

An abstract network diagram consisting of numerous nodes (small dots) connected by thin lines. The nodes are arranged in a complex, interconnected pattern, with some nodes having more connections than others. The lines are thin and grey, and the nodes are small dots in various colors (blue, red, black). The overall shape is roughly triangular, with the base at the bottom and the apex at the top.

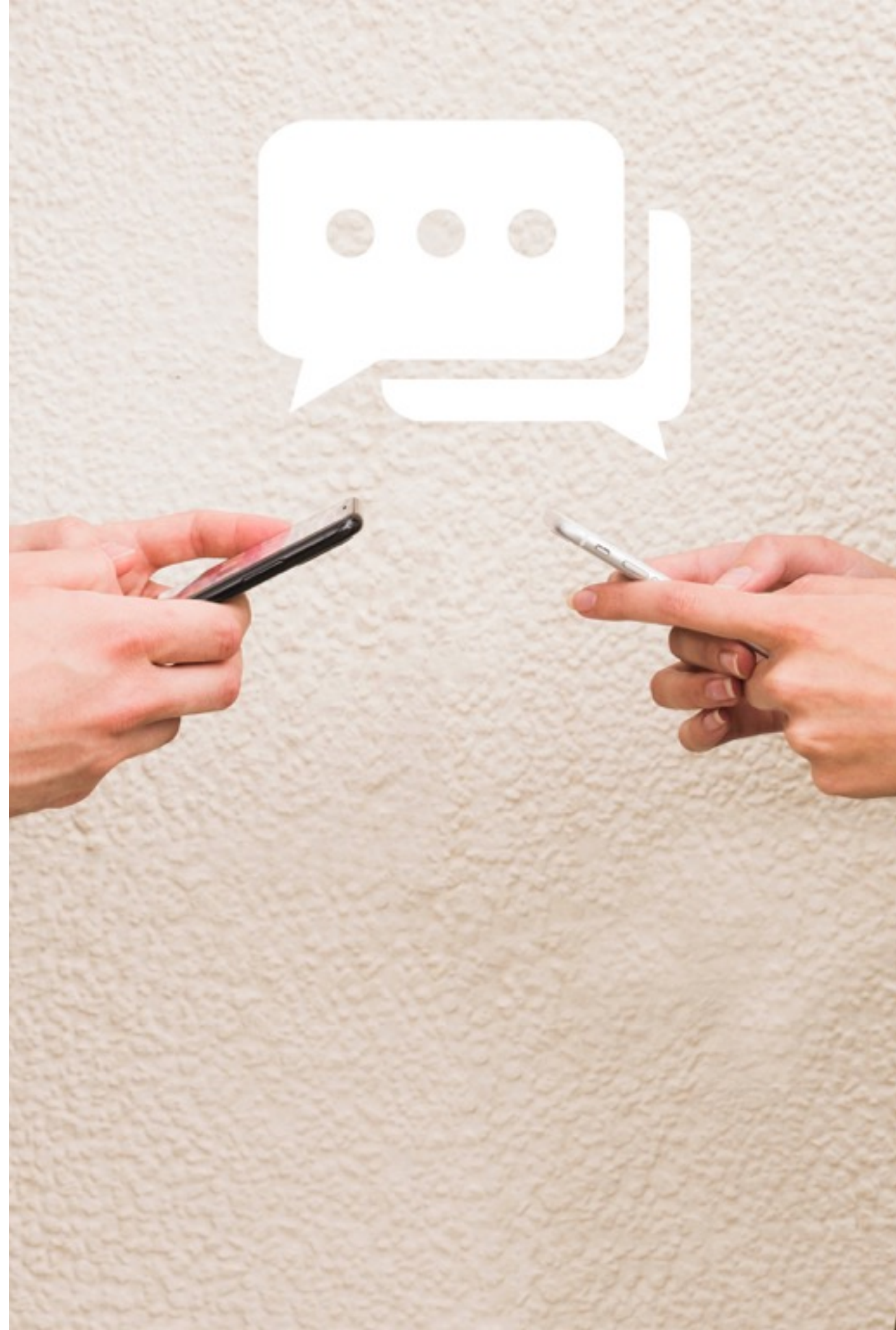
DEFINICIÓN DE ESQUEMAS Y VOCABULARIO EN XML

Para un correcto intercambio de información.

Intercambio de información independiente de plataformas

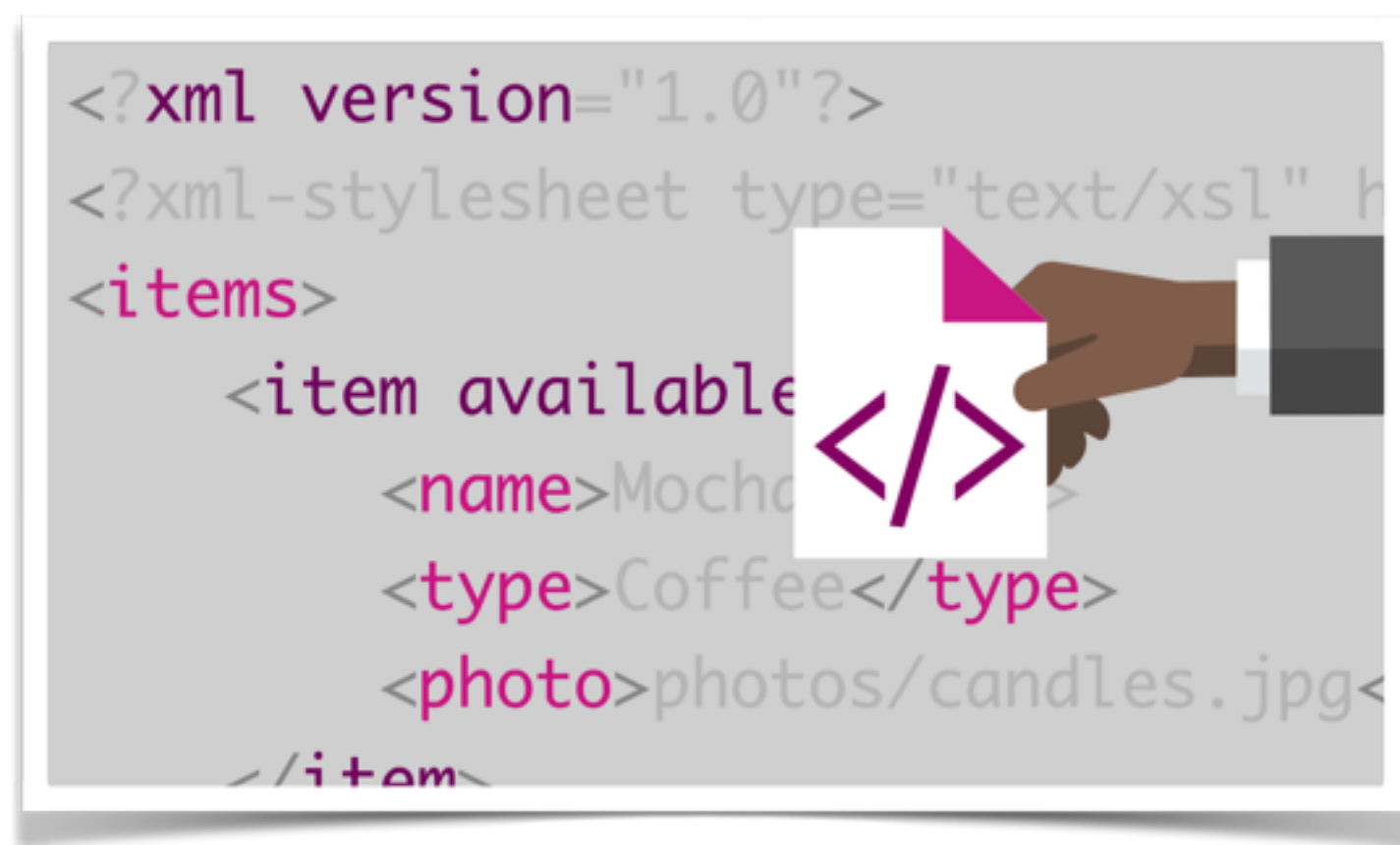
No es solo imprescindible que el documento esté bien formado, sino que ambos actores dispongan de una misma definición de estructura de datos.

DTD  XML Schema



XML SCHEMA

- Es un mecanismo para comprobar la validez de un documento XML. Es una forma alternativa y más actual que los DTD. Los documentos XML que se validan contra un esquema se llaman instancias del esquema.
- Fue diseñado completamente alrededor de namespaces y soporta tipos de datos típicos de los lenguajes de programación, como también tipos personalizados simples y complejos. Un esquema se define pensando en su uso final.



VENTAJAS

- Un esquema es un documento XML y, por tanto, se puede comprobar que está bien formado.
- Tiene gran número de tipos de datos y atributos predefinidos que se pueden ampliar o restringir con la creación de nuevos tipos, ya que está basado en la misma sintaxis que XML.
- Se puede determinar con precisión la cardinalidad de un elemento, es decir, el número de veces que aparece en un documento.
- Permite utilizar varios espacios de nombres, esto permite mezclar distintos vocabularios o juegos de etiquetas.

DECLARACIÓN DEL ELEMENTO SCHEMA

para definir el XML Schema :

Primer ejemplo XSD. libro.xsd

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



para asociar el documento XML al Schema, en el documento INSTANCIA

Instancia XML. libro.xml

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




- “xmlns:xsi = http://www.w3.org/2001/XMLSchema-instance”: indica que queremos utilizar los elementos definidos en http://www.w3.org/2001/XMLSchema-instance.
- “xsi:noNamespaceSchemaLocation="libro.xsd": indica que vamos a usar ese fichero (libro.xsd) que contiene el XSchema, pero sin asociar un espacio de nombres a esas definiciones. Sin esta sentencia, no tendremos esquema de validación.

- Los XML schemas son documentos XML. Así, todo esquema debe comenzar con una declaración XML: esta es la primera línea del documento.
- En la segunda línea encontramos la declaración del elemento esquema:
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- Como todo documento XML, un schema debe tener un elemento raíz, este es el elemento schema
 - </xsd:schema>

EL ELEMENTO RAIZ

```
<?xml version="1.0"?>  
<xs:schema>  
...  
</xs:schema>
```



Primer ejemplo XSD. libro.xsd

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

EN ESTE ELEMENTO RAIZ TIENE UN ATRIBUTO MUY INTERESANTE:

- EL ESPACIO DE NOMBRES

ESPACIO DE NOMBRES

`xmlns(:<prefijo>)?=<nombre_del_espacio_de_nombres(URI)>`

- Un espacio de nombres XML es una recomendación W3C para proporcionar elementos y atributos con nombre único en una instancia XML. Una instancia XML puede contener nombres de elementos o atributos procedentes de más de un vocabulario XML.
- Fue diseñado completamente alrededor de namespaces y soporta tipos de datos típicos de los lenguajes de programación, como también tipos personalizados simples y complejos. Un esquema se define pensando en su uso final.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<lb:libro xmlns:lb="urn:loc.gov:libros" xmlns:isbn="urn:ISBN:0-395-36341-6">
  <lb:titulo>Más barato que una Docena</lb:titulo>
  <isbn:numero>1568491379</isbn:numero>
</lb:libro>
```

ESPACIO DE NOMBRES

EN EL XSD SCHEMA:

- xmlns: identifica el espacio de nombres al que pertenecen los componentes incluidos en el esquema, asignando opcionalmente un prefijo (este prefijo suele ser “xs” o “xsd”. Así, la sentencia
- xmlns:xs=”http://www.w3.org/2001/XMLSchema”
- indica que los elementos y tipos de datos utilizados en el esquema provienen del espacio de nombres “http://www.w3.org/2001/XMLSchema”, al que se le asigna el prefijo “xs”.
- Además podemos hacer que el esquema que estamos creando tenga asociado un espacio de nombres propio (target namespace). Para ello se puede utilizar el atributo targetNamespace del elemento “schema”:
- targetNamespace:”http://www.prueba.es/esquema1”

EN EL DOCUMENTO INSTANCIA:

- La referencia a un esquema se hace mediante el atributo raiz, y generalmente se hace a una especificación estandar, que generalmente se le da el nombre de xsi.

xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”

- Con esto podemos utilizar los atributos “noNameSpaceLocation” y “schemaLocation”, que nos permiten asociar un documento instancia XML con un esquema:
- noNamespaceSchemaLocation: identifica un documento de schema que no tiene un espacio de nombres de destino (no incluye el atributo “targetNamespace”) y lo asocia al documento XML instancia.
- schemaLocation: Asocia un documento de esquema que tiene un espacio de nombres de destino (targetNamespace) con un documento de instancia. Este atributo tendrá dos valores, separados por un espacio en blanco. El primer valor coincide con el del “targetNamespace” especificado en el schema. El segundo es la ubicación donde se encuentra definido el XML schema.

ESPACIO DE NOMBRES

EN EL XSD SCHEMA:

Tenemos un schema definido: “apuntes.xsd” , si en dicho schema se definió:

targetNamespace:”<http://www.prueba.es/esquema1>”

EN EL DOCUMENTO INSTANCIA:



xmlns:xsi=”<http://www.w3.org/2001/XMLSchema-instance>”

xsi:schemaLocation:”<http://www.prueba.es/esquema1> apuntes.xsd”

Si por el contrario en el documento “apuntes.xsd” no se especifica Un “targetNamespace”, en el documento instancia XML tendremos:



xmlns:xsi=”<http://www.w3.org/2001/XMLSchema-instance>”

xsi:noNamespaceSchemaLocation=”apuntes.xsd”

Para establecer cual es el prefijo de un determinado espacio de nombres dentro del documento instancia (XML) utilizamos:

xmlns: prefijo= ”espacio de nombres”

xmlns:apu=”<http://www.prueba.es/esquema1>”

DECLARACIÓN DE ELEMENTOS

```
<xsd:element name="nombreElemento"
  type="tipoSimple/tipoComplejo"
  minOccurs="valor"
  maxOccurs="valor"
  fixed="valor"
  default="valor"/>
```

- **name:** Nombre del elemento
- **type:** el tipo de elemento.
 - **Tipos simples:** son elementos que sólo pueden contener datos carácter; no pueden incluir otros elementos ni tampoco atributos.
 - **Tipos complejos:** estos elementos pueden incluir otros elementos y/o atributos. El contenido de estos elementos se define entre la etiqueta de inicio
- **minOccurs y maxOccurs** (Opcionales) : estos dos atributos indican el mínimo (minOccurs) y máximo (maxOccurs) número de ocurrencias del elemento. El valor por defecto para ambos atributos es 1. Si se quiere indicar que el elemento puede aparecer un número ilimitado de veces, el atributo maxOccurs tomará el valor “**unbounded**”.
- **fixed** (Opcional): especifica un valor fijo para el elemento.
- **default** (Opcional): especifica un valor por defecto para el elemento.

MODELOS DE CONTENIDOS PARA ELEMENTOS SIMPLES

TIPOS CADENA

- string: secuencia de longitud finita de caracteres*
- anyURI: una uri estándar de Internet
- NOTATION: declara enlaces a contenido externo no-XML
- QName: una cadena legal QName (nombre con cualificador)

TIPOS NUMERICOS

- decimal: número decimal de precisión (dígitos significativos) arbitraria *
- float: número de punto flotante de 32 bits de precisión simple *
- double: número de punto flotante de 64 bits de doble precisión *

TIPOS FECHA/HORA

- duration: duración de tiempo
- dateTime: instante de tiempo específico, usando calendario gregoriano, en formato "YYYYMM-DDThh:mm:ss"
- date: fecha específica del calendario gregoriano, en formato "YYYY-MM-DD" *
- time: una instancia de tiempo que ocurre cada día, en formato "hh:mm:ss"
- gYearMonth: un año y mes del calendario gregoriano
- gYear: año del calendario gregoriano
- gMonthDay: día y mes del calendario gregoriano
- gMonth: un mes del calendario gregoriano
- gDay: una fecha del calendario gregoriano (día)

TIPOS BINARIO CODIFICADO

boolean: toma los valores "true" o "false" *

hexBinary: dato binario codificado como una serie de pares de dígitos hexadecimales

base64Binary: datos binarios codificados en base 64

MODELOS DE CONTENIDOS PARA ELEMENTOS SIMPLES

EJEMPLOS DE TIPOS SIMPLES

xs:integer

```
<xs:element name="vueltas" type="xs:integer">
```

```
<vueltas>1205</vueltas>
```

```
<vueltas>-1205</vueltas>
```

xs:boolean

```
<xs:element name="pagado" type="xs:boolean">
```

```
<pagado>>true</pagado>
```

```
<pagado>>false</pagado>
```

xs:hexBinary

```
<xs:element name="imagen" type="xs:hexBinary">
```

```
<xs:element name="foto" type="xs:base64Binary">
```

```
<!-- entre las etiquetas va una codificación binaria  
documentos o formatos binarios. -->
```

xs:string

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

```
<nombre>Javier Toledano</nombre>
```

xs:date

```
<xs:element name="fechaNacimiento" type="xs:date">
```

```
<fechaNacimiento>1979-02-04</fechaNacimiento>
```

```
<fechaNacimiento>1979-02-04+01:00</fechaNacimiento>
```

xs:time

```
<xs:element name="hora" type="xs:date">
```

```
<hora>23:55:15</hora>
```

```
<hora>23:55:15+01:00</hora>
```

```
<hora>23:55:15.1</hora>
```

xs:dateTime

```
<xs:element name="fecha" type="xs:date">
```

```
<fecha>1979-02-04T23:55:15</fecha>
```

```
<fecha>1979-02-04T23:55:15+01:00</fecha>
```

```
<fecha>1979-02-04T23:55:15.1</fecha>
```

xs:decimal

```
<xs:element name="precio" type="xs:decimal">
```

```
<precio>1205.74</precio>
```

```
<precio>-1205.74</precio>
```

EJEMPLOS

ejemplo1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<nota>hola</nota>
```

ejemplo1.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="nota" type="xs:string"/>
</xs:schema>
```


MODELOS DE CONTENIDOS PARA ELEMENTOS

ELEMENTO: el elemento puede contener dentro sub-elementos. Existen tres formas de especificar los sub-elementos dentro del elemento, mediante tres tipos de elementos predefinidos en XML Schema: "sequence", "choice" y "all".

LOS ELEMENTOS COMPLEJOS PUEDEN CONTENER:

- Elementos que contienen otro elementos en su interior
- Elementos que contienen atributos en su interior
- Elementos que contienen otros elementos y atributos en su interior

MODELOS DE CONTENIDOS PARA ELEMENTOS

ELEMENTO SEQUENCE: Se utiliza este elemento para indicar una secuencia de elementos que tienen que aparecer en el documento XML. Deben aparecer todos, y en el mismo orden en que se especifican

```
<xsd:element name="camiseta">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="color" type="xsd:string"/>
      <xsd:element name="talla" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

MODELOS DE CONTENIDOS PARA ELEMENTOS

ELEMENTO CHOICE: Especifica una lista de elementos de los cuales sólo puede aparecer uno en el documento XML.

El elemento “choice” puede incluir opcionalmente los atributos minOccurs y maxOccurs, para especificar el mínimo y máximo número de elementos hijos que pueden incluirse en el documento.

```
<xsd:element name="vehiculoMotor">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="coche" type="xsd:string"/>
      <xsd:element name="moto" type="xsd:string"/>
      <xsd:element name="furgoneta" type="xsd:string"/>
      <xsd:element name="camion" type="xsd:string"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

MODELOS DE CONTENIDOS PARA ELEMENTOS

ELEMENTO ALL: Se comporta igual que el elemento `<xsd:sequence>`, pero no es obligado que en el documento XML aparezcan todos los elementos especificados, ni en el mismo orden.

```
<xsd:element name="camiseta">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="color" type="xsd:string"/>
      <xsd:element name="talla" type="xsd:string"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```


MODELOS DE CONTENIDOS PARA ELEMENTOS

ELEMENTO MIXED: el elemento puede contener tanto datos carácter como elementos hijo.

Los elementos hijo se definen igual que en el modelo anterior, mediante los elementos “sequence”, “choice” o “all”.

Para indicar que el elemento puede además incluir datos carácter se usa el atributo “mixed” con valor igual al “true” en el elemento “complexType”

```
<xsd:element name="confirmacionPedido">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="intro" type="xsd:string"/>
      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="fecha" type="xsd:string"/>
      <xsd:element name="titulo" type="xsd:string"/>
    <xsd:sequence>
      </xsd:complexType>
    </xsd:element>
```

```
<confirmacionPedido>
  <intro>Para:</intro>
  <nombre>Antonio Lara</nombre>
  Confirmamos que con fecha <fecha>24-01-2005</fecha> hemos recibido su pedido
  de <titulo>Raices</titulo>. Su título será enviado en 2 días hábiles desde la
  recepción del pedido. Gracias,Ediciones Aranda.
</confirmacionPedido>
```

MODELOS DE CONTENIDOS PARA ELEMENTOS

Vacío: el elemento no puede contener datos carácter ni otros sub-elementos, pero sí puede incluir atributos. En este caso hay que declararlos como tipos complejos. Si no contienen atributos pueden declararse como tipos simples.

```
<xsd:element name="antiguedad">  
  <xsd:complexType>  
    <xsd:attribute name="anyosDeServicio"  
      type="xsd:positiveInteger"/>  
  </xsd:complexType>  
</xsd:element>
```

Ejemplo: declaramos el elemento “antiguedad”, que es de contenido vacío (no contiene ni texto ni otros subelementos), y que es de tipo complejo porque contiene un atributo, “anyosDeServicio”, que es de tipo “positiveInteger” (entero positivo)

EJEMPLOS

libro1.xml

```
<libro>
  <autor>Miguel de Cervantes Saavedra</autor>
  <titulo>El Quijote de la Mancha</titulo>
</libro>
```

libro1.xsd

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

EJEMPLOS

libro2.xml

```
<biblioteca>
  <libro>
    <autor>Miguel de Cervantes Saavedra</autor>
    <titulo>El Quijote de la Mancha</titulo>
  </libro>
  <libro>
    <autor>Pablo Neruda</autor>
    <titulo>Veinte poemas de amor y una canción desesperada</titulo>
  </libro>
</biblioteca>
```

libro2.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="biblioteca">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" minOccurs="0" maxOccurs="unbounded"
          <xs:complexType>
            <xs:sequence>
              <xs:element name="autor" />
              <xs:element name="titulo" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


EJEMPLOS

libro3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <autor>Miguel de Cervantes Saavedra</autor>
    <titulo>El Quijote de la Mancha</titulo>
    <codigo>123</codigo>
  </libro>
  <libro>
    <autor>Pablo Neruda</autor>
    <titulo>Veinte poemas de amor y una canción desesperada</titulo>
    <codigo>124</codigo>
  </libro>
</biblioteca>
```

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="biblioteca">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="autor" />
              <xs:element name="titulo" />
              <xs:element name="codigo">
                <xs:simpleType>
                  <xs:restriction base="xs:integer">
                    <xs:minInclusive value="1"/>
                    <xs:maxInclusive value="9999"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

EJEMPLOS

```
<?xml version="1.0" encoding="utf-8"?>
<biblioteca>
  <libro>
    <autor>Miguel de Cervantes Saavedra</autor>
    <titulo>El Quijote de la Mancha</titulo>
    <codigo>123</codigo>
    <ubicacion>estantería 1</ubicacion>
  </libro>
  <libro>
    <autor>Pablo Neruda</autor>
    <titulo>Veinte poemas de amor y una canción desesperada</titulo>
    <codigo>124</codigo>
    <ubicacion>estantería 11</ubicacion>
  </libro>
</biblioteca>
```

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="biblioteca">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="libro" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="autor" />
            <xs:element name="titulo" />
            <xs:element name="codigo">
              <xs:simpleType>
                <xs:restriction base="xs:integer">
                  <xs:minInclusive value="1"/>
                  <xs:maxInclusive value="9999"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="ubicacion">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="estantería 1"/>
                  <xs:enumeration value="estantería 2"/>
                  ...
                  <xs:enumeration value="estantería 12"/>
                  <xs:enumeration value="estantería 13"/>
                  <xs:enumeration value="estantería 14"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

DECLARACIÓN DE ATRIBUTOS

```
<xsd:attribute name="nombreAtributo"
type="tipoSimple"
use="valor"
default="valor"
fixed="valor"/>
```

- **name:** es el nombre del atributo.
- **type:** el tipo del atributo. Los atributos sólo pueden contener tipos simples.
- **use** (Opcional): puede tomar uno de los siguientes valores:
 - **required:** el atributo debe aparecer en el documento XML.
 - **optional:** el atributo puede aparecer o no aparecer en el documento XML. Este es el valor por defecto.
 - **prohibited:** el atributo no debe aparecer en el documento XML.
- **default** (Opcional): si el atributo no aparece en el documento XML, se le asigna el valor especificado en el atributo “default”. Los valores por defecto sólo tienen sentido si el atributo es opcional, de lo contrario tendremos un error.
- **fixed** (Opcional): define un valor fijo para el atributo.
 - si el valor del atributo está presente en la instancia del documento XML, el valor debe ser el mismo que el que indica el atributo “fixed”
 - si el atributo no está presente en el documento XML, se le asigna el valor contenido en el atributo “fixed”

DECLARACIÓN DE ATRIBUTOS

- Las declaraciones de atributos para un elemento deben aparecer siempre al final del bloque delimitado por la etiqueta de inicio `<complexType>` y la de fin `</complexType>`, después de las especificaciones de todos los demás componentes.
- Los valores de los atributos “default” y “fixed” son mutuamente exclusivos, por lo tanto habrá un error si una declaración contiene ambos.

```
<xsd:element name="cliente">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nombre" type="xsd:string"/>
      <xsd:element name="apellido" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="numCliente" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```


EJERCICIOS

Realizar los 3 primeros ejercicios XML Schema subidos al Classroom

RECORDAMOS QUE LOS MODELOS DE TIPOS DE DATOS PRIMITIVOS SON:

TIPOS CADENA

- string: secuencia de longitud finita de caracteres*
- anyURI: una uri estándar de Internet
- NOTATION: declara enlaces a contenido externo no-XML
- QName: una cadena legal QName (nombre con cualificador)

TIPOS NUMERICOS

- decimal: número decimal de precisión (dígitos significativos) arbitraria *
- float: número de punto flotante de 32 bits de precisión simple *
- double: número de punto flotante de 64 bits de doble precisión *

TIPOS FECHA/HORA

- duration: duración de tiempo
- dateTime: instante de tiempo específico, usando calendario gregoriano, en formato "YYYYMM-DDThh:mm:ss"
- date: fecha específica del calendario gregoriano, en formato "YYYY-MM-DD" *
- time: una instancia de tiempo que ocurre cada día, en formato "hh:mm:ss"
- gYearMonth: un año y mes del calendario gregoriano
- gYear: año del calendario gregoriano
- gMonthDay: día y mes del calendario gregoriano
- gMonth: un mes del calendario gregoriano
- gDay: una fecha del calendario gregoriano (día)

TIPOS BINARIO CODIFICADO

boolean: toma los valores "true" o "false" *

hexBinary: dato binario codificado como una serie de pares de dígitos hexadecimales

base64Binary: datos binarios codificados en base 64

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES

- Para crear un tipo de dato propio, hay varios métodos:
- Crear una restricción de un tipo simple predefinido se utiliza `xs:restriction`, en `xs:simpleType`

```
<xs:simpleType name="monedaUSA">  
  <xs:restriction base="xsd:decimal">  
    <xs:fractionDigits value="2"/>  
  </xs:restriction>  
</xs:simpleType>
```

En este ejemplo creamos un nuevo tipo simple y le asignamos el nombre “monedaUSA”. Mediante el uso del elemento “xsd:restriction” definimos el tipo “monedaUSA” como un subtipo del tipo base “xsd:decimal”, en el que el número de cifras decimales es 2 (xsd:fractionDigits value=”2”).

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES

- Podemos reutilizar el tipo de dato creado o no:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superhero" type="nombreSuperhero"/>

  <xs:simpleType name="nombreSuperhero">
    <xs:restriction base="xs:string">
      <xs:minLength value="1" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Tipo de datos reutilizable, ya que le hemos dado un nombre a simpleType

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superhero">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1" />
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
</xs:schema>
```

Tipo de datos NO reutilizable, está dentro de un elemento solo funcionará dentro de él.

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES

para crear restricciones en los valores de un tipo de datos hemos utilizado el nuevo elemento `<xs:restriction>`. Estas restricciones deben ser detalladas a través de las llamadas facetas:

- `xs:pattern`: patrón o expresión regular. Se estudiará a fondo un poco más adelante. Se usa en tipos de datos de texto.
- `xs:whitespace`: especifica lo que hacer con los espacios en blanco, saltos de línea, retornos de carro y tabuladores. Los valores aceptados son: `preserve` (conservar), `replace` (reemplazar por espacios en blanco) y `collapse` (convertir en un sólo espacio). Se usa en tipos de datos de texto.
- `xs:length`: longitud exacta de caracteres. Se usa en tipos de datos de texto.
- `xs:minLength`: longitud mínima de caracteres. Se usa en tipos de datos de texto.
- `xs:maxLength`: longitud máxima de caracteres. Se usa en tipos de datos de texto.
- `xs:fractionDigits`: número máximo de posiciones decimales en números flotantes. Sólo se usa en datos numéricos flotantes.
- `xs:totalDigits`: número exacto de dígitos. Se usa en datos de tipo numérico.
- `xs:minInclusive`: límite inferior de un intervalo de valores, con el límite incluido. Se usa en tipos numéricos y de fecha y hora.
- `xs:maxInclusive`: límite superior de un intervalo de valores, con el límite incluido. Se usa en tipos numéricos y de fecha y hora.
- `xs:minExclusive`: límite inferior de un intervalo de valores, con el límite no incluido. Se usa en tipos numéricos y de fecha y hora.
- `xs:maxExclusive`: límite superior de un intervalo de valores, con el límite no incluido. Se usa en tipos numéricos y de fecha y hora.
- `xs:enumeration`: lista de valores admitidos. Se usa en cualquier tipo de datos.

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES / ejemplo

- **Crear un tipo enumerado (xsd:enumeration)**

Una forma muy útil de utilizar facetas es crear tipos enumerados para limitar los valores que puede tomar un elemento o atributo

```
<xsd:attribute name="demanda">  
<xsd:simpleType>  
<xs:restriction base="xsd:string">  
<xs:enumeration value="bajo"/>  
<xs:enumeration value="medio"/>  
<xs:enumeration value="alto"/>  
</xs:restriction>  
</xsd:simpleType>  
</xsd:attribute>
```

Definimos el atributo “demanda” como una restricción del tipo base “xsd:string”, donde sólo existen tres valores posibles: “alto”, “medio” y “bajo”. El atributo “demanda” debe tomar uno de estos tres valores.

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES

Realizar los siguientes tipos de datos con restricciones:

- Tipo numérico entero contenido en el límite inferior de 10 (no incluido) y 89 (incluido).
- Tipo de datos que almacena el nombre de uno de los cuatro posibles cantantes: John Lennon, Sting, Lady Gaga. Es posible que necesites buscar en Internet el uso de la faceta que permite hacer esto en concreto.
- Crear un tipo de datos numérico que permita números con 2 dígitos decimales y 10 dígitos enteros.
- Tipo de datos de cadena de texto, que tenga un mínimo de 6 caracteres de longitud y un máximo de menos de 16 caracteres.

```
<xs:simpleType name="numeropropio">
  <xs:restriction base="xs:integer">
    <xs:minExclusive value="10" />
    <xs:maxInclusive value="89"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="cantante">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Jhon Lennon"/>
      <xs:enumeration value="Sting"/>
      <xs:enumeration value="Lady Gaga"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: EXPRESIONES REGULARES

Las expresiones regulares nos van a permitir toda la potencia necesaria para validar cualquier tipo de dato simple. Para ello se basan en una forma de especificar los diferentes tipos de caracteres que podemos encontrar en un texto, y una cardinalidad asociada a cada uno de ellos.

REPRESENTACION

- `.` -> Representa cualquier caracter.
- `\w` -> Cualquier letra, mayúscula o minúscula.
- `\d` -> Un dígito.
- `\D` -> Cualquier carácter que no sea un dígito.
- `\s` -> Cualquier carácter similar a un espacio, como tabuladores, saltos de línea, etc.
- `\S` -> Cualquier carácter que no sea similar a un espacio.
- `[abc]` -> Cualquiera de los caracteres contenidos dentro de los corchetes, sólo se permitirá un único carácter.
- `[A-Z]` -> Intervalo de valores, se permitirá cualquiera que este dentro del intervalo. Recuerda que los caracteres están representados a través de datos numéricos.
- `[^abc]` -> Significa cualquier caracter que no sea alguno de los contenidos entre corchetes.
- `(a|b)` -> uno de los dos caracteres. A efectos prácticos sería igual a `[ab]`.

CARDINALIDAD

- `?` -> De 0 a 1 ocurrencias.
- `*` -> De 0 a infinitas ocurrencias.
- `+` -> De 1 a infinitas ocurrencias.
- `{n}` -> n ocurrencias.
- `{n,m}` -> Mínimo de n ocurrencias y máximo de m.
- `{n,}` -> Mínimo de n ocurrencias y máximo de infinitas.

`\d{4,8}` -> Sucesión de dígitos de un mínimo de 4 y un máximo de 8.

`\d{8}[A-Z]` -> DNI con letra final en mayúscula.

`\w+` -> de 1 a infinitos caracteres

`\w+@\w+.\w+` -> Correo electrónico

`\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}` -> Dirección IPv4

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: EXPRESIONES REGULARES/EJEMPLO

VALIDACIÓN DE UN DNI:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superhero" type="nombreSuperhero"/>

  <xs:simpleType name="nombreSuperhero">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{8}[A-Z]" />
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES/ ejemplo

- Derivación de elementos

La derivación de tipos de datos simples se basa en utilizar un tipo de datos simple como punto de partida para crear otro tipo de datos simple, añadiéndole diferentes facetas que sirvan para restringir, aún más, el dato. El tipo de datos "original" recibe el nombre de tipo de datos "padre" y el tipo de datos "derivado" recibe el nombre de tipo de datos "hijo".

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="NumeroDelimitado" type="numero-50-60"/>

  <xs:simpleType name="numero-50-60">
    <xs:restriction base="numero-10-100">
      <xs:minInclusive value="50" />
      <xs:maxInclusive value="60" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="numero-10-100">
    <xs:restriction base="xs:positiveInteger">
      <xs:minInclusive value="10"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

CREANDO NUESTROS PROPIOS TIPOS DE DATOS: RESTRICCIONES/ ejemplo

- **Listas (xsd:list)**

Las listas son similares a la faceta “enumeration”, pero permiten incluir valores múltiples, separados mediante espacios. Las listas permiten elegir el tipo de datos de los valores en la lista, mediante el atributo “itemType”, incluyendo tipos definidos por el usuario. También se pueden aplicar otras facetas, como “length”, etc. Ejemplo:

```
<xsd:simpleType name="posiblesTiendas">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="oBoy"/>
<xsd:enumeration value="Yoohoo!"/>
<xsd:enumeration value="ConJunction"/>
<xsd:enumeration value="Anazone"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="listaTiendass">
<xsd:list itemType="posiblesTiendas"/>
</xsd:simpleType>
<xsd:simpleType name="tiendas">
<xsd:restriction base="listaTiendass">
<xsd:maxLength value="3"/>
</xsd:restriction>
</xsd:simpleType>
```

Se define el tipo “tiendas” como una restricción del tipo “listaTiendass” donde “length” no puede ser mayor que 3. Es decir, en “tiendas” puede haber hasta 3 valores de la lista. El tipo “listaTiendass” se define como una lista donde cada elemento de la lista es del tipo “posiblesTiendas”. Este último se define como un “enumeration”. Podríamos por ejemplo definir elemento llamado “vendedores” de la siguiente manera:

```
<xsd:element name="vendedores" type="tiendas"/>
```

Este elemento puede tomar hasta tres valores de los que se han especificado en el tipo “posiblesTiendas”. Por ejemplo, en un documento instancia XML podríamos encontrarnos algo como:

```
<tiendas>oBoy Yoohoo!</tiendas>
```


DEFINICIÓN DE TIPOS DE DATOS PROPIOS

- Para crear un tipo de dato propio, hay varios métodos:
- **Añadir atributos a tipos simples por extensión (xsd:extensión)**

Sabemos que los elementos de tipos simples son los que sólo pueden contener datos carácter, pero no atributos ni elementos hijo. Por otro lado, los elementos de tipo complejo pueden tener tanto atributos como elementos hijo. ¿Qué pasa si queremos que un elemento pueda tener atributos, pero no elementos hijo? Existe una forma de hacer esto, utilizando los elementos “xsd:simpleContent” y “xsd:extension”. Veámoslo con un ejemplo:

```
<xsd:element name="nombreOriginal">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="confirmado" default="no"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
```

En este ejemplo definimos el elemento “nombreOriginal” de tipo complejo, pero al usar el elemento <xsd:simpleContent> estamos diciendo que el contenido de “nombreOriginal” tiene que ser sólo datos carácter. Sin embargo, este elemento sí tiene un atributo. Esto se especifica definiendo el elemento “xsd:simpleContent” como una extensión del tipo base “xsd:string” a la que se le añade el atributo “confirmado”, cuyo valor por defecto es “no”.

DEFINICIÓN DE TIPOS DE DATOS PROPIOS

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="superheroe" type="superheroeDerivado"/>

  <xs:complexType mixed="true" name="superheroePrimitivo">
    <xs:sequence>
      <xs:element name="nombre" type="xs:string"/>
      <xs:element name="amigo" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="superheroeDerivado">
    <xs:complexContent mixed="true">
      <xs:extension base="superheroePrimitivo">
        <xs:sequence>
          <xs:element name="poderPrincipal" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="edad" type="xs:string"/>
        <xs:attribute name="color" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

O como extender un dato complejo a nuevas características (xs:extensión) y (xs:complexContent)

Observa el ejemplo para comprender que se ha comenzado creando un tipo de datos complejo llamado "superheroePrimitivo" que tiene en su interior simplemente una secuencia de dos elementos. A continuación se ha partido de este tipo de datos para crear otro, llamado "superheroeDerivado" que incorpora un nuevo elemento y dos atributos.

Una última advertencia: si utilizas el atributo "mixed=true" en el elemento "xs:element" de un tipo de datos "padre", lo tienes que utilizar obligatoriamente en el tipo de datos "hijo".

REFERENCIAS A OTROS ELEMENTOS EXTERNOS

- En un schema es posible declarar elementos de forma global y luego hacer referencias a ellos desde otros elementos.
- VENTAJAS:
 - La principal ventaja de esto es que permite reutilizar una misma definición de un elemento en varios sitios del schema.
 - Otra ventaja de este mecanismo es que puede ayudar a que los schema estén mejor estructurados y sean más fácilmente legibles.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Libro">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Título"/>
        <xsd:element ref="Autores"/>
        <xsd:element ref="Editorial"/>
      </xsd:sequence>
    <xsd:attribute name="precio" type="xsd:double"/>
  </xsd:complexType>
</xsd:element>
  <xsd:element name="Título" type="xsd:string"/>
  <xsd:element name="Autores" type="xsd:string" maxOccurs="10"/>
  <xsd:element name="Editorial" type="xsd:string"/>
</xsd:schema>
```

Para referenciar a un elemento se utiliza el atributo “ref” cuyo valor es el nombre del elemento referenciado, en lugar del atributo “name” seguido de la definición del elemento.

INDICADORES DE GRUPO

- Facilitan la manera de establecer un conjunto de elementos asociados entre sí.
- VENTAJAS:
 - Los grupos permiten modularizar los elementos en pequeños trozos y reutilizarlos si fuera necesario en otras partes del esquema.
 - Otra ventaja de este mecanismo es que puede ayudar a que los schema estén mejor estructurados y sean más fácilmente legibles.

```
<xs:group name="grupoCoche">
  <xs:sequence>
    <xs:element name="marca" type="xs:string"/>
    <xs:element name="modelo" type="xs:string"/>
    <xs:element name="caballos" type="xs:integer"/>
  </xs:sequence>
</xs:group>
<xs:element name="coche" type="TipoCoche"/>
<xs:complexType name="TipoCoche">
  <xs:sequence>
    <xs:element name="codigo" type="xs:integer"/>
    <xs:group ref="grupoCoche"/>
    <xs:element name="combustible" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

El mismo concepto se puede seguir con grupos de atributos:

```
<xs:attributeGroup name="grupoCoche">
  <xs:element name="marca" type="xs:string"/>
  <xs:element name="modelo" type="xs:string"/>
  <xs:element name="caballos" type="xs:integer"/>
</xs:attributeGroup>
```

DEFINICIÓN DE TIPOS DE DATOS PROPIOS

- Para crear un tipo de dato propio, hay varios métodos:

- **CREAR UN TIPO COMPLEJO CON UN NOMBRE.**

- Otra ventaja de este mecanismo es que puede ayudar a que los schema estén mejor estructurados y sean más fácilmente legibles.

```
<xsd:complexType name="precioInfo">  
<xsd:sequence>  
<xsd:element ref="precioTipo"/>  
<xsd:element ref="precioN"/>  
</xsd:sequence>  
</xsd:complexType>  
<xsd:element name="precioTipo" type="xsd:string"/>  
<xsd:element name="precioN" type="xsd:decimal"/>
```

```
<xsd:element name="precio" type="precioInfo"/>
```

Para referenciar a un elemento se utiliza el atributo “ref” cuyo valor es el nombre del elemento referenciado, en lugar del atributo “name” seguido de la definición del elemento.

EJERCICIOS

Realizar los ejercicios XML Schema subidos al Classroom

METODOS DE DISEÑO DE ESQUEMAS

- Diseño Anidado o de muñecas rusas.
- Diseño Plano o de uso de referencias a elementos y atributos
- Diseño con tipos con nombre.

METODOS DE VALIDACIÓN DE DOCUMENTOS

- xsddiagram