Qué es un esquema XSD

Un **esquema XSD** es un mecanismo para comprobar la validez de un documento XML, es decir, definir su estructura: qué elementos, qué tipos de datos, que atributos, en qué orden, cuántas veces se repiten, etc. Se trata de una forma alternativa a los esquema DTD, pero con bastantes *ventajas* sobre estos:

- Es un documento XML, por lo que se puede comprobar si está bien formado.
- Existe una extensa lista de tipos de datos predefinidos para elementos y atributos que pueden ser ampliados o restringidos para crear nuevos tipos.
- Permiten concretar con precisión la cardinalidad de un elemento, es decir, las veces que puede aparecer en un documento XML.
- Permite mezclar distintos vocabularios (juegos de etiquetas) gracias a los espacios de nombres.

Pero también tienen alguna *desventaja*:

- Son documentos más difíciles de interpretar por el ojo humano.

Herramientas para el manejo de esquemas XSD

Para diseñar esquemas XSD podemos utilizar el **software** libre XML Copy Editor o cualquier otro software que comentamos aquí.

Estructura de un esquema XSD

Un **esquema XSD** es un documento XML con la siguiente estructura:

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

Donde aparecen los siguientes elementos:

- La etiqueta *xml* define la versión de XML utilizada y la codificación de caracteres del documento.
- La etiqueta raíz *xs:schema* define el espacio de nombres del esquema XSD así como una serie de atributos opcionales que podéis ampliar aquí.
- La etiqueta *xs:element* que contiene la definición del elemento raíz del documento XML al que se le va a aplicar el esquema y que explicaremos más adelante.

El **documento XML** deberá referenciar el esquema XSD con el que se va a validar utilizando la siguiente cabecera:

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

Donde la etiqueta raíz del documento define los atributos:

- *xmlns:xsi* para declarar el espacio de nombres del esquema XSD.
- *xsi:noNamespaceSchemaLocatio*n para vincular el documento XML con el esquema local XSD.

Componentes básicos de un esquema XSD

Los componentes imprescindibles para construir un esquema XSD son los siguientes:

- xs:element
- xs:attribute

xs:element

Este componente permite declarar los elementos del documento XML. Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:

- *name*: Indica el nombre del elemento. Obligatorio si el elemento padre es <*xs:schema*>.
- *ref*: Indica que la declaración del elemento se encuentra en otro lugar del esquema. No se puede usar si el elemento padre es <*xs:schema*>. No puede aparecer junto con *name*.
- *type*: Indica el tipo de dato que almacenará el elemento. No puede aparecer junto con *ref*.
- *default*: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
- *fixed*: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual.
- *minOccurs*: Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <*xs:schema*>. Va desde 0 hasta ilimitado (*unbounded*). Por defecto 1.
- *maxOccurs*: Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <*xs:schema*>. Va desde 0 hasta ilimitado (*unbounded*). Por defecto 1.

```
<xs:element name="nombre" type="xs:string" default="TicArte" minOccurs="1"
maxOccurs="unbounded" />
<xs:element ref="contacto" minOccurs="1" maxOccurs="unbounded" />
```

xs:attribute

Este componente permite declarar los atributos de los elementos del documento XML. Entre otros, los principales atributos que podemos utilizar en la declaración son los siguientes:

- *name*: Indica el nombre del atributo.
- *ref*: Indica que la declaración del atributo se encuentra en otro lugar del esquema. No puede aparecer junto con *name*.
- *type*: Indica el tipo de dato que almacenará el atributo. No puede aparecer junto con *ref*.
- *use*: Indica si la existencia del atributo es opcional (*optional*), obligatoria (*required*) o prohibida (*prohibited*). Por defecto opcional.
- *default*: Es el valor que tomará el elemento al ser procesado si en el documento XML no ha recibido ningún valor. Sólo se puede usar con tipo de dato textual.
- *fixed*: Indica el único valor que puede contener el elemento en el documento XML. Sólo se puede usar con tipo de dato textual.

```
<xs:attribute name="moneda" type="xs:string" default="euro" use="required" />
<xs:attribute ref="moneda" use="required" />
```

Tipos de datos

Tipos de datos simples y complejos

Existe dos grandes grupos de tipos de datos que se pueden utilizar en los esquemas XSD:

- **Tipos de datos simples** (*xs:simpleType*): Se dividen en los siguientes.
  - Tipos de datos predefinidos.
  - Tipos de datos construidos con nuestras propias restricciones y basados en los tipos de datos predefinidos.
- **Tipos de datos complejos** (*xs:complexType*): Se dividen en los siguientes.
  - Elementos dentro de otros elementos.
  - Elementos que tienen atributos.
  - Elementos mixtos que tienen datos y otros elementos.
  - Elementos vacíos.

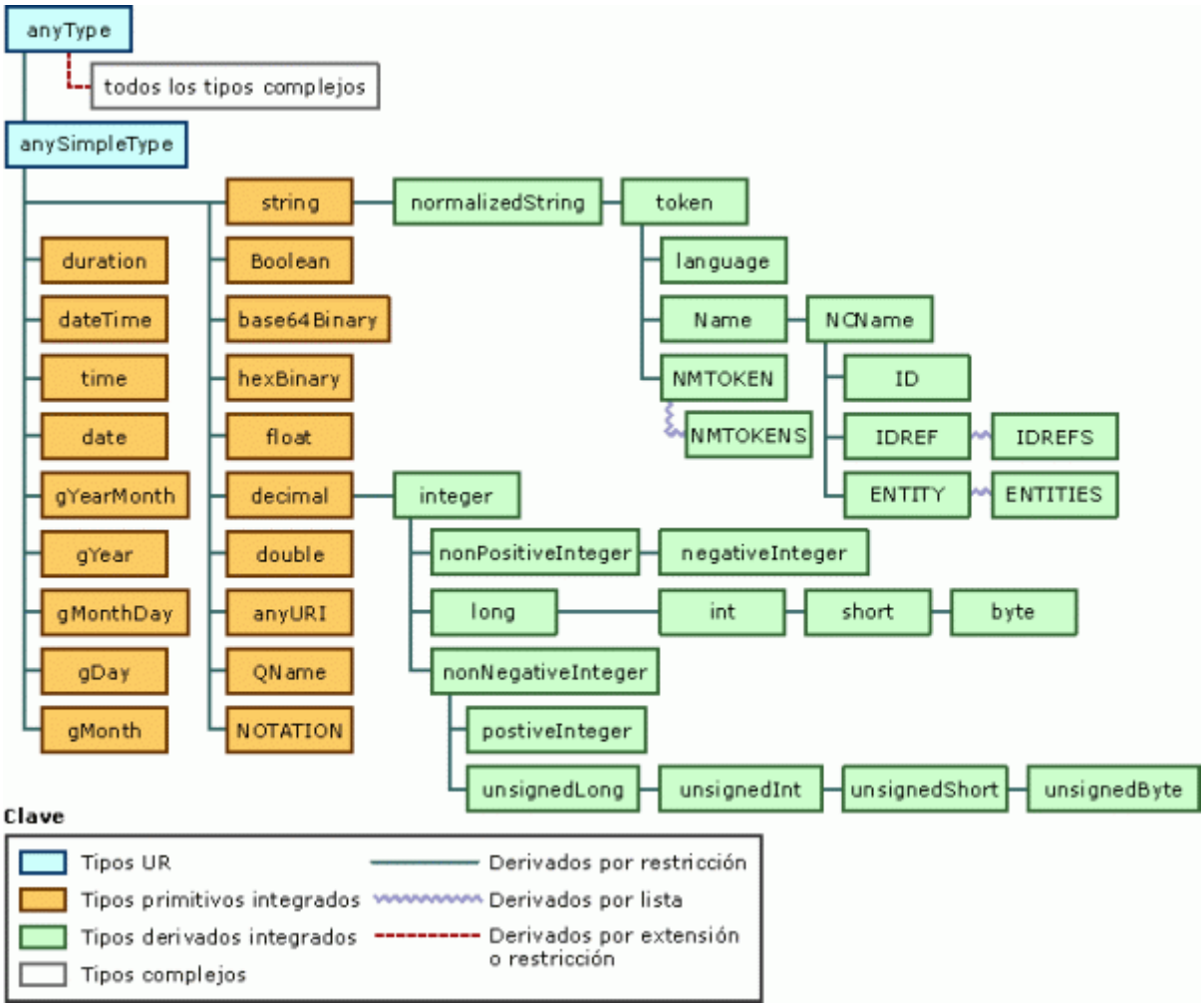
Tipos de datos predefinidos

Son 44 los tipos de datos que define el esquema XSD, clasificados de una manera jerárquica, en el que los elementos hijos poseen las mismas características del padre más alguna particularidad añadida que los distinga

El tipos de datos predefinido principal es *xs:anyType*, el más genérico y del cual derivan el resto de tipos de datos predefinidos como podemos ver en la siguiente imagen.

Los tipos de datos predefinidos los podemos dividir en:

- **Primitivos**: Descienden directamente de *xs:anySimpleType*.
- **No primitivos**: Descienden de alguno de los primitivos.



Tipos de datos predefinidos (imagen extraída de Microsoft)

Los **tipos de datos primitivos** que podemos utilizar son los siguientes.

Tipo de Dato	Descripción
string	Cadena de caracteres.

boolean	Valores booleanos que pueden ser **true** o **false**.
decimal	Números reales.
float	Números en punto flotante de 32 bits.
double	Números en punto flotante de 64 bits.
duration	Duración representada en "Años + Meses + Días + Horas + Minutos + Segundos" (P10Y5M4D10H12M50S).
dateTime	Fecha y hora en formato CCYY-MM-DDThh:mm:ss donde CC es el siglo, YY el año, MM el mes, DD el día, precedido del carácter (-) indica un número negativo. La T es el separador de hh horas, mm minutos, y ss segundos. Puede seguirse de una Z para indicar la zona UTC (Universal Time Coordinated).
time	Hora en formato hh:mm:ss.sss.
date	Fecha en formato CCYY-MM-DD.
gYearMonth	Sólo el año y mes en formato Gregoriano CCYY-MM.
gYear	Sólo el año en formato Gregoriano CCYY.
gMonthDay	Sólo el mes y el día en formato Gregoriano --MM-DD.
gDay	Sólo el día en formato Gregoriano ---DD.
gMonth	Sólo el mes en formato Gregoriano --MM--.
hexBinary	Secuencia de dígitos hexadecimales.
base64Binary	Secuencia de dígitos hexadecimales en base 64.
anyURI	Cualquier identificador URI.
QName	Cualquier nombre cualificado del espacio de nombres
NOTATION	Tipo de datos para atributo compatible con DTD.

Los **tipos de datos no primitivos** que podemos utilizar son los siguientes.

Tipo de dato	Descripción
normalizedString	Representa cadenas normalizadas de espacios en blanco. Este tipo de datos se deriva de string.
token	Representa cadenas convertidas en tokens. Este tipo de datos se deriva de normalizedString.
language	Representa identificadores de lenguaje natural (definidos por RFC 1766). Este tipo de datos se deriva de token.
IDREFS	Representa el tipo de atributo IDREFS. Contiene un conjunto de valores de tipo IDREF.
ENTITIES	Representa el tipo de atributo ENTITIES. Contiene un conjunto de valores de tipo ENTITY.
NMTOKEN	Representa el tipo de atributo NMTOKEN. NMTOKEN es un juego de caracteres de nombres (letras, dígitos y otros caracteres) en cualquier combinación. A diferencia de Name y NCName, NMTOKEN, no tiene restricciones del carácter inicial. Este tipo de datos se deriva de token.

NMTOKENS	Representa el tipo de atributo NMTOKENS. Contiene un conjunto de valores de tipo NMTOKEN.
Name	Representa nombres en XML. Name es un token que empieza con una letra, carácter de subrayado o signo de dos puntos, y continúa con caracteres de nombre (letras, dígitos y otros caracteres). Este tipo de datos se deriva detoken.
NCName	Representa nombres sin el signo de dos puntos. Este tipo de datos es igual que Name, excepto en que no puede empezar con el signo de dos puntos. Este tipo de datos se deriva de Name.
ID	Representa el tipo de atributo ID definido en la recomendación de XML 1.0. El IDno debe incluir un signo de dos puntos (NCName) y debe ser único en el documento XML. Este tipo de datos se deriva de NCName.
IDREF	Representa una referencia a un elemento que tiene un atributo ID que coincide con el ID especificado. IDREF debe ser un NCName y tener un valor de un elemento o atributo de tipo ID dentro del documento XML. Este tipo de datos se deriva de NCName.
ENTITY	Representa el tipo de atributo ENTITY definido en la recomendación de XML 1.0. Es una referencia a una entidad sin analizar con un nombre que coincide con el especificado. ENTITY debe ser un NCName y declararse en el esquema como nombre de entidad sin analizar. Este tipo de datos se deriva de NCName.
integer	Representa una secuencia de dígitos decimales con un signo inicial (+ o -) opcional. Este tipo de datos se deriva dedecimal.
nonPositiveInteger	Representa un número entero menor o igual que cero. nonPositiveInteger consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de integer.
negativeInteger	Representa un número entero menor que cero. Consta de un signo negativo (-) y una secuencia de dígitos decimales. Este tipo de datos se deriva de nonPositiveInteger.
long	Representa un entero con un valor mínimo de -9223372036854775808 y un valor máximo de 9223372036854775807.Este tipo de datos se deriva de integer.
int	Representa un entero con un valor mínimo de -2147483648 y un valor máximo de 2147483647. Este tipo de datos se deriva de long.
short	Representa un entero con un valor mínimo de -32768 y un valor máximo de 32767. Este tipo de datos se deriva

	deint.
byte	Representa un entero con un valor mínimo de -128 y un valor máximo de 127. Este tipo de datos se deriva de short.
nonNegativeInteger	Representa un número entero mayor o igual que cero. Este tipo de datos se deriva de integer.
unsignedLong	Representa un entero con un valor mínimo de cero y un valor máximo de 18446744073709551615. Este tipo de datos se deriva de nonNegativeInteger.
unsignedInt	Representa un entero con un valor mínimo de cero y un valor máximo de 4294967295. Este tipo de datos se deriva de unsignedLong.
unsignedShort	Representa un entero con un valor mínimo de cero y un valor máximo de 65535. Este tipo de datos se deriva deunsignedInt.
unsignedByte	Representa un entero con un valor mínimo de cero y un valor máximo de 255. Este tipo de datos se deriva deunsignedShort.
positiveInteger	Representa un número entero mayor que cero. Este tipo de datos se deriva de nonNegativeInteger.

Tipos de datos construido

Los tipos de datos construidos son generados por el usuario a partir de un tipo de dato predefinido y aplicándoles restricciones si se desea (estas restricciones se explicarán más adelante). Se pueden diseñar de dos maneras.

- Pueden usarse **directamente en la definición de un elemento**, en lugar de usar el atributo *type*:

```
<xs:element name="edad" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- También pueden **definirse asignándoles un nombre** y pudiéndose usar en cualquier elemento del documento mediante el atributo *type.* Esta sección es indiferente colocarla antes o después de la definición del elemento raíz:

```
<xs:simpleType name="longitudMaxima">
  <xs:restriction base="xs:string">
    <xs:minLength value="0"/>
    <xs:maxLength value="10"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:element name="nombre" type="longitudMaxima"/>
```

- Y se pueden construir **tipos de datos extendiendo** otros tipos de datos ya existentes.

```
<xs:complexType name="infoPersona">
  <xs:sequence>
    <xs:element name="nombre" type="xs:string"/>
    <xs:element name="edad" type="xs:integer"/>
  </xs:sequence>
  <xs:attribute name="numero" type="xs:integer"/>
</xs:complexType>

<xs:complexType name="infoPersonaAmpliada">
  <xs:complexContent>
    <xs:extension base="infoPersona">
      <xs:sequence>
        <xs:element name="ciudad" type="xs:string"/>
        <xs:element name="pais" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:element name="persona" type="infoPersonaAmpliada" />
```

Contenido simple y complejo

Dentro de los *tipos de datos complejos*, podemos definir dos tipos de contenidos que pueden ir entre las etiquetas de apertura y cierre de los elementos:

- **Contenido simple** (*xs:simpleContent*): Cuando el elemento declarado sólo tienen contenido textual, sin elementos descendientes.
- **Contenido complejo** (*xs:complexContent*): Cuando el elemento declarado tiene elementos descendientes, pudiendo tener o no contenido textual.

Indicadores

Permiten establecer las características de los elementos que van a utilizarse dentro de otro elemento, por tanto dentro del Contenido complejo.

Indicadores de orden: Asignan el orden de aparición de los elementos descendientes se pueden utilizar las siguientes opciones, tanto individualmente como combinados.

- **Secuencia** (*xs:sequence*): Define el orden exacto de aparición de los elementos.
- **Alternativa** (*xs:choice*): Define una serie de elementos entre los cuales sólo se puede elegir uno de ellos.
- **Todos** (*xs:all*): Define una serie de elementos que pueden aparecer en cualquier orden.

Indicadores de ocurrencia: Asignan cuántas veces puede aparecer dicho elemento.

- **Mínimo** *(minOccurs)*: Indica el mínimo número de ocurrencias que deben aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <*xs:schema*>. Va desde 0 hasta ilimitado (*unbounded*). Por defecto 1.
- **Máximo** *(maxOccurs)*: Indica el máximo número de ocurrencias que pueden aparecer de ese elemento en el documento XML. No se puede usar si el elemento padre es <*xs:schema*>. Va desde 0 hasta ilimitado (*unbounded*). Por defecto 1.

Indicadores de grupo: Agrupan un conjunto de elementos o de atributos.

- **Grupo de elementos** (xs:group): Sirve para agrupar un conjunto de declaraciones de elementos relacionados.
- **Grupo de atributos** (xs:attributeGroup): sirve para agrupar un conjunto de declaraciones de atributos relacionados.

```
<xs:element name="alumno" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nif" type="xs:string" />
      <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="nombre" type="xs:string" minOccurs="1"
maxOccurs="1" />
        <xs:element name="nombre_compuesto" type="xs:string" minOccurs="1"
maxOccurs="1" />
      </xs:choice>
      <xs:element name="apellido" type="xs:string" minOccurs="2"
maxOccurs="2" />
      <xs:group ref="direccion" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:group name="direccion">
  <xs:sequence>
    <xs:element name="calle" type="xs:string"/>
    <xs:element name="localidad" type="xs:string"/>
    <xs:element name="provincia" type="xs:string"/>
```

```
  </xs:sequence>
</xs:group>
```

Restricciones

Permiten establecer restricciones (o también llamadas facetas) a los Contenidos simples.

- **Rango de números**: Se utiliza en los tipos de datos numéricos y de fecha/hora. Define el mínimo y máximo, tanto inclusive como exclusive, de los números permitidos.

```
<xs:element name="edad" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="edad" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minExclusive value="0"/>
      <xs:maxExclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Dígitos**: Se utiliza en los tipos de datos numéricos. Define el número máximo de dígitos permitidos.

```
<xs:element name="telefono" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:totalDigits value="9"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Lista de valores**: Se utiliza en todos los tipos de datos. Define una lista de valores permitidos en el elemento.

```
<xs:element name="coche" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
```

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

- **Longitud**: Se utiliza en tipos de datos de texto. Define el número exacto de caracteres permitidos, o el mínimo y máximo de ellos.

```
<xs:element name="clave" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="clave" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="5"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Plantilla de caracteres**: Se utiliza en tipos de datos de texto. Especifica un patrón o expresión regular que debe cumplir el contenido del elemento (leer más sobre Expresiones regulares).

```
<xs:element name="iniciales" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Za-z][A-Za-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

- **Tratamiento de espacios en blanco**: Se utiliza en tipos de datos de texto. Especifica cómo se tratan los espacios en blanco (que incluyen también saltos de línea y tabuladores). Los puede respetar (*preserve*), los puede reducir a uno (*collapse*) y los puede reemplazar (*replace*).

```
<xs:element name="direccion" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
```

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

Declaración de elementos

En la siguiente tabla podemos observar todas las combinaciones posibles sobre declaraciones de elementos en esquemas XSD (si contienen contenido textual, elementos descendientes o atributos), y que vamos a ir explicando a continuación en los siguientes subapartados.

Tipo dato	Contenido	Contenido textual	Elementos	Atributos	Subapartado
Simple	Simple	X			1. Elementos con contenido textual 2. Elementos con contenido textual restringido
Complejo	Simple	X		X	7. Elementos con contenido textual y atributos 8. Elementos con contenido textual y atributos restringidos 9. Elementos con contenido textual restringido y atributos 10. Elementos con contenido textual restringido y atributos restringidos
Complejo	Complejo				3. Elementos vacíos
Complejo	Complejo			X	6. Elementos vacíos con atributos (con y sin restringir)
Complejo	Complejo		X		4. Elementos que contienen sólo elementos
Complejo	Complejo		X	X	5. Elementos que contienen

Tipo dato	Contenido	Contenido textual	Elementos	Atributos	Subapartado
					sólo elementos y atributos (con y sin restringir)
Complejo	Complejo mixto	X	X		
Complejo	Complejo mixto	X	X	X	

Elementos con contenido textual

Se trata de elementos de tipo de dato simple y contenido simple. Su definición es la más simple. Permite utilizar cualquier tipo de datos predefinido.

```
<xs:element name="color" type="xs:string" default="rojo" minOccurs="1"
maxOccurs="1" />
```

Elementos con contenido textual restringido

Se trata de elementos de tipo de dato simple y contenido simple. Permite utilizar cualquier tipo de datos predefinido, pero sobre él se declara una **restricción** o faceta (facet) para reducir el rango de valores permitidos en el contenido del elemento.

```
<xs:element name="edad" minOccurs="1" maxOccurs="1">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<edad>56</edad>
```

Elementos vacíos

Se trata de elementos de tipo de dato complejo y contenido complejo. Estos elementos no podrán contener ninguna clase de contenido. Si nos fijamos no existe el atributo "type".

```
<xs:element name="h1" minOccurs="1" maxOccurs="1">
  <xs:complexType />
</xs:element>
```

El anterior esquema validaría los siguientes elementos del documento XML:

```
<h1></h1>
<h1 />
```

Elementos que contienen sólo elementos

Se trata de elementos de tipo de dato complejo y contenido complejo. Estos elementos definen los elementos que pueden contener en su interior, así como el orden de ellos, su cardinalidad y su agrupamiento ya vistos anteriormente.

En el siguiente ejemplo, *alumno* debe contener de manera obligatoria y en este orden un *nombre* y dos *apellidos*. El número de veces que se repite *apellidos* lo conseguimos con la cardinalidad.

```
<xs:element name="alumno" minOccurs="1" maxOccurs="1">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nombre" type="xs:string" minOccurs="1"
maxOccurs="1"/>
        <xs:element name="apellido" type="xs:string" minOccurs="2"
maxOccurs="2"/>
      </xs:sequence>
  </xs:complexType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<alumno>
   <nombre>Juan</nombre>
   <apellido>Luque</apellido>
   <apellido>Ostos</apellido>
</alumno>
```

En el siguiente ejemplo combinamos los indicadores de orden para incluir dentro de una secuencia una alternativa entre dos elementos. En este caso, hay que elegir primero entre *nombre* o *nombre_compuesto*, y posteriormente aparecerán dos *apellido.*

```
<xs:element name="alumno" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="1" maxOccurs="1">
```

```
      <xs:element name="nombre" type="xs:string" minOccurs="1"
maxOccurs="1" />
      <xs:element name="nombre_compuesto" type="xs:string" minOccurs="1"
maxOccurs="1" />
    </xs:choice>
    <xs:element name="apellido" type="xs:string" minOccurs="2"
maxOccurs="2" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Elementos que contienen sólo elementos y atributos (con y sin restringir)

Se trata de elementos de tipo de dato complejo y contenido complejo. Estos elementos definen los elementos que pueden contener en su interior, así como el orden de ellos y su cardinalidad. Además pueden definir atributos.

```
<xs:element name="alumno" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="apellido" type="xs:string" minOccurs="2"
maxOccurs="2"/>
    </xs:sequence>
    <xs:attribute name="dni" type="xs:string" />
 </xs:complexType>
</xs:element>
```

Y si queremos meter restricciones en el atributo:

```
<xs:element name="alumno" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      <xs:element name="apellido" type="xs:string" minOccurs="2"
maxOccurs="2"/>
    </xs:sequence>
    <xs:attribute name="dni" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:pattern value="[0-9]{8}[A-Z]" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
 </xs:complexType>
</xs:element>
```

Los anteriores esquemas validarían el siguiente documento XML:

```
<alumno dni="10203040A">
   <nombre>Juan</nombre>
   <apellido>Luque</apellido>
   <apellido>Ostos</apellido>
</alumno>
```

Elementos vacíos con atributos (con y sin restringir)

Se trata de elementos de tipo de dato complejo y contenido complejo. Estos elementos no podrán contener ninguna clase de contenido, pero sí tendrán atributos, que podrán tener o no restricciones.

```
<xs:element name="producto" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:integer" use="required" />
    <xs:attribute name="dni" use="optional">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minLength value="0"/>
          <xs:maxLength value="100"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<producto prodid="132" catid="25" />
```

Elementos con contenido textual y atributos

Se trata de elementos de tipo de dato complejo y contenido simple. Estos elementos podrán contener sólo contenido textual. Se definen atributos para el elemento.

```
<xs:element name="producto" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="prodid" type="xs:integer" use="optional" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
```

```
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<producto prodid="132">Manzana</producto>
```

Elementos con contenido textual y atributos restringidos

Se trata de elementos de tipo de dato complejo y contenido simple. Estos elementos podrán contener sólo contenido textual. Los atributos que se definen poseen restricciones.

```
<xs:element name="producto" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="prodid" use="optional">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[A-Z][0-9]+" />
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<producto prodid="M032">Manzana</producto>
```

Elementos con contenido textual restringido y atributos

Se trata de elementos de tipo de dato complejo y contenido simple. Estos elementos tendrán contenido textual con restricciones. Los atributos se definen sin restricciones.

En este caso, utilizaremos un *tipo de dato construido* con su propio nombre para darle restricciones al contenido del elemento.

```
<!-- Esta sección irá antes o después del elemento raíz -->
<xs:simpleType name="longitudMaxima">
    <xs:restriction base="xs:string">
        <xs:minLength value="0"/>
        <xs:maxLength value="10"/>
```

```
    </xs:restriction>
</xs:simpleType>
<xs:element name="producto" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="longitudMaxima">
        <xs:attribute name="prodid" type="xs:string" use="optional" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

```
<producto prodid="M0876">Manzana</producto>
```

Elementos con contenido textual restringido y atributos restringidos

Se trata de elementos de tipo de dato complejo y contenido simple. Estos elementos tendrán contenido textual con restricciones. Los atributos se definen con restricciones también.

En este caso, utilizaremos un *tipo de dato construido* con su propio nombre para darle restricciones al contenido del elemento.

```
<!-- Esta sección irá antes o después del elemento raíz -->
<xs:simpleType name="longitudMaxima">
    <xs:restriction base="xs:string">
        <xs:minLength value="0"/>
        <xs:maxLength value="10"/>
    </xs:restriction>
</xs:simpleType>
<xs:element name="producto" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="longitudMaxima">
        <xs:attribute name="prodid" use="optional">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:pattern value="[A-Z][0-9]+" />
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

El anterior esquema validaría el siguiente documento XML:

<producto prodid="M0876">Manzana</producto>

Métodos de diseño de esquemas XSD

Hay varios métodos a la hora de enfrentarnos a al diseño de un esquema XSD. Se puede utilizar uno u otro o combinar ambos para realizar un diseño entendible.

Diseño plano (Elementos y atributos referenciados + Tipos de datos construidos)

Consisten en realizar de manera independiente las declaraciones de los elementos, atributos o tipos de datos construidos, para luego referenciarlos en los elementos que lo necesiten. Nos permite reutilizar ese mismo elemento en diferentes partes del mismo sin tener que repetir su declaración. La declaración de los elementos y atributos se realizará fuera elemento raíz, pero es indiferente si se realiza antes o después del mismo.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <!-- Declaración de elementos simples -->
  <xs:element name="titulo" type="xs:string"/>
  <xs:element name="autor" type="xs:string"/>
  <xs:element name="nombre" type="xs:string"/>
  <xs:element name="amigoDe" type="xs:string"/>
  <xs:element name="desde" type= "xs:date"/>
  <xs:element name="calificacion" type="xs:string"/>

  <!-- Declaración de atributos -->
  <xs:attribute name="isbn" type="xs:string"/>

  <!-- Declaración de elementos de complejos -->
  <xs:element name="personaje">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nombre"/>
        <!-- Los indicadores de ocurrencia se utilizan
             al referenciar el elemento y no en su declaración -->
        <xs:element ref="amigoDe" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="desde"/>
        <xs:element ref="calificación"/>
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>

  <!-- Declaración del elemento raíz -->
  <xs:element name="libro">
    <xs:complexType>
      <xs:sequence>
         <xs:element ref="titulo"/>
         <xs:element ref="autor"/>
         <xs:element ref="personaje" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute ref="isbn"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Diseño anidado

Consiste en realizar las declaraciones de los elementos y atributos unos dentro de otros, según van apareciendo en el documento XML. Este método de diseño es muy sencillo, pero puede dar lugar a esquemas XSD difíciles de leer y mantener cuando los documentos son complejos. Además produce duplicidades en la descripción de elementos con tipos iguales y puede haber elementos con igual nombre y distintas descripciones. Es el método de diseño menos recomendable.

```
<?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="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"/>
        <xs:element name= "personaje" minOccurs= "0" maxOccurs ="
unbounded" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="amigoDe" type= "xs:string" minOccurs="0"
maxOccurs="unbounded"/>
              <xs:element name="desde" type="xs:date"/>
              <xs:element name="calificación" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
  </xs:element>
```

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

Generación automática de esquemas XSD

La mayoría de herramientas especializadas en lenguajes XML poseen alguna utilidad para generar de manera automática un esquema XSD a partir de un documento XML bien formado.

Por ejemplo, en XML Copy Editor, podemos encontrar esta opción en el menú *XML -> Create Schema...*

Validación de documentos XML con esquemas XSD

La mayoría de herramientas especializadas en lenguajes XML poseen la funcionalidad de validar documentos XML mediante su esquema XSD asociado.