Declaración en XML Schema XSD

1 Cómo se declaran elementos

Todos los elementos que se vayan a usar en el ejemplar XML tienen que declararse en el esquema. Las declaraciones de elementos en XML Schema tienen esta sintaxis:

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

- **name**: es el nombre del elemento
- **type**: el tipo de elemento. XML Schema define dos tipos de elementos:

  Tipos simples: son elementos que solo pueden contener datos carácter; no pueden incluir otros elementos ni tampoco atributos. Ejemplo:

  ```
  <xsd:element name=”fecha” type=”xsd:date”/>
  ```

  Declaramos un elemento llamado “fecha”, de tipo “date” (el prefijo xsd: indica que este tipo de datos “date” es parte del vocabulario de XML Schema).

  Tipos complejos: estos elementos pueden incluir otros elementos y/o atributos. El contenido de estos elementos se define entre la etiqueta de inicio

  ```
  <xsd:complexType> y la correspondiente de cierre
  </xsd:complexType>. Ejemplo:
  <xsd:element name=”libro”>
  <xsd:complexType>
  <xsd:attribute name=”formato” type=”xsd:string” use=”required”/>
  </xsd:complexType>
  </xsd:element>
  ```

  Declaramos el elemento “libro”, que es de tipo complejo. Entre la etiquetas <xsd:complexType> y la de cierre </xsd:complexType> se especifica la definición de este elemento. En este caso se trata de un elemento que incluye un atributo “formato”, que es de tipo string, y que es obligatorio.

- **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.

A diferencia de lo que ocurre con los valores de atributos, los valores “fixed” y “default” de los elementos solo se añaden al documento XML si el elemento está presente. En el caso de los elementos con un valor “fixed”, el elemento puede estar vacío, o si no lo está, su contenido debe coincidir con el especificado en el atributo “fixed”.

2 Modelos de contenido para elementos

El contenido de un elemento se define mediante un modelo de contenido. En XML Schema existen cuatro modelos de contenido para elementos:

1) Texto: el elemento solo puede contener datos carácter. Ejemplo: veamos una definición de un elemento que solo contiene texto (usamos el tipo “string”, que representa una secuencia de caracteres”)

```
<xsd:element name=”autor” type=”xsd:string”/>
```

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

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

3) Elementos: 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”.

- **El elemento <xsd:sequence>**. Utilizamos 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. Ejemplo:

```
<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>
```

- **El elemento <xsd:choice>**. Especifica una lista de elementos de los cuales solo puede aparecer uno en el documento XML. Ejemplo:

```
<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>
```

```
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.
```

- **El elemento <xsd: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. Ejemplo:

```
<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>
```

4) Mixto (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”. Ejemplo:

```
<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>
```

El elemento definido arriba podría usarse dentro de un documento XML de la siguiente manera:

```
<confimacionPedido>
<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.
</confimacionPedido>
```

3 Referencias a otros elementos

En un schema es posible declarar elementos de forma global y luego hacer referencias a ellos desde otros elementos. 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. 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.

Vamos a ver cómo quedaría el siguiente ejemplo de un XML inicial:

```
<?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>
```

utilizando referencias a elementos:

```
<?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>
```

Tipos de datos simples predefinidos en XML Schema XSD

Los tipos de datos se dividen en tipos de datos simples (o primitivos) y tipos de datos complejos. Los tipos de datos simples se pueden utilizar en los valores de los atributos y en los elementos que contienen sólo datos carácter. Existe una serie de tipos de datos definidos en el estándar y que por tanto se pueden usar directamente en los esquemas. Además de estos, el usuario puede definir sus propios tipos de datos, tanto simples como complejos, como veremos más adelante.
Existen 19 tipos de datos simples predefinidos primitivos, que se pueden agrupar en 4 categorías:

- **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)

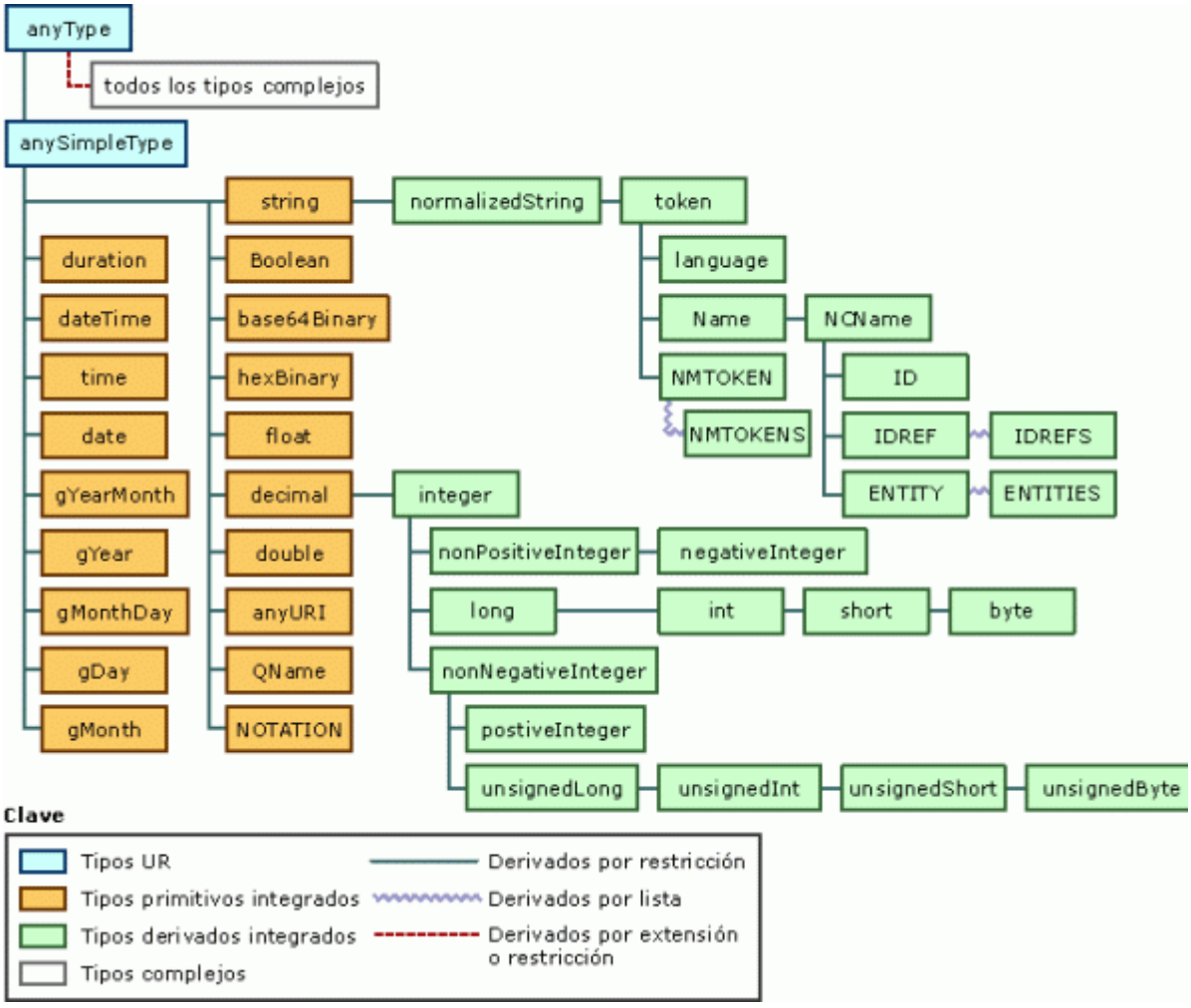
- **Tipos binario codificado**
  - **boolean**: toma los valores “true” o “false” *
  - **hexBinary**: dato binario codificado como una serie de pares de dígitos hexadecimales
  - **base64Binary**: datos binarios codificados en base 64

- **Tipos numéricos**
  - **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 de 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)

De cada uno de estos tipos primitivos se pueden obtener tipos derivados, como se muestra en el siguiente diagrama:



Nota:

Es posible definir tipos de datos simples a partir de estos tipos predefinidos utilizando las llamadas facetas.

Definición de tipos de datos en XML Schema XSD

XML Schema permite al usuario la creación de sus propios tipos de datos. Existen varias opciones para hacer esto.

Crear un tipo complejo con un nombre

La forma más sencilla de crear un nuevo tipo es crear un elemento complexType al que se le asigna un nombre mediante el atributo “name”. Ejemplo:

```
<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”/>
```

Con esto creamos un tipo nuevo llamado “precioInfo” que está formado por una secuencia de dos elementos, “precioTipo” y “precioN”, a los que referencia. Podremos utilizar este tipo que hemos creado para definir elementos, por ejemplo:

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

Observamos que en la definición del elemento precio, el nombre del tipo “precioInfo” no lleva el prefijo “xsd”, porque no pertenece al espacio de nombres del estándar XML Schema. La principal ventaja de definir tipos de datos propios es, por un lado, que estos tipos se pueden utilizar donde se quiera, y además, que estos tipos se pueden utilizar como tipos base para definir otros tipos. A continuación, vamos a ver los mecanismos que existen para definir tipos derivados de otros.

Crear un tipo por restricción de un tipo simple predefinido (xsd:restriction)

La forma más sencilla de crear un nuevo tipo a partir de uno ya existente es añadir condiciones a alguno de los tipos predefinidos en el XML Schema. Esto se hace con el elemento <xsd:restriction>. Por ejemplo:

```
<xsd:simpleType name="monedaUSA">
<xsd:restriction base="xsd:decimal">
<xsd:fractionDigits value="2"/>
</xsd:restriction>
</xsd: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”). Las restricciones (también llamadas facetas) que pueden aplicarse a los tipos simples son:

- ***minInclusive***: mínimo valor que puede tomar el número; por ejemplo, si mininclusive vale 5, el número tiene que ser mayor o igual que 5.
- ***minExclusive***: el número debe ser mayor que este valor; por ejemplo, si minExclusive vale 5, el número debe ser mayor que 5.

- ***maxInclusive***: máximo valor que puede tomar el número; por ejemplo, si maxinclusive vale 5, el número tiene que ser menor o igual que 5.
- ***maxExclusive***: el número debe ser menor que este valor; por ejemplo, si maxExclusive vale 5, el número debe ser menor que 5.
- ***totalDigits***: total de cifras en el número, incluyendo las enteras y las decimales.
- ***fractionDigits***: número de cifras decimales.
- ***length***: número de unidades del valor literal; para la mayoría de los tipos (por ejemplo, los tipos "string" y sus derivados, la faceta "length" se refiere a caracteres, pero para listas (que veremos más adelante) se refiere al número de elementos en la lista, y para valores binarios se refiere al número de octetos. No se puede aplicar a los tipos "integer", "float" o "double" (para estos se puede utilizar "totalDigits").
- ***minLength*** y ***maxLength***: valor mínimo y máximo respectivamente para la faceta "length".
- ***pattern***: formato que debe tener el valor, especificado mediante una expresión regular tradicional.
- ***enumeration***: conjunto de posibles valores que puede tomar el dato (lo veremos en el siguiente apartado)
- ***whiteSpace***: controla la forma que tendrá el contenido de este dato una vez haya sido procesado; puede tomar los siguientes valores:
  - *"preserve"*: los datos no se modifican, quedan tal y como aparecen escritos.
  - *"replace"*: los tabuladores, saltos de línea y retornos de carro son sustituidos por espacios.
  - *"collapse"*: hace lo mismo que "replace", pero además sustituye espacios múltiples por un solo espacio.

Estas facetas se pueden combinar, y se pueden usar tanto en las definiciones de tipos con nombre como en las definiciones de elementos. Veamos algunos ejemplos, en los que las facetas se aplican a definiciones de elementos. Ejemplo 1: facetas "minInclusive" y "maxInclusive".

```
<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>
```

Define el elemento "edad" de tipo simple, como un entero en el rango [0-120].
Ejemplo 2: facetas "minLength" y "maxLength".

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

Define el elemento "contraseña" como una cadena de entre 5 y 8 caracteres.

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. Ejemplo:

```
<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>
</xs: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.

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="listaTiendas">
<xsd:maxLength value="3"/>
</xsd:restriction>
</xsd:simpleType>
```

Se define el tipo “tiendas” como una restricción del tipo “listaTiendas” donde “legnth” no puede ser mayor que 3. Es decir, en “tiendas” puede haber hasta 3 valores de la lista. El tipo “listaTiendas” 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>
```

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”. También es posible definir tipos derivados de los tipos complejos definidos por el usuario, utilizando los mismos mecanismos que hemos visto.