UD 04 - DEFINICIÓN DE ESQUEMAS Y VOCABULARIOS EN XML

> "¿Algún modo de garantizar la estructura de datos de los diferentes tipos de documentos XML?"

XML (*eXtensible Markup Language*, Lenguaje de Marcado eXtensible) es un lenguaje desarrollado por **W3C** (*World Wide Web Consortium*) que está basado en **SGML** (*Standard Generalized Markup Language*, Lenguaje de Marcado Generalizado Estándar).

- lenguaje de etiquetas
- utilizado para el **almacenamiento e intercambio de datos estructurados** entre diferentes equipos y de manera segura, fiable y sencilla
- **metalenguaje**, es decir, puede ser empleado para definir otros lenguajes, llamados dialectos XML (como XHTML, GML, MathML, SVG, RSS)
- compatible con protocolos como HTTP y URL, legible para los humanos
- extensible, adaptable, aplicable a gran variedad de situaciones
- orientado a objetos
- fácil de crear
- difusión asegurada porque cualquier procesador XML puede leer documento XML

Los datos carácter son los que forman el verdadero XML. El marcado puede ser tan rico como se quiera (detectar futuras necesidades y conseguir documentos fácilmente actualizables).
Documentos XML pueden tener comentarios pero no son interpretados por XML. Estos se deben incluir entre `<!-- y --!>` y pueden estar en cualquier lugar salvo antes del prólogo o dentro de una etiqueta.

Pasos para definir un documento XML

Etapas para definir:

- **Especificación de requisitos**: Recoger la estructura que tiene la información
- **Diseño de etiquetas**: Se crean etiquetas para representarla
- **Marcado de los documentos**: Se utilizan las etiquetas para crear documentos con contenido real. (Todo lo situado entre "<" y ">" o "&" y ";")

1. Las partes de un documento XML

Un documento XML básico está formado por PRÓLOGO Y EJEMPLAR :

- **Prólogo**: Informa al intérprete que va a procesar el documento de los datos que necesita para realizar su trabajo.
  Es opcional pero si se incluye debe ir obligatoriamente al inicio del documento.
  Tiene dos partes:
  - **La definición XML**: Va entre `<? ?>` No es obligatoria (en contraste con "de no ser así se genera error que impide que el documento sea procesado") pero, de incluirse, debe estar en la primera línea del documento y el carácter ' <' debe ser el primero de dicha línea (sin espacios en blanco).
    Posee los atributos `version` (obligatorio, indica versión XML utilizada aunque solo hayu 1.0. y 1.1 y la que se usa es la 1.0), `encoding` (opcional debe ir tras version, indica codificación de caracteres empleada, valor por defecto "UTF-8". Podría ser ISO-8859-1 (Caracteres europeos occidentales)) y `standalone` (es opcional, debe ir en último lugar, cuando está a "yes" indica que el documento es independiente de otros (de un DTD externo por ejemplo). Su valor por defecto es "no", o sea, que puede estar asociado a otros.).

    (En los apuntes no viene esto): También se pueden indicar en el prólogo líneas de instrucciones de procesamiento (opcionales) para indicar cierta información al intérprete. Estas van también entre `<? ?>` . Por ej. se puede asociar un archivo CSS:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="estilo-animales.css"?>
<animales>
   <animal>
      <nombre>perro</nombre>
      <patas>4</patas>
   </animal>
   <animal>
      <nombre>pato</nombre>
      <patas>2</patas>
   </animal>
</animales>
```

Y el CSS, hará que el XML aparezca formateado

```css
nombre{color:blue;font-size:40px}
patas{color:red;font-size:22px}
```

  - **La declaración del tipo de documento** (La declaración del tipo propiamente dicha es el nombre del ejemplar precedido de `<!DOCTYPE` y separado por un espacio Ej.: `<!DOCTYPE animales>`). Pero esta declaración permite al autor del XML definir restricciones y características del documento (asociar el documento a una definición de tipo DTD).
- **Ejemplar**: Parte más importante del documento. Contiene los datos reales del documento a procesar. El ejemplar es el elemento raíz de un documento XML. Es decir, formado por un elemento, el principal o raíz, que **debe ser único**. Ya que está compuesto de elementos estructurados en árbol. El ejemplar es el elemento raíz (raíz, única) y las hojas son los elementos terminales (los que no contiene más elementos). Los elementos **pueden contener atributos**. Se puede usar el marcador especial `<![CDATA[texto que se quiera]]>` para introducir en un documento texto que pueda contener caracteres como & o < o >. Se tomará como texto.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<!DOCTYPE ejemplo>
<ejemplo>
</ejemplo>
```

> Si preguntan por "componentes de un documento XML": Prólogo OK, Ejemplar OK, Definición de codificación OK, Cabecera NO

Es recomendable mantener todo el criterio en el documento. Etiquetas en minúsculas. Para visualización óptima se anidarán los elementos identando o tabulando en el código.

2. Sintaxis de documentos XML: Elementos vs Atributos. Namespaces: espacios de nombres

La estructura del documento XML viene dada por etiquetas de marcado.

Elemento: Grupo formado por etiqueta de apertura, etiqueta de cierre y contenido entre ambas. Siguen la estructura de árbol (pueden ser anidadas pero no entrelazadas).
El nombre de la etiqueta debe estar formado por letras mayúsculas o minúsculas, números, guiones y punto. No puede contener espacios y debe comentar por una letra. El nombre xml está prohibido.

```
<etiqueta>valor</etiqueta>
```

Si está vacío:

```
<etiqueta></etiqueta>
<etiqueta/>
```

Todos los datos del documento XML deben pertenecer a un elemento. Pueden estar formados por otros elementos. Nombres de etiquetas deben ser autodescriptivos.

Normas:

- Debe existir un elemento raíz y solo uno
- Todos los elementos tienen etiqueta de inicio y de cierre
- No puede cerrarse elemento con otro que aún no se haya cerrado
- Nombres de etiquetas de inicio y de cierre deben ser iguales
- No puede contener cadena "]]>" por compatibilidad con SGME. Ni usar directamente `>,` `<,&",apostrofe`
- Para usar caracteres especiales deben usarse &#D; &#H; (numeros decimal y hexadecimal...)

Atributos: Características o propiedades que pueden añadirse a los elementos de un documento. Entrecomillados. No tienen que seguir un orden significativo. Son hojas, no tienen hijos. Tampoco pueden repetirse en un elemento. Si se usa, y se deja vacío debe escribirse 'atributo=""'. No puede haber espacios entre el nombre del atributo y el signo igual, ni entre el signo igual y el valor.

> *Recomendado elemento si*: contiene subestructuras, es de tamaño considerable, su valor cambia con frecuencia, su valor va a ser mostrado.
> *Recomendado atributo si*: no tiene subestructuras, es de pequeño tamaño, su valor raramente cambia, solo puede tener unos cuantos valores fijos, no va a mostrarse al usuario y solo sirve para guiar el procesamiento XML.

No se debe usar atributo para información susceptible de ser dividida. Solo para proporcionar información adicional sobre el elemento.

Espacios de nombres: Asigna un identificador único a elementos y atributos. Son definidos con xmlns. Se declaran indicando la URI (localización del conjunto del vocabulario del espacio de nombres al que hace referencia) `xmlns:"URI_namespace"`. Parecerá una URL pero no se usa como tal, solo se usa para que sea distinto a cualquier otro posible espacio de nombres.
Pueden usar, además de la URI, un prefijo que indique cuál es el vocabulario al que está asociado esa definición. `xmnls:prefijo="URI_namespace"`

- Permiten diferenciar elementos y atributos que comparten nombre (pero son de distintos vocabularios, con diferentes significados) evitando conflictos.
- Permiten agrupar los elementos y atributos relacionados de una aplicación XML para que el software lo reconozca con facilidad.

Ejemplos:

```
<!-- Espacio de nombres -->
<nombre
xmlns="https://educacionadistancia.juntadeandalucia.es/EspacioNombres">Ejemplo</nombre>

<!-- Espacio de nombres con prefijo -->
<EN:nombre
xmlns:EN="https://educacionadistancia.juntadeandalucia.es/EspacioNombres">Ejemplo</EN:nombre
>

<!-- añadir mas de JSF, Icefaces por ejemplo -->
```

(Ojo: Facilita que el software localice y permite tener varios elementos homónimos pero no define los atributos que forman parte del documento, ni determina los elementos que forman parte)

Entidades: Forma de almacenar texto para usarlo en el documento. Se representan entre el símbolo `&` y el `;`. Pueden ser definidas por el desarr4ollador pero también hay 5 entidades predefinidas:

Entidad	Significado
`>`	Símbolo >
`<`	Símbolo <
`&`	Símbolo &
`&apos;`	Símbolo '
`&quot;`	Símbolo

Entidad	Significado
`&numero;`	Código
`&#numero;`	Código hexadecimal

Ejemplo: `<fórmula> a > 3</fórmula`

Comentarios: Está prohibido usar comentarios antes del prólogo y dentro de una etiqueta.

Espacios en blanco: XML elimina los espacios en blanco al inicio y al final del contenido de un elemento. Si se desea que se mantengan espacios en blanco sin procesar usar el atributo `xml:space = "preserve"`.

```
<text xml:space="preserve">
   Este es un ejemplo de texto
con espacios en blanco
preservados. </text>
```

El documento bien formado

Documento bien formado: Es sintácticamente correcto, es decir, cumple con las reglas de creación de documentos XML ya definidas

Documentos válidos: Están bien formados y cumplen definición de estructura (DTD, XML Schema...)

Para estar bien formado deben verificarse las reglas sintácticas que define la recomendación del W3C para el estándar XML:

- *El documento **ha de tener definido una declaración XML en el prólogo:***

  éstos serán los valores por defecto si no se incluye el prólogo, pero recordamos que es muy recomendable incluirlo, ya que algunos navegadores nos devolverán errores si no lo hacemos.
- ***Existe un único elemento raíz para cada documento**: es un solo elemento en el que todos los demás elementos y contenidos se encuentran anidados.*
- *Los elementos se organizan entre sí en **estructura jerárquica y no se permite el solapamiento de los elementos.***
- *Hay que **cumplir las reglas sintácticas del lenguaje XML para definir los distintos elementos y atributos** del documento. Los resumimos a continuación:*
  - *El nombre de los elementos pueden tener como primer carácter [A-Z], [a-z] y "_", y para el resto de caracteres, además de los citados: [0-9], "-" y "."*
  - ***Las etiquetas de apertura y de cierre tienen que ser idénticas**, XML es sensible a las mayúsculas y minúsculas, por lo que, por ejemplo, ... sería incorrecto. Recordemos que se permiten elementos vacíos.*
  - *Los **valores de los atributos se escribirán siempre entre comillas dobles o simples**. Si queremos incluir un atributo que incluya alguno de esos caracteres, se hará de la siguiente forma: ": (") y ' (). Existen atributos reservados que no podremos usar, salvo para la finalidad en la que están reservados: "xml:", "xml:lang", "xml:space: default | preserve" y "xml:id"*
  - ***Los comentarios en XML** se escribirán así:*

3. Declaración del tipo de documento, DOCTYPE

Es la primera línea de código requerida en el documento. Está formada por el elemento `<!DOCTYPE>`:

- **Declaración del tipo de documento propiamente dicha**: El tipo (nombre del ejemplar) precedido de `<!DOCTYPE` y separado por un espacio Ej.: `<!DOCTYPE animales>`)
- **Las definiciones de tipo de documento**: Permite asociar al documento una **definición de tipo DTD** con las cualidades de este (tipos de los elementos y atributos, notaciones que pueden usarse, restricciones, valores por defecto) . Esto se hace mediante **declaraciones de marcado** que pueden ser:
  - **Declaraciones internas** (DTD Interno): Son exclusivas del documento y se procesan primero (sobrescriben a las externas). Se localizan dentro de unos corchetes que siguen a la declaración del tipo de documento.
  - **Declaraciones externas** (DTD Externo): Pueden (suelen) ser compartidas por varios XML. Se indican en un documento con extensión DTD (que puede estar en el mismo directorio que el XML.
    Importante considerar:

- La declaración de documento autónomo en el prólogo (`standalone`) debe haberse puesto negativa. (nunca positiva)
- El procesado del documento será más lento porque primero el procesador obtendrá las entidades y luego el documento.
- No se usan corchetes. Para localizar las declaraciones del tipo de documento externo se podría utilizar:
Una URI: `<!DOCTYPE nombre_ejemplar SYSTEM "url"` (SYSTEM indica que es definida externamente mediante un indicador del sistema).
Una URI y un identificador (id público), que el procesador XML podría usar para generar un URI alternativo (posiblemente basado en alguna tabla). Se usa suponiendo que hay un repositorio en el que se almacenan los DTDs y mediante un ID se recupera.
`<!DOCTYPE nombre_ejemplar PUBLIC "id_publico" "URI"`
Seria mas correcto esto último: `<!DOCTYPE cine PUBLIC "filmoteca" "http://cine.com/filmoteca.dtd">`, siendo filmoteca el nombre de la DTD

Ejemplos de declaraciones internas y externas (recuerda, pueden coexistir ambas)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE pelicula [
  <!ELEMENT pelicula (titulo)>
  <!ELEMENT titulo (#PCDATA)>
]>
<pelicula>
  <titulo>Titanic</titulo>
</pelicula>
```

(Si no pones standalone "yes", tomará por defecto el "no" y en principio no pasa nada malo pero algunos procesadores podrían asumir que necesita esa dependencia y ralentizarse o presentar alguna vulnerabilidad de seguridad)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE pelicula SYSTEM "cine.dtd">
<pelicula>
  <titulo>Titanic</titulo>
</pelicula>
```

(No haría falta poner standalone "no", por defecto lo toma)

4. Definiciones de tipo de documento, DTD

Es la descripción de la estructura y sintaxis de un documento XML o SGML (Lenguaje de marcado generalizado estándar)

Permite diseño común y consistencia entre los documentos que usan la misma DTD.

Un DTD permite incluir en el fichero XML ficheros binarios, además no se construyen en XML.

Un XML válido es...: Aquel que está bien formado y sigue las especificaciones que dicta la DTD.

Las restricciones de los DTD permiten...

- Especificar estructura
- Restricción de integridad referencial (normas que usan las bases de datos relacionales para garantizar que los registros de tablas relacionadas son validas) mínima (ID, IDREF)
- Mecanismos de abstracción comparables a las macros (instrucción almacenada en la aplicación que la usa y ejecutada de forma secuencial) -> entidades
- Incluir documentos externos

Inconvenientes...

- Su sintaxis no es XML
- No soportan espacios de nombres
- No definen distintos tipos para los datos (solo hay tipo en elementos terminales)

- No permite secuencias no ordenadas
- No permite claves con varios atributos o elementos (no permiten formación de identificadores compuestos)
- Definido el DTD, no pueden añadirse nuevos vocabularios

Como se ha visto arriba cuando están definidas dentro del documento XML se ubican entre corchetes después del nombre del ejemplar en el elemento `<!DOCTYPE>`. Y cuando están definidas fuera del XML se utiliza la URI con extensión ".dtd" usando una URI o una URI y un IDENTIFICADOR.

Una DTD externa es ventajosa porque si es compartida por varios documentos evita tener que modificarlos todos; porque puede ubicarse en un servidor web permitiendo la validación del XML por cualquiera.

4.1. Declaraciones de elementos

Deben incluirse tantas declaraciones de elementos como tipos de etiquetas haya en el vocabulario.

Para cada elemento se utiliza la sintaxis `<!ELEMENT nombre (contenido)>` indicando:

- La palabra clave `ELEMENT`
- El nombre del elemento (etiqueta)
- Los contenidos que se permiten dentro del elemento (otros elementos, texto, combinación de ambos.

Elemento sin contenido: Se indica con `EMPTY` `<!ELEMENT br EMPTY>`

Contenido libre: Se indica con `ANY`. Estos elementos deben de haber sido declarados en el DTD.
`<!ELEMENT empresa ANY>`

Por ejemplo:

```
<mascota>
        <nombre>Shiro</nombre> es mi mascota. <br/> Es un <tipo>gato</tipo> de
<color>blanco</blanco>
</mascota>
```

(Curioso, pero se puede poner texto fuera de etiquetas: es bien formado y en este caso también es válido)

Solo texto: Se indica con `#PCDATA` `<!ELEMENT nombre (#PCDATA)>`. Solo puede contener datos de tipo carácter (exceptuando <, &,]], >).

Solo otro elemento: Se indica, debiendo declarar el otro también
`<!ELEMENT importe (cantidad)>`
`<!ELEMENT cantidad (#PCDATA)>`

Secuencias de elementos: Se indican en el mismo orden que aparecen en la declaración separando los elementos por una coma.

La repetición o la opcionalidad de elementos en la secuencia debe indicarse.

- Operador opción. **Cero o uno (?)** - Elemento opcional. De aparecer solo lo hará una vez
- Operador cero-o-mas. **Cero o más (*)** - Elemento opcional. Si aparece puede hacerlo sin límite
- Operador un 111 o-o-mas. **Uno o más (+)**- Elemento obligatorio. Puede aparecer sin límite.

No es posible especificar cardinalidad del tipo "hasta dos" en el DTD
Las opciones son uno, ninguno o más, uno o más

Operador de elección. **Elementos alternativos**: Contener un elemento y solo uno de entre todos los posibles, separándolos por una barra vertical |

Combinaciones: Puede mezclarse lo anterior para representar cualquier tipo de contenido
No es solo "opciones entre elementos". Puede ser algo más complejo como:

Uno o varios padres o madres en una familia (o ninguno)
`<!ELEMENT familia (((padre, madre?) | (madre?, padre?) | (padre, padre) | (madre, madre)), hijos)>`

Mínimo un hijo/hija (apareciendo indistintamente uno u otro):
`<!ELEMENT hijos ((hijo | hija)+)>`

nombre y email o teléfono
`<!ELEMENT nombre, (email | telefono)`

calle y número o teléfono e email
`<!ELEMENT (calle, numero) | (telefono, email)`

email o direccion y descripcion
`<!ELEMENT (email | (direccion, descripcion+))`

Contenido mixto: Puede especificarse texto y elementos. Por ejemplo:

```
<!ELEMENT parrafo ((#PCDATA | negrita | cursiva)*)>
<!ELEMENT negrita (#PCDATA)>
<!ELEMENT cursiva (#PCDATA)>
```

4.2. Declaraciones de atributos

Deben declararse después de haber declarado el elemento en el DTD y solo se puede hacer una por elemento. Con la declaración `<!ATTLIST elemento atributos>`

Se declaran con indicaciones de nombre, tipo y valor.

El tipo puede ser:

- `CDATA`: Texto

```
<!ATTLIST ejemplo color CDATA #REQUIRED>
```

- `NMTOKEN`: Solo letras dígitos y otros caracteres permitidos en nombres XML (letras ,números, caracteres : , _ - .. Ni espacios en blanco, ni caracteres especiales.
- `NMTOKENS`: Varios tokens separados por espacios.
- `ENTITY`: Representa a una entidad dentro del documento (referencia a elemento externo o valor predefinido en el documento)
- `ENTITIES`: Múltiples entidades separadas por espacios
- `ID`: No puede repetirse
- `IDREF`: Referencia a elemento que tiene ID definido en el mismo documento. Restricción de integridad referencial.
- `IDREFS`: Múltiples referencias separados por espacios
- `NOTATION`: Define el tipo de notación que puede ser usada en un atributo particular. Se utiliza para describir el formato de los datos (como MIME type u otra notación)

```
<!DOCTYPE imagen [
<!ELEMENT imagen (ruta)>
<!ELEMENT ruta (#CDATA)>
<!ATTLIST imagen formato NOTATION (JPEG | PNG | PDF) #REQUIRED >
 <!NOTATION JPEG SYSTEM "image/jpeg">
<!NOTATION PNG SYSTEM "image/png">
<!NOTATION GIF SYSTEM "image/gif">
]>
<imagen formato="JPEG">
    <ruta>/ruta/a/imagen.jpg</ruta>
</imagen>
```

- **Enumeración**: Especificar lista de posibles valores poniéndolos separados por barra vertical y entre paréntesis. Solo puede contener letra, números, subrayado y guion no pudiendo contener

nada no permitido.

```
<!ATTLIST fecha dia_semana (lunes|martes|miércoles|jueves|viernes|sábado|domingo) #REQUIRED>
```

El valor son restricciones sobre el valor del atributo:

- Valor entre comillas. Opcional. Puede aparecer en el elemento o no. Si no aparece el procesador supone que tiene el valor indicado.
- `#REQUIRED`: Obligatorio y toma el valor que se le da
- `#IMPLIED`: Opcional y si no aparece, no se le da valor por defecto
- `#FIXED`: Obligatorio y debe llevar el valor indicado en la declaración

Ejemplo:

```
<!ELEMENT body ANY>
<!ATTLIST body
background-color CDATA "#FFFFFF"
href CDATA #REQUIRED
version CDATA #IMPLIED
padre CDATA #FIXED "html"
alineacion (izquierda | centro | derecha) "derecha">
```

4.3. Declaraciones de entidades

Se pueden definir entidades en el DTD que más tarde se referencien en el documento como:
`<!ENTITY nombre contenido`

Puede haber

Entidades generales

Se proporciona el texto a sustituir

`<!ENTITY nombre "texto"`

Ejemplo:
`<!ENTITY onu "Organización de Naciones Unidas"`

```
<parrafo>La &onu; es la más chula</parrafo>
```

El contenido de la entidad debe ser bien formado. Las entidades generales no pueden referenciarse desde dentro del DTD

Entidades externas

El contenido está en archivo externo al documento en el que se declara el DTD. Se relacionan a través de la URI de este último.

`<!ENTITY nombre SYSTEM "url">`

Ejemplo
`<!ENTITY cabecera SYSTEM "cabecera.xml">`
Declara la entidad cabecera y le asocia el contenido del archivo. El documento debe estar bien formado.

Cuando va a ser usada por varias aplicaciones es:
`<!ENTITY nombre_entidad PUBLIC "identificador público formal" "camino hasta la DTD (uri)">`

Si se quiere incluir ficheros binarios (que no sean analizados), se usa la palabra reservada `NDATA` en la definición de la entidad y debe asociarse a dicha entidad una **declaración de notación**.

Entidades parámetricas

Internas: Pueden usarse dentro del DTD. Deben declararse de la forma:

`<!ENTITY % nombre "valor">`

Por ejemplo si en HTML todos los atributos deben llevar id, class y style puede hacerse:

```
<!ENTITY % common-attrs ("id ID #IMPLIED class ID #IMPLIED style CDATA #IMPLIED">
<!ATTLIST html %common-attrs;>
<!ATTLIST html %common-attrs;>
`<!ATTLIST html %common-attrs;>
```

Externas: Permiten incluir en un DTD elementos externos (se divide la definición DTD en varios documentos)
`<!ENTITY %persona SYSTEM "persona.dtd">`

4.4. Declaraciones de notación

Para indicar cual debe ser la aplicación que ha de procesar un fichero binario asociado a un XML deben usarse declaraciones de notación... `<!NOTATION nombre SYSTEM aplicacion>`

Por ejemplo una notación gif referenciando al editor de GIF
`<!NOTATION gif SYSTEM "gifEditor.exe">`

Por ejemplo para asociar una entidad externa no analizada
`<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>`

4.5. Secciones condicionales

Permiten incluir o ignorar partes de la declaración de un DTD. Para ello se usa:

- `INCLUDE`: Permite que se vea esa parte de la declaración. (`
- `IGNORE`: Permite ocultar esa sección de declaraciones dentro del DTD. (`

```
<![ %datos_basicos; [
   <!ELEMENT persona (nombre, edad)>
]]>

<![ %datos_ampliados; [
   <!ELEMENT persona (nombre, apellidos, edad, ciudad)>
]]>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE persona SYSTEM "persona.dtd" [
   <!ENTITY % datos_basicos "INCLUDE">
   <!ENTITY % datos_ampliados "IGNORE">
]>

<persona>
   <nombre>Elsa</nombre>
   <edad>23</edad>
</persona>
```

"Las sentencias condicionales permiten definir unos elementos u otros dentro del fichero XML en función de una determinada condición." --> Falso

4.6. Ejemplito sencillo de DTD

Para:

```xml
<?xml version="1.0"?>
<!DOCTYPE familias SYSTEM "familias.dtd">
<familias>
    <familia codigo="P01-12">
        <madre>
            <nombre>Maria</nombre>
            <apellido1>González</apellido1>
            <apellido2>López</apellido2>
            <dni>12345678A</dni>
            <edad>35</edad>
        </madre>
        <padre>
            <nombre>&FJ;</nombre>
            <apellido1>Pérez</apellido1>
            <apellido2>Rodríguez</apellido2>
            <dni>87654321B</dni>
            <edad>40</edad>
        </padre>
        <hijos>
            <hijo posicion="1">
                <nombre>Carlos</nombre>
                <apellido1>González</apellido1>
                <apellido2>Pérez</apellido2>
                <edad>10</edad>
            </hijo>
            <hijo posicion="2">
                <nombre>Sofía</nombre>
                <apellido1>Pérez</apellido1>
                <apellido2>González</apellido2>
                <edad>8</edad>
            </hijo>
        </hijos>
    </familia>
    <familia codigo="M02-24">
        <madre>
            <nombre>Lucía</nombre>
            <apellido1>Ruiz</apellido1>
            <apellido2>García</apellido2>
            <dni>98765432C</dni>
            <edad>38</edad>
        </madre>
        <madre>
            <nombre>Patricia</nombre>
            <apellido1>Gómez</apellido1>
            <apellido2>Fernández</apellido2>
            <edad>39</edad>
        </madre>
        <hijos>
            <hija posicion="1">
                <nombre>Marta</nombre>
                <apellido1>Ruiz</apellido1>
                <apellido2>Gómez</apellido2>
                <edad>12</edad>
            </hija>
        </hijos>
    </familia>
</familias>
```

```
<!--
1. En una familia puede haber un padre y una madre, o bien un solo padre, o una sola madre
o ninguno de ellos. También hay posibilidad de que sean dos padres o dos madres.
2. El elemento padre y el elemento madre puede aparecer en cualquier orden
3. Posteriormente a los padres aparecerán los hijos.
4. El orden de aparición de hijo o hija es indiferente.
5. Una familia tiene al menos un hijo/a
6. Tanto el elemento padre, como madre, hijo o hija contienen los elementos: nombre,
apellido (siempre dos), dni (opcional) y edad (opcional).
7. Limitar el tamaño de nombre y apellidos a una longitud de 30
8. El atributo posición toma valores del 1 al 30
9. El atributo código de la familia está formado por Letra mayúsculas y dos números, un
guión
y dos números, por ejemplo P01-12
10. La edad es un número entero positivo.
```

```
!-->
<!ENTITY FJ "Francisco José">
<!-- Entidad general-->
<!ENTITY % info_familia "(nombre, apellido1, apellido2, dni?, edad?)">
<!-- Las entidades paramétricas solo funcionan en DTD externo. Pueden hacerse al declarar
elementos o atributos -->
<!ELEMENT familias (familia*)>
<!ELEMENT familia (((padre, madre?) | (madre?, padre?) | (padre, padre) | (madre, madre)),
hijos)>
<!-- cubramos los casos
padre padre
madre madre
padre madre
madre padre
padre
madre
ninguno
-->
<!ELEMENT hijos ((hijo | hija)+)>

<!ELEMENT padre (%info_familia;)>
<!ELEMENT madre (%info_familia;)>
<!ELEMENT hijo (%info_familia;)>
<!ELEMENT hija (%info_familia;)>

<!ELEMENT nombre (#PCDATA)>
<!ELEMENT apellido1 (#PCDATA)>
<!ELEMENT apellido2 (#PCDATA)>
<!ELEMENT dni (#PCDATA)>
<!ELEMENT edad (#PCDATA)>

<!ATTLIST familia codigo ID #REQUIRED>
<!ATTLIST hijo posicion NMTOKEN #REQUIRED>
<!ATTLIST hija posicion NMTOKEN #REQUIRED>
```

Más ejemplos en:
https://www.youtube.com/playlist?list=PLM8XywipQpGA4Siojsb3yhSaZth5lFgSD
https://www.abrirllave.com/

5. XML Schema

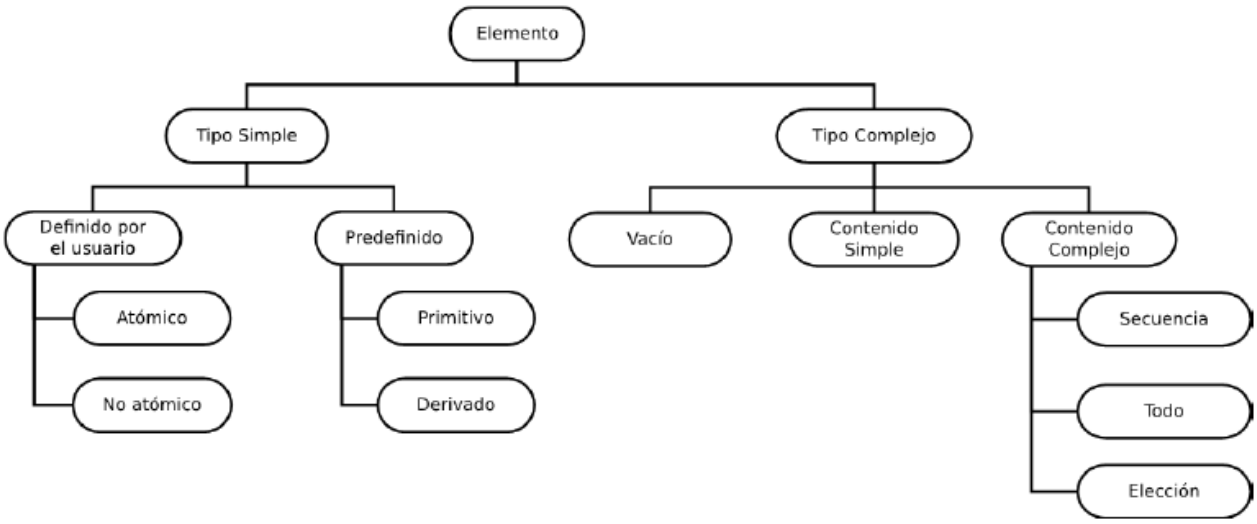
5.1. Introducción a XML Schema

- Lenguaje basado en XML
- Se usa para crear otros lenguajes basados en XML y modelos de datos
- Define los nombres de los elementos y los atributos del documento XML y especifica la estructura que deben seguir y el tipo de documento que puede haber dentro de cada elemento o atributo.
- Los documentos XML que intentan seguir un XML Schema se llaman instancias del esquema. Si siguen el esquema se dice que son válidos.

(No puede tenerse asociado a la vez un DTD y un XSD)

XML Schema es más potente que DTD porque:

- Los DTD no tienen tipos de datos predefinidios
- Los DTD no soportan tipos de datos derivados
- Los DTD tienen un control limitado de la cardinalidad
- Los DTD no soportan espacios de nombres ni formas simples de reutilizar o importar de otros esquemas



Es un lenguaje basado en XML que se usa para crear otros lenguajes basados en XML y modelos de datos.

Algunos principios

- Los elementos pueden ser de tipo simple o complejo
- Los elementos de tipo simple solo contienen texto. No tienen ni hijos, ni atributos
- Los tipos predefinidos son simples
- Se pueden derivar nuevos tipos a partir de los simples, restringiéndolos.
- Los tipos simples pueden ser atómicos (cadenas, números) o no atómicos (listas)
- Elementos de tipo complejo pueden contener otros elementos, atributos y texto
- Elementos de tipo complejo tienen, por defecto, elementos hijos
- Elementos de tipo complejo pueden ser limitados para que tengan contenido simple (solo texto), pero también podrán tener atributos (cosa que los simples no)
- Elementos de tipo complejo pueden limitarse para no tener contenido pero sí atributos
- Tipos complejos pueden contener contenido mixto, mezcla entre texto y elementos hijos.

Cómo empezar un XML Schema

El fichero `.xsd` debe estructurarse como:

myElement.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
<!-- Reglas XML Schema -->
</xs:schema>
```

xmlns:xs="http://www.w3.org/2001/XMLSchema" (http://www.w3.org/2001/XMLSchema%22)

- indica que los elementos y tipos de datos usados en el esquema vienen del espacio de nombres "http://www.w3.org/2001/XMLSchema" (http://www.w3.org/2001/XMLSchema%22). También especifica que los elementos y los tipos de datos que provengan de dicho espacio de nombres deben tener el prefijo **xs:. Este fragmento es el único obligatorio para que la definición sea correcta.**
  targetNamespace="https://www.w3schools.com" (https://www.w3schools.com%22)
- indica que los elementos definidos en el esquema pertenecen al espacio de nombres de "https://www.w3schools.com" (https://www.w3schools.com%22). Por defecto toma este valor.
  xmlns="https://www.w3schools.com" (https://www.w3schools.com%22)
- indica que el espacio de nombres por defecto es "https://www.w3schools.com" (https://www.w3schools.com%22).
  elementFormDefault="qualified"

- indica que cualquier elemento usado en una instancia XML que ha sido declarada con este esquema debe ser identificado con el espacio de nombres. Por defecto toma este valor.

Asociación con documentos XML

Se asocia mediante un espacio de nombres al ejemplar del documento en el elemento raíz.

Debe indicarse el espacio por defecto de nombres en el documento (coincide con el declarado en el propio archivo del esquema), se indica el espacio de nombres correspondiente al esquema (siempre es la misma dirección de Internet) y se asocia a este espacio el prefijo xs (se puede elegir otro prefijo, pero no es nada conveniente).

En el caso de que en el Schema únicamente definamos el espacio de nombres xmlns:xs="http://www.w3.org/2001/XMLSchema" (http://www.w3.org/2001/XMLSchema%22), en el documento xml habrá que indicar:

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

Y, si hace uso de espacios de nombres, pueden relacionarse cada uno con su XSD como:

```
<myElement xmlns:bk="http://example.com/books"
                  xmlns:au="http://example.com/author"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation="http://example.com/books book.xsd
                                      http://example.com/author author.xsd">
       <!-- Contenido del documento XML -->
</myElement>
```

5.2. Elementos de tipo simple

No tienen hijos, ni atributos. Pueden ser predefinidos o definidos por el usuario.

Tipos simples predefinidos

Tipo	Descripción	Tipo	Descripción
`string`	Caracteres	`gYear`	Un año `yyyy`
`boolean`	0, 1, true, false	`gMonthDay`	Día en un año. (12 de noviemre) Format `--mm-dd`
`decimal`	Real con decimales. Con punto	`gDay`	Día de cualquier mes de cualquier año `---dd`
`float`	Con notación científica	`gMonth`	Mes del año `--mm`
`double`	Float pero con doble precisión	`hexBinary`	Hexadecimal. Longitud par.
`duration`	Duración de tiempo en formato `PnYnMnDTnHnMnS` Número de años, meses, días, horas, minutos y segundos (P y T para separar)	`base64Binary`	Base 64. Alfanumericos, +, /, =
`dateTime`	Con formato `yyyy-mm-ddThh:mm:ss.nnnn:zzz`	`anyURI`	URL, absoluta o relativa

Tipo	Descripción	Tipo	Descripción
`time`	Hora en un día `hh:mm:ss.nnnn:zzz`	`QName`	Nombre cualificado XML. Cadena que valga como nombre de elemento XML.
`date`	Fecha en un día `yyyy-mm-dd`	`NOTATION`	Notación
`gYearMonth`	U mes. Formato `yyyy-mm`	``	

(gDayMonth no existe. Cuidao)

Tipos derivados predefinidos

Derivan de los primitivos, introduciendo alguna restricción.
normalizedString, token, language, NMTOKEN, NMTOKENS, Name, NCName, ID, IDREF, IDREFS, ENTITY, ENTITIES, integer, nonPositiveInteger, negativeInteger, long, int, short, byte, nonNegativeInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte, positiveInteger.

Veamos algún ejemplo sencillo de uso

```
<?xml version="1.0"?>
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema">
	<xs:element name="autor">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="nombre" type="xs:string">
				<xs:element ref="apellidos" type="xs:string">
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Tipos simples derivados por el usuario. Uso de facetas

Se definen con `xs:simpleType`
A base de tomar un tipo simple base y restringirlo `xs:restriction` con facetas

Las facetas disponibles son:

- `length`: Longitud exacta
- `minLength`: Longitud mínima
- `maxLength`: Longitud máxima
- `pattern`: Patrón mediante expresión regular
- `enumeration`: Conjunto de valores válidos (Solo uno de los dados)
- `whiteSpace`: Cómo deben tratarse los espacios en blanco
- `minInclusive`: Valor mínimo del rango incluido
- `minExclusive`: Valor mínimo del rango excluido
- `maxInclusive`: Valor máximo del rango incluido
- `maxExclusive`: Valor máximo del rango excluido
- `totalDigits`: Total de dígitos de un número
- `fractionDigits`: Número de dígitos decimales

Elementos para hacer patrones.

Patrón	Significado
[A-Z a-z]	Letra.
[A-Z]	Letra mayúscula.
[a-z]	Letra minúscula.
[0-9]	Dígitos decimales.
\D	Cualquier carácter excepto un dígito decimal.
(A)	Cadena que coincide con A.
A | B	Cadena que es igual a la cadena A o a la B.

Elementos para hacer patrones.

Patrón	Significado
AB	Cadena que es la concatenación de las cadenas A y B.
A?	Cero o una vez la cadena A.
A+	Una o más veces la cadena A.
A*	Cero o más veces la cadena A.
[abcd]	Alguno de los caracteres que están entre corchetes.
[^abcd]	Cualquier carácter que no esté entre corchetes.

Ejemplo, restricción de longitud mínima y máxima de contraseñas

```
<xs:complexType>
	<xs:sequence>
		<xs:element name="PW" type="password"/>
	</xs:sequence>
</xs:complexType>
<xs:simpleType name="password">
	<xs:restriction base="xs:string">
		<xs:minLength value="8"/>
		<xs:maxLength value="12"/>
	</xs:restriction>
</xs:simpleType>
```

Podría hacerse con un patrón regex: `<xs:pattern value="[A-Za-z_]{6,12}"/>`

Enumeraciones

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:simpleType name="salario">
		<xs:restriction base="xs:decimal">
			<xs:minInclusive value="10000"/>
			<xs:maxInclusive value="90000"/>
			<xs:fractionDigits value="2"/>
			<xs:totalDigits value="7"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:simpleType name="puestoTrabajo">
		<xs:restriction base="xs:string">
			<xs:enumeration value="Gestor de Ventas"/>
			<xs:enumeration value="Vendedor"/>
			<xs:enumeration value="Recepcionista"/>
			<xs:enumeration value="Desarrollador"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:element name="empleado">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="salario" type="salario"/>
				<xs:element name="puesto" type="puestoTrabajo"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
```

Espacios en blanco

Al `xs:string` se le puede restringir el espacio en blanco con `xs:whiteSpace` indicando los valores `preserve` (se queda como está), `replace` (todos los tabuladores, saltos de línea... se reemplazan por espacios en blanco), `collapse` (todos se reemplazan por un único espacio en blanco y se eliminan espacios al inicio y al final)

Tipos no atómicos

Listas

Se ponen con `xs:list itemType=""` Se pueden tener listas de enteros o de fechas. No deben confundirse con las enumeraciones (valor único). Estas representan uno o más valores dentro del elemento.

```
<xs:simpleType name="listaFechas">
	<xs:list itemType="xs:date"/>
</xs:simpleType>
```

Resultará algo como esto:

```
<diasVacaciones>2006-08-13 2006-08-14 2006-08-15</diasVacaciones>
```

Uniones

Agrupaciones de tipos permitiendo que el valor de un elemento sea más de un tipo.

```
<xs:simpleType name="evento">
	<xs:union memberTypes="carrera gimnasia"/>
</xs:simpleType>
```

Elementos simples globales vs locales

Cuando se declaran elementos globales estos pueden ser referenciados a lo largo del esquema haciéndolo más modular y fácil de mantener.
El inconveniente es que deben tener un nombre único.

Los elementos globales son aquellos en los que la declaración del elemento es un hijo directo del elemento raíz `xs:schema`.
Cuando se quiere hacer uso de él se le llama con el atributo ref.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="nombre" type="xs:string"/>
	<xs:element name="apellidos" type="xs:string"/>
	<xs:element name="autor">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="nombre"/>
				<xs:element ref="apellidos"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

Valores por defecto

Un elemento que no tiene hijos, puede tener un valor por defecto. Para ello se usa el atributo `default`. Este valor solo se usará cuando el elemento aparezca en la instancia y no tiene contenido.

`<xs:element name="puesto" type="xs:string" default="Vendedor">`

Valores fijos

Se obliga a que, si aparecen una instancia, contengan el valor indicado:

```
<xs:element name="estado" type="xs:string"
fixed="actualizado" minOccurs="0"/>
```

5.3. Elementos de tipo complejo

Son aquellos que tienen atributos, elementos hijos o una combinación de ambos.

No es necesario declarar de forma explícita que un elemento simple es simple; pero sí que es necesario indicar que un elemento complejo es complejo con el elemento `xs:complexType`

El modelo de contenido de un elemento complejo puede ser:

- `xs:sequence`: Elementos deben aparecer en el orden especificado
- `xs:all`: Elementos deben aparecer pero el orden no es importante
- `xs:choice`: Solo uno de los elementos debe aparecer

```
<xs:element name="nombreElemento">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="ElementoHijo1" type="xs:string"/>
			<xs:element name="ElementoHijo2" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>

<xs:element name="nombreElemento">
	<xs:complexType>
		<xs:all>
			<xs:element name="ElementoHijo1" type="xs:string"/>
			<xs:element name="ElementoHijo2" type="xs:string"/>
		</xs:all>
	</xs:complexType>
</xs:element>

<xs:element name="nombreElemento">
	<xs:complexType>
		<xs:choice>
			<xs:element name="ElementoHijo1" type="xs:string"/>
			<xs:element name="ElementoHijo2" type="xs:string"/>
		</xs:choice>
	</xs:complexType>
</xs:element>
```

Los elementos de tipo complejo pueden contener otros elementos de tipo complejo:

```
<xs:element name="nombreElemento">
	<xs:complexType>
		<xs:choice>
			<xs:element name="ElementoHijo1" type="xs:string"/>
			<xs:element name="ElementoHijo2">
				<xs:complexType>
					<xs:sequence>
					<xs:element name="ElementoNieto1" type="xs:string"/>
					<xs:element name="ElementoNieto2" type="xs:string"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="ElementoHijo3" type="xs:string"/>
		</xs:choice>
	</xs:complexType>
</xs:element>
```

Restricciones de ocurrencia

En el `xs:element` Puede ponerse ocurrencias infinitas `maxOccurs="unbounded"

Si `maxOccurs="unbounded"` es de 1 a infinito
Si `minOccurs="0" maxOccurs="unbounded"` es de 0 a infinito
Si `minOccurs="0"` se pone entre 0 y 1 (opcional)

Contenido mixto

Un elemento puede contener tanto los elementos hijos como texto. Esto se hace indicando `mixed="true"`

```
<xs:element name="elementoPadre">
	<xs:complexType mixed="true">
		<xs:sequence>
			<xs:element name="elementoHijo1" type="xs:string"/>
			<xs:element name="elementoHijo2" type="xs:string"/>
		</xs:sequence>
	</xs:complexType>
</xs:element>
```

5.5. Atributos en XML Schema

Elementos vacíos con atributos

Se declara como elemento complejo, solo incluyendo el atributo.

Elementos con contenido complejo con atributos

Se declaran después del modelo de grupos del elemento

Elementos con contenido simple con atributos

Se declaran usando `xs:simpleContent` y extendiendo el elemento usando `xs:extension` que debe especificar el tipo del contenido simple usado en el atributo base.

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

Restringiendo los valores de los atributos

De la misma forma que los elementos

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="libro">
	<xs:complexType>
		<xs:sequence>
			<xs:element name="titulo" type="xs:string"/>
			<xs:element name="autor">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="nombre" type="xs:string"/>
					</xs:sequence>
					<xs:attribute name="titulo">
						<xs:simpleType>
							<xs:restriction base="xs:string">
							<xs:enumeration value="Sr."/>
							<xs:enumeration value="Sra."/>
							<xs:enumeration value="Dr."/>
							</xs:restriction>
```

```
                                        </xs:simpleType>
                                    </xs:attribute>
                                </xs:complexType>
                                </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:element>
```

Valores por defecto y requiriendo atributos
Para ponere uno por defecto se usa `default`
Para fijarlo se usa el atributo `fixed`
Para requerirlo se pone el valor `use:"required"`

5.4. Claves en XML Schema

Unicidad

Puede usarse `<xs:unique>` para reforzar la unicidad.
Tiene dos elementos hijo:

- `<xs:selector>` tiene un atributo xpath con la expresión que referencia a los elementos afectados por la restricción
- `<xs:field>` Atributo xpath que especifica qué parte del elemento debe ser única

Claves

El uso de claves y referencias a claves se puede hacer mediante `xs:key` y `xs:keyref`
Ambos contienen también los elementos `xs:selector` y `xs:field`
El elemento `xs:key` se usa para identificar los elementos que van a ser referenciados por los elementos especificados con `xs:keyref`

Este mecanismo asegura que:

- Los valores de claves deben ser únicos. No puede haber dos con el mismo valor.
- Debe haber integridad referencial. No puede haber referencia a una clave por un valor que no exista entre las claves.

5.5. Documentación del schema

Los analizadores no garantizan que los comentarios no se modifiquen al procesar los documentos, luego estos podrían perderse.
Para documentación, XML Schema tiene definido un elemento `<xs:annotation>` para guardar información adicional.

Además puede contener:

- `<xs:documentation>` Puede contener elementos de esquema y elementos XML bien estructurados.
- `<xs:appinfo>` Parecido a `<documentation` aunque inicialmente se pensó que `<documentation>` fuese legible para usuarios y que este guardase información para los programas.

6. Creación y validación

- Notepad++
- Editix XML
- XML Copy Editor
- Netbeans
- Utilidades online