, pero estricto en otros, nos permite hacer que un programa de ordenador también pueda de manera sencilla detectar rápidamente qué parte del documento son datos y qué son metadatos, pero que además pueda establecer cuál es la relación entre estos datos.

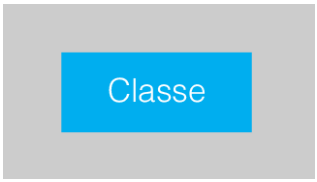
En ningún momento se ha especificado nada sobre cómo se deberán representar los datos, ya que ésta es una de las bases fundamentales del lenguaje XML:

> El objetivo de la XML es representar la estructura de los datos olvidándose de cómo se hará para presentarlas.

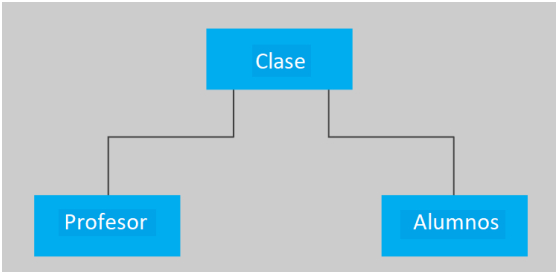
Representación en forma de árbol

Una de las características interesantes de los documentos XML es que, al tener los datos estructurados de manera jerárquica, esta jerarquía hace que los documentos XML puedan ser representados gráficamente en forma de árbol. La representación en árbol es útil para comprender mejor cuál es la estructura del documento.

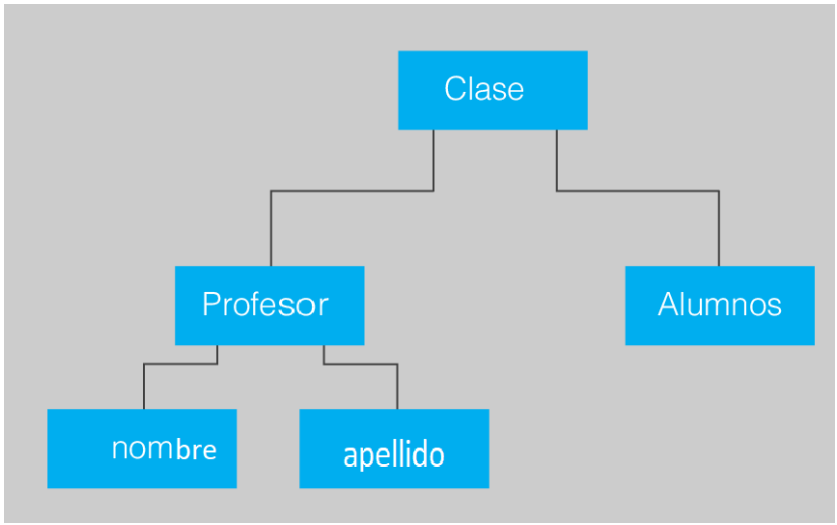
Para hacer la representación en forma de árbol siempre se parte del nodo raíz, que en este caso es <clase>



A partir del nodo raíz se toman los hijos directos que tenga y se enlazan con un arco, que será el que indicará la relación que tienen los elementos entre ellos:



Lo mismo se puede hacer para los hijos de cada uno de los nodos. Por ejemplo, profesor tiene dos hijos más, que son nombre y apellido, que se unirán a profesor con un arco para indicar que tienen relación padre/hijo.



Como ya no hay más elementos para representar ya se pueden pintar los nodos de datos, que generalmente se suelen pintar de color diferente