



Resumen tema 4

Lenguajes de Marcas y Sistemas de Gestión de la Información

9 de enero de 2023

CIFP Carlos III - Cartagena

Santiago Francisco San Pablo Raposo

2º curso DAW

Contenido

# Índice de ilustraciones.

# Índice de tablas.

Resumen tema 4.

# 1.- XML. Estructura y sintaxis.

En la primera unidad vimos que un documento XML básico estaba formado por:

* **Prólogo**: informa al intérprete encargado de procesar el documento de todos aquellos datos que necesita para realizar su trabajo. Consta de dos partes
  + **Definición de XML**: Donde se indica la versión de XML que se utiliza, el código de los datos a procesar y la autonomía del documento. Este último dato hasta ahora siempre ha sido "yes", puesto que los documentos no requerían otros
  + **Declaración del tipo de documento**: es el nombre del ejemplar precedido de la cadena <!DOCTYPE y separado de ésta por (al menos) un espacio. Veremos más posteriormente
* **Ejemplar**: Contiene los datos del documento que se quiere procesar
  + Es el elemento raíz del documento y ha de ser único.
  + Está compuesto de elementos estructurados.
  + El elemento raíz es el ejemplar y las hojas los elementos terminales.
  + Los elementos pueden estar a su vez formados por atributos .

**Lo que nos falta por abordar es la “validez” de dichos documentos**, en términos de poder definir con qué estructura y de qué naturaleza son los datos que aparecen en él, qué cosas deben aparecer obligatoriamente, qué valores se espera que sean usados, etc. Podremos así pasar a generar documentos XML “válidos”, y no sólo “bien formados”.

Veamos a continuación algunos conceptos previos necesarios para abordar con garantías las **dos técnicas de validación de documentos XML más extendidas**: DTD y XML Schema.

## 1.1.- Declaración de tipo de documento (para DTDs).

Permite al autor definir restricciones y características en el documento, aunque no habíamos profundizado en las partes que la forman.

### 1.1.1.- Declaración.

Comienza con *<!DOCTYPE*, seguido de al menos un espacio, y después el nombre del tipo, que ha de ser idéntico al del ejemplar del documento XML en el que se está trabajando.

<!DOCTYPE nombre\_ejemplar>

### 1.1.1.- Definición.

Permite asociar al documento una definición de tipo DTD (Document Type Definition), la cual se encarga de definir las cualidades del tipo.

De momento sólo diremos que los DTD’s se componen de declaraciones de marcado, las cuales pueden ser internas o externas.

* **Internas**: definidas en el mismo documento. No se comparten con ningún otro. Se localizan dentro de unos corchetes que siguen a la declaración de tipo del documento:

<!DOCTYPE nombre\_ejemplar [

<!ELEMENT elemento1 (elemento2, elemento3+)>

<!ELEMENT elemento2 (#PCDATA)>

<!ELEMENT elemento3 (#PCDATA)>

]>

* **Externas**: definidas en un archivo separado (generalmente con extensión .dtd). Pueden ser compartidas por varios archivos XML.

<!DOCTYPE nombre\_ejemplar SYSTEM "../dtds/fic\_definicion.dtd">

En un archivo XML puede coexistir una mezcla de declaraciones internas y externas.

Si existen externas, **se debe indicar además dos cosas**:

* **Dónde encontrar las declaraciones**, lo cual se hará **mediante una URI**, como hemos hecho en el ejemplo.
* **Indicar en la declaración de XML que el documento no es autónomo**.
  + Un documento autónomo se indica mediante standalone=”yes”.
  + Un documento que debe ser validado por DTD, como en este caso, se indica mediante standalone=”no”

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>

**Sobre estas declaraciones externas, caben dos posibilidades**:

* **Usar identificadores de sistema** (SYSTEM, como en el ejemplo anterior): se destinan a un uso **privado**.
  + Se acepta que el identificador y la URI de localización del DTD son la misma cosa.
* **Usar identificadores públicos** (PUBLIC): se usan para identificar una entrada en un catálogo, supuestamente **público**, y del que tienen conocimiento los sistemas a los que se destinan los XMLs.
  + Se indica el identificador y la URI separadamente.
  + Por ejemplo:

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

En los públicos, el identificador puede ser utilizado por el procesador XML para intentar generar un URI alternativo, posiblemente basado en alguna tabla.

También conviene saber que **primero se procesa el subconjunto interno y después el externo**, lo que permite sobreescribir declaraciones externas compartidas entre varios documentos para ajustarlas a un documento concreto de forma más específica.

## 1.2.- Definición de la sintaxis de documentos XML.

Las etiquetas de marcado describen la estructura del documento.

### 1.2.1.- Elementos.

Un elemento es un grupo formado por una etiqueta de apertura (donde puede opcionalmente contener atributos), otra de cierre y el contenido que hay entre ambas.

En los documentos de lenguajes de marcas, **la distribución de los elementos está jerarquizada según una estructura de árbol**.

**El orden de los elementos es importante, pues se pueden recorrer secuencialmente** (sobre todo si su etiqueta es la misma).

### 1.2.2.- Atributos.

El orden de los atributos no es importante, pero no puede haber dos con el mismo nombre.

**Los atributos no pueden tener nodos que dependan de ellos**, por tanto, solo pueden corresponder con hojas de la estructura de árbol que jerarquiza los datos. De todos modos, aunque es cierto que los atributos son hojas, pero **las hojas pueden ser atributos o elementos**.

### 1.2.3.- Elección de diseño: ¿atributo o elemento?

El dato será un **elemento** si cumple alguna de las siguientes condiciones:

* **Contiene subestructuras** de datos (es un dato complejo).
* Es de un tamaño considerablemente grande.
* Su valor posee una **longitud** o una naturaleza muy **variable**.
* Su **valor** va a ser mostrado a un usuario o aplicación que **requiere de formato**, como el texto enriquecido.

Los casos en los que el dato será un **atributo** son:

* El dato es de **pequeño tamaño** y su valor raramente cambia.
* El dato tiene una naturaleza y conjunto de posibles valores bien definidos
* El dato guía el procesamiento XML, pero no se va a mostrar.

### 1.2.4.- Namespaces (espacios de nombres).

Los espacios de nombres nos permiten:

* **Diferenciar entre los elementos** y atributos de distintos vocabularios (con diferentes significados) **que comparten nombre de identificador**.
* **Agrupar todos los elementos y atributos relacionados de una aplicación XML** para que el software pueda reconocerlos con facilidad

**Declaración**:

xmlns:"URI\_namespace"

La declaración anterior **establece el espacio de nombres predeterminado** (por defecto) **para cualquier etiqueta** que aparezca en el fichero XML.

**En ocasiones, es necesario hacer referencia a más de un espacio de nombres en el mismo XML**, en cuyo caso necesitamos “**alias**”, en forma de prefijos, que diferencien a unos de otros:

xmlns:prefijo="URI\_namespace"

# 2.- DTD – Definiciones de tipo de documento.

Es una técnica de validación consistente en el establecimiento de una serie de declaraciones que imponen requisitos que el XML ha de cumplir para garantizar su validez.

**Estas definiciones indican qué elementos pueden aparecer en un documento y dónde**, así como el contenido y los atributos del mismo. Por ejemplo, permiten:

* **Especificar la estructura** del documento .
* Reflejar una **restricción de integridad referencial** mínima utilizando (ID e IDREF).
* Utilizar unos pequeños **mecanismos de abstracción** comparables a las **macros**, que son las **entidades**.
* Incluir documentos externos.

**¿Cuáles son los inconvenientes de los DTD?** Los principales son:

* 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 permite las secuencias no ordenadas.
* 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

**También hemos visto que**:

* **Cuando están definidas dentro del documento XML se ubican entre corchetes**, dentro del elemento <!DOCTYPE>
* **Cuando está definido en un fichero**, éste **es de texto plano**, con extensión “.dtd”.

## 2.1.- Declaraciones de tipos de elementos terminales.

<!ELEMENT nombre\_elemento declaración\_de\_contenido>

**Aclaración**: TIPO TERMINAL = ELEMENTO SIN SUB-ELEMENTOS.

En el caso de elementos terminales, la declaración de contenido es dada por uno de los siguientes valores:

* **EMPTY**: indica que el elemento no es contenedor.
  + **Por ejemplo**: la siguiente definición muestra un elemento “E” que no contiene nada:

<!ELEMENT E EMPTY>

* **ANY**: Permite que el contenido del elemento sea cualquier cosa.
  + **Un ejemplo** de definición de un elemento “A” de este tipo es:

<!ELEMENT A ANY>

* **(#PCDATA): Parsed Character Data**. Indica que el elemento solo contiene texto, cualquier etiqueta que se detecte será invalidada. Por ello, **no puede contener elementos**, solo datos de tipo carácter, excepto los siguientes: < & [ ] >.
  + **Ejemplo**:

<!ELEMENT P (#PCDATA)>

## 2.2.- Declaraciones de tipos de elementos no terminales.

Son **aquellos elementos que están formados por otros elementos**. Para definirlos utilizamos referencias a los grupos que los componen tal y como muestra el ejemplo.

<!ELEMENT A (B,C)>

En este caso se ha definido un elemento “A” que está formado por un elemento “B” seguido de un elemento “C”.

También hay que indicar cuando un elemento puede no aparecer.

### 2.2.1.- Opción: ?

Indica que el elemento no es obligatorio. En el siguiente ejemplo el subelemento trabajo es opcional, mientras que casa no lo es. Ambos sólo pueden aparecer una vez como mucho

<!ELEMENT telefono ( trabajo? , casa )

### 2.2.2.- Uno-o-más: +.

El elemento debe estar presente al menos una vez, pudiendo aparecer varias veces.

<!ELEMENT pc ( placa , cpu , disco+ )

La repetición puede afectar a un grupo de sub-elementos:

<!ELEMENT provincia ( nombre , ( cp , ciudad )+ )

### 2.2.3.- Cero-o-más: \*

El elemento es opcional, pudiendo estar presente cero, una o varias veces.

<!ELEMENT provincia (nombre , ( cp , ciudad )\* )

### 2.2.4.- Elección: |

Cuando se utiliza (sustituyendo a las comas en la declaración de grupos) indica que para formar el documento XML hay que elegir entre uno de los elementos separados por este operador.

<!ELEMENT autorización ( fecha , alumno , (padre|madre|tutor) )

## 2.3.- Declaraciones de atributos.

Se declaran como una lista de atributos:

<!ATTLIST elemento atributo tipo modificador>

* El **tipo** define su “**naturaleza**”, algo **parecido al tipo de dato**.
* El modificador determina la obligatoriedad y si se refiere a un valor constante.

**!ATTLIST puede usarse de dos formas**:

* **Para declarar varios atributos** asociados a un mismo elemento, **todos de una vez**.

<!ATTLIST elemento

atributo1 tipo1 modificador1

atributo2 tipo2 modificador2

atributo3 tipo3 modificador3

>

* **Repetirse varias veces** para enumerar uno a uno los atributos de un elemento.

<!ATTLIST elemento atributo1 tipo1 modificador1>

<!ATTLIST elemento atributo2 tipo2 modificador2>

<!ATTLIST elemento atributo3 tipo3 modificador3>

### 2.3.1.- Tipos de atributos.

* **Enumeración**: el atributo solo puede tomar uno de los valores determinados dentro de un paréntesis y separados por el operador |.

<!ATTLIST fecha dia\_semana

(lunes|martes|miércoles|jueves|viernes|sábado|domingo)

>

* **CDATA**: se utiliza cuando el atributo es una cadena de texto, sin ninguna restricción en su contenido.

<!ATTLIST persona nombre CDATA #REQUIRED >

* **ID**: permite declarar un atributo identificador en un elemento. Esto determina que **no puede repetirse el valor de dicho atributo**, pues ha de ser único en el documento.

<!ATTLIST ticket numero ID #REQUIRED >

Serían válidos estos casos:

...

<ticket numero="TK00001" fecha="1/1/2016">...</ticket>

<ticket numero="TK00002" fecha="1/1/2016">...</ticket>

<ticket numero="TK00003" fecha="1/1/2016">...</ticket>

<ticket numero="TK00004" fecha="2/1/2016">...</ticket>

...

Pero **no serían válidos identificadores de tickets que fuesen números enteros simplemente**.

* **IDREF**: Permite hacer referencias a identificadores. En este caso el valor del atributo ha de corresponder con el de un identificador de un elemento existente en el documento.
  + **Por ejemplo**: podría existir un elemento “abono” que hiciera referencia al “ticket” que se abona.

<!ATTLIST abono

numero ID #REQUIRED

ticket IDREF #REQUIRED

fecha...

>

Siguiendo el mismo ejemplo de documento XML anterior, podría aparecer una referencia así:

...

<abono

numero="AB00002"

ticket=”TK00003”

fecha="4/1/2016">

</abono>

...

* [**NMTOKEN**](https://www.abrirllave.com/dtd/atributos-de-tipo-nmtoken.php): permite determinar que **el valor de un atributo ha de ser una sola palabra compuesta por los caracteres permitidos por XML para componer nombres de etiquetas**, atributos e identificadores en general.

### 2.3.2.- Modificadores de atributos.

* **#IMPLIED**: determina que el atributo sobre el que se aplica es **opcional**.

<!ATTLIST persona color\_ojos CDATA #IMPLIED >

* **#REQUIRED**: determina que el atributo tiene caracte4r obligatorio, como en los ejemplo anteriores.
* **#FIXED**: Permite definir un valor fijo para un atributo, independientemente de que ese atributo se defina explícitamente en una instancia del elemento en el documento XML (como debe ser).
  + Al menos, al definirlo así forzamos a que no puede contener otro valor que no sea ese.
  + Al usar este modificador, debemos indicar también dicho valor fijo. Por ejemplo:

<!ATTLIST regla habilitada CDATA #FIXED "true" >

De esta forma, se entenderá **que la ausencia de este atributo es la que marca la inhabilitación de esta**.

## 2.4.- Declaraciones de entidades.

Las entidades nos permiten definir **constantes** en un documento XML. Cuando se usan dentro del documento XML se limitan por "&" y ";", por ejemplo:

&entidad;

Al procesar el documento XML, el intérprete sustituye la entidad por el valor que se le ha asociado en el DTD.

Para definir una entidad en un DTD **se usa el elemento <!ENTITY>**.

No admiten recursividad, es decir, **una entidad no puede hacer referencia a ella misma**.

**Las entidades pueden ser de tres tipos**: externas, internas y de parámetro.

### 2.4.1.- Internas.

**Existen 5 entidades predefinidas en el lenguaje**, son:

|  |  |  |
| --- | --- | --- |
| Entidad | Valor (literal) | Valor (nombre) |
| &lt; | < | Signo “menor qué” |
| &gt; | > | Signo “mayor qué” |
| &quot; | “ | Comillas dobles |
| &apos; | ‘ | Comilla simple (apóstrofe) |
| &amp; | & | Ampersand |

**Es posible definir una entidad interna nueva**, mediante la sintaxis:

<!ENTITY nombre\_entidad "valor de la entidad">

**Por ejemplo**, la definición:

<!ENTITY dtd "Definiciones de Tipo de Documento">

Podría ser usada:

<doc id="D0001" tipo="&dtd;">

Este es un documento de tipo "&dtd;"

</doc>

### 2.4.2.- Externas.

**El valor de la entidad** no se indicará entre comillas, junto a su nombre, sino que **estará contenido en un archivo separado**.

A diferencia de las entidades internas, **el contenido de los ficheros es analizado**, por lo que:

* Puede que contengan un mero texto plano, con el valor de la entidad.
* O puede que contenga un trozo de XML interpretable.

En todo caso, **su contenido se reemplazará por la entidad, y tras ello, se procesará como parte del XML en el que aparece la entidad**.

Un ejemplo de declaración de una entidad externa de sistema es:

<!ENTITY nom\_entidad SYSTEM "http://localhost/dtd/entidad.txt">

**Cuando es necesario incluir ficheros con formatos binarios, es decir ficheros que no se analicen**, se utiliza la palabra reservada **NDATA** en la definición de la entidad.

<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>

No obstante, se requiere además de una entidad de notación, que son un tipo diferente y que vamos a estudiar en el apartado 2.5.

### 2.4.3.- De parámetro.

Permite **dar nombres a partes de un DTD y hacer referencia a ellas** a lo largo del mismo DTD.

Son útiles cuando varios elementos del DTD comparten lista de atributos o especificaciones de contenidos. Se denotan por *%entidad;*

<!ENTITY %direccion "calle, numero?, ciudad, cp">

<!ELEMENT alumno (dni, %direccion;)>

<!ELEMENT ies (nombre, %direccion;)>

### 2.4.4.- De parámetros externas.

Son como las anteriores, sólo que la definición de la entidad puede residir en otro archivo DTD:

<!ENTITY %persona SYSTEM "persona.dtd">

## 2.5.- Declaraciones de notación.

Cuando se incluyen ficheros binarios en un fichero XML, ¿cómo le decimos qué aplicación ha de hacerse cargo de ellos?

<!NOTATION nombre SYSTEM aplicacion>

Por ejemplo, siguiendo con el ejemplo anterior del gif, una notación llamada gif donde se indica que se hace referencia a un editor de formatos gif para visualizar imágenes será:

<!NOTATION gif SYSTEM "gifEditor.exe">

No obstante, no es rigurosamente necesario indicar la aplicación. **Si no se desea vincular el contenido binario a una aplicación, bastaba con la declaración de la entidad ya vista**:

<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>

## 2.6.- Secciones condicionales.

Permiten incluir o ignorar partes de la declaración de un DTD. Para ello se usan dos tokens:

### 2.6.1.- Include.

**Permite que se vea esa parte de la declaración del DTD**. Su sintaxis es:

<![INCLUDE [Declaraciones visibles] ] >

Por ejemplo:

<![INCLUDE [ <!ELEMENT nombre (#PCDATA)>] ] >

### 2.6.2.- Ignore.

Permite **ocultar esa sección de declaraciones dentro del DTD**: La forma de uso es:

<![IGNORE [Declaraciones ocultas] ] >

Por ejemplo:

<![IGNORE [<!ELEMENT clave (#PCDATA)>] ] >

### 2.6.3.- Ejemplo de secciones condicionadas y entidades de parámetros.

**A continuación se muestra un DTD en el que aparecen dos secciones**, que resultarán condicionadas según el valor de las entidades de parámetros (%datos\_basicos y %datos\_ampliados):

<![ %datos\_basicos; [

<!ELEMENT persona (nombre, edad)>

]]>

<![ %datos\_ampliados; [

<!ELEMENT persona (nombre, apellidos, edad, ciudad)>

]]>

<!ELEMENT nombre (#PCDATA)>

<!ELEMENT apellidos (#PCDATA)>

<!ELEMENT edad (#PCDATA)>

<!ELEMENT ciudad (#PCDATA)>

Según sean los valores de %datos\_basicos y %datos\_ampliados, puede optarse por forzar a que el elemento “persona” posea 2 o 4 sub-elementos.

La aplicación práctica podría verse en estos dos ejemplos de documento XML, cada uno de los cuales opta por una posibilidad:

**Para limitar a datos básicos**:

<?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>

**Para obligar a datos ampliados**:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!DOCTYPE persona SYSTEM "persona.dtd" [

<!ENTITY % datos\_basicos "IGNORE">

<!ENTITY % datos\_ampliados "INCLUDE">

]>

<persona>

<nombre>Ana</nombre>

<apellidos>Sanz Tin</apellidos>

<edad>19</edad>

<ciudad>Pamplona</ciudad>

</persona>

Haber puesto IGNORE en ambos o INCLUDE en ambos podría haber ocasionado que ningún XML fuera válido.

# 3.- XML Schema.

Es una técnica de validación de XML diferente. **DTD presenta algunas carencias, que están subsanadas con XML Schema**:

* Poder **especificar tipos de datos** para los elementos y atributos, indicando además algunas **restricciones** **en sus valores**.
* Poder hacer **uso de espacios de nombres** para las reglas de validación.
* **Usar sintaxis XML**, al igual que en el documento a validar.

Los **archivos serán de texto plano**, aunque para ellos se usa la **extensión “.xsd”**.

Los elementos XML que se utilizan para generar un esquema han de pertenecer al espacio de nombre “XML Schema”, que es: <http://www.w3.org/2001/XMLSchema>.

El ejemplar (**elemento raíz**) de estos ficheros es <xs:schema>, contiene declaraciones para todos los elementos y atributos que puedan aparecer en un documento XML asociado válido. Con lo dicho hasta ahora, **un fichero XSD básico tendría el siguiente aspecto**.

<?xml version="1.0"?> <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

...

</xs:schema>

Los elementos hijos inmediatos de este ejemplar (representados con puntos suspensivos en el ejemplo), son etiquetas <xs:element> (elementos). Obviamente, el ejemplar del XML a validar debe figurar como elemento. **Por ejemplo**: si imaginamos que el XSD anterior valida un XML con raíz <nota>, su aspecto será el siguiente:

<?xml version="1.0"?>

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

<xs:element name="nota">

</xs:element>

</xs:schema>

## 3.1.- Tipos de datos simples.

Por ejemplo, para indicar que el elemento raíz, su contenido es un texto:

<?xml version="1.0" encoding="UTF-8"?>

<nota>

Hola mundo.

</nota>

Para ello, en el XSD, **usamos el atributo “type”** de la siguiente forma:

<?xml version="1.0"?>

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

<xs:element name="nota" type="xs:string">

</xs:element>

</xs:schema>

Estos tipos de datos **se pueden aplicar al contenido de los elementos y a los valores de los atributos**. A continuación, se explican los más importantes:

|  |  |
| --- | --- |
| Tipo de dato | Explicación |
| string | Cadena de caracteres UNICODE. |
| boolean | Valores lógicos “true” o “false”. |
| integer | Número entero, positivo o negativo. |
| positiveInteger | Número entero positivo. |
| negativeInteger | Número entero negativo. |
| decimal | Número decimal, en coma fija, por ejemplo: 8,97. |
| dateTime | Fecha y hora absolutas (no una duración de tiempo). |
| date | Fecha absoluta, en formato CCYY-MM-DDD (año-mes-día) (\*). |
| time | Hora, en el formato hh:mm:ss (horas:minutos:segundos). |
| gYearMonth | Mes y año determinado, con formato CCYY-MM (\*). |
| gYear | Año gregoriano, el formato usado es CCYY(\*). |
| gMonthDay | Día (1-31) y mes (1-12), mediante el formato -MM-DD. |
| gDay | Día del mes (1-31), mediante el formato -DD. |
| gMonth | Mes, mediante el formato -MM. Por ejemplo, febrero es -02. |
| anyURI | Cualquier URI válida. |
| duration | Representa una duración de tiempo expresado en años, meses, días, horas, minutos, segundos. |
| **El formato utilizado es**: PnYnMnDTnHnMnS |
| Por ejemplo, para representar una duración de 2 años, 4 meses, 3 días, 5 horas, 6 minutos y 10 segundos, habría que poner:  P2Y4M3DT5H6M7S |
| Se pueden omitir los valores nulos, luego una duración de 2 años será P2Y. |
| Para indicar una duración negativa se pone -P. |
| language | Representa identificadores de idiomas, sus valores están definidos en el estándar RFC 1766. |
| Varios presentes en DTD | Nos referimos a los tipos: ID, IDREF, ENTITY, NOTATION, MTOKEN. Representan lo mismo que en los DTD’s ([ver apartado 2.3](#_2.3.-_Declaraciones_de)). |
| Más información sobre tipos de datos en la URL: [XML Schema Part 2: Datatypes Second Edition (w3.org)](https://www.w3.org/TR/xmlschema-2/) |

*\*CCYY significa que el año puede expresarse con 4 dígitos o sólo con los 2 últimos. Los valores cuyo carácter alias se duplica significan que debe rellenarse con ceros por la izquierda. Así, el mes (MM) de abril es “04”, las 8:05 (hora hh:mm:ss) es “08:05:00”.*

## 3.2.- Restricciones de los tipos de datos simples.

Se refieren a las **restricciones que podemos aplicar sobre los valores** de los datos de un elemento o atributo (longitud máxima, enumeración de posibilidades, etc.).

Se aplican **al definir** lo que se denomina como “**tipos simples**” (tipos de datos definidos por el usuario, pero que almacenan valores simples). **Los tipos simples se definen mediante la etiqueta <xs:simpleType>**.

**Por ejemplo**: vamos a **definir un tipo simple** para los valores del elemento “nota”:

<xs:element name="nota">

<xs:simpleType>

...

</xs:simpleType>

</xs:element>

Entre los puntos suspensivos, habrá que indicar las restricciones. **Se hace mediante la etiqueta <xs:restriction>**, quedando el conjunto:

<xs:element name="nota">

<xs:simpleType>

<xs:restriction base="xs:string">...</xs:restriction>

</xs:simpleType>

</xs:element>

Como puede verse, **al definir la restricción se indica también el tipo “base”**, que como su nombre indica es el tipo de dato que sirve como base y sobre el cual aplicar las restricciones

**Si no existieran restricciones**, habría bastado con el ejemplo anterior, usando ***type=”xs:string”***.

A continuación, veremos algunas posibilidades respecto a las **restricciones**:

|  |  |
| --- | --- |
| Tipo de restricción | Explicación |
| length, minlength, maxlength | Longitud del tipo de datos (fija, mínima y máxima). |
| <xs:element name="nota">  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:maxLength value="160"/>  </xs:restriction>  </xs:simpleType>  </xs:element> |
| enumeration | Restringe a un determinado conjunto de valores. |
| <xs:element name="estado">  <xs:simpleType>  <xs:restriction base="xs:string">  <xs:enumeration value="conectado"/>  <xs:enumeration value="ocupado"/>  </xs:restriction>  </xs:simpleType>  </xs:element> |
| maxInclusive, minInclusive, maxExclusive, minExclusive | Fija los límites superiores/inferiores del tipo de datos.   * **Inclusive**: el valor que se determine es parte del conjunto de valores válidos para el dato. * **Exclusive**: el valor dado no pertenece al conjunto de valores válidos. |
| totalDigits, fractionDigits | Número de dígitos totales y decimales de un número decimal. |

**Ejemplo de totalDigits, fractionDigits, minExclusive y maxInclusive**: sirve para validar notas, asumiendo que:

* Han de ser estrictamente mayores que 0.
* Su valor no puede ser superior a 10 (pero sí igual).
* Deben expresarse a lo sumo con dos decimales.

<xs:element name="nota">

<xs:simpleType>

<xs:restriction base="xs:decimal">

<xs:totalDigits value="4"/>

<xs:fractionDigits value="2"/>

<xs:minExclusive value="0"/>

<xs:maxInclusive value="10"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

**NOTA**: el nº total de dígitos es 4 porque hay que descontarle los 2 que serán parte decimal (2 para la parte entera + 2 para la parte decimal = 4 dígitos totales).

Un tipo de restricción más: el tipo pattern

Permite construir máscaras que han de cumplir los datos de un elemento. La siguiente tabla muestra algunos de los **caracteres que tienen un significado especial para la generación de las máscaras**:

|  |  |  |  |
| --- | --- | --- | --- |
| Patrón | Significado | Patrón | Significado |
| [A-Z a-z] | Letra. | AB | Cadena que es la concatenación de las cadenas A y B. |
| [A-Z] | Letra mayúscula. | A? | Cero o una vez la cadena A. |
| [a-z] | Letra minúscula. | A+ | Una o más veces la cadena A. |
| [0-9] | Dígitos decimales. | A\* | Cero o más veces la cadena A. |
| \D | Cualquier carácter excepto un dígito decimal. | [abcd] | Alguno de los caracteres que están entre corchetes. |
| (A) | Cadena que coincide con A. | [^abcd] | Cualquier carácter que no esté entre corchetes. |
| A | B | Cadena que es igual a la cadena A o a la B. | \t | Tabulación. |

Las máscaras son un mecanismo muy potente para validar valores alfanuméricos.

**Ejemplo**: cómo validar que un DNI sea introducido a base de 8 dígitos y una letra.

<xs:element name="dni">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:pattern value="[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][A-Z]"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

## 3.3.- Elementos del lenguaje.

*INFO: El contenido de este punto 4.3.3 es diferente y complementario a los contenidos de la plataforma. Se recomienda leer primero el punto 4.3.4*

Veamos algunas técnicas adicionales para sacar partido a la declaración de XSDs.

### 3.3.1.- Definición de atributos.

Casi todo lo visto para el contenido de un elemento, es válido también para los atributos.

**Caben dos salvedades importantes**:

* **Los atributos se definen con la etiqueta <attribute>**, **dentro del elemento (que será sí o sí complexType)** al que han de pertenecer.
* Los tipos de datos solo pueden ser simples.

**Ejemplo de definición de atributos**:

<xs:element name="terminal">

<xs:attribute name="numero" type=”xs:string” />

<xs:attribute name="marca" type=”xs:string” />

<xs:attribute name="pin" type=”xs:string”>

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:pattern value="[0-9][0-9][0-9][0-9] "/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

</xs:element>

Donde vemos que “terminal” puede poseer como atributos “numero”, “marca” y “pin”. Este último posee una restricción de que ha de componerse con 4 números del 0 al 9.

**Podemos complementar la definición de los atributos con algunos atributos** (valga la redundancia) **de la etiqueta <attribute>**. Algunos de ellos muy interesantes:

* **Establecer un valor por defecto**:

<xs:attribute name="idioma" type="xs:string" default="EN"/>

* **Fijar un valor constante**:

<xs:attribute name=" idioma " type="xs:string" fixed="EN"/>

* **Hacer que sea obligado**:

<xs:attribute name="lang" type="xs:string" use="required"/>

### 3.3.2.- Reutilizar tipos declarados.

**Los tipos de datos pueden declararse y reutilizarse**. Para ello, basta con:

* **No declararlos en el ámbito de un elemento, sino fuera de él**.
* Indicar un nombre mediante su atributo “name”.
* Reutilizarlo donde haga falta, mediante el atributo “type”

**Ejemplo**: declaración de un tipo compuesto “dirección”, y usarlo después:

<xs:complexType name="direccion">

<xs:sequence>

<xs:element name="tipo\_via" type="xs:string" />

<xs:element name="nombre\_via" type="xs:string" />

<xs:element name="numero” type="xs:integer" /> <xs:element name="localidad” type="xs:string" />

<xs:element name="provincia” type="xs:string" />

</xs:sequence>

</xs:complexType>

...

<xs:element name="persona">

<xs:complexType>

<xs:sequence>

<xs:element name="nombre" type="xs:string" />

<xs:element name="direccion" type="direccion"/>

</xs:sequence>

</xs:complexType>

</xs:element>

### 3.3.3.- Cardinalidad de los elementos.

Tras estudiar el punto 3.4, vemos que **se pueden componer tipos complejos a base de indicar secuencias (ordenadas o no) y selecciones de elementos**.

La **cardinalidad** se refiere al **número de ocurrencias mínimo y máximo** con el que cada elemento de estas estructuras complejas puede presentarse. **Por defecto**:

* **Cada elemento de una secuencia ha de aparecer una y sólo una vez** (ordenada o no).
* Por cada selección debe aparecer obligatoriamente uno y sólo uno de sus opciones.

**Podemos cambiar este comportamiento indicando el nº mínimo y/o máximo de veces que un elemento puede aparecer en una secuencia**.

**Ejemplo**: supongamos una forma de validar una etiqueta <familia>:

<xs:element name="familia">

<xs:complexType>

<xs:sequence>

<xs:element name="padre" type="xs:string" />

<xs:element name="madre" type="xs:string" />

<xs:element name="hijo" type="xs:string" />

</xs:sequence>

</xs:complexType>

</xs:element>

**Si queremos que cualquiera de los elementos sea opcional**, sería así:

<xs:element name="padre" type="xs:string" minOccurs="0" />

<xs:element name="madre" type="xs:string" minOccurs="0" />

<xs:element name="hijo" type="xs:string" minOccurs="0"/>

**Si queremos limitar la enumeración a 2 padres, 2 madres y 20 hijos**:

<xs:element name="padre" type="xs:string" minOccurs="0" maxOccurs="2" />

<xs:element name="madre" type="xs:string" minOccurs="0" maxOccurs="2" />

<xs:element name="hijo" type="xs:string" minOccurs="0" maxOccurs="20" />

Es decir, podrán aparecer 0, 1 o 2 padres, después 0, 1 o 2 madres, y luego podrán aparecer hijos o no, hasta un máximo de 20. El orden es importante porque hemos utilizado <sequence> y no <all>.

### 3.3.4.- Agrupaciones.

La etiqueta *<xs:group>* permite nombrar agrupaciones de elementos y de atributos, para hacer referencia a ellas en múltiples puntos en los que puede hacer falta. **Por ejemplo**:

<?xml version="1.0"?>

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

<xs:group name="datos\_cliente">

<xs:sequence>

<xs:element name="razon\_social" type="xs:string"/>

<xs:element name="nombre\_comercial" type="xs:string"/>

<xs:element name="facturar\_a" type="xs:string"/>

<xs:element name="enviar\_a" type="xs:string"/>

</xs:sequence>

</xs:group>

<xs:complexType name="tipo\_pedido">

<xs:group ref="datos\_cliente"/>

<xs:attribute name="fecha" type="xs:date"/>

<xs:attribute name="estado" type="xs:string"/>

</xs:complexType>

<xs:element name="pedido" type="tipo\_pedido"/>

</xs:schema>

En este XSD se ha definido el grupo “datos\_cliente”, que es utilizado en la definición del tipo “tipo\_pedido”, que finalmente será el que tipifique al elemento “pedido” que aparece abajo.

### 3.3.5.- Contenido mixto.

Un tipo de datos complejo puede contener a su vez contenido propio como si se tratara de un elemento terminal, declarando el atributo “mixed” = “true”, de la siguiente forma:

<xs:element name="carta">

<xs:complexType mixed="true">

<xs:sequence>

<xs:element name="nombre" type="xs:string"/>

<xs:element name="id\_pedido" type="xs:positiveInteger"/>

<xs:element name="fecha\_envio" type="xs:date"/>

</xs:sequence>

</xs:complexType>

</xs:element>

De esta forma, en un fichero XML, podríamos tener un elemento carta como el que figura a continuación:

Texto

Descripción generada automáticamente

## 3.4.- Tipos de datos complejos.

Existen como forma de construir tipos estructurados, partiendo de la base de los tipos simples. Es decir, **los tipos complejos se refieren a elementos que se componen de otros elementos**. Por otra parte, son los que nos permiten incluir atributos en los elementos.

Para definirlos usamos la etiqueta *<xs:complexType>.* Por ejemplo, un elemento <correo>, se compone de varios sub-elementos:

<xs:element name="correo">

<xs:complexType>

...

</xs:complexType>

</xs:element>

Para especificar cómo se realiza esa composición, **existen varias alternativas**:

|  |  |
| --- | --- |
| Tipos de composiciones | Explicación |
| xs:sequence | Define que los elementos englobados en ella han de aparecer en ese orden. |
| **Ejemplo**: Si entendemos que el correo se compone de “para”, “de”, “asunto” y “cuerpo” (en ese orden), el aspecto del XSD sería: |
| <xs:element name="correo">  <xs:complexType>  <xs:sequence>  <xs:element name="para" type="xs:string"/>  <xs:element name="de" type="xs:string"/>  <xs:element name="asunto" type="xs:string"/>  <xs:element name="cuerpo" type="xs:string"/>  </xs:sequence>  </xs:complexType  </xs:element> |
| xs:all | **Secuencias no ordenadas**. Representa secuencias, como las anteriores, salvo por el hecho de que no importa en qué orden aparezcan los elementos contenidos en ella. |
| xs:choice | **Representa alternativas**, hay que tener en cuenta que es una o-exclusiva. Es decir, **de entre los elementos incluidos sólo será válido que aparezca uno de ellos**. |

## 3.5.- Asociación con documentos XML.

Vamos a ver como un fichero “test.xsd” se encargará de validar a otro “test.xml”. En primer lugar, debemos asegurarnos de tener el plugin “XML” de RedHat, así como el plugin “RedHat Commons”.

En segundo lugar, **creamos el archivo XSD**, con el siguiente contenido, guardándolo en una ubicación a nuestra conveniencia, con el nombre “test.xsd”:

<?xml version="1.0" encoding="UTF-8"?>

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

    <xs:element name="carta" type="xs:string">

    </xs:element>

</xs:schema>

Ahora, creamos el documento a validar. Lo llamaremos “test.xml” y tendrá este contenido, de forma que quedará vinculado al fichero de XML Schema que lo validará automáticamente:



Aunque, para vincularlo con archivos validadores que estén en nuestro sistema (en lugar de en la red), es mejor utilizar **xsi:noNamespaceSchemaLocation**.

Con esto, y los plugins de Visual Studio Code, ya podremos validar nuestro fichero .xml con nuestro fichero validador de forma totalmente automática.

## 3.6.- Aclaración.

* **Simple type**: elementos que solo tienen texto en su interior (contenido), o bien, para atributos.
* **Complex type**: elementos que contienen atributos y/o subelementos (hijos).
  + Por defecto, no pueden tener texto (contenido) en su interior.
  + Para poder poner contenido en su interior, declarar el complexType así:
    - <xs:complexType mixed="true">