2022

Tema 4

DEFINICIÓN DE ESQUEMAS Y VOCABULARIO XML. DTD Y XSD

STUDIUM

www.grupostudium.com
informacion@grupostudium.com
954 539 952

4.1 Introducción

Al inicio de la asignatura se habló de los lenguajes de marcas en general, entre los que destacaban HTML y XML. Ya se ha estudiado HTML como lenguaje de marcas para el diseño de páginas webs soportando la información y la visualización dejándosela a las hojas de estilo CSS.

Ahora nos disponemos a trabajar con detalle con XML. Tal como se comentó es mucho más flexible que HTML a la hora de representar la información, porque, entre otras cosas, permite crear nuestras propias etiquetas, las necesarias para representar la información de la mejor forma posible. Admite también hojas de estilo CSS como veremos más adelante, pero realmente la utilidad de XML es representar de la forma óptima información relevante. Es el estándar de intercambio de datos entre sistemas informáticos basados en la web. Estamos hablando de mensajería, videoconferencias, correo, streaming, ... En los últimos años está ganando terreno, en cuanto al intercambio de información, otro formato llamado JSON. Lo estudiaremos al final del tema.

www.grupostudium.com
informacion@grupostudium.com
954 539 952

4.2 ¿Qué es un DTD?

DTD significa Document Type Definition o Definición del tipo de Documento. Pero ¿qué define? Pues los elementos y atributos que pueden aparecer en el documento XML. Especifica la estructura que debe tener la información expresada en el XML.

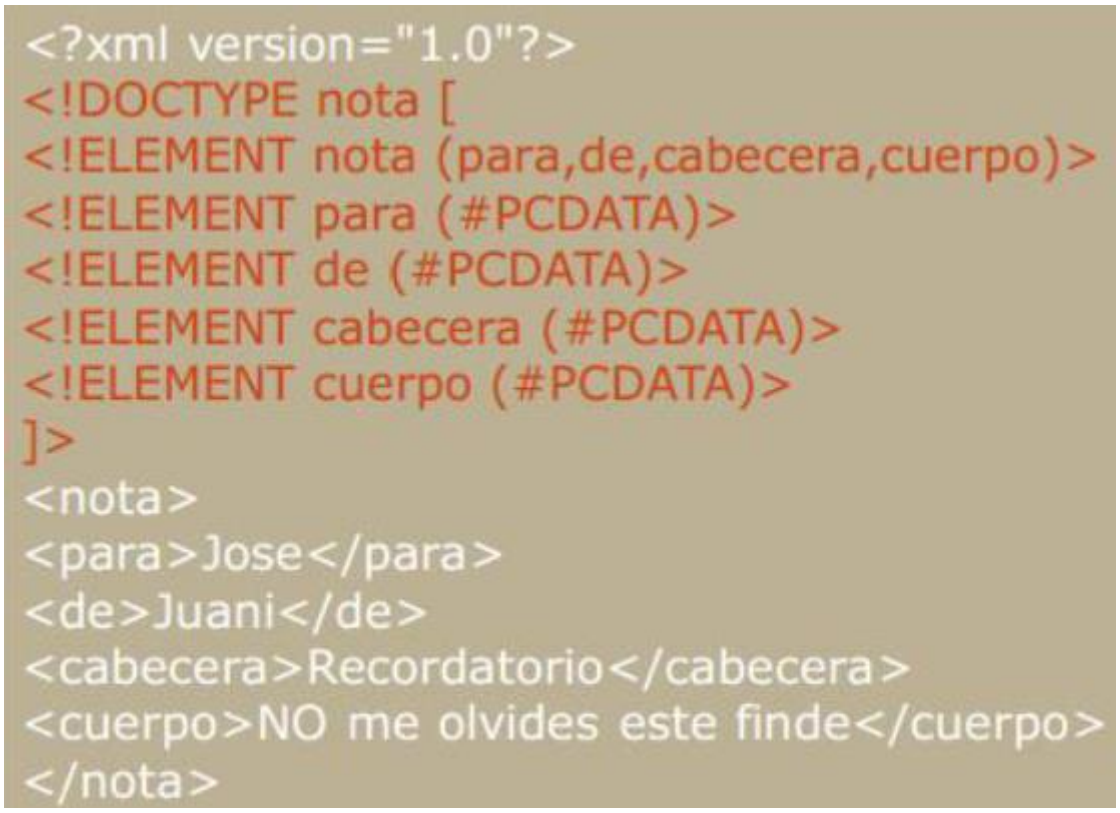
Un DTD puede ser declarado en línea dentro de un documento XML (interna), o como una referencia externa (externa) o como ambos.

Declaración interna

Debe seguir la siguiente sintaxis:

```
<!DOCTYPE elemento_raíz [declaración de elementos]>
```

Ejemplo de documento XML con una DTD interna:



En rojo se ha marcado la Definición, el DTD, mientras que, en blanco, aparecen los datos. El DTD se puede asemejar al diseño de una tabla en SQL: se especifican los nombres de los campos, los tipos, ...

Declaración externa

En este caso, tenemos la definición en un fichero externo, es decir, la definición de la estructura y los datos se separan físicamente en dos ficheros independientes, pero relacionados. Debe seguir la siguiente sintaxis:

<!DOCTYPE elemento raíz SYSTEM “archivo">

Vemos el mismo ejemplo anterior, pero ahora, en ficheros separados:

nota.xml

```
<?xml version="1.0"?>
 <!DOCTYPE nota SYSTEM "nota.dtd">
<nota>
<para>Jose</para>
<de>Juani</de>
<asunto>Recordatorio</asunto>
<mensa>NO me olvides este finde</mensa>
</nota>
```

nota.dtd

```
<!ELEMENT nota
(para,de,cabecera,cuerpo)>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT mensa (#PCDATA)>
```

Ventajas e inconveniente de usar DTD

Ventajas:

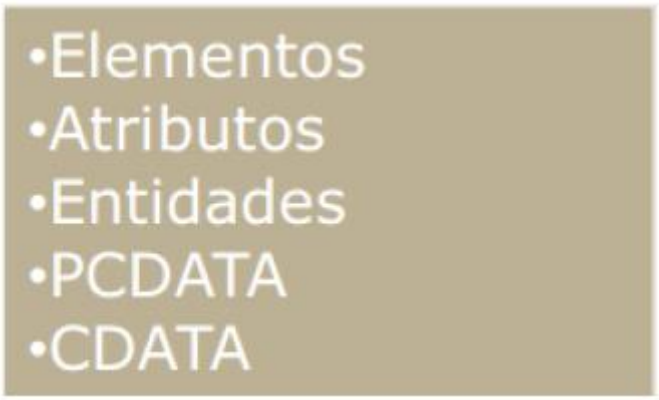
- Con una DTD, cada uno de los archivos XML se puede llevar a una descripción de su propio formato.
- Con una DTD, grupos independientes de personas se ponen de acuerdo para utilizar una DTD estándar para intercambiar datos.
- Su aplicación puede utilizar una norma DTD para verificar que los datos que recibimos del mundo exterior es válida.
- También puede utilizar un DTD para verificar sus propios datos.

Inconvenientes:

- Su sintaxis no es XML.
- No soportan espacios de nombres.
- No definen tipos para los datos. Solo hay un tipo de elementos terminales, que son los datos textuales.
- No es posible formar claves a partir de varios atributos o elementos.
- Una vez que se define un DTD no es posible añadir nuevos vocabularios.

Bloques de Construcción de documentos XML

Visto desde un punto de vista DTD, todos los documentos XML (y los documentos HTML) están compuestos por los siguientes apartados:



Elementos

Los elementos son los bloques de construcción principales de documentos HTML y documentos XML.

Ejemplos de elementos HTML son <body> y <table>.

Ejemplos de elementos XML podrían ser <nota> y <mensaje>.

Los elementos pueden contener texto, otros elementos, o estar vacío.

Ejemplos:

```
<body>texto</body>
<mensaje>texto</mensaje>
```

En una DTD, los elementos XML se declaran con una declaración del elemento con la siguiente sintaxis:

```
<!ELEMENT nombre-elemento categoría>
```

o

```
<!ELEMENT nombre-elemento (elemento)>
```

Veamos algunos ejemplos:

1 Elementos vacíos: Elementos que no contienen información. Son más bien estructurales.

Su sintaxis sería:

```
<!ELEMENT nombre-elemento EMPTY>
```

Si definimos un salto de línea en un DTD sería:

```
<!ELEMENT br EMPTY>
```

Si lo definimos en un XML sería:

```
<br/>
```

2 Elementos de datos PCDATA: Los elementos con los datos de caracteres se analizan con la declaración #PCDATA entre paréntesis. Indica que el elemento contiene un tipo de dato analizable o usable.

Su sintaxis sería:

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

Si definimos un elemento que represente un origen de un mensaje en un DTD sería:

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

Si lo definimos en un XML sería:

```
<de>Juani</de>
```

3 Elementos con cualquier contenido ANY: Los elementos que pueden tener cualquier contenido son declarados con ANY. Pueden contener cualquier combinación de los datos apta para su procesamiento.

Su sintaxis sería:

```
<!ELEMENT nombre-elemento ANY>
```

Si definimos un elemento que represente una nota en un DTD sería:

```
<!ELEMENT nota ANY>
```

Si lo definimos en un XML podría ser:

```
<nota>
        <para>José Antonio</para>
        <de>Juani</de>
        <asunto>Quedada</asunto>
        <mensaje>No os olvidéis de avisar a Chema</mensaje>
</nota>
```

4 Elementos con hijos (hijo1, hijo2...): Los elementos con uno o más hijos se declaran con el nombre de los elementos de los hijos entre paréntesis.

Su sintaxis sería:

```
<!ELEMENT nombre-elemento (hijo1)>

<!ELEMENT nombre-elemento (hijo1, hijo2, ...)>
```

www.grupostudium.com
informacion@grupostudium.com
954 539 952

Si definimos un elemento que represente una nota en un DTD sería:

```
<!ELEMENT nota (para, de, asunto, mensaje>
<!ELEMENT para (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT asunto (#PCDATA)>
<!ELEMENT mensaje (#PCDATA)>
```

Si lo definimos en un XML podría ser:

```
<nota>
        <para>José Antonio</para>
        <de>Juani</de>
        <asunto>Quedada</asunto>
        <mensaje>No os olvidéis de avisar a Chema</mensaje>
</nota>
```

En esta declaración cada elemento secundario o hijo debe aparecer una vez, y sólo una vez dentro del elemento padre. Si necesitamos o queremos dar la posibilidad de que aparezca una o más de una vez, añadimos el signo de más, como podemos ver en el siguiente ejemplo:

Si definimos un elemento que represente una nota en un DTD sería:

```
<!ELEMENT nota (mensaje+)>
```

Si lo definimos en un XML podría ser:

```
<nota>
        <mensaje>No os olvidéis de avisar a Chema</mensaje>
        <mensaje>Al final llevo yo las bebidas</mensaje>
</nota>
```

Si lo que deseamos es indicar la posibilidad de que aparezca entre 0 y muchas veces, usaremos el asterisco. Modificando el anterior ejemplo tendríamos:

```
<!ELEMENT nota (mensaje*)>
```

Por último, si lo que queremos expresar es la posibilidad de 0 o una ocurrencia del hijo, usaremos el signo de interrogación:

```
<!ELEMENT nota (mensaje?)>
```

También podemos expresar que aparezca un hijo u otro de forma excluyente. Con el signo |, denotamos que pude aparecer uno u otro elemento secundario. El ejemplo declara que el elemento "nota“ debe contener los elementos “para“, "de“, “asunto“, y el elemento "mensaje" o “cuerpo”:

Su sintaxis sería:

```
<!ELEMENT nombre-elemento (hijo1, hijo2, ... , (hijoX | hijoY))>
```

Si definimos un elemento que represente un origen de un mensaje en un DTD sería:

```
<!ELEMENT nota (para, de, asunto, (mensaje | cuerpo))>
```

Si lo definimos en un XML sería:

```
<nota>
        <para>José Antonio</para>
        <de>Juani</de>
        <asunto>Quedada</asunto>
        <mensaje>No os olvidéis de avisar a Chema</mensaje>
</nota>
```

O bien:

```
<nota>
        <para>José Antonio</para>
        <de>Juani</de>
        <asunto>Quedada</asunto>
        <cuerpo>No os olvidéis de avisar a Chema</cuerpo>
</nota>
```

5 Contenido mixto elemento (#PCDATA|hijo1|hijo2|hijo3|hijo4)*: El ejemplo define que el elemento "nota“ puede contener cero o más ocurrencias de datos de: caracteres analizados, o “para", o "de", o “asunto", o "mensaje“, o “cuerpo. (pero solo uno de ellos).

Su sintaxis sería:

```
<!ELEMENT nombre-elemento (#PCDATA | hijo1 | hijo2 | hijo3 | hijo4)*>
```

Si definimos un elemento que represente un origen de un mensaje en un DTD sería:

```
<!ELEMENT nota (#PCDATA | para | de | asunto | mensaje | cuerpo)*>
```

En XML podríamos tener:

```
<nota>
        <para>José Antonio</para>
        <para>Juani</para>
        <para>Luis</para>
</nota>
```

Y también sería válido:

```
<nota>para José Antonio</nota>
```

A modo resumen:

Símbolo	*Descripción*
Ninguno	El elemento aparece sólo una vez.
Signo más (+)	El elemento aparece una o más veces.
Asterisco (*)	El elemento es opcional y puede aparecer cualquier número de veces.
Signo de interrogación (?)	El elemento es opcional y puede aparecer sólo una vez.

Atributos

Como hemos visto en temas anteriores, los atributos proporcionan información adicional acerca de los elementos.

Los atributos se colocan siempre dentro de la etiqueta de apertura de un elemento, y siempre vienen en pares de: nombre / valor.

Ejemplo: El siguiente elemento "img" tiene información adicional acerca de un archivo de origen:

```
<img src="computer.gif" />
```

En una DTD, los atributos se declaran con una declaración ATTLIST con la siguiente sintaxis:

```
<!ATTLIST elemento atributo tipo-atributo valor-por-defecto>
```

Donde:

- "elemento" es el nombre del elemento para el que se define un atributo
- “atributo" es el nombre del atributo
- "tipo-atributo" es el tipo de datos
- "valor-por-defecto" es el valor predeterminado del atributo (aunque también puede indicar otras cosas)

Ejemplo aclaratorio:

```
<!ATTLIST pago tipo CDATA "cheque">
```

Estamos definiendo un atributo llamado “tipo” sobre la etiqueta “pago” cuyo contenido contendrá caracteres (CDATA) y que tendrá como valor por defecto “cheque”.

El tipo de atributo puede ser:

Tipo	Descripción
CDATA	El valor es un dato de carácter
(*en1* | en2 | ..)	El valor debe ser uno de una lista enumerada
ID	El valor es un identificador único
IDREF	El valor es el identificador de otro elemento
IDREFS	El valor es una lista de identificadores de otros
NMTOKEN	El valor es un nombre XML válido
NMTOKENS	El valor es una lista de nombres XML válidos
ENTITY	El valor es una entidad
ENTITIES	El valor es una lista de entidades
NOTATION	El valor es un nombre de una notación
xml:	El valor es un valor predefinido xml

NOTA: El tipo ID permite que un atributo determinado tenga un nombre único que podrá ser referenciado por un atributo de otro elemento que sea de tipo IDREF. Los atributos ID, no pueden empezar por número ni contener espacios.
NOTA: Los atributos CDATA (character data) son los más sencillos, y pueden contener casi cualquier cosa. Los atributos NMTOKEN (name token) son parecidos, pero sólo aceptan los caracteres válidos para nombrar cosas (letras, números, puntos, guiones, subrayados y los dos puntos).

No es obligatorio indicar el valor por defecto, pero en caso de no hacerlo debemos expresar alguna de estas otras situaciones:

Valor	Explicación
valor	El valor predeterminado del atributo
# REQUIRED	El atributo es necesario
# IMPLIED	El atributo no es necesario
# FIXED *valor*	El valor del atributo es fijo

Veamos algunos ejemplos.

Definición:

```
<!DOCTYPE ejemplo [
<!ELEMENT ejemplo EMPTY>
<!ATTLIST ejemplo color CDATA #REQUIRED>
]>
```

Uso:

```
<ejemplo></ejemplo> <!-- No es correcto, falta el atributo "color" -->
<ejemplo color=""></ejemplo>
<ejemplo color="amarillo"></ejemplo>
<ejemplo color="azul marino #000080"></ejemplo>
```

Definición:

```
<!DOCTYPE cuadrado [
<!ELEMENT cuadrado EMPTY>
<!ATTLIST cuadrado ancho CDATA "100">
]>
```

Uso:

```
<cuadrado/>
<cuadrado ancho="120" />
<cuadrado /><!--Toma el valor por defecto 100-->
<cuadrado ancho=""/><!--Toma el valor por defecto 100-->
```

Definición:

```
<!DOCTYPE persona [
<!ELEMENT persona EMPTY>
<!ATTLIST persona numero CDATA #REQUIRED>
]>
```

Uso:

```
<persona numero="5677" />


<persona  /> <!-- XML inválido -->
```

Definición:

```
<!DOCTYPE contacto [
<!ELEMENT contacto EMPTY>
<!ATTLIST contacto fax CDATA #IMPLIED>
]>
```

Uso:

```
<contacto fax="555-667788" />
<contacto />
```

Definición:

```
<!DOCTYPE producto [
<!ELEMENT producto CDATA>
<!ATTLIST producto gen CDATA #FIXED 'informatica' >
]>
```

Uso:

```
<producto gen='informatica'>portátil</producto>
<producto>portátil</producto>
<producto gen='literatura'>portátil</producto><!--incorrecto-->
<producto gen=''>portátil</producto><!--incorrecto-->
```

Definición:

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

En este caso, una etiqueta como...

```
<enlace destino="seccion-3">
```

haría referencia a otra etiqueta ...

```
<capitulo referencia="seccion-3">
```

de forma que el procesador XML lo podría convertir en un hipervínculo, u otra cosa.

Definición:

```
<!DOCTYPE ejemplo [
<!ELEMENT ejemplo (libro*)>
<!ELEMENT libro (#PCDATA) >
<!ATTLIST libro codigo ID #REQUIRED>
]>
```

Uso:

```
<ejemplo>
	<libro codigo="L1">Poema de Gilgamesh</libro>
</ejemplo>
<ejemplo>
	<libro codigo="1">Poema de Gilgamesh</libro>
<!-- Incorrecto porque el valor de un atributo de tipo ID no puede empezar con un número -
->
	<libro codigo="L2">Los preceptos de Ptah-Hotep</libro>
</ejemplo>
<ejemplo>
	<libro codigo="L1">Poema de Gilgamesh</libro>
	<libro codigo="L1">Los preceptos de Ptah-Hotep</libro>
	<!-- No es correcto porque el valor no puede repetirse -->
</ejemplo>
```

Definición:

```
<!DOCTYPE ejemplo [
<!ELEMENT ejemplo ((libro | prestamo)*)>
<!ELEMENT libro (#PCDATA) >
<!ATTLIST libro codigo ID #REQUIRED>
<!ELEMENT prestamo (#PCDATA) >
<!ATTLIST prestamo libro IDREF #REQUIRED>
]>
```

Uso:

```
<ejemplo>
	<libro codigo="L1">Poema de Gilgamesh</libro>
	<prestamo libro="L1">Numa Nigerio</prestamo>
</ejemplo>
<ejemplo>
	<libro codigo="L1">Poema de Gilgamesh</libro>
	<prestamo libro="L2">Numa Nigerio</prestamo>
```

```
        <!-- No es correcto porque el valor "L2" no es ID de ningún elemento -->
</ejemplo>
```

¿No suena esto a Bases de Datos Relacionales?

Llegados a este punto, se nos puede presentar la siguiente duda: ¿Usar elementos o atributos?

Algunos de los problemas con los atributos son:

- Los atributos no pueden contener varios valores (los elementos secundarios o hijos sí pueden)
- Los atributos no son fácilmente extensibles (para futuros cambios)
- Los atributos no pueden describir las estructuras (elementos secundarios si pueden)
- Los atributos son más difíciles de manipular por el código del programa
- Los valores de los atributos no son fáciles de probar con una DTD

Recomendación: Usar los atributos lo menos posible, sólo utilizar identificador único, o para indicar restricciones que no pueden expresarse con los atributos.

Entidades

Las entidades son las variables utilizadas para definir los accesos directos a texto estándar o caracteres especiales.

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

Pueden ser internas:

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

o externas:

```
<!ENTITY nombreEntidad SYSTEM "uri">
```

Ejemplos de cada una:

Definición de interna:

```
<!ENTITY escritor “Donald Duck.”>
<!ENTITY copyright “Copyright W3Schools” >
```

Uso en XML:

```
<autor>&escritor;&copyright;</autor>
```

Definición de externa:

```
<!ENTITY escritor SYSTEM "http://...">
<!ENTITY copyright SYSTEM "http://...">
```

Uso en XML (Quedaría igual):

```
<autor>&escritor;&copyright;</autor>
```

Al realizar la transformación tanto en un caso como el otro, el fichero XML que tendría que interpretar el navegador sería:

```
<autor>Donald Duck.Copyright W3Schools</autor>
```

4.3 XML Schema Definition (XSD)

Un XML Schema Definition (XSD) es un vocabulario para expresar las reglas de los datos que usaremos. Nos sirve de referencia para validar los datos que aparecen en el XML. Al igual que los DTD, un esquema XML describe la estructura de un documento XML.

XML Schema Definition (XSD) es una alternativa para la DTD basada en XML.

Su finalidad es definir los elementos válidos de un documento XML, al igual que un DTD.

Podemos decir que un Esquema XML (XSD):

- Define los elementos que pueden aparecer en un documento XML
- Define los atributos que pueden aparecer en un documento
- Define los elementos que son los elementos secundarios o hijos
- Define el orden de los elementos secundarios
- Define el número de elementos secundarios
- Define si un elemento está vacío o puede incluir texto
- Define los tipos de datos de los elementos y los atributos
- Define los valores por defecto y fijos para los elementos y los atributos

Referencia a un esquema en el documento XML

Una vez escrita una especificación XSD, podemos escribir un documento XML, y hacer constar en el mismo que debe ser conforme a dicha especificación. Esto se hace añadiendo algunos atributos al elemento raíz del documento XML.

Estructura de un Esquema XML

Tiene la estructura habitual de todo documento XML con la obligación de que el elemento raíz se llame *schema* y la extensión del fichero es xsd.

En esta etiqueta se declara el espacio de nombres estándar que utilizan los esquemas (y que permite diferenciar las etiquetas XML del esquema, respecto a las del documento XML). Se pueden dar estas tres posibilidades:

- <schema xmlns="http://www.w3.org/2001/XMLSchema">
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

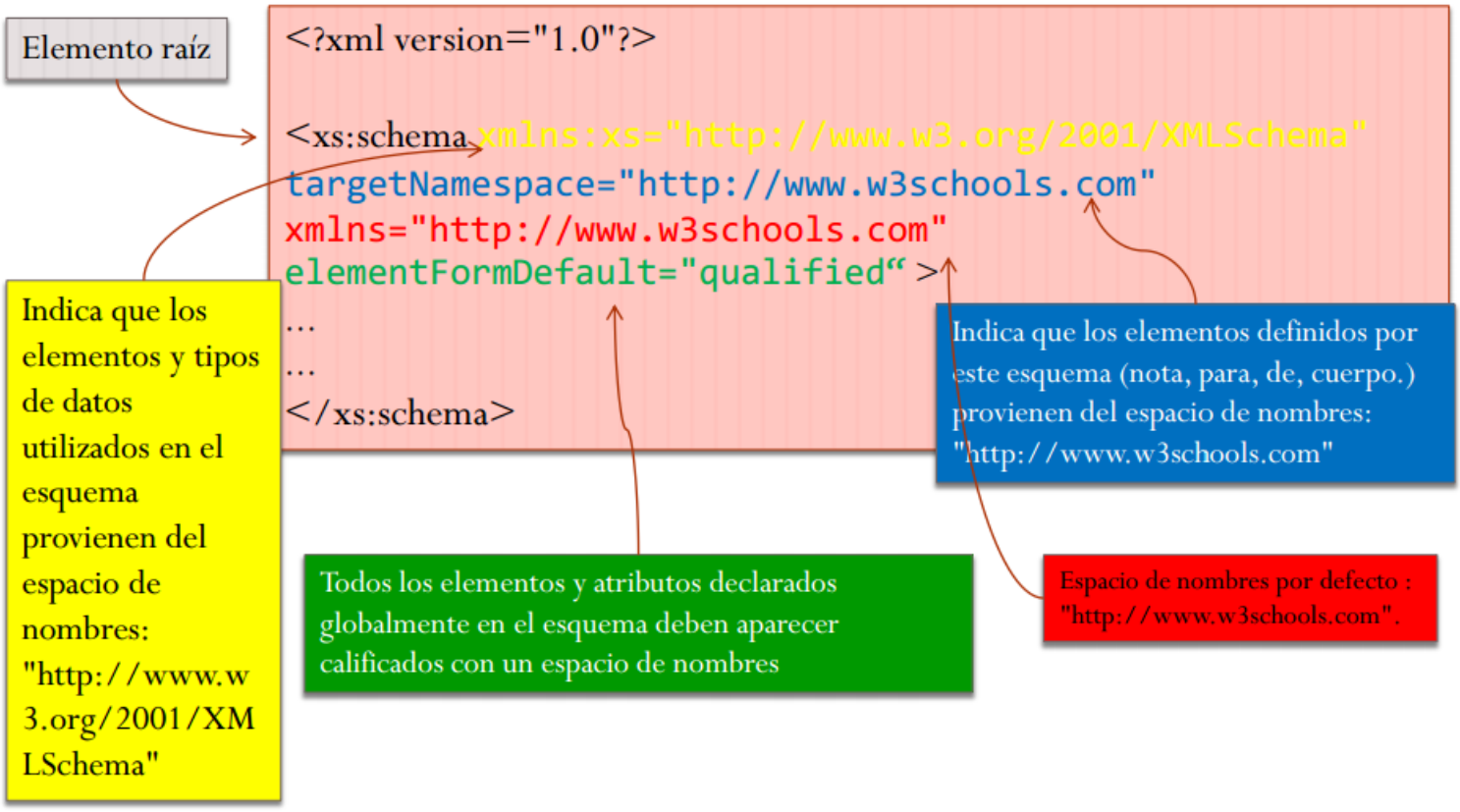
En el primero el espacio de nombre se define por defecto, el segundo se usa el prefijo xs (es la forma habitual) o bien definir un prefijo xsd.

A partir de ahí las etiquetas pertenecientes a XML Schema, usarán el prefijo indicado en su espacio de nombres (normalmente xs).

El atributo targetNamespace permite indicar el espacio de nombres sobre el que se aplica el esquema (si se aplica a varios espacios de nombres, aparecerán separados con espacios), que es como decir a qué documentos se aplicará el esquema.

Por último, se define el espacio de nombres al que se aplica el esquema. Normalmente un esquema XML se aplica a un espacio de nombres privado. Como siempre la etiqueta que declara el espacio de nombres es xmlns.

Veamos una declaración de documento XSD, completa:



www.grupostudium.com
informacion@grupostudium.com
954 539 952

Asociar un esquema a un documento XML

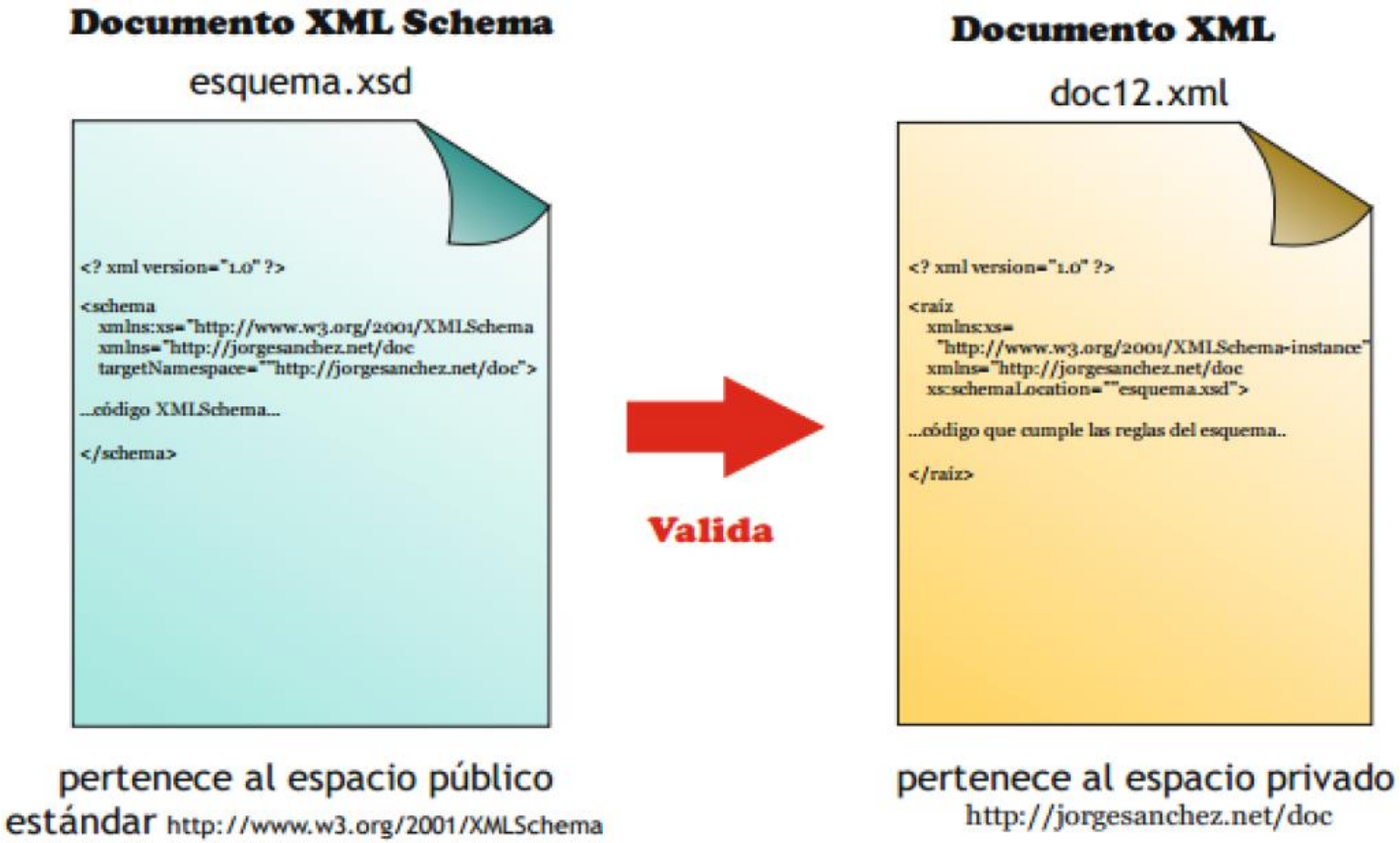
Para que un documento XML siga las reglas definidas en un esquema, no disponemos de etiqueta !DOCTYPE como en DTD; en su lugar utilizamos atributos especiales en el elemento raíz del documento XML.

Como ya hemos visto al igual que en el documento XMLSchema, necesitamos:

- Definir los dos espacios de nombres, el correspondiente al documento XML (sin abreviatura, es decir como espacio por defecto) y el espacio de nombres de XML Schema (que suele utilizar el prefijo xs)
- Indicar dónde está el archivo XMLSchema que contiene las reglas de validación que se aplican al documento. Esto se hace gracias al atributo llamado schemaLocation

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns="http://www.jorgesanchez.net/doc"
   xmlns:xs="http://w3.org/2001/XMLSchema-instance"
   xs:schemaLocation="esquema.xsd">
  ....
</documento>
```

El atributo schemaLocation indica la localización del documento XMLSchema que contiene la definición de las reglas a cumplir por el documento actual.



Partes de un esquema

- Elementos, definidos con etiquetas xs:element. Se utiliza para indicar los elementos permitidos en los documentos que sigan el esquema.
- Atributos, etiqueta xs:attribute.
- Tipos simples, que permiten definir los tipos simples de datos que podrá utilizar el documento XML. Lo hace la etiqueta xs:simpleType.
- Tipos complejos, mediante la etiqueta xs:complexType.
- Documentación, información utilizable por aplicaciones que manejen los esquemas. Etiquetas xs:annotation, xs:documentation y xs:appInfo.

Sintaxis de la definición de un elemento

En XML Schema la definición de un elemento XML se realiza mediante la etiqueta element. La sintaxis completa es:

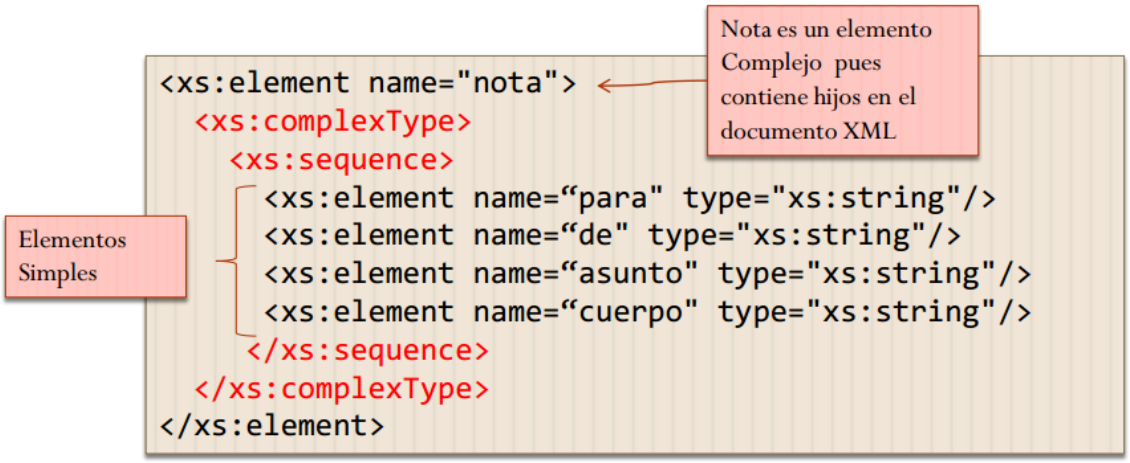
```
<xs:element
   name="nombre del elemento"
   type="tipo global de datos"
   ref="declaración del elemento global"
   id="identificador"
   form="cualificación" <!--qualified o unqualified →
   minOccurs="número mínimo de veces"
   maxOccurs="máximo número de veces"
   default="valor por defecto"
   fixed="valor fijo"
>
```

Al menos hay que indicar el nombre; el tipo de datos también es necesario indicarlo casi siempre; el resto de los atributos sólo si se necesitan.

Tipos de Datos

Un esquema contiene elementos que pueden ser:

- Simples: No pueden tener ni elementos ni atributos
- Complejos: Pueden contener otros elementos y atributos



Elementos simples

Un elemento simple es un elemento XML que sólo contiene texto. (No puede contener otros elementos o atributos).

Los tipos de datos que pueden contener son:

- xs: string
- xs: decimal
- xs: integer
- xs: boolean
- xs: date (formato dd-mm-yyyy)
- xs: time (formato hh:mm:ss)

La sintaxis sería:

```
<xs:element nombre ="xxx" type="yyy />
```

La relación entre XML y XSD sería:

XML	XSD
<nombre>Juan</nombre> <edad>18</edad> <fecha>15-03-2012</fecha>	<xs:element name="nombre" type="xs:string"/> <xs:element name="edad" type="xs:integer"/> <xs:element name="fecha" type="xs:date"/>

Valores por defecto y valores fijos para elementos simples

Podemos indicar que un elemento toma un valor por defecto (default), o un valor fijo (fixed) de la siguiente forma:

```
<xs:element name="color" type="xs:string" default="rojo"/>
<xs:element name="color" type="xs:string" fixed="rojo"/>
```

Restricciones o facetas

Podemos restringir valores para los elementos o atributos XML. Las restricciones se denominan "FACETS". Se indican mediante etiquetas restriction, dentro de la cual se establecen las posibles restricciones.

El atributo base sirve para indicar en qué tipo nos basamos al definir la restricción Ejemplo: el valor de la temperatura no puede ser inferior a -20 o superior a 40:

Restricciones sobre los valores

```
<xs:element name="temperatura">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="-20"/>
   <xs:maxInclusive value="40"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Podemos limitar el contenido de un elemento XML a un conjunto de valores aceptables, usando la restricción enumeration. El siguiente ejemplo define un elemento llamado "semana" con una restricción. Los únicos valores aceptables son: Lunes, Miércoles y Viernes:

Restricciones a un conjunto de valores

```
<xs:element name="semana">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:enumeration value="Lunes"/>
   <xs:enumeration value="Miércoles"/>
   <xs:enumeration value="Viernes"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

El ejemplo anterior también se puede escribir así…

Restricciones a un conjunto de valores

```
<xs:element name="semana" type="diasSemana"/>
.....
<xs:simpleType name="diasSemana">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Lunes"/>
  <xs:enumeration value="Miércoles"/>
  <xs:enumeration value="Viernes"/>
 </xs:restriction>
</xs:simpleType>
```

Se trata de un tipo personalizado "diasSemana". La ventaja es que el tipo diasSemana podemos utilizarlo en otras partes del esquema.

Las posibles restricciones que se pueden establecer son:

- Tamaños de texto. Indica tamaños máximos y mínimos que debe de tener el texto. Ejemplos:
  - minLength. Indica el mínimo número de caracteres. Se especifica mediante el atributo value, en el que se indica un número con el tamaño mínimo que deseamos.
  - maxLength. Indica un tamaño máximo de caracteres o de dígitos numéricos. Usa el mismo atributo value.
  - length. Indica un tamaño fijo de caracteres para el tipo. Es decir, si indicados length con value="9" el texto deberá tener exactamente nueve caracteres.
- Máximos y mínimos numéricos. Restringe valores numéricos asignando topes a los mismos. Sirve para números y para valores de tiempo o duración. Se hace con:
  - minExclusive. Establece un valor mínimo. El valor debe ser mayor que el establecido por la etiqueta a través del atributo value.
  - maxExclusive. Establece un valor máximo. El valor debe ser menor que el establecido por la etiqueta a través del atributo value.
  - minInclusive. Establece un valor mínimo. El valor debe ser mayor o igual que el establecido por la etiqueta a través del atributo value.
  - maxInclusive. Establece un valor máximo. El valor debe ser menor o igual que el establecido por la etiqueta a través del atributo value. (Podemos ver como se usa, en el ejemplo anterior de la temperatura).

```
<xs:simpleType name=“nombres">
        <xs:restriction base="xs:string">
                <xs:maxLength value="15" />
                <xs:minLength value="4" />
        </xs:restriction>
</xs:simpleType>
```

- Espacios en blanco. Sirve para especificar como se van a tratar los caracteres en blanco en los textos. La etiqueta que lo controla es whiteSpace y tiene tres posibles valores para el atributo value:
    - preserve. No modificar espacios en blanco, ni tabuladores ni saltos de línea. Es decir, se les tendrá en cuenta.
    - replace. Cada doble espacio o tabulador o salto de línea se cambia por un espacio
    - collapse. Como el anterior, pero además elimina los espacios a izquierda y derecha. Es muy útil para usar en combinación con las propiedades de tamaño de texto vistas anteriormente. (Es el más utilizado)
- Enumeraciones. Limitan el contenido de un elemento XML a una lista de valores. Se realizan con una sucesión de etiquetas enumeration.
- Plantillas (pattern). Permite establecer expresiones regulares; es decir, un texto con símbolos especiales que permiten establecer expresiones que ha de cumplir el contenido. Las plantillas se manejan con etiquetas pattern a las que, en el atributo value, se indica un texto con símbolos especiales que especifica la expresión a cumplir. (La veremos más adelante).
- Dígitos máximos. Indica las posibles cifras que puede tener el número.
    - totalDigits. Número máximo de dígitos del número, incluyendo los decimales. El atributo value indica el número máximo deseado
    - fractionDigits. Máximo número de decimales que puede tener el número

Las restricciones se pueden combinar:

```
<xs:simpleType name="diasSemanaTipo">
   <xs:restriction base="xs:string">
       <xs:whiteSpace value="collapse" />
       <xs:enumeration value="Lunes" />
       <xs:enumeration value="Martes" />
       <xs:enumeration value="Miércoles" />
       <xs:enumeration value="Jueves" />
       <xs:enumeration value="Viernes" />
       <xs:enumeration value="Sábado" />
       <xs:enumeration value="Domingo" />
   </xs:restriction>
</xs:simpleType>
```

A modo de resumen tendríamos:

Restricción	Descripción
enumeration	Define una lista de valores permitidos.
fractionDigits	Especifica el número máximo de decimales permitidos. Debe ser mayor o igual que cero.
length	Especifica el número exacto de caracteres o ítems permitidos. Debe ser mayor o igual que cero.
maxExclusive	Especifica el límite superior para un valor numérico (el valor debe ser inferior al número especificado).
maxInclusive	Especifica el límite superior para un valor numérico (el valor debe ser inferior o igual al número especificado).
maxLength	Especifica el número máximo permitido de caracteres o de ítems. Debe ser mayor o igual que cero.
minExclusive	Especifica el límite inferior para un valor numérico (el valor debe ser superior al número especificado).
minInclusive	Especifica el limite inferior para un valor numérico (el valor debe ser superior o igual al número especificado).
minLength	Especifica el numero mínimo permitido de caracteres o de ítems. Debe ser mayor o igual que cero.
pattern	Define la secuencia exacta de caracteres permitidos.
totalDigits	Especifica el número exacto de dígitos permitidos. Debe ser mayor que cero.
whiteSpace	Especifica como se manejan los espacios en blanco (LF, CR, tabulaciones, espacios).

Restricciones usando patrones

Permite establecer expresiones regulares; es decir, un texto con símbolos especiales que permiten establecer expresiones que ha de cumplir el contenido.

El siguiente ejemplo define un elemento denominado "carta" con una restricción. El único valor aceptable es UNA de las letras minúsculas de la A a la Z:

Restricciones usando patrones

```
<xs:element name="carta">
 <xs:simpleType>
  <xs:restriction base="xs:string">
   <xs:pattern value="[a-z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Los símbolos que se pueden utilizar son:

Símbolo	Significado
texto tal cual	Hace que sólo se pueda escribir ese texto. Por ejemplo si se indica "*Hombre*", la restricción será escribir como valor posible exactamente el texto *Hombre*.
[xyz]	Permite elegir entre los caracteres *x*, *y* o *z*
[^xyz]	Prohibe usar cualquiera de los caracteres entre corchetes
[a-z]	Vale cualquier carácter de la *a* a la *z*.
^	Inicio de línea
$	Final de línea
+	Repite acepta el carácter precedente una o más veces
*	Acepta el carácter precedente 0 o más veces
?	Acepta el carácter precedente 0 o una vez.
{n}	Acepta exactamente *n* repeticiones del carácter precedente.
{n,}	Acepta al menos *n* repeticiones del carácter precedente.
{n,o}	Acepta entre *n* y *o* repeticiones del carácter precedente.
\s	Permite indicar los caracteres especiales. Por ejemplo \^ representa el carácter circunflejo ^ para que sea tomado como texto y no como código especial.

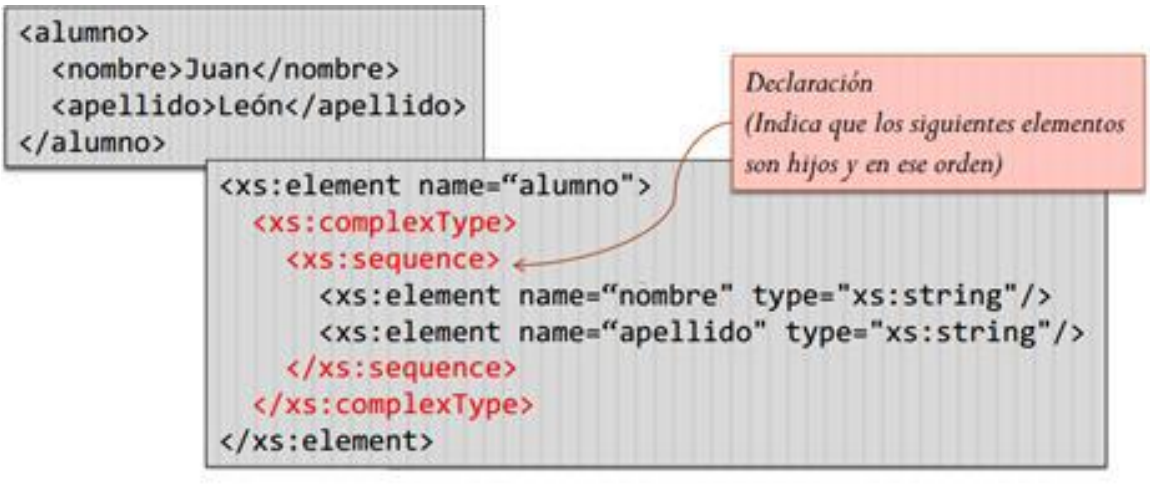
Por ejemplo, la validación para un dato tipo DNI (8 cifras y un número), sería:

```
<xs:simpleType name="dniTipo">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse" />
    <xs:pattern value="[0-9]{8}[A-Z]" />
  </xs:restriction>
</xs:simpleType>
```

Elementos complejos

Son elementos XML que contienen otros elementos y / o atributos. Se definen con la etiqueta complexType. Hay cuatro tipos de elementos complejos:

- Elementos con contenido vacío (con y sin atributos)
- Elementos con contenido y atributos
- Elementos que contienen hijos (con y sin atributos)
- Elementos que contienen elementos hijos y contenido (mixto) (con y sin atributos)



Atributos XSD

Los atributos se definen parecidos a los elementos. Su sintaxis de definición es:

```
<xs:attribute
   name="nombre del elemento"
   type="tipo global de datos"
   ref="declaración del elemento global"
   form="cualificación" <!--qualified o unqualified →
   id="identificador"
   default="valor por defecto"
   fixed="valor fijo"
   use="uso" <!-- prohibited, optional o required →
>
```

Donde…

- use="required" El atributo debe estar presente.
- use="optional" El atributo puede estar presente o no. Es el valor por defecto, de modo que no es necesario emplearlo.
- use="prohibited" Está prohibido que el atributo aparezca en el XML.

Los elemento con atributos son elementos complejos, se utiliza la palabra xs:attribute

Los tipos pueden ser: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time

El uso más normal es:

```
<xs:attribute name ="xxx" type="yyy"/>
```

XML	XSD
`<curso letra="A">1</curso>`	`<xs:element name="curso" type="xs:integer"/>` `<xs:attribute name="letra" type="xs:string"/>`

Valores por defecto, fijos u opcionales

```
<xs:attribute name="letra" type="xs:string" default="A"/>
<xs:attribute name="letra" type="xs:string" fixed="A"/>
<xs:attribute name="letra" type="xs:string" use="required"/>
```

Elementos complejos vacíos

```
<xs:element name="producto" >
    <xs:complexType/>
</xs:element>
```

Elementos complejos vacíos y con atributos

XML

```
<producto id="1345" />
```

XSD

```
<xs:element name="producto">
  <xs:complexType>
    <xs:attribute name="id" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Elementos complejos con contenido y atributos

Se indica que el elemento posee contenido simple en la etiqueta complexType a través de la etiqueta (simpleContent), en la cual debemos indicar el tipo de datos para el contenido del elemento.

Hay dos formas de indicar contenido simple: por extensión (mediante etiqueta extension) y por restricción (etiqueta restriction).

Los atributos se deben indicar en el apartado extension que es el encargado de indicar los tipos.

XML

```
<nombre repetidor="No" >juan</nombre>
```

XSD

```
<xs:element name="nombre" >
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="repetidor" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Tipo complejo con sólo hijos

Los contenidos compuestos se refieren a los elementos que contienen otros elementos (pero nunca texto libre).

Hay tres posibles tipos de elementos a contener: secuencias, elecciones y contenidos libres (all).

Además, se pueden incorporar atributos.

Ejemplo de secuencia:

XML

```
<alumno>
  <nombre>Juan</nombre>
  <apellido>León</apellido>
</alumno>
```

XSD

```
<xs:element name=“alumno">
  <xs:complexType>
    <xs:sequence>
      <xs:element name=“alumno"
type="xs:string"/>
      <xs:element name=“apellido"
type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Secuencias

Dentro de un elemento es habitual indicar su contenido como una secuencia de elementos.

Esto se permite con la etiqueta sequence, dentro de la cual se añaden etiquetas element para indicar los elementos que entran en la secuencia. Ejemplo:

```
<xs:simpleType name="emailT">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z]{3,}@.{3,}" />
  </xs:restriction>
</xs:simpleType>
<xs:element name="email">
  <xs:complexType >
    <xs:sequence>
      <xs:element name="remite" type="emailT" />
      <xs:element name="para" type="emailT" minOccurs="1"
                  maxOccurs="unbounded" />
      <xs:element name="CC" type="emailT" minOccurs="0"
                  maxOccurs="unbounded" />
      <xs:element name="CCO" type="emailT" minOccurs="0"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

El elemento email está compuesto de cuatro elementos. El remite (que tiene obligatoriamente que aparecer una vez), el para aparecerá al menos una vez y puede aparecer tantas veces como se desee y los apartados opcionales CC y CCO que pueden aparecer repetidos

La etiqueta sequence posee los atributos minOccurs y maxOccurs para indicar que el bloque de la secuencia se puede repetir

Elecciones

Sirven para permitir elegir uno de entre varios elementos.

Su funcionamiento es el mismo que en las secuencias, pero en este caso se utiliza una etiqueta llamada choice.

Ejemplo:

```
<xs:element name="identificación">
<xs:complexType>
  <xs:choice>
    <xs:element name="firma" type="xs:NCName"/>
    <xs:element name="código" type="xs:NCName" />
  </xs:choice>
</xs:complexType>
```

El elemento identificación, consta de dos posibles elementos firma y código de los que sólo se podrá incluir uno.
La etiqueta choice también posee los atributos minOccurs y maxOccurs.

Contenido Libre (all)

Se trata de una posibilidad similar a choice y sequence que se utiliza de la misma forma y que tiene como diferencia principal que los elementos que contiene pueden aparecer cero o una vez y además en el orden que quieran.

```
<xs:element name="identificación">
<xs:complexType>
  <xs:all>
    <xs:element name="firma" type="xs:NCName"/>
    <xs:element name="código" type="xs:NCName" />
  </xs:all>
</xs:complexType>
```

El elemento identificación, consta de dos elementos firma y código. En este caso la firma y el código pueden aparecer o no, aparecer los dos e incluso el orden será indiferente. Es una etiqueta muy potente que ahorra mucho trabajo.
Esta etiqueta tiene los atributos minOccurs y maxOccurs, pero sólo se puede indicar como valores cero o uno.

A veces los contenidos de un documento XML son extremadamente complejos y por eso se permite en los esquemas colocar etiquetas choice dentro de etiquetas sequence y viceversa.

Y lo mismo ocurre con las etiquetas all. Estas posibilidades permiten crear cualquier tipo de esquema por complejo que resulte.

Ejemplo:

```
<xs:element name="correo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="remite" type="xs:string"/>
        <xs:element name="para" type="xs:string"
              minOccurs="1" maxOccurs="unbounded"/>
        <xs:choice>
          <xs:element name="cc" type="xs:string"
              minOccurs="1" maxOccurs="unbounded" />
          <xs:element name="cco" type="xs:string"
              minOccurs="1" maxOccurs="unbounded" />
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

El elemento correo consta de tres elementos: remite, para y un tercero que puede ser cc o cco

E incluso la estructura puede complicarse aún más si añadimos atributos.

En los apartados complexType, los atributos del elemento se definen al final del apartado complexType (justo antes de cerrarle).

Ejemplo:

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="apellidos" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="sexo">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Hombre" />
          <xs:enumeration value="Mujer" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="fechaNacimiento"
                  type="xs:date" use="required" />
  </xs:complexType>
</xs:element>
```

El elemento persona contienen dos elementos en secuencia (nombre y apellidos) y dos atributos: uno opcional (sexo) que sólo pueden tomar los valores Hombre o Mujer y uno obligatorio para la fecha de nacimiento

Tipos complejos con contenido mixto

Es el caso más complejo. Se trata de elementos que contienen otros elementos y además texto (e incluso atributos).

Para permitir esta posibilidad hay que marcar el atributo mixed de la etiqueta complexType a true.

Ejemplo:

Lo malo es que no se puede controlar el tipo de datos del texto interior. Los elementos se controlan completamente, pero el texto no.

XML

```
<carta>
  Estimado Sr.<nombre>Juan León</nombre>.
  Su pedido <pedido>1032</pedido>
  será enviado el <fechaenvio>25-03-2012</fechaenvio>
</carta>
```

XSD

```
<xs:element name="carta">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="pedido" type="xs:positiveInteger"/>
      <xs:element name="fechaenvio" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Pasar de DTD a XML Schema (XSD)

Como ya habremos observado, tanto DTD como XSD expresan lo mismo, pero de diferente forma: cómo se deben comportar tanto la estructura como los datos de los ficheros XML. Y por dicha relación, se puede pasar de uno a otro.

Por ejemplo, el siguiente DTD:

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>To write Tove!</body>
</note>
```

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

Se expresaría, igualmente en XSD de la siguiente manera:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.mysite.com"
xmlns="http://www.mysite.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

4.4 JSON

JSON (JavaScript Object Notation o Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es fácil para los humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Todos ganan. Está basado en un subconjunto del Lenguaje de Programación JavaScript, más concretamente la *Standard ECMA-262 3rd Edition* de diciembre de 1999. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un array asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como tablas o arrays, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Recordemos que un fichero JSON (al igual que uno XML) no es más que información, datos, que se necesitan transmitir desde un origen a un destino. Estamos hablando, por ejemplo, de la información que se intercambia entre una fuente de datos en MySQL con su posible visualización en un navegador en una página web: una búsqueda en un portal de compras.

El proceso podría ser tal que así:

1. El usuario/cliente accede a una página web en su navegador y hace una consulta.
2. El navegador solicita la información al servidor donde está alojada la web
3. El programa que tramita la solicitud hace una consulta a la base de datos obteniendo un resultado
4. Dicho resultado (registros) se envía en formato JSON o XML al navegador

5. El navegador interpreta la información tal como le llega, pero la transforma en un formato más amigable para el usuario

Veamos una serie de ejemplos para afianzar conceptos.

Comencemos con algo básico, la representación de los datos de una persona, tanto en formato XML como en formato JSON:

XML	JSON

<persona>
<nombre>Elena</nombre>
<apellidos>Sánchez Ortiz</apellidos>
 	
<fechaNacimiento>25/06/1998</fechaNacimiento>
</persona>
	{
	"nombre":"Elena",
	"apellidos":"Sánchez Ortiz",
	"fechaNacimiento":"25/06/1998"
	}

Como se puede observar, se ha representado la misma información, pero en el caso de JSON es mucho más sencillo de implementar, y, por tanto, más fácil de interpretar por los programas, webs, dispositivos que reciban dicha información para su posterior tratamiento.

A favor de XML diremos que JSON carece de algo similar tanto de DTD como de XSD, por lo que la información que nos llega en XML será siempre de mayor fiabilidad que con JSON.

El siguiente ejemplo, basado en el anterior, muestra cómo sería la estructura para representar más de una persona:

XML	JSON

<personas>
<persona>
<nombre>Elena</nombre>
<apellidos>Sánchez Ortiz</apellidos>
 	
<fechaNacimiento>25/06/1998</fechaNacimiento>
</persona>
<persona>
<nombre>Ernesto</nombre>
<apellidos>Ramírez Vega</apellidos>
 	
<fechaNacimiento>05/12/2000</fechaNacimiento>
</persona>
</personas>
	{
	[
	{
	"nombre":"Elena",
	"apellidos":"Sánchez Ortiz",
	"fechaNacimiento":"25/06/1998"
	},
	{
	"nombre":"Ernesto",
	"apellidos":"Ramírez Vega",
	"fechaNacimiento":"05/12/2000"
	}
	]
	}

En el caso del XML simplemente tendremos una etiqueta <personas> que engloba a tantas etiquetas <persona> como sea necesario.

En el caso del JSON, tenemos un array, delimitado por corchetas [ ], de varios objetos JSON delimitados por llaves { }.

Sigue siendo mucho más sencillo de representar en formato JSON que en formato XML.

Además, si contamos los caracteres de la representación de XML tendremos 346, incluyendo los espacios en blanco frente a los 240 de JSON. Si extrapolamos este resultado a miles de registros viajando por la red, el ahorro de envío de datos aumentando la velocidad, se hace evidente a favor de JSON.

Indicar finalmente, que independientemente de la representación en XML como en JSON, ambos se pueden asimilar perfectamente a registros de Bases de datos Relacionales.

4.5 CSS para XML

Acabamos el tema indicando una posibilidad que se usa más bien poco y es la de dar formato CSS a ficheros XML. Como hemos comentado, la utilidad del formato XML y del formato JSON es la de representar datos, información. Y mientras más simple, mejor.

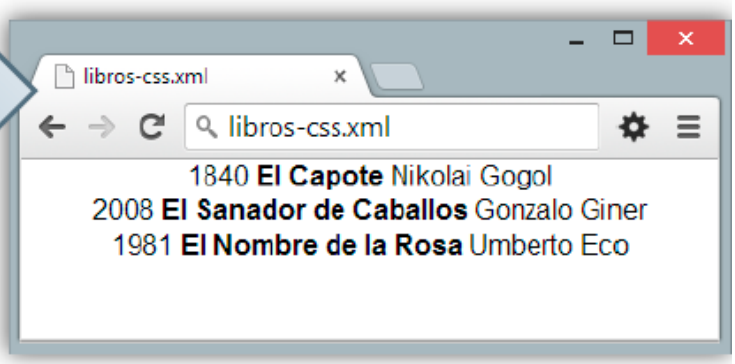
No obstante, como podemos ver en un navegador un fichero XML tal cual, podemos darle algo de estilo para que no sea tan básico, aunque, insistimos, la finalidad de los datos contenidos en un fichero XML no es que sean vistos tal cual, sino como fuente de datos para programas que los traten y los manipulen para mostrar resultados, y ahora sí, con una buena presentación. Pero esto ya es responsabilidad del programa o la web, no del fichero XML tal cual.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="libros.css" type="text/css"?>

<!DOCTYPE Libros SYSTEM "libros.dtd">

<Libros>
   <Libro>
         <Fecha>1840</Fecha>
         <Titulo>El Capote</Titulo>
         <Autor>Nikolai Gogol</Autor>
   </Libro>
...
```

```
Libros {
     font-family: Arial, sans-serif;
     text-align: center;
}

Libro {
     display: block;
}

Libro Titulo {
     font-weight: bold;
}
```

libros.css



RENDERIZADO EN NAVEGADOR WEB

Hojas de estilos CSS aplicadas a documentos XML

Otro ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="libros2.css" type="text/css"?>

<!DOCTYPE Libros SYSTEM "libros.dtd">

<Libros>
   <Libro>
          <Fecha>1840</Fecha>
          <Titulo>El Capote</Titulo>
          <Autor>Nikolai Gogol</Autor>
   </Libro>
...
```

```
Libros {
     font-family: Arial, sans-serif;
}

Libro {
     display: table-row;
}

Fecha,Titulo,Autor {
     display: table-cell;
     padding: 10px;
     border: 1px solid green;
}
```

libros-css-2.xml

libros-css-2.xml

1840	El Capote	Nikolai Gogol
2008	El Sanador de Caballos	Gonzalo Giner
1981	El Nombre de la Rosa	Umberto Eco

libros2.css

RENDERIZADO EN NAVEGADOR WEB

Hojas de estilos CSS aplicadas a documentos XML (II)

12/07/2022