XML con JS

Antonio Espín Herranz

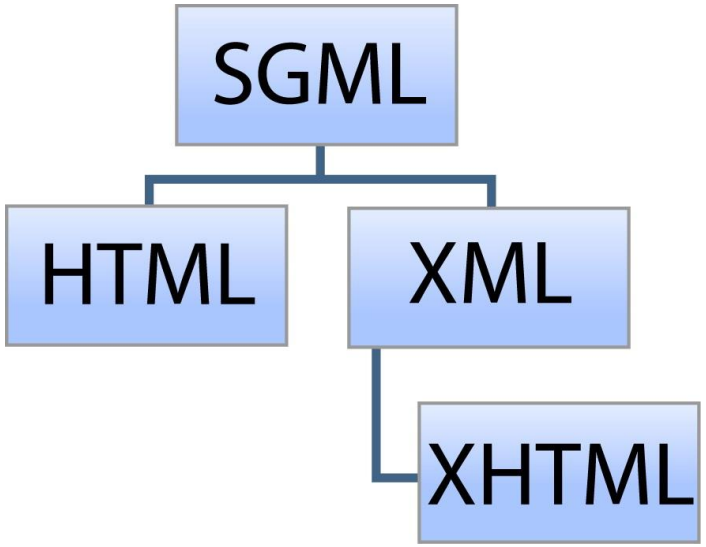
Contenidos

- Introducción al lenguaje XML
- Lenguaje XML
- Validación con DTDs
- NameSpaces
- Validación con XML Schema

Introducción XML

Introducción

- Todos los lenguajes de marcas heredan del lenguaje **SGML** (Lenguaje de Marcado Generalizado Estándar).



Introducción

- A partir del lenguaje XML surgieron multitud de lenguajes derivados de este.

- Necesidad de procesar programáticamente estos documentos lo que da lugar a distintas APIs como pueden ser SAX y DOM.
  - **SAX**: Simple API for XML.
  - **DOM**: Document Object Model.
  - Distintos lenguajes de programación implementan las APIs: Java, PHP, Visual Basic, etc.

Introducción

- Integración de distintas plataformas mediante la tecnología XML → SOA.

- SOA: Arquitectura Orientada a Servicio.
- Posibilita la comunicación entre máquinas sin necesidad del humano integrando distintas tecnologías y plataformas (**Web Services**).

Introducción

- El lenguaje XML y sus derivados está estandarizado y forman parte del **W3C**.
- **World Wide Web Consortium**:
  - Consorcio a nivel internacional que desarrolla estándares relacionados con la Web (recomendaciones). Desde OCT 1994.
- Direcciones de interés:
  - http://www.w3c.es/
  - http://www.w3.org

Introducción

- Otras necesidades que van surgiendo dentro del lenguaje XML (a parte de almacenar información) debido al gran volumen de los documentos.
  - **Localizar información**: Hacer búsquedas dentro de los documentos.
  - **Formatear documentos**: Aplicar estilos a los documentos. Transformar en otros documentos.
  - **Validación de documentos**: Si queremos comunicar aplicaciones entre si, todas tienen que “hablar el mismo lenguaje” .

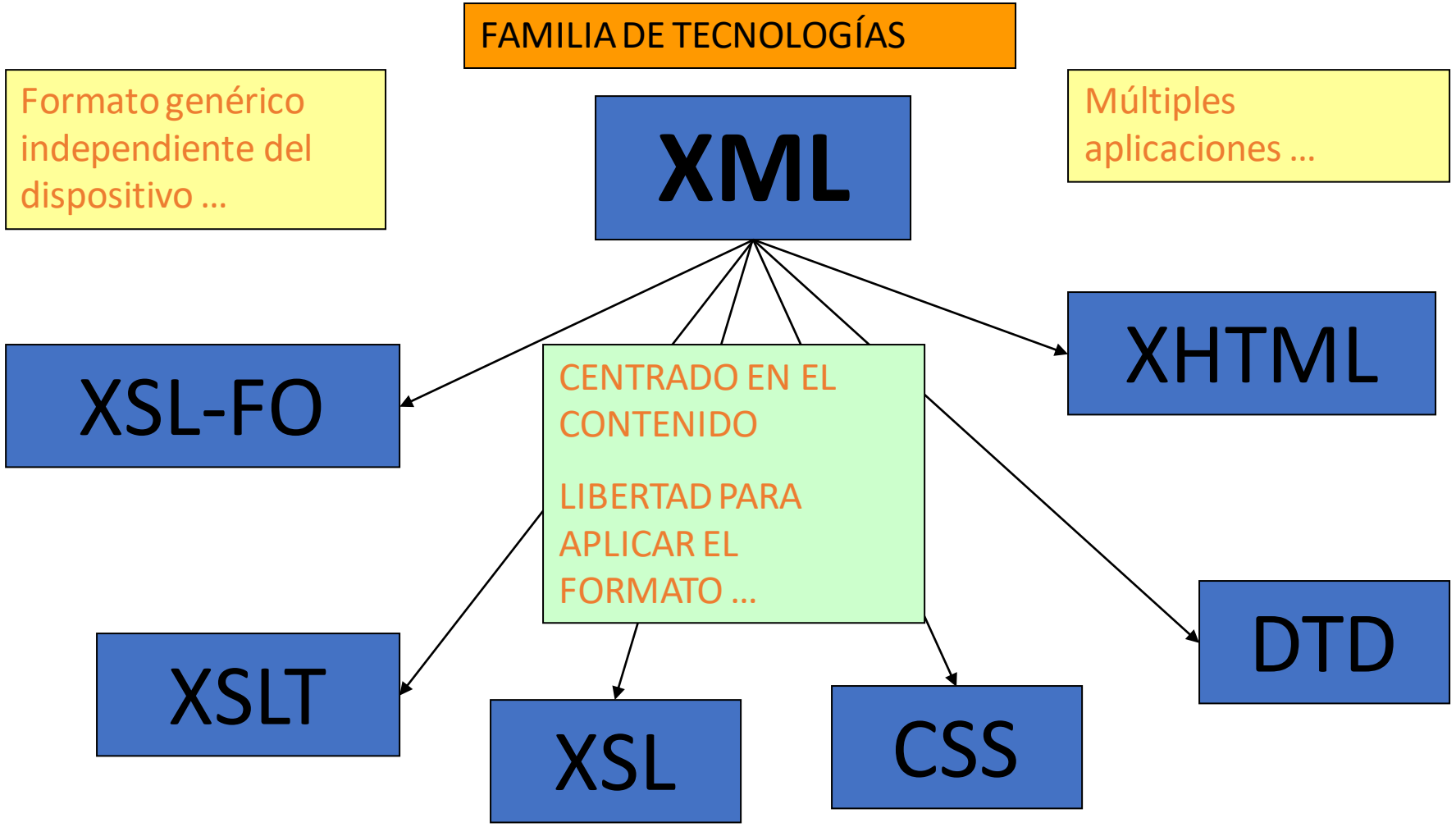
Introducción

- **Validación**: Estructura del documento, orden y colocación de la etiquetas, atributos, etc.
  - **XSD**, **DTD**.
- **Transformaciones**: Convertir documentos XML en otro tipo de documentos aplicando formatos y estilos.
  - **XSLT, XSL-FO**.
- **Localizar Información**:
  - **XPath, XQuery**.

OTROS VOCABULARIOS XML

- **XHTML**: Versión de HTML adaptada a XML.
- **WML** (Wireless markup language): Teléfonos móviles.
- **SVG** (Scalable Vector Graphics): Gráficos Vectoriales.
- **XSL**: Hojas de estilo para documentos XML
  - Formado por XPath, XSLT y XSL-FO.
- **SMIL**: Multimedia.
- **VoiceXML**: Portales de Voz.
- **MathML**: Fórmulas Matemáticas.
- **X3D**: Representaciones tridimensionales.

LENGUAJES XML



Lenguaje XML

DOCUMENTOS XML

- Se centra en el contenido de información.
- La idea era diseñar un tipo de documento independiente del formato y de la plataforma así como del dispositivo.
- Con una sintaxis similar al HTML.
- Se componen de marcas diseñadas por nosotros mismos.
- Tienen una sintaxis mas estricta que HTML.

COMPARACIÓN CON HTML

```
<html>
<head>
<title>Poema</title>
</head>
<body>
<h1>Alba</h1>
<h2>Abril de 1915 </h2>
<h2><i>Granada</i></h2>
<p>Mi corazón oprimido</p>
<p>siente junto a la alborada</p>
<p>el dolor de sus amores</p>
<p>y el sueño de las distancias.</p>
</body>
</html>
```

```
<poema fecha="Abril de 1915"
lugar="Granada">
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>siente junto a la alborada</verso>
<verso>el dolor de sus amores</verso>
<verso>y el sueño de las distancias. </verso>
</poema>
```

CARACTERÍSTICAS

- Utilizable en Internet.
- Soporte a gran variedad de aplicaciones.
- Compatible con **SGML** (Standard Generalized Markup Language).
- Debe ser fácil escribir programas que procesen XML.
- Número de características opcionales = Mínimo.
- Documentos legibles por personas.
- El diseño de XML debe poder hacerse rápidamente.
- El diseño de XML debe ser formal y conciso.
- La creación de documentos XML debe ser fácil.
- La concisión de las marcas XML no tiene importancia (es preferible la claridad a la brevedad).

ESTRUCTURA

<?xml version="1.0"?> ***Declaración de XML.***

<!DOCTYPE raiz [
....
]>

Declaración DTD opcional

<raiz>
  <elemento>
    ...
  </elemento>
</raiz>

Documento

SINTAXIS

Declaración de XML:

<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>

- **version**: Actual = 1.0
  - Borrador de versión 1.1
  - Mayor compatibilidad con Unicode
- Identificadores: Permite cualquier carácter Unicode:
  - **encoding**: UTF-8, UTF-16, iso-8859-1, etc.
- **standalone**: Indica si el documento no hace referencias a entidades externas.

Ejemplos declaración

- <?xml version='1.0' ?>
- <?xml version='1.0' encoding='US-ASCII' ?>
- <?xml version='1.0' encoding='US-ASCII' standalone='yes' ?>
- <?xml version='1.0' encoding='UTF-8' ?>
- <?xml version='1.0' encoding='UTF-16' ?>
- <?xml version='1.0' encoding='ISO-10646-UCS-2' ?>
- <?xml version='1.0' encoding='ISO-8859-1' ?>
- <?xml version='1.0' encoding='Shift-JIS' ?>

SINTAXIS

- Los documentos consisten en una serie de datos marcados mediante etiquetas.
- Las etiquetas describen la estructura del documento.
- En los documentos XML se distingue entre mayúsculas y minúsculas.
- Un elemento = grupo formado por etiqueta inicial, etiqueta final y contenido entre ambas.
- La etiqueta inicial puede incluir atributos.

  <etiqueta attributo="valor">……</etiqueta>

  <etiqueta attributo="valor"></etiqueta>
- Elemento vacío: Entre la etiqueta inicial y final no hay información.

  <etiqueta atributo="valor"/>

SINTAXIS

- Se pueden anidar elementos

  <externo>
  <interno>texto</interno>
  </externo>

- ...pero no se pueden entrelazar:

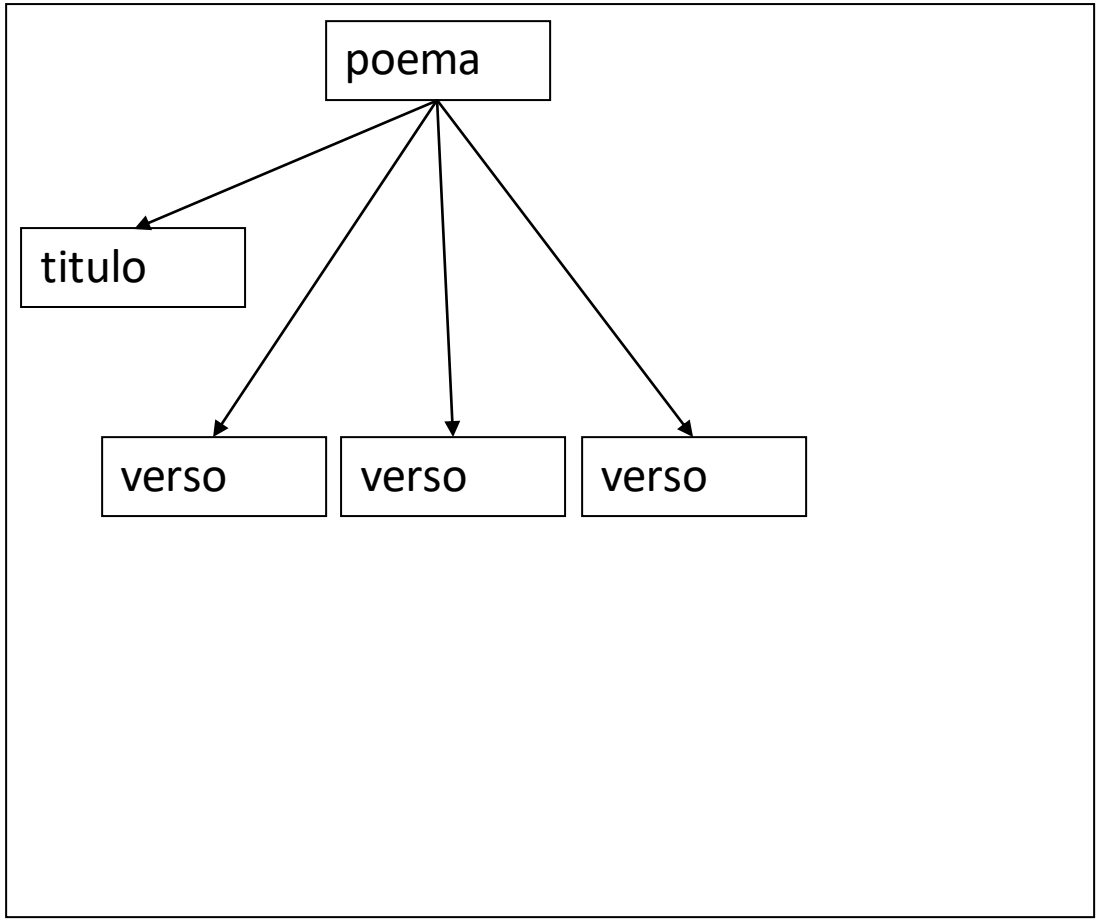
  <externo>
  <interno>texto</externo>
  </interno>

SINTAXIS

- Sólo puede haber un único elemento raíz.

- Cada documento XML equivale a un árbol.

<poema fecha=" Abril de 1915 "
lugar=" Granada">
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>siente junto a la alborada</verso>
<verso>el dolor de sus amores</verso>
<verso>y el sueño de las distancias. </verso>
</poema>



SINTAXIS

- Cada elemento puede contener atributos en la etiqueta inicial

  <pizza nombre="Margarita" precio="6">

  . . .

  </pizza>

- El orden de los atributos no es significativo.

- No puede haber 2 atributos con el mismo nombre.

- Atributos predefinidos:
  - **xml:lang:** Especifica el idioma.
    - Por ejemplo: en (inglés), sp (español)
  - **xml:space:** Especifica cómo tratar el espacio en blanco.
    - Valores:
    - **preserve** = Mantenerlo
    - **default** = Permitir a la aplicación que lo trate como quiera.

SINTAXIS

- Comentarios:

  <!- - el texto de un comentario no es analizado - - >

- El código debe ir encerrado en secciones CDATA:

```
<![CDATA [
        if x < 3 && x > 4 then
          print “Hola”
 ]]>
```

- Caracteres especiales: No pueden incluirse directamente:

  < <
  > >
  " “
  ' ‘
  & &

  Este código no es correcto:

```
<codigo>
        if x &lt; 4 then x:=x + 1;
</codigo>
```

SINTAXIS

- Instrucciones de procesamiento:
- Es posible incluir instrucciones que indican al procesador alguna acción a realizar
  <?aplicacion datos ?>

- De echo la declaración del documento es una instrucción de procesamiento:
  <?xml version=“1.0” ?>

- También se pueden adjuntar hojas de estilos al documento:
  <?xml-stylesheet type="text/xsl" href=“hoja.xsl"?>

DOCUMENTO BIEN FORMADO

- Sigue las reglas sintácticas.

- Importante:
  - Contiene un único elemento raíz.
  - Todas las etiquetas están correctamente anidadas.

- Ejemplo:

```
<pizzas>
<pizza nombre="Margarita" precio="6">
<ingrediente nombre="Tomate" />
<ingrediente nombre="Queso" />
</pizza>
</pizzas>
```

VENTAJAS DEL XML

- Es un formato estructurado
- Contiene información y meta-información
- Ha sido diseñado específicamente para Internet
- Soportado por visualizadores y servidores
- Numerosas herramientas de procesamiento
- Legible por personas humanas
- Admite la definición de vocabularios específicos
- Separa contenido del procesamiento y visualización
- Aumenta la seguridad mediante la validación de documentos
- Formato abierto, respaldado por numerosas organizaciones
- Una vez definido un DTD común, facilita intercambio de información.

INCONVENIENTES DEL XML

- Puede requerir demasiado espacio, ancho de banda y tiempo de procesamiento.
- Documentos largos con mucha información redundante.
- Es una sintaxis de documentos, no un lenguaje de programación.
- Es posible crear formatos y vocabularios propietarios.
- Puede fomentar la proliferación de vocabularios específicos.
- Poco eficiente como lenguaje de almacenamiento de bases de datos.
- Bueno para texto, malo para datos binarios.

VALIDACIÓN MEDIANTE DTDs

DTDs: VALIDACIÓN DE DOCUMENTOS

- Una vez que los documentos XML están correctos, sintácticamente hablando.
- El siguiente paso sería comprobar su estructura, para ello lo vamos a validar con el DTD.
- Definición del Tipo del Documento. Para incluirlo en el XML: **<!DOCTYPE raiz SYSTEM "poema.dtd">**
- **Documento válido**
  - Está bien formado y
  - La estructura encaja con la declaración del tipo de documento

EJEMPLO

```
<?xml version="1.0"?>
<!DOCTYPE poema SYSTEM "poema.dtd">
<poema fecha=" Abril de 1915 "
lugar=" Granada" >
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
<verso>siente junto a la alborada</verso>
<verso>el dolor de sus amores</verso>
<verso>y el sueño de las distancias. </verso>
</poema>
```

```
<!ELEMENT poema (titulo,autor?,verso+)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT autor(#PCDATA)>
<!ELEMENT verso (#PCDATA)>
<!ATTLIST poema fecha CDATA #REQUIRED
lugar CDATA #IMPLIED>
```

DTDs

- La definición del tipo del documento puede ser de dos formas:
  - Interna: Se detalla dentro del propio fichero XML.
  - Externa:
    - Privada: En un fichero propio.
    - Pública: Haciendo referencia a una organización.

DTDs

DTD INTERNO

```
<?xml version="1.0"?>
<!DOCTYPE poema [
<!ELEMENT poema (titulo, autor?,
verso+)>
. . .
]>
<poema>. . . </poema>
```

DTD EXTERNO

SYSTEM (DTDs de ámbito local), URL o File

```
<?xml version="1.0"?>
<!DOCTYPE poemaSYSTEM
"http://www.poemas.org/poema.dtd">
<poema>
. . .
</poema>
```

PUBLIC (DTDs compartidos por diversas organizaciones)

```
<?xml version="1.0"?>
<!DOCTYPE HTML PUBLIC
"-//W3C//DTD HTML 4.0//EN"
"http://www.w3c.org/TR/REC-
html/strict.dtd">
```

Ejemplo declaración interna

```
<!DOCTYPE person [
    <!-- internal subset -->
    <!ELEMENT person (name, age)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT age (#PCDATA)>
]>
<person>
    <name>Billy Bob</name>
    <age>33</age>
</person>
```

TIPOS DE DECLARACIONES

- ELEMENT
  - Elementos del documento XML.
- ATTLIST
  - Lista de atributos de un elemento.
- ENTITY
  - Entidades (variables o macros).
- NOTATION
  - Definen tipos de contenidos.
  - Facilitan la inclusión de formatos binarios (imágenes, vídeos, sonidos, ...).

ELEMENTOS

- (?) = 0, 1 elemento.
- (*) = 0 ó más elementos.
- (+) = 1 ó más elementos.
- (|) = alternativa.
- (,) = secuencia.
- EMPTY = vacío.
- ANY = cualquier estructura de subelementos.
- #PCDATA = cadena de caracteres analizados.

```
<!ELEMENTpoema
  (titulo,autor?,verso+)>
<!ELEMENT publicacion
  (poema | novela | ensayo) >
<!ELEMENT autor EMPTY>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT sección (título,
  (contenido | sección+))>
<!ELEMENT p (#PCDATA | a |
  ul | em )* >
```

RECURSIVIDAD

PCDATA = Parsed Character Data
Indica que los datos son analizados buscando etiquetas

ATRIBUTOS

- Tipos de datos
  - CDATA = Cadena de caracteres.
  - NMTOKEN = Palabra (sin espacios).
  - NMTOKENS = Lista de palabras.
  - Enumeración separada por |
  - ID = Nombre único (sin duplicados)
  - IDREF = Su valor debe apuntar a un ID.
- Valor de los Atributos
  - #REQUIRED Obligatorio
  - #IMPLIED Opcional
  - #FIXED Constante
  - Valor Valor por defecto.

EJEMPLO

```
<!ATTLIST poema fecha CDATA #REQUIRED
lugar CDATA #IMPLIED>
<!ATTLIST precio moneda (euros|dólares)
   #REQUIRED
valor CDATA #REQUIRED>
<!ATTLIST persona código ID #REQUIRED>
<!ATTLIST autor código IDREF #REQUIRED>
<!ATTLIST enEstantería (sí|no) “sí” >
<!ATTLIST impuesto tipo CDATA #FIXED “IVA”>
```

```
<poema lugar=“Oviedo” fecha="2004">
<precio moneda=“euros” valor=“20” />
<autor código="35" />
</poema>
<persona código=“23” nombre =“Juan” />
<persona código=“35” nombre =“Pepe” />
<persona código=“37” nombre =“Luis” />
<impuesto tipo=“IVA” />
```

Ejemplo 2

```
<!-- person.dtd -->
<!ELEMENT person (name, age, children?)>
<!ELEMENT name (fname, (mi|mname)?, lname)?>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT mi (#PCDATA)>
<!ELEMENT mname (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT children (person*)>
```

```
<!-- person.xml -->
<!DOCTYPE person SYSTEM "person.dtd">
<person>
    <name>
        <fname>Billy</fname>
        <lname>Smith</lname>
    </name>
    <age>43</age>
    <children>
        <person>
            <name/>
            <age>0.1</age>
        </person>
        <person>
            <name>
                <fname>Jill</fname>
                <mi>J</mi>
                <lname>Smith</lname>
            </name>
            <age>21</age>
        </person>
    </children>
</person>
```

Contenido mixed

<!-- p.dtd -->

```
<!ELEMENT p (#PCDATA | b | i)*>
<!ELEMENT b (#PCDATA)>
<!ELEMENT i (#PCDATA)>
```

<!-- p.xml -->

```
<!DOCTYPE p SYSTEM "p.dtd">
<p>This <i>is</i> an <b>example</b> of <i>mixed</i>
<i>content</i><b>!</b></p>
```

Ej. Atributos

```
<!-- emp.dtd -->
<!ELEMENT employees
   (employee*)>
<!ELEMENT employee (#PCDATA)>
<!ATTLIST employee
name CDATA #REQUIRED

species NMTOKEN #FIXED "human“

id ID #REQUIRED

<!--  Referencia a un id -- >
mgr IDREF #IMPLIED
<!– Puede referenciar a varios -- >
manage IDREFS #IMPLIED>
```

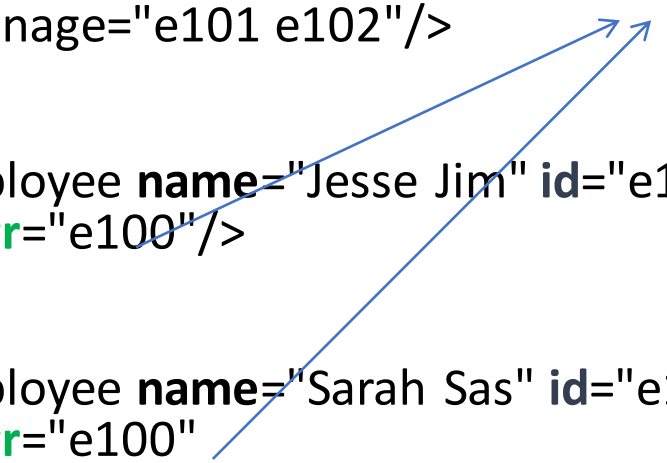
```
<!-- emp.xml -->
<!DOCTYPE employees SYSTEM
   "emp.dtd">
<employees>
<employee name="Billy Bob" id="e100"
   manage="e101 e102"/>

<employee name="Jesse Jim" id="e101"
   mgr="e100"/>

<employee name="Sarah Sas" id="e102"
   mgr="e100"
manage="e103" species="human"/>

<employee name="Nikki Nak" id="e103"
   mgr="e102"/>

<employee name="Peter Pan" id="e104"/>
</employees>
```



Ej. Atributos 2

```
<!-- emp.dtd -->
<!ELEMENT employee (address)>
<!-- NMTOKEN enumeration -->
<!ATTLIST employee
          title (president|vice-pres|secretary|sales)
          #REQUIRED>
<!ELEMENT address (#PCDATA)>
<!-- NOTATION enumeration -->
<!ATTLIST address
          format NOTATION (cs|lf)| "cs">
<!NOTATION cs PUBLIC "urn:addresses:comma-separated">
<!NOTATION lf PUBLIC "urn:addresses:line-breaks">
```

```
<!-- emp.xml -->
<!DOCTYPE employee SYSTEM "emp.dtd">
<employee title='vice-pres'>
  <!-- notation informs consuming application how to
       process element content -->
  <address format='cs'>1927 N 52 E, Layton, UT, 84041
    </address>
</employee>
```

ENTIDADES GENERALES

- **Entidades**: Asignan nombres a ciertos elementos (similar a variables)
  - Se denotan por **&entidad;**
  - No se admite recursividad.

- **Ejemplo**:

```
<!ENTITY pepe “Jose Luis López López”>
<!ENTITY humanidades “<lugar>Biblioteca de Humanidades</lugar>” >
```

- **Uso**:

```
<poema autor=”&pepe;" >&humanidades;
</poema>
```

- **Resultado**:

```
<poema nombre=” Jose Luis López López" >
   <lugar>Biblioteca de Humanidades</lugar>
</poema>
```

ENTIDADES NUMERICAS Y PREDEFINIDAS

- Entidades numéricas: Código numérico del carácter.

  ∀ ∃

  ∀ ∃

- Entidades predefinidas: Permite incluir etiquetas sin analizar:

  < < " “ ' ‘

  > > & &

ENTIDADES EXTERNAS

- P1.XML

```
<poema fecha="1915"
   lugar="Granada">
<titulo>Alba</titulo>
<verso>Mi corazón oprimido</verso>
…
<verso>de las distancias.</verso>
</poema>
```

- P2.XML

```
<poema fecha="l920" lugar="Zujaira">
<titulo>CANTOS NUEVOS</titulo>
<verso>Dice la tarde</verso>
…
<verso>y suspira el viento.</verso>
</poema>
```

```
<!DOCTYPE poemas [
<!ENTITY p1 SYSTEM "p1.xml">
<!ENTITY p2 SYSTEM "p2.xml">]>
<poemas>
&p1;
&p2;
</poemas>
```

```
<poemas>
    <poema fecha="1915" lugar="Granada">
    <titulo>Alba</titulo>
    <verso>Mi corazón oprimido</verso>
    …
    <verso>de las distancias.</verso>
    </poema>
    <poema fecha="l920" lugar="Zujaira">
    <titulo>CANTOS NUEVOS</titulo>
    …
    </poema>
<poemas>
```

ENTIDADES EXTERNAS NO ANALIZABLES

- El contenido de los ficheros es analizado (deben seguir sintaxis XML).
- Para incluir ficheros externos sin analizar se utiliza NDATA (Notation Data).
- Aplicaciones: Incluir formatos binarios.
- Mediante NOTATION se puede indicar qué aplicación se hará cargo de dichas notaciones.

```
<!NOTATION gif SYSTEM “gifEditor.exe”>
<!ENTITY dibujo SYSTEM “logotipo.gif” NDATA gif>
```

ENTIDADES PARAMETRO

- Permiten dar nombres a partes de un DTD.

- Se denotan por %entidad;

- Ejemplo:

  <!ELEMENT establecimiento (nombre,dueño?,**calle,número?,ciudad,país,códigoPostal**) >

  <!ELEMENT persona (dni, nombre, **calle,número?,ciudad,país,códigoPostal**) >

- Definición:

  <!ENTITY % localización “calle,número?,ciudad,país,códigoPostal” >
  <!ELEMENT establecimiento (nombre,dueño?,**%localización;**)>
  <!ELEMENT persona (dni, nombre, **%localización;**)>

ENTIDADES EXTERNAS

- Entidades externas: Permiten incluir elementos externos en una DTD.
- Aplicación: Dividir la definición de una DTD en varios documentos.

```
<!ENTITY %persona SYSTEM “persona.dtd”>
<!ENTITY %establecimiento SYSTEM
        “establecimiento.dtd”>
        %persona;
        %establecimiento;
```

FACTURA, PELICULA, FORO

Ejemplo Entidad Interna

```
<!DOCTYPE person [
  <!ELEMENT person (name)>
  <!ENTITY % nameDecl "<!ELEMENT name (#PCDATA)>">
  <!-- parameter entity expands to
       complete declaration -->
  %nameDecl;
]>
<person><name>Billy Bob</name></person>
```

Ejemplo Entidad Externa

```
<!-- person.dtd -->
<!ENTITY % person-content "name, age">
<!ELEMENT person (%person-content;)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
```

```
<!-- person1.xml -->
<!DOCTYPE person SYSTEM "person.dtd">
```

```
<person>
  <name>Billy Bob</name>
  <age>33</age>
</person>
```

Ejemplo Entidad Externa

```
<!-- person2.xml -->
<!DOCTYPE person SYSTEM "person.dtd" [
  <!-- change person's content model -->
  <!ENTITY % person-content "age, name">
]>
<person>
  <age>33</age>
  <name>Billy Bob</name>
</person>
```

Ejemplos DTD con prefijos I

```
<!-- person.dtd -->
<!ENTITY % prefix "p">
<!ENTITY % personName "%prefix;:person">
<!ENTITY % nameName "%prefix;:name">
<!ENTITY % ageName "%prefix;:age">
<!ENTITY % xmlnsPerson "xmlns:%prefix;">
<!ELEMENT %personName; (%nameName;, %ageName;)>
<!ATTLIST %personName;
          %xmlnsPerson; CDATA #REQUIRED>
<!ELEMENT %nameName; (#PCDATA)>
<!ELEMENT %ageName; (#PCDATA)>
```

Ejemplo DTD con prefijos II

```
<!-- person1.xml -->
<!DOCTYPE p:person SYSTEM "person.dtd">
<p:person xmlns:p='urn:person:demo'>
  <p:name>Billy Bob</p:name>
  <p:age>33</p:age>
</p:person>
```

```
<!-- person2.xml -->
<!DOCTYPE x:person SYSTEM "person.dtd" [
  <!-- override the prefix to be 'x' -->
```

```
  <!ENTITY % prefix "x">
]>
<x:person xmlns:x='urn:person:demo'>
  <x:name>Billy Bob</x:name>
  <x:age>33</x:age>
</x:person>
```

XML NameSpace

Contenidos

- ¿Qué son los namespaces en XML?
- ¿Porqué definir namespace?
- Declarar Namespaces

Namespaces

- XML propone namespaces para distinguir elementos que coinciden en la misma etiqueta pero en distintos niveles.

- Por ejemplo:

```
<?xml version="1.0"?>
<person>
   <name>
      <title>Sir</title>
   </name>
   <htmlFormat>
      <title>Movie</title>
   </htmlFormat>
</person>
```

- El nombre de la persona y htmlFormat tienen la misma etiqueta "**title**".
- Una solución NO buena sería cambiar la etiqueta title y hacer: titlePersona y titleHTMLFormat. XML propone **Nombres cualificados**.

Nombres cualificados

- Podemos utilizar nombres cualificados para hacer estas distinciones.

  <pers:person xmlns:pers="http://www.ingenieria-informatica.es"/>

- xmlns significa Espacio de nombres XML.
- **pers** es el prefijo de espacio de nombres.
- http://www.ingenieria-informatica.es es el URI del espacio de nombres.
- Prefijo de sí mismo (**pers**) no tiene ningún sentido - su único propósito es apuntar al nombre de espacio de nombres.
- Se necesita el prefijo en ambos las etiquetas iniciales y finales etiquetas de los elementos.
- Los elementos ya no están simplemente siendo identificados por sus nombres, sino por sus **QNames**.
- Este prefijo se puede utilizar para cualquier descendientes de los <pers: Persona> elemento para denotar que también pertenecen al espacio de nombres http://www.ingenieria-informatica.es .

Tipos de URIs

- **URI** (Uniform Resource Identifier) es un string que identifica un recurso. Puede ser una URL o una URN.
  - **URL** (Uniform Resource Locator)
  - **URN** (Universal Resource Name). Ejemplo: urn:foo:a123

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.java2s.com"
 xmlns="http://www.java2s.com"
 elementFormDefault="qualified“>
 <xsd:element name="name" type="xsd:string" />
 <xsd:element name="source" type="xsd:string" />
</xsd:schema>
```

Nombres cualificados

- <pers:person xmlns:pers="http://www.ingenieria-informatica.com">
  <pers:name>
  <pers:title>Sir</pers:title>
  </pers:name>
  </pers:person>

- **No es necesario declarar todos los namespace en el elemento raíz, se puede definir en el elementos secundarios**.
- <person xmlns="http://www.java2s.com">
  <name/>
  <xhtml:p xmlns:xhtml="http://www.w3.org/1999/xhtml">
  This is XHTML
  </xhtml:p>
  </person>

Alcance

```
<person xmlns="http://www.java2s.com">
    <name/>
    <paragraph xmlns="http://www.w3.org/1999/xhtml">
        This is XHTML
    </paragraph>
</person>
```

http://www.java2s.com

- Es el namespace por defecto para todo el documento.

http://www.w3.org/1999/xhtml

- Es el namespace por defecto para el elemento <**paragraph**> y cualquiera de sus descendentes. El namespace http://www.w3.org/1999/xhtml sobreescribe al primer namespace http://www.java2s.com.

Prefijos

- URI deben ser utilizados para los nombres de prefijo.
- Un URI (Uniform Resource Identifier) es una cadena de caracteres que identifica un recurso.
- Puede ser URL (Uniform Resource Locator) o URN (Universal Resource Name).
- La URL que estamos utilizando se utiliza simplemente como un nombre para el espacio de nombres.
- Analizador XML no tratará descargar todos los recursos de ese lugar.
- Analizador XML lo utiliza para dar nombre a los espacios de nombres en el documento.

```
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>Book List</title>
 </head>
 <body>
  <pub:publications
  xmlns:pub="http://www.java2s.com/namespaces/pub">
   <pub:book>
    <pub:title>Mastering XHTML</pub:title>
    <pub:author>Ed Tittel</pub:author>
   </pub:book>
   <pub:book>
    <pub:title>Java Developer's Guide to E-Commerce
    with XML and JSP</pub:title>
    <pub:author>William Brogden</pub:author>
   </pub:book>
  </pub:publications>
 </body>
</html>
```

Formas de especificar Namespaces

- Dentro de un fichero XML.
- Dentro del fichero DTD.

XML File

- Se puede especificar un Namespace por cada elemento .

```
<lower:aaa xmlns:lower = “http://website/lowercase”>
   <lower:bbb xmlns:lower = “http://website/lowercase”>
         <lower:ccc xmlns:lower = “http://website/lowercase” />
   </lower:bbb>
   <upper:BBB xmlns:upper = “http://website/uppercase” >
         <upper:CCC xmlns:upper = “http://website/uppercase” />
   </upper:BBB>
</lower:aaa>
```

XML File

- Especificar todos los namespace dentro del elemento raíz.
- La especificación también sería válida para todos los elementos que se dan dentro de este.

```
<lower:aaa xmlns:lower = “http://website/lowercase” xmlns:upper = “http://website/uppercase”>
   <lower:bbb >
         <lower:ccc />
   </lower:bbb>
   <upper:BBB >
         <upper:CCC />
   </upper:BBB>
</lower:aaa>
```

XML File

- Cuando declaramos un namespace no tiene que llevar necesariamente un prefijo.

- El atributo **xmlns** define por defecto el namespace que se usa en el elemento.

```
<aaa >
   <bbb xmlns = "http://website/lowercase“>
            <ccc />
   </bbb>
   <BBB xmlns = "http://website/uppercase" >
            <CCC />
   </BBB>
</aaa>
```

XML File

- A los atributos se les puede asignar explicítamente un valor asociado con un namespace usando diferentes prefijos.

```
<lower:aaa xmlns:lower = “http://website/lowercase”
   xmlns:upper = “http://website/uppercase”>
   <lower:bbb lower:zz = “11” >
         <lower:ccc upper:VV = “22” />
   </lower:bbb>
   <upper:BBB lower:sss = “***” />
</lower:aaa>
```

XML File

- Los atributos sin prefijos no pertenecen a ningún namespace.

```
<lower:aaa xmlns:lower = “http://website/lowercase” xmlns:upper =
  “http://website/uppercase”>
        <lower:bbb zz = “11” >
            <lower:ccc WW = “22” />
        </lower:bbb>
        <upper:BBB sss = “***” />
</lower:aaa>
```

XML File

- Una especificación de namespace se puede sobreescribir usando los prefijos.

```
<lower:aaa xmlns:lower="http://website/lowercase">
  <lower:bbb>
        <lower:ccc xmlns:lower="http://website/uppercase">
              <lower:ddd>It’s uppercase now.</lower:ddd>
        </lower:ccc>
  </lower:bbb>
</lower:aaa>
```

DTD File

- Ejemplo de especificación de namespace dentro de un DTD.

<!ELEMENT cars>

<!ATTLIST cars

xmlns:part CDATA #FIXED “http://www.w3.org/1999/cars”>

XML Schema

XML - Schema

- El propósito del estándar XML Schema es definir la ***estructura*** de los documentos XML que estén asignados a tal esquema y los ***tipos de datos*** válidos para cada elemento y atributo.

- En este sentido las posibilidades de control sobre la estructura y los tipos de datos son muy amplias. Mas efectivo que las **DTDs**.

- Al restringir el contenido de los ficheros XML es posible ***intercambiar*** información entre aplicaciones con gran seguridad. Disminuye el trabajo de comprobar la estructura de los ficheros y el tipo de los datos.

- XML Schema tiene un ***enfoque modular*** que recuerda a la programación orientada a objetos y que facilita la reutilización de código.

Introducción a XSD

- Los ***tipos de datos*** tienen en XML Schema la función de las ***clases*** en la POO.

- El usuario puede construir tipos de datos a partir de tipos predefinidos, agrupando elementos y atributos de una determinada forma y con mecanismos de extensión parecidos a la ***herencia***.

- Los tipos de datos se clasifican en función de los elementos y atributos que contienen.

- Los tipos de datos en XML Schema pueden ser simples o complejos.

Introducción a XSD

- XML Schema incluye el uso de ***namespaces***.

- Los "espacios de nombres" permiten definir elementos con igual nombre dentro del mismo contexto, siempre y cuando se anteponga un prefijo al nombre del elemento. El uso de *namespaces* también evita confusiones en la reutilización de código.

- Es posible agrupar atributos, haciendo más comprensible el uso de un grupo de aspectos de varios elementos distintos, pero con denominador común, que deben ir juntos en cada uno de estos elementos.

- Los ficheros XML Schema se escriben en el propio lenguaje XML.

Tipos Simples

- Tipos simples son aquellos que no tienen ni elementos hijos ni atributos.

- Son tipos simples:
  - Tipos predefinidos de XML: string, double, boolean, etc.
  - List (lista de datos separados por espacios).
  - Union (tipo de dato derivado de la unión de tipos predefinidos).

Tipos Complejos

- Son tipos complejos aquellos que tienen elementos hijos y/o atributos.

- Pueden tener nombre o ser anónimos. Si tienen nombre pueden ser ***reutilizados*** dentro del mismo XML Schema o por otros XML Schemas.

- Es posible "mezclar" o combinar elementos y texto.

Ejemplos

XSD:

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="Libro">
          <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Título" type="xsd:string"/>
                <xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
                <xsd:element name="Editorial" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="precio" type="xsd:double"/>
          </xsd:complexType>
        </xsd:element>
    </xsd:schema>
```

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<Libro xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="libro.xsd" precio="20">
        <Título>Fundamentos de XML Schema</Título>
        <Autores>Allen Wyke</Autores>
        <Autores>Andrew Watt</Autores>
        <Editorial>Wiley</Editorial>
</Libro>
```

Ejemplo XML

```
<?xml version="1.0"?>
<persona nacimiento="1999-10-20"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ej_persona.xsd" >
  <datos>
      <nombre>Pepe</nombre>    <apellidos>Garcia</apellidos>
      <dni>25390952</dni>
  </datos>
  <comentario>buena gente...</comentario>
</persona>
```

Ejemplo XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/08/XMLSchema">
<xsd:element name="persona" type="tipoPersona"/>
<xsd:element name="comentario" type="xsd:string"/>

<xsd:complexType name="tipoPersona">
        <xsd:sequence>
                <xsd:element name="datos" type="info"/>
                <xsd:element ref="comentario" minOccurs="0"/>
         </xsd:sequence>
<xsd:attribute name="nacimiento" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="info">
<xsd:sequence>
        <xsd:element name="nombre"  type="xsd:string"/>
        <xsd:element name="apellidos"  type="xsd:string"/>
         <xsd:element name="dni"  type="xsd:string"/>
</xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Esquemas XML – elemento schema

- Los elementos utilizados en la creación de un esquema “proceden” del espacio de nombres: http://www.w3.org/2001/XMLSchema

- El elemento *schema* es el elemento raíz del documento en el que se define el esquema:

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

Esquemas XML – elementos “simples”

- Un elemento simple es un elemento que sólo puede contener texto (cualquier tipo de dato), pero no a otros elementos ni atributos
- Para definir un elemento simple, utilizamos la sintáxis:

```
<xsd:element name="xxx" type="yyy"/>
```

- Ejemplos:

```
<xsd:element name=“apellido” type="xs:string"/>
<xsd:element name=“edad" type="xs:integer"/>
<xsd:element name=“fecNac" type="xs:date"/>
```

Esquemas XML – elementos “simples”, tipos de datos

- Los tipos de datos más utilizados son:
  - xsd:string
  - xsd:decimal
  - xsd:integer
  - xsd:boolean
  - xsd:date
  - xsd:time
- Un elemento simple puede tener un valor por defecto y un valor “fijo”
- Esto se indica mediante los atributos default y fixed

```
<xsd:element name="color" type="xsd:string" default="red"/>
```

Esquemas XML – atributos (1)

- Los atributos se deben declarar de forma similar a los “elementos simples”.
- Si un elemento puede ir acompañado de atributos, el elemento se deberá declarar como un elemento “complejo”.
- Un atributo se declara de la siguiente forma:

  <xsd:attribute name="xxx" type="yyy"/>

  Ejemplo:

  <xsd:attribute name=“idioma" type="xs:string"/>
- Los atributos tienen un tipo de dato: xsd:string, xsd:decimal, xsd:integer, xsd:boolean, xsd:date, xsd:time

Esquemas XML – atributos (2)

- Los atributos pueden tener valores por defecto y valores fijos:

  <xsd:attribute name=“idioma" type="xsd:string" default=“ES"/>

- Por defecto, los atributos son opcionales.
- Para indicar que un atributo debe ser obligatorio, se debe añadir a su declaración en el esquema es atributo “use”

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

- El atributo use puede tomar el valor “optional” si el atributo no es obligatorio (opción por defecto)

Esquemas XML – facetas

- Las facetas o restricciones permiten restringir el valor que se puede dar a un elemento o atributo XML
- Mediante restricciones podemos indicar que un valor debe estar comprendido en un rango determinado, debe ser un valor de una lista de valores “cerrada”, o debe ser mayor o menor que otro valor…
- Tipos de facetas:
  - Valor comprendido en un rango.
  - El valor está restringido a un conjunto de valores posibles.
  - Restringir el valor de un elemento a una serie de caracteres.
  - Longitud de los valores de los elementos…

Esquemas XML – facetas (ej. 1)

```
<xsd:element name="age">
  <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="100"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Esquemas XML – facetas (ej. 2)

```
<xsd:element name="car">
  <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:enumeration value="Audi"/>
              <xsd:enumeration value="Golf"/>
              <xsd:enumeration value="BMW"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Esquemas XML – facetas (ej. 2, alt.)

```
<xsd:element name="car" type="carType"/>

<xsd:simpleType name="carType">
  <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Audi"/>
        <xsd:enumeration value="Golf"/>
        <xsd:enumeration value="BMW"/>
  </xsd:restriction>
</xsd:simpleType>
```

Esquemas XML – facetas (ej. 3)

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

En este ejemplo, el elemento “letter” debe tomar como valor 1 letra minúscula (sólo 1)

Esquemas XML – facetas (ej. 4)

```
<xsd:element name="initials">
  <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
        </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

En este ejemplo, el elemento “initials” debe tomar como valor 3 letras mayúsculas o minúsculas (sólo 3)

Esquemas XML – facetas (ej. 5)

```
<xsd:element name="choice">
  <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:pattern value="[xyz]"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

En este ejemplo, el elemento “choice” debe tomar como valor una de estas letras: x, y o z

Esquemas XML – facetas (ej. 6)

```
<xsd:element name="prodid">
  <xsd:simpleType>
        <xsd:restriction base="xsd:integer">
        <xsd:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
        </xsd:restriction>
</xsd:simpleType>
</xsd:element>
```

Esquemas XML – facetas (ej. 7)

```
<xsd:element name="letter">
  <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:pattern value="([a-z])*"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Esquemas XML – facetas (ej. 8)

```
<xsd:element name="password">
  <xsd:simpleType>
        <xsd:restriction base="xs:string">
              <xsd:pattern value="[a-zA-Z0-9]{8}"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- En este ejemplo, el valor del campo “password” debe ser 8 caracteres.

Esquemas XML – facetas (ej. 9)

```
<xsd:element name="password">
  <xsd:simpleType>
        <xsd:restriction base="xsd:string">
              <xsd:length value="8"/>
        </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

- Los elementos length, minLength y maxLength permiten indicar el número exacto, mínimo y máximo de caracteres que puede tener un valor de un elemento.

Elementos para restricciones

enumeration	Establece una lista de valores “aceptados”
fractionDigits	Número de cifras decimales
length	Número de caracteres obligatorios
maxExclusive y maxInclusive	Valor máximo de un rango
minExclusive y minInclusive	Valor mínimo en un rango
maxLength y minLength	Número máximo y mínimo de caracteres permitidos
pattern	Define una secuencia de caracteres permitida
totalDigits	Número exacto de dígitos permitidos
whiteSpace	Indica cómo se deben de tratar los espacios en blanco

Elementos complejos

- Son elementos que contienen a otros elementos hijos, o que tienen atributos.

- Se suelen dividir en 4 tipos:
  - Elementos vacíos con Atributos.
  - Elementos no vacíos con atributos.
  - Elementos con elementos hijos.
  - Elementos con elementos hijos y con “texto” o valor propio (como el contenido mixto de las DTD).

Elementos complejos

// Elementos vacíos con Atributos.

<product pid="1345" />

```
<xsd:element name="product">
  <xsd:complexType>
        <xsd:attribute name="pid" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

Elementos complejos

// Elementos no vacíos con Atributos.
<shoesize country="España">42</shoesize>

```
<xsd:element name="shoesize">
  <xsd:complexType>
        <xsd:simpleContent>
              <xsd:extension base="xsd:integer">
                    <xsd:attribute name="country" type="xsd:string" />
              </xsd:extension>
        </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Elementos complejos

```
// Elementos con Hijos:
<employee>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
</employee>
```

```
<xsd:element name="employee">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:element name="firstname" type="xsd:string"/>
               <xsd:element name="lastname" type="xsd:string"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

En este caso, el tipo de define de forma anónima.

Declarar elementos complejos

- Podemos usar otra sintaxis para reutilizar la “definición” de los elementos hijos en varios elementos:

```
<xsd:element name="employee" type="personinfo"/>
<xsd:element name="student" type="personinfo"/>

<xsd:complexType name="personinfo">
   <xsd:sequence>
         <xsd:element name="firstname" type="xsd:string"/>
         <xsd:element name="lastname" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

En este caso el tipo está definido.
No es anónimo.

Elementos complejos

// Elementos Mixtos:

<letter>Estimado cliente:

<name>Juan Perez</name>. Su pedido número <orderid>1032</orderid> se enviará el día <shipdate>2001-07-13</shipdate>

</letter>

```
<xsd:element name="letter">
   <xsd:complexType mixed="true">
         <xsd:sequence>
               <xsd:element name="name" type="xsd:string"/>
               <xsd:element name="orderid" type="xsd:positiveInteger"/>
               <xsd:element name="shipdate" type="xsd:date"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

Herencia

```
<xsd:element name="employee" type="fullpersoninfo"/>

<xsd:complexType name="personinfo">
   <xsd:sequence>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="fullpersoninfo">
   <xsd:complexContent>
          <xsd:extension base="personinfo">
                 <xsd:sequence>
                        <xsd:element name="address" type="xsd:string"/>
                        <xsd:element name="city" type="xsd:string"/>
                        <xsd:element name="country" type="xsd:string"/>
                 </xsd:sequence>
          </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

En la declaración de elementos complejos, es posible utilizar un mecanismo de "herencia" para reutilizar o extender elementos definidos con anterioridad.

Declarar elementos complejos: Indicadores

- En los ejemplos anteriores hemos utilizado el elemento xsd:sequence como elemento hijo del elemento xsd:complexType.
- **xsd:sequence** indica que los elementos anidados en él deben aparecer en un **orden determinado**.
- Los esquemas XML nos ofrecen otras alternativas, además de xsd:sequence, para indicar cómo se deben tratar los elementos que aparecen anidados en un elemento complejo.
- Las opciones o “indicadores” son: **xsd:all** y **xsd:choice**

Declarar elementos complejos: Indicador xsd:all

- El indicador xsd:all indica que los elementos que contiene pueden aparecer en cualquier orden, pero como máximo sólo una vez

```
<xsd:element name="person">
  <xsd:complexType>
      <xsd:all>
          <xsd:element name="firstname" type="xsd:string"/>
          <xsd:element name="lastname" type="xsd:string"/>
      </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Declarar elementos complejos: Indicador xsd:choice

- El indicador xsd:choice indica que puede aparecer sólo uno de los elementos que contiene.

```
<xsd:element name="person">
  <xsd:complexType>
        <xsd:choice>
              <xsd:element name="firstname" type="xsd:string"/>
              <xsd:element name="lastname" type="xsd:string"/>
        </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Declarar elementos complejos: Indicadores maxOccurs y minOccurs

- Estos indicadores se utilizan para indicar el número **máximo** y **mínimo** de veces que puede aparecer un elemento hijo de un elemento complejo.
- El atributo maxOccurs puede tomar el valor "**unbounded**", que indica que no existe ningún límite.

```
<xsd:element name="person">
   <xsd:complexType>
         <xsd:sequence>
         <xsd:element name="full_name" type="xsd:string"/>
         <xsd:element name="child_name" type="xsd:string" maxOccurs="10"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

El modelo de contenido ANY

- En esquemas XML también contamos con un modelo de contenido ANY, que permite incluir elementos no declarados inicialmente en el esquema

```
<xsd:element name="person">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:element name="firstname" type="xsd:string"/>
               <xsd:element name="lastname" type="xsd:string"/>
               <xsd:any minOccurs="0"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

El modelo de contenido ANY

- También contamos con un elemento que permite extender el número de atributos de un elemento:

```
<xsd:element name="person">
   <xsd:complexType>
         <xsd:sequence>
               <xsd:element name="firstname" type="xsd:string"/>
               <xsd:element name="lastname" type="xsd:string"/>
         </xsd:sequence>
         <xsd:anyAttribute/>
   </xsd:complexType>
</xsd:element>
```

Ejemplo: Fichero XML

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

<course>
        <teacher id="jp">
          <name>John Punin</name>
        </teacher>

        <student id="js">
                <name>John Smith</name>
                <hw1>30</hw1>
                <hw2>70</hw2>
                <project>80</project>
                <final>85</final>
        </student>

        <student id="gl">
                <name>George Lucas</name>
                <hw1>80</hw1>
                <hw2>90</hw2>
                <project>100</project>
                <final>40</final>
        </student>

    <! - - Mas estudiantes, un numero indeterminado pero al menos debe de haber uno -- >
</course>
```

Course:
Se compone de un elemento **teacher** y una serie de elementos **student**.

Ambos elementos son complejos.

Ejemplo: XSD desde Eclipse

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/notas"
    xmlns:tns="http://www.example.org/notas"
    elementFormDefault="qualified">


<element name="course">
<complexType>
<sequence>
      <element name="teacher" type="tns:tTeacher" />
      <element name="student" type="tns:tStudent"
          minOccurs="1" maxOccurs="unbounded"/>
</sequence>
</complexType>
</element>
```

Definición de los espacios de nombres. ***El prefijo tns → this namespace,*** *definido dentro del propio documento.*

También hacemos referencia a la definición del Schema. En este caso va sin prefijo.

Ejemplo: XSD desde Eclipse

```
<complexType name="tTeacher">
    <sequence>
        <element name="name" type="string" />
    </sequence>
    <attribute name="id" type="string" />
</complexType>

<complexType name="tStudent">
    <sequence>
        <element name="name" type="string" />
        <element name="hw1" type="float" />
        <element name="hw2" type="float" />
        <element name="project" type="float" />
        <element name="final" type="float" />
    </sequence>
    <attribute name="id" type="string" />
</complexType>
</schema>
```

Validar documentos XSD

- Validador de XSD:
  - http://www.ltg.ed.ac.uk/~ht/xsv-status.html

Enlaces

- https://browntreelabs.com/parse-xml-to-dom-tree-and-deal-with-it/