

# DTD y XML SCHEMA

Ingeniería de la Información

# Tabla de Contenidos

## Plantillas de validación de documentos XML

- DTD
  - Declaración de tipos
  - Elemento raíz
  - Elementos
  - Atributos
- XML Schema
  - Elemento raíz
  - Elementos simples
  - Elementos complejos
  - Restricciones
  - Tipos de datos

# DTD

- ✧ *DTD - Document Type Definition.*
- ✧ Define la gramática a seguir en el documento XML para que éste sea considerado como válido.
- ✧ Puede incluirse en un fichero externo al XML, y/o incluirse dentro del propio fichero XML.

## DTD. Declaración de tipo (i)

Libros1.dtd {

- <!ELEMENT Libros (Libro+)>
- <!ELEMENT Libro (Titulo, Autor)>
- <!ELEMENT Titulo (#PCDATA)>
- <!ELEMENT Autor (#PCDATA)>

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Libros SYSTEM "Libros1.dtd">
<Libros>
  <Libro>
    <Titulo>Don Quijote de la Mancha</Titulo>
    <Autor>Miguel de Cervantes</Autor>
  </Libro>
  <Libro>
    <Titulo>La vida es suenno</Titulo>
    <Autor>Calderon de la Barca</Autor>
  </Libro>
</Libros>
```

## DTD. Declaración de tipo (ii)

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE Libros [
    <!ELEMENT Libros (Libro+)>
    <!ELEMENT Libro (Titulo, Autor)>
    <!ELEMENT Titulo (#PCDATA)>
    <!ELEMENT Autor (#PCDATA)>
]>
<Libros>
    <Libro>
        <Titulo>Don Quijote de la Mancha</Titulo>
        <Autor>Miguel de Cervantes</Autor>
    </Libro>
    <Libro>
        <Titulo>La vida es sueno</Titulo>
        <Autor>Calderon de la Barca</Autor>
    </Libro>
</Libros>
```

## DTD. Elemento raíz

- ✎ Toda DTD debe tener uno y sólo un *elemento raíz* (también conocido como *elemento documento*).
- ✎ Este elemento raíz debe coincidir con el nombre que aparece a continuación del DOCTYPE.

## DTD. Contenido

- Un documento DTD puede contener:
  - Declaraciones de elementos.
  - Declaraciones de atributos para un elemento.
  - Declaraciones de entidades.
  - Declaraciones de notaciones.
  - Instrucciones de procesamiento.
  - Comentarios.
  - Referencias a entidades de parámetro.

## DTD. Elementos (i)

- ✎ A partir del elemento raíz, pueden opcionalmente colgar (de forma jerárquica) otros elementos.

```
<!ELEMENT Libros (Libro+)>  
<!ELEMENT Libro (Titulo, Autor)>  
<!ELEMENT Titulo (#PCDATA)>  
<!ELEMENT Autor (#PCDATA)>
```



## DTD. Elementos (ii)



### Contenido de un elemento:

- EMPTY: El elemento está vacío (puede contener atributos). Ej.: `<!ELEMENT IMAGEN EMPTY>`
- ANY: El elemento puede almacenar cualquier tipo de contenido. Ej.: `<!ELEMENT IMAGEN ANY>`
- Otros elementos: Un elemento puede contener uno o más elementos hijos en una cierta secuencia. Ej.: `<!ELEMENT LIBRO (TITULO, AUTOR)>`
- #PCDATA: Texto *a procesar por el parser*.  
Ej.: `<!ELEMENT LIBRO (#PCDATA)>`
- Mixto: el elemento puede incluir secuencias de caracteres opcionalmente mezcladas con elementos hijos. `<!ELEMENT LIBRO (#PCDATA | AUTOR)*>`

## DTD. Elementos (iii)

### Secuencias de hijos de un elemento:

- Secuencia:
  - Secuencia en orden: hijos separados por comas.
  - Opciones: hijos separados por | (barra)
  - Conjuntos de elementos pueden agruparse entre paréntesis.
- Cardinalidad: un elemento, o un conjunto de ellos puede repetirse 0, 1 ó más veces:
  - *elemento*      Elemento repetido 1 única vez
  - ?                  Elemento repetido 0 ó 1 vez
  - \*                  Elemento repetido 0 ó más veces
  - +                  Elemento repetido 1 ó más veces

## DTD. Elementos (iv)

<!ELEMENT LIBRO (Autor, Editorial)>

<!ELEMENT Autor (#PCDATA)>

<!ELEMENT PELICULA (Actor|Actriz|Director)+>

<!ELEMENT PELICULA ((Actor | Actriz)\*, Director, Maquillaje?)>

<!ELEMENT PELICULA (#PCDATA | Actor)\*>

<!ELEMENT PELICULA (Titulo, Genero, (Actor | Actriz | Narrador)\*)>

<!ELEMENT FICHA (Nombre+, Apellido+, Direccion\*, foto?,  
TelFijo\*|TelMovil\*)>

<!ELEMENT FICHA ((Sr | Sra | Srta)?, Nombre, Apellido\*)>

<!ELEMENT Sr EMPTY>

## DTD. Atributos (i)

- ✎ Un elemento puede opcionalmente declarar uno o más atributos
  - `<!ATTLIST Elemento Atributo Tipo Modificador>`
- ✎ Los atributos de un elemento pueden incluirse en una o más declaraciones `<!ATTLIST ...>`.
- ✎ Si se hace en la misma declaración, basta con separar con un espacio (espacio, tabulador, retorno de carro).

## DTD. Atributos (ii)



### Tipo de un atributo:

- Tipo cadena: CDATA

<!ATTLIST Autor Nacionalidad CDATA>

- Tipo enumerado:

<!ATTLIST Pelicula Genero (Ficcion | Terror | Humor)>

- Tipo simbólico:

- ID: valdrá como identificador en el resto del documento, sólo un atributo ID por cada elemento.
- IDREF, IDREFS: su valor debe coincidir con algún otro atributo de tipo ID en el resto del documento XML. IDREFS separa las referencias por espacio. Ej.: "ID1 ID2 ID3".
- ENTITY, ENTITIES: su valor debe coincidir con una o más entidades no analizadas.
- NMTOKEN, NMTOKENS: su valor ha de ser una cadena de tipo *token*. Ej.: <LIBRO ISBN="9-34532-33-81"></LIBRO>

## DTD. Atributos (iii)

### Modificadores:

- **#REQUIRED:** Este atributo debe introducirse obligatoriamente.  
Ej.: `<!ATTLIST Pelicula Titulo CDATA #REQUIRED>`
- **#IMPLIED:** Indica que el atributo es opcional.
- **ValorPredeterminado:** Si se omitiese el atributo, los procesadores recogerían este valor por omisión. Ej.:  
`<!ATTLIST Pelicula Genero (Ficcion | Terror | Humor)`  
`"Humor">`  
`<!ATTLIST Autor Nacionalidad CDATA "Espanola">`
- **#FIXED:** se incluya o no se incluya el atributo, los procesadores siempre obtendrán este mismo valor  
`<!ATTLIST Autor Nacionalidad CDATA #FIXED "Espanyola">`

## DTD. Problemas

- ✧ Una DTD no sigue el formato de un documento XML estándar.
- ✧ Esto representa un problema para los *parsers*.
- ✧ No se soportan distintos tipos de datos al estilo de los lenguajes de programación.
- ✧ No se pueden crear tipos de datos personalizados.
- ✧ No se soportan los espacios de nombres.
- ✧ El número de ocurrencias no se puede controlar al 100%.
- ✧ **Por estas y otras razones, surgen los *Schemas* (Esquemas) XML.**

## XML Schema


- ✧ XML Schema es una alternativa más potente a las DTDs.
- ✧ XML Schema permite escribir esquemas detallados para documentos XML, utilizando la sintaxis estándar de XML.
- ✧ XML Schema describe la estructura de un documento XML.
- ✧ El lenguaje XML Schema también se denomina XML Schema Definition (XSD).



## ¿Para que sirve un XML Schema?

- ✎ El objetivo de un XML Schema es definir los elementos que permiten construir un documento XML válido, igual que las DTDs.
- ✎ Un XML Schema define:
  - los elementos que pueden aparecer en un documentos
  - atributos que pueden aparecer en un documento
  - qué elementos son elementos hijos
  - el orden de los elementos hijos
  - el número de elementos hijos
  - si un elemento está vacío o puede incluir texto
  - los tipos de datos de sus elementos y atributos
  - los valores por defecto y fijos para elementos y atributos

## XML Schema es el sucesor de DTD's

-  En un futuro, los XML Schema serán utilizados en la mayoría de las aplicaciones Web y reemplazarán a las actuales DTDs:
- Los XML Schemas son extensibles.
  - Los XML Schemas son más ricos semánticamente que las DTDs.
  - Los XML Schemas están escritos siguiendo la sintaxis estándar de XML.
  - Los XML Schemas soportan tipos de datos.
  - Los XML Schemas soportan *namespaces*.

XML Schema es un estándar de **W3C**

## XML Schema soporta tipos de datos

- La característica mas importante de los XML Schemas es que soportan tipos de datos.
- Los tipos de datos permiten:
  - describir qué elementos están permitidos
  - validar si los datos son correctos
  - trabajar con los datos de una base de datos
  - definir *Facets* (restricciones)
  - definir patrones de datos (formatos)
  - convertir datos considerando diferentes tipos de datos

## XML Schema utiliza sintaxis de XML

- ✎ Otra característica importante de los XML Schemas es que están escritos en XML.
- ✎ Beneficios:
  - No hay necesidad de aprender un lenguaje nuevo.
  - Se puede utilizar un editor de XML para editar el XML Schema.
  - Se puede utilizar un procesador de XML para procesar un XML Schema.
  - Se puede manipular un XML Schema utilizando XML DOM.
  - Se puede transformar un XML Schema utilizando XSLT.

# Transmisión de datos sin ambigüedad

- ✧ Cuando se transmite información desde un emisor a un receptor, es necesario que ambas partes conozcan el mismo protocolo de comunicación.
- ✧ Con XML Schemas, el emisor puede describir los datos de manera que el receptor lo entienda.
- ✧ Una fecha como: "03-11-2006", dependiendo del país, puede ser interpretada como "3 de noviembre" y en otros países como "11 de marzo".
- ✧ Con lo cual, al definir un elemento de la siguiente manera:
  - `<fechaInicio type="date">2006-03-11</fechaInicio>`  
se asegura un mutuo entendimiento sobre el contenido, porque el tipo de dato "date" requiere el formato "YYYY-MM-DD". MCV1

## Diapositiva 21

---

### MCV1

El formato Date tomará el formato de fecha que se tiene configurado en Configuración regional, si no recuerdo mal. Eso habría que comprobarlo.

Maricruz Valiente; 11/03/2007

## Los XML Schema son extensibles

- ✧ Los XML Schemas son extensibles porque están escritos en XML.
- ✧ Con una definición de XML Schemas extensible se podrá:
  - reutilizar el Schema en otros Schemas
  - crear tipos de datos propios derivados de los tipos estándar
  - referenciar varios Schemas en el mismo documento

## No es suficiente que el XML Schema esté bien formado

- ✎ Un documento XML bien formado es un documento que cumple las reglas sintácticas de XML como:
  - Empezar con la declaración XML.
  - Tener un único elemento raíz.
  - Las etiquetas de apertura deben tener su correspondientes etiquetas de cierre.
  - Los elementos son "case-sensitive".
  - Todos los elementos deben cerrarse.
  - Todos los elementos tienen que estar adecuadamente anidados.
  - Los atributos deben estar entre comillas.
  - Se deben utilizar las entidades para utilizar caracteres especiales.
- ✎ **A pesar de que los documentos estén bien formados pueden seguir teniendo errores.**



## Ejemplo XML

```
<?xml version="1.0"?>  
<nota>  
  <a>Juan</a>  
  <de>Susana</de>  
  <cabecera>Recordatorio</cabecera>  
  <cuerpo>iRecuerda que tenemos reunión!</cuerpo>  
</nota>
```

## Ejemplo XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

  <xs:element name="nota">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="a" type="xs:string"/>
        <xs:element name="de" type="xs:string"/>
        <xs:element name="cabecera" type="xs:string"/>
        <xs:element name="cuerpo" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

## Referencia a un XML Schema

- ❧ Cómo se referencia un archivo XML Schema desde un documento XML.

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

<nota
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">

  <a>Juan</a>
  <de>Susana</de>
  <cabecera>Recordatorio</cabecera>
  <cuerpo>iRecuerda que tenemos reunión!</cuerpo>
</nota>
```

## Diapositiva 26

---

### MCV2

`xmlns="http://www.w3schools.com"`

specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3schools.com" namespace.

Once you have the XML Schema Instance namespace available:

`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` you can use the `schemaLocation` attribute. This attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

`xsi:schemaLocation="http://www.w3schools.com note.xsd"`

Maricruz Valiente; 11/03/2007

## MCV3 XML Schema. Elemento raíz

- El elemento **<schema>** es el elemento raíz (elemento documento) de todo XML Schema:

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

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
```

```
...
```

```
...
```

```
</xs:schema>
```

## Diapositiva 27

---

### MCV3

`xmlns:xs="http://www.w3.org/2001/XMLSchema"`

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

`targetNamespace="http://www.w3schools.com"`

indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

`xmlns="http://www.w3schools.com"`

indicates that the default namespace is "http://www.w3schools.com".

`elementFormDefault="qualified"`

indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

## XML Schema. Elementos simples (i)

- Un elemento simple es un elemento XML que sólo puede contener texto. No puede contener ni elementos ni atributos.
- Sin embargo, la restricción "sólo texto" no es del todo cierta. El texto puede contener muchos tipos de datos. Puede ser de uno de los tipos incluidos en la definición de XML Schema (boolean, string, date, etc.), o puede ser de un tipo definido por el usuario.
- Además, se pueden añadir restricciones (*facets*) a los tipos de datos, para limitar así su contenido, o para requerir que ciertos datos sean conformes a un patrón específico.

## XML Schema. Elementos simples (ii)

- La sintaxis de un elemento simple es:

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

- donde xxx es el nombre del elemento e yyy es el tipo de datos del elemento.
- Los tipos de datos más comunes en XML Schema son:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date



## XML Schema. Elementos simples (iii)

### Ejemplo de XML:

```
<nombre>Jana</nombre>  
<edad>36</edad>  
<fechanacimiento>1970-03-27</fechanacimiento>
```

### Ejemplo de XML Schema:

```
<xs:element name="nombre" type="xs:string"/>  
<xs:element name="edad" type="xs:integer"/>  
<xs:element name="fechanacimiento" type="xs:date"/>
```

## XML Schema. Elementos simples (iv)

- Se pueden especificar valores por defecto o valores fijos para los elementos simples.
- Se asignará automáticamente el valor por defecto si no se especifica otro valor para el elemento.

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

- Se asignará automáticamente un valor para el elemento, pero en este caso, dicho valor no puede ser modificado

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

## XML Schema. Atributos (i)

- Los elementos simples no pueden tener atributos.
- Un elemento con atributos será considerado un tipo complejo (complex type).
- Sin embargo, los atributos se definen como un tipo simple.

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

- donde xxx es el nombre del elemento e yyy es el tipo de datos del atributo.

## XML Schema. Atributos (ii)

 Código en XML:

```
<apellido idioma="EN">Smith</apellido>
```

 Código en XML Schema:

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

## XML Schema. Atributos (iii)

- Se pueden especificar valores por defecto o valores fijos para los atributos.
- Se asignará automáticamente el valor por defecto si no se especifica otro valor para el atributo.

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

- Se asignará automáticamente un valor para el atributo, pero en este caso, dicho valor no puede ser modificado

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

- Por defecto los atributos son opcionales, pero se puede hacer que éstos sean requeridos

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

## XML Schema. Restricciones (i)

- Las restricciones en XML Schema se denominan **"FACETS"**

```
<xs:element name="edad">  
  
  <xs:simpleType>  
    <xs:restriction base="xs:integer">  
      <xs:minInclusive value="0"/>  
      <xs:maxInclusive value="120"/>  
    </xs:restriction>  
  </xs:simpleType>  
  
</xs:element>
```

## XML Schema. Restricciones (ii)

- Para limitar el contenido de un elemento XML a un conjunto de valores, se utilizará la restricción **enumeration**.

MCV4

```
<xs:element name="coche">

  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>

</xs:element>
```

## Diapositiva 36

---

### MCV4

Vemos que la restricción se encuentra incluida dentro del elemento.

Maricruz Valiente; 11/03/2007



## XML Schema. Restricciones (iii)

 Otra alternativa:

```
<xs:element name= "coche" type="TipoCoche"/>

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

## XML Schema. Restricciones (iv)

- Para limitar que el contenido de un elemento XML solo pueda tener una serie de números o letras se deberá utilizar la restricción **pattern**.

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

## XML Schema. Restricciones (v)

- Los únicos valores permitidos son tres letras de la 'a' a la 'z' escritas en mayúscula.

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

- Los únicos valores permitidos son tres letras de la 'a' a la 'z' escritas en mayúscula o en minúscula:

```
<xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
```

- El único valor permitido es una de las siguientes letras: 'x', 'y', o 'z':

```
<xs:pattern value="[xyz]"/>
```

- Los únicos valores permitidos son cinco dígitos comprendidos entre '0' y '9':

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

## XML Schema. Restricciones (vi)

- Los únicos valores permitidos son cero o más ocurrencias de letras en minúscula desde la 'a' a la 'z':

```
<xs:pattern value="([a-z])*"/>
```

- Los únicos valores permitidos son una o más ocurrencias de dos letras. En cada par de letras, la primera irá en minúscula y la segunda en mayúscula. Por ejemplo, "sToP" sería validado por el patrón, pero no "Stop", "STOP" ó "stop":

```
<xs:pattern value="([a-z][A-Z])+"/>
```

- Los únicos valores permitidos son exactamente ocho caracteres, donde esos caracteres pueden ser letras minúsculas, letras mayúsculas o un dígito del 0 al 9:

```
<xs:pattern value="[a-zA-Z0-9]{8}"/>
```

## XML Schema. Restricciones (vii)

- Para especificar como se van a tratar los caracteres en blanco, se utilizará la restricción whiteSpace.

```
<xs:elementMCV5 name="direccion">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:whiteSpace value="preserve"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

## Diapositiva 41

---

### MCV5

Este ejemplo define un elemento llamado "direccion" con una restricción. La restricción "whiteSpace" se establece a "preserve", lo cual significa que el procesador XML NO ELIMINARÁ ningún carácter en blanco :

Maricruz Valiente; 11/03/2007

## XML Schema. Restricciones (viii)

```
<xs:element name="direccion">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="replace"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- Si se desea que XML elimine todos los espacios en blanco; LF, CR, tabulaciones, los espacios de principio y de final de cadena, y los espacios intermedios (dejando para este caso sólo uno):

```
<xs:whiteSpace value="collapse"/>
```

## Diapositiva 42

---

### MCV6

#### Ejemplo 1:

Este ejemplo además define un elemento llamado "address" con una restricción. La restricción "whiteSpace" se ha establecido a "replace", lo que significa que el procesador de XML REEMPLAZARÁ todos los espacios en blanco (LF, CR, tabulaciones) por espacios:

Maricruz Valiente; 11/03/2007



## XML Schema. Restricciones (ix)




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

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

## XML Schema. Restricciones (x)

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

# XML Schema. Elementos complejos (I)

-  Un elemento complejo es un elemento XML que puede tener más elementos y/o atributos.
-  Hay cuatro clases de elementos complejos:
  - Elementos vacíos.
  - Elementos que contienen otros elementos.
  - Elementos que contienen sólo texto.
  - Elementos que contienen ambos: otros elementos y texto.
-  Nota: Cada uno de estos elementos pueden contener atributos.

# XML Schema. Elementos complejos (ii)

- Elemento XML complejo vacío:

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

- Elemento XML complejo que contiene sólo otros elementos:

```
<empleado>  
<nombre>Juan</nombre>  
<apellidos>García López</apellidos>  
</empleado>
```


- Elemento XML complejo que contiene sólo texto:

```
<comida tipoC="postre">Helado</comida>
```

- Elemento XML complejo que contiene ambos (texto y otros elementos):

```
<carta>  
Estimado Sr.<nombre>Juan García</nombre>.  
Su pedido <pedidoid>1032</pedidoid>  
será enviado el <fechaenv>2007-03-25</fechaenv>.  
</carta>
```

# XML Schema. Elementos complejos (iii)

 Se puede definir un elemento complejo en XML Schema de dos formas distintas :


1. El elemento "empleado" se puede declarar directamente nombrando el elemento :

```
<xs:element name="empleado">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellidos" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# XML Schema. Elementos complejos (iv)

2. El elemento "empleado" se puede declarar directamente nombrando el elemento :

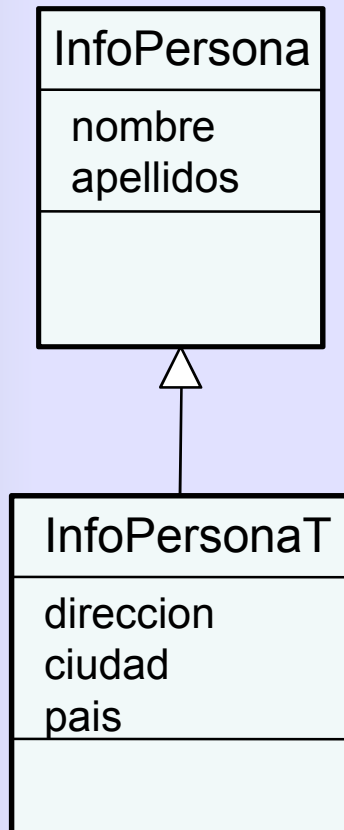
```
<xs:element name="empleado" type="InfoPersona"/>
<xs:complexType name="InfoPerson a">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

-  Usando este método, muchos elementos se pueden referir a la misma definición de tipo complejo

```
<xs:element name="empleado" type="InfoPersona"/>
<xs:element name="alumno" type="InfoPersona"/>
<xs:element name="Socio" type="InfoPersona"/>
<xs:complexType name="personinfo">
  ...
</xs:complexType>
```

# XML Schema. Elementos complejos (v)

- Además, se puede describir un elemento complejo basándose en otro elemento complejo añadiendo más elementos:



```
<xs:element name="empleado" type="fullpersoninfo"/>
<xs:complexType name="InfoPersona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name=" InfoPersonaT">
  <xs:complexContent>
    <xs:extension base=" InfoPersona">
      <xs:sequence>
        <xs:element name="direccion" type="xs:string"/>
        <xs:element name="ciudad" type="xs:string"/>
        <xs:element name="pais" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

# XML Schema. Elementos complejos vacíos

## Definición en XML Schema

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

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



# XML Schema. Elementos complejos con sólo elementos

- La etiqueta `<xs:sequence>` indica que los elementos definidos deben aparecer en el orden descrito dentro del elemento "persona".

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellidos" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="persona" type="TipoPersona"/>
<xs:complexType name="TipoPersona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

# Elementos complejos con texto y otros elementos

- Para permitir que aparezcan caracteres entre los elementos hijos, el atributo mixed debe estar a "true".

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

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

# XML Schema. Indicadores para elementos complejos

- ✎ Permiten realizar el control de qué elementos van a ser utilizados en un documento XML.
- ✎ Hay siete tipos de indicadores:
  - **Indicadores de orden:**
    - All
    - Choice
    - Sequence
  - **Indicadores de ocurrencia:**
    - maxOccurs
    - minOccurs
  - **Indicadores de grupo:**
    - Group name
    - attributeGroup name

# XML Schema. Indicadores de orden

## Indicador **<xs:all>**

- Los hijos pueden aparecer en cualquier orden y una sola vez.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:all>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellidos" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

## Indicador **<xs:choice>**

- Solo puede aparecer un hijo de los descritos.

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="empleado" type="Empleado"/>
      <xs:element name="socio" type="Socio"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

## Indicador **<xs:sequence>**

- Deben aparecer los hijos en la secuencia descrita y solo una vez.

# XML Schema. Indicadores de ocurrencia

- Indicador **<maxOccurs>**: El número máximo de veces que se puede dar un elemento.
- Indicador **<minOccurs>**: El número mínimo de veces que un elemento puede aparecer

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre_completo"
type="xs:string"/>
      <xs:element name="nombre_hijo" type="xs:string"
        maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Por defecto, **<minOccurs>** = "1"

# XML Schema. Indicadores de grupo de elementos

- Define grupos de elementos para poder referenciarlos varias veces.

```
<xs:group name="GrupoPersonas">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="apellidos" type="xs:string"/>
    <xs:element name="fechanac" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="persona" type="InfoPersona"/>

<xs:complexType name="InfoPersona">
  <xs:sequence>
    <xs:group ref="GrupoPersonas"/>
    <xs:element name="pais" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

# XML Schema. Indicadores de grupos de atributos

- Define grupos de atributos para poder referenciarlos varias veces.

```
<xs:attributeGroup name="GrupoAtrPersonas">
  <xs:attribute name="nombre" type="xs:string"/>
  <xs:attribute name="apellidos" type="xs:string"/>
  <xs:attribute name="fechanac" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="persona">
  <xs:complexType>
    <xs:attributeGroup ref="GrupoAtrPersonas"/>
  </xs:complexType>
</xs:element>
```

# XML Schema. <any> (i)

- El elemento **<any>** permite extender documentos con elementos que no han sido descritos en el Schema.

Familia.xsd

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellidos" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- Después del elemento <apellidos> se podrían poner otros elementos que no están definidos en el Schema



# XML Schema. <any> (ii)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<personas xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com familia.xsd
http://www.w3schools.com hijos.xsd">
```

```
<persona>
<nombre>Jana</nombre>
<apellidos>Martínez Sanz</apellidos>
<hijos>
  <nombrehijo>Cecilia</nombrehijo>
</hijos>
</persona>
```

```
<persona>
<nombre>Miguel</nombre>
<apellidos>Campos Pérez</apellidos>
</persona>
```

```
</personas>
```

# XML Schema.

## <anyAttribute> (i)

- El elemento **<anyAttribute>** permite extender documentos con atributos que no han sido descritos en el Schema.

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="apellido" type="xs:string"/>
    </xs:sequence>
    <xs:anyAttribute/>
  </xs:complexType>
</xs:element>
```

- Cuando se describa una persona en el documento XML se podrá extender con atributos de otro Schema.

# XML Schema. <anyAttribute > (ii)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<personas xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.microsoft.com familia.xsd
http://www.w3schools.com atributo.xsd">
```

```
<person sexo="femenino">
<nombre>Jana</nombre>
<apellidos>Martínez Sanz</apellidos>
</persona>
```

```
<persona sexo="masculino">
<nombre>Miguel</nombre>
<apellidos>Campos Pérez</apellidos>
</persona>
```

```
</personas>
```

# XML Schema. String

- El tipo de dato **String** permite contener cualquier tipo de caracteres, LF (line feeds), CR (carriage returns), y tabulaciones.

```
<xs:element name="cliente" type="xs:string"/>
```

```
<cliente>Juan García</cliente>
```

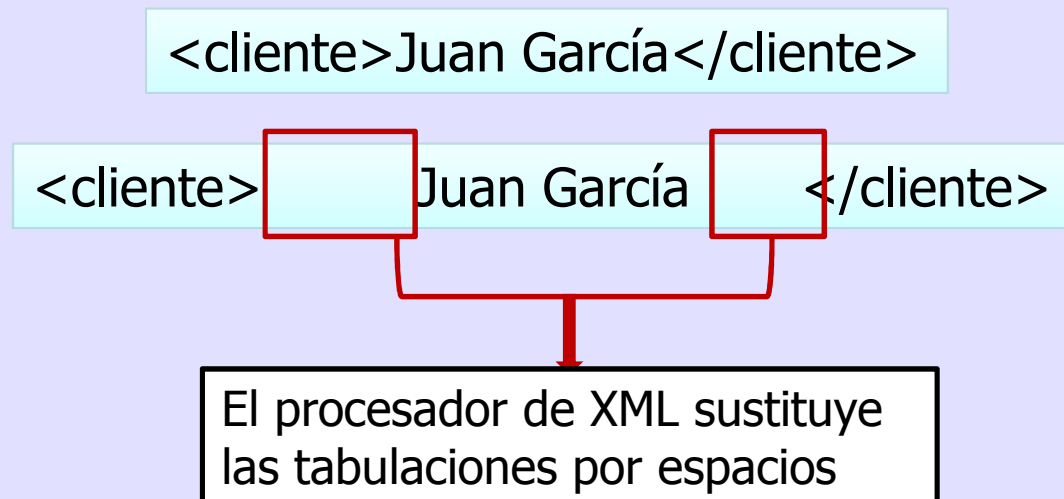
```
<cliente>      Juan García      </cliente>
```

# XML Schema.

## normalizedString

- El tipo de datos **normalizedString** permite contener cualquier carácter pero el procesador de XML sustituye los LF, CR y caracteres de tabulación por espacios.

```
<xs:element name="cliente" type="xs:normalizedString"/>
```



# XML Schema. token

- El tipo de datos **token** permite contener cualquier carácter, pero el procesador de XML eliminará los LF, CR, caracteres de tabulación, espacios al principio, espacios al final, y espacios entre medias

```
<xs:element name="cliente" type="xs:token"/>
```

```
<cliente>Juan García</cliente>
```

```
<cliente> Juan García </customer>
```



El procesador de XML elimina las tabulaciones, espacios del principio, final, entre medias (dejando solo uno)

# XML Schema. Cadenas de caracteres

Nombre	Descripción
ID	Cadena de caracteres que representa un atributo ID en XML (sólo se usa con atributos de esquema).
IDREF	Cadena de caracteres que representa el atributo IDREF en XML (sólo se usa con atributos de esquema).
language	Cadena de caracteres que contiene un ID de lenguaje válido.
Name	Cadena de caracteres que contiene un nombre XML válido.
NMTOKEN	Cadena de caracteres que representa el atributo NMTOKEN en XML (sólo se usa con atributos de esquema).

# XML Schema. Fechas

<b>Nombre</b>	<b>Descripción</b>
date	Define un valor para una fecha.
dateTime	Define valores de fecha y hora.
duration	Define intervalos de tiempo.
gDay	Define una parte de una fecha: el día (DD).
gMonth	Define una parte de una fecha: el mes (MM).
gMonthDay	Define una parte de una fecha: el mes y el día (MM-DD).
gYear	Define una parte de una fecha: el año (YYYY).
gYearMonth	Define una parte de una fecha: el año y el mes (YYYY-MM).
time	Define un valor de hora.



# XML Schema. decimal

- El tipo de dato **decimal** se utiliza para especificar valores numéricos decimales.

```
<xs:element name="rendimiento" type="xs:decimal"/>
```

```
<rendimiento>999.50</rendimiento>
```

```
<rendimiento>+999.5450</rendimiento>
```

```
<rendimiento>-999.5230</rendimiento>
```

```
<rendimiento>0</rendimiento>
```

```
<rendimiento>14</rendimiento>
```

# XML Schema. integer

- El tipo de dato **integer** sirve para poder describir números sin la parte fraccional.

```
<xs:element name="coste" type="xs:integer"/>
```

```
<coste>999</coste>
```

```
<coste>+999</coste>
```

```
<coste>-999</coste>
```

```
<coste>0</coste>
```

# XML Schema. Datos numéricos

Nombre	Descripción
byte	Entero de 8 bits con signo.
int	Entero de 32 bits con signo.
long	Entero de 64 bits con signo.
negativeInteger	Entero que contiene sólo valores negativos ( ..., -2, -1.).
nonNegativeInteger	Entero que contiene sólo valores positivos (0, 1, 2, ...).
nonPositiveInteger	Entero que contiene sólo valores no positivos (... , -2, -1, 0).
positiveInteger	Entero que contienen sólo valores positivos (1, 2, ...).
short	Entero de 16 bits con signo.
unsignedLong	Entero de 64 bits sin signo.
unsignedInt	Entero de 32 bits sin signo.
unsignedShort	Entero de 16 bits sin signo.
unsignedByte	Entero de 8 bits sin signo.