Contenido

[**[3.6] Validación por XML Schema** 2](#_Toc497935254)

[**[3.6.1] XML Schema y DTD** 2](#_Toc497935255)

[**[3.6.2] Ventajas de los esquemas** 2](#_Toc497935256)

[**[3.6.3] Desventajas de los esquemas** 2](#_Toc497935257)

[**[3.6.4] Estructura de los esquemas** 3](#_Toc497935258)

[**[3.7] Definición de componentes XML Schema** 7](#_Toc497935259)

[**[3.7.1] Componentes locales y globales** 7](#_Toc497935260)

[**[3.7.2] Sintaxis de la definición de un elemento XML Schema** 7](#_Toc497935261)

[**[3.7.3] Tipos simples de datos** 8](#_Toc497935262)

[**[3.7.4] Tipos de datos ya implementados** 8](#_Toc497935263)

[**[3.7.5] Definir tipos simples personales** 12](#_Toc497935264)

[**[3.7.6] Establecer restricciones para tipos simples** 12](#_Toc497935265)

[**[3.7.7] Definir tipos simples por lista** 16](#_Toc497935266)

[**[3.7.8] Definiciones locales de tipos** 17](#_Toc497935267)

[**[3.7.9] Definición de atributos en XML Schema** 17](#_Toc497935268)

[**[3.7.10] Tipos compuestos** 18](#_Toc497935269)

[**[3.7.11] Definición de contenidos compuestos** 20](#_Toc497935270)

[**[3.7.12] Contenidos mixtos** 22](#_Toc497935271)

[**[3.7.13] Uso de elementos y atributos globales** 23](#_Toc497935272)

[**[3.7.14] Grupos de elementos** 24](#_Toc497935273)

[**[3.7.15] Grupos de atributos** 24](#_Toc497935274)

**[3.6] Validación por XML Schema**

**[3.6.1] XML Schema y DTD**

Los DTD en realidad son una reminiscencia del lenguaje SGML que permite un mecanismo de validación para XML al ser éste un subconjunto de SGML.

Por ello, se planteó en torno a 1999 una sintaxis que permitiera más capacidades de validación y que fuera más coherente con el lenguaje XML. Hoy en día es la arquitectura predominante para la validación, ya que se usa en **XQuery** y sobre todo en los **servicios Web**.

**[3.6.2] Ventajas de los esquemas**

* La sintaxis es XML, por lo que son analizables como cualquier otro documento XML
* Soportan íntegramente los espacios de nombres
* Permiten validaciones de datos avanzadas
* Proporcionan una mayor facilidad para crear validaciones complejas y reutilizables
* Soportan conceptos avanzados como herencia y sustitución de tipos

**[3.6.3] Desventajas de los esquemas**

**[1]**Son más complejas de entender que las DTD

**[2]**Presentan más incompatibilidades con software que las DTD

**[3]**No permiten definir entidades

**[4]**Tecnologías como **SAX** o **DOM**tienen utilidades especiales para las DTD, pero no para los XML Schemas.

**[3.6.4] Estructura de los esquemas**

Un esquema es un documento XML al que se le coloca la extensión **xsd**. Al ser un archivo XML tiene la estructura habitual de todo documento XML con la obligación de que el elemento raíz se llame **schema**.

1. [**Etiqueta schema**](#EtiquetaSchema)
2. [**Asociar un esquema a un documento XML**](#AsociarEsquema)
3. [**Partes de un esquema**](#PartesEsquema)

**Etiqueta schema**

La etiqueta **schema** identifica la raíz de un documento XML Schema. En esta etiqueta se declara el espacio de nombres estándar que utilizan los esquemas (y que permite diferenciar las etiquetas XML del esquema, respecto a las del documento XML), el cual se puede definir como el espacio de nombres por defecto, definir un prefijo **xs** para él (es la forma habitual) o bien definir un prefijo **xsd**. Es decir estas tres posibilidades:

* <schema xmlns=”http://www.w3.org/2001/XMLSchema”>
* <xs:schema xmlns:xs=”http://www.w3.org/2001/XMLSchema”>
* <xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>

A partir de ahí las etiquetas pertenecientes a XML Schema, usarán el prefijo indicado en su espacio de nombres. Normalmente se usa la segunda opción, la que implica usar el prefijo **xs**. En estos apuntes se dará por hecho que hemos utilizado esa segunda opción.

Las etiquetas privadas, las propias de nuestro documento pueden tener asignado un espacio de nombres. Normalmente se le aplica un espacio de nombres por defecto. De esa forma los elementos sin prefijo serán los nuestros y los que tienen prefijo los correspondientes a la sintaxis de XML Schema. Por ejemplo:

|  |
| --- |
| **<xs:schema** **xmlns:xs=**”http://www.w3.org/2001/XMLSchema”  **xmlns=**”http://www.jorgesanchez.net/doc”  **targetNamespace=**”http://www.jorgesanchez.net/doc”**>** |

En el ejemplo anterior, las etiquetas correspondientes al espacio estándar de XML Schema usarán el prefijo **xs**, mientras que las etiquetas pertenecientes a los documentos XML correspondientes al espacio privado de la dirección *jorgesanchez.net/doc* usarán el espacio de nombres por defecto.

El atributo **targetNamespace** sirve para indicar a qué espacio de nombres se aplican las normas definidas por XML Schema.

En este otro ejemplo:

|  |
| --- |
| **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”  **xmlns:doc=**”http://www.jorgesanchez.net/doc”  **targetNamespace=**”http://www.jorgesanchez.net/doc”**>** |

Ahora los elementos de XMLSchema usarán el prefijo **xs**, mientras que las nuestras etiquetas usan el prefijo **doc**.

Finalmente veamos un ejemplo más enrevesado:

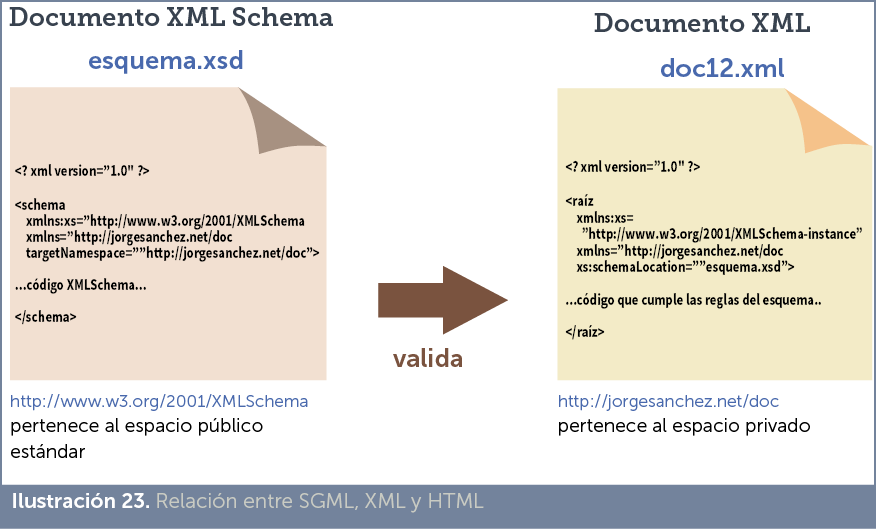
|  |
| --- |
| **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”  **xmlns:doc=**”http://www.jorgesanchez.net/doc”  **xmlns:img=**”http://www.jorgesanchez.net/img”  **targetNamespace=** “http://www.jorgesanchez.net/doc http://www.jorgesanchez.net/img “**>** |

En este último caso, las etiquetas XMLSchema usan el prefijo xs, y nuestras etiquetas usan dos prefijos (puesto que usamos dos espacios de nombres) *doc*e *img*. Ambos espacios están también indicados en el **targetNamespace**; eso indica que el esquema se aplicará a documentos pertenecientes a ambos espacios de nombres.

En todos los casos, a esta etiqueta le siguen los elementos que permiten especificar las reglas de validación del documento XMl Schema.

**Asociar un esquema a un documento XML**

Para que un documento XML siga las reglas definidas en un esquema, no disponemos de etiqueta **!DOCTYPE**; en su lugar utilizamos atributos especiales en el elemento raíz del documento XML.



Primero, al igual que en el documento XMLSchema, necesitamos definir los dos espacios de nombres, el correspondiente al documento XML (que se suele usar sin abreviatura, es decir como espacio por defecto) y el espacio de nombres de XML Schema (que suele utilizar el prefijo **xs**, aunque se puede utilizar otro).

Además es necesario indicar dónde está el archivo XMLSchema que contiene las reglas de validación que se aplican al documento. Esto se hace gracias al atributo llamado **schemaLocation** (perteneciente al espacio de nombres del esquema, por lo que se usa normalmente como **xs:schemaLocation**).

Ejemplo:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>**  **<documento** **xmlns=**”http://www.jorgesanchez.net/doc”  **xmlns:xs=**”http://w3.org/2001/XMLSchema-instance”  **xs:schemaLocation=”esquema.xsd”>**  ....  **</documento>** |

En este código se indica el espacio por defecto de nombres en el documento (coincide con el declarado en el propio archivo del esquema), se indica el espacio de nombres correspondiente al esquema (siempre es la misma URL) y se asocia a este espacio el prefijo **xs**(se puede elegir otro prefijo, pero no es lo normal).

El atributo **schemaLocation** (acompañado del prefijo asociado al espacio de nombres de XMLSchema) indica la localización del documento XMLSchema que contiene la definición de las reglas a cumplir por el documento. Es un par formado por el espacio de nombres que será validado por el esquema y por la ruta al documento XMLSchema (con extensión **xsd**).

Se pueden indicar varios esquemas de validación, por lo que habría que indicar a qué espacio se aplica cada uno:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>**  **<documento xmlns:doc=**”http://www.jorgesanchez.net/doc”  **xmlns:img=**”http://www.jorgesanchez.net/img”  **xmlns:xs=**”http://w3.org/2001/XMLSchema-instance”  **xs:schemaLocation=**  ”http://www.jorgesanchez.net/doc esquemaDoc.xsd  “http://www.jorgesanchez.net/img esquemaImg.xsd”  ....  **</documento>** |

La dirección del documento se ha puesto en estilo de ruta relativa, que se calcula a partir de la dirección del documento XML (es decir el documento XMLSchema en el ejemplo estará en la misma carpeta del XML). Pero lo habitual es que a los esquemas se acceda por URL completa; por ejemplo  
*http://www.obj.com/esq.xsd*

Además podemos indicar un esquema para un documento, pero sin que dicho esquema utilice espacio de nombres. Por ejemplo el esquema podría tener esta cabecera (archivo *esquema1.xsd*):

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schem**a **xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**   **<xs:element name=**”descripción” **type=**”xs:string” **/>  </xs:schema>** |

El archivo XML Schema anterior no indica ningún espacio de nombres al que aplicarse. Un documento que hiciera referencia al esquema podría ser:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<descripción xmlns:xs=**  ”http://www.w3.org/2001/XMLSchema-instance”  **xs:noNamespaceSchemaLocation=**”esquema1.xsd”**>**  **</descripción>** |

El atributo **noNameSchemaLocation** permite indicar un esquema para el documento sin que éste utilice espacio de nombres alguno (lo cual no es nada aconsejable, pero vale para hacer pruebas).

**Partes de un esquema**

* **Elementos**, definidos con etiquetas **xs:element**. Para indicar los elementos permitidos en los documentos que sigan el esquema.
* **Atributos**, etiqueta **xs:attribute**.
* **Tipos simples**, que permiten definir los tipos simples de datos que podrá utilizar el documento XML. Lo hace la etiqueta **xs:simpleType**.
* **Tipos complejos**, mediante la etiqueta **xs:complexType**.
* **Documentación**, información utilizable por aplicaciones que manejen los esquemas. Etiquetas **xs:annotation**, **xs:documentation** y **xs:appInfo**.

**[3.7] Definición de componentes XML Schema**

**[3.7.1] Componentes locales y globales**

El orden de los elementos en un esquema no es significativo, es decir las declaraciones se pueden hacer en cualquier orden. Pero sí que hay que tener en cuenta que dependiendo de dónde coloquemos la definición de los elementos del esquema, varía su ámbito de aplicación. Se distinguen dos posibilidades de declarar elementos:

* En **ámbito global**. Se trata de los elementos del esquema que se coloquen dentro de la etiqueta raíz **schema** y que no están dentro de ninguna otra. Estos elementos se pueden utilizar en cualquier parte del esquema.
* En **ámbito local**. Se trata de elementos definidos dentro de otros elementos. En ese caso se pueden utilizar sólo dentro del elemento en el que están inmersos y no en todo el documento. Es decir si, por ejemplo, si dentro de la definición de un atributo colocamos la definición de un tipo de datos, este tipo de datos sólo se puede utilizar dentro del elemento **xs:attribute** en el que se encuentra la definición del tipo de datos.

Ejemplo:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>**  **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”  **xmlns:doc=**”http://www.jorgesanchez.net/doc”  **targetNamespace=**”http://www.jorgesanchez.net/doc”**>**  **<xs:element …>** <!--Definición global  **<xs:simpleType …**> <!--Definición local,  …  **</xs:simpleType>**  **…**  **</xs:element >**  **<xs:simpleType …>** <!--Definición global  …  **</xs:simpleType …>**  **</xs:schema>** |

El componente local definido sólo se podría utilizar en la zona resaltada. Si fuera global se podría utilizar en todo el documento.

**[3.7.2] Sintaxis de la definición de un elemento XML Schema**

En XML Schema la definición de un elemento XML se realiza mediante la etiqueta **element**. La sintaxis completa es:

|  |
| --- |
| **<xs:element**  **name=”**nombre del elemento**”**  **type=”**tipo global de datos**”**  **ref=”**declaración del elemento global**”**  **id=”**identificador**”**  **form=”**cualificación**”** <!--qualified o unqualified -->  **minOccurs=”**número mínimo de veces**”**  **maxOccurs=”**máximo número de veces**”**  **default=”**valor por defecto**”**  **fixed=”**valor fijo**”**  **>** |

Al menos hay que indicar el nombre; el tipo de datos también es necesario indicarle casi siempre; el resto de atributos sólo si se necesitan. Por ejemplo se puede definir un elemento como:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**   **<xs:element** **name=**”descripción” **type=**”xs:string” **/>  </xs:schema>** |

El elemento descripción será de tipo string. Los tipos de datos en XML Schema son muchos y además permiten personalizar sus posibilidades para adaptarles a cualquier necesidad.

Poco a poco todos los atributos de la etiqueta **element** se irán describiendo a lo largo del presente documento.

**[3.7.3] Tipos simples de datos**

Tenemos dos posibles variantes:

* **Primitivos**. Los tipos más básicos de XML, están ya definidos por el propio lenguaje XML.
* **Derivados**. Tipos de datos más complejos creados a partir de los anteriores.

**[3.7.4] Tipos de datos ya implementados**

1. [**Tipos de datos primitivos**](#DatosPrimitivos)
2. [**Tipos de datos derivados**](#DatosDerivados)
3. [**Tipos equivalentes a los existentes en el lenguaje DTD**](#DatosEquivalentesDTD)
4. [**Uso de tipos de datos en componentes de XMLSchema**](#UsoTiposDatos)

**Tipos de datos primitivos**

Son los tipos básicos de XML. Sirven para formar los tipos derivados y tipos más complejos.

Para usarlos basta indicarlos en el atributo **type** de una etiqueta **element** o **attribute**(que son las que permiten crear elementos y atributos). De esta forma:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**  **<xs:element name=**”descripción” **type=”xs:string”/>**  **</xs:schema>** |

Él código remarcado es el que indica que en los documentos XML basados en esta plantilla XMLSchema, habrá un elemento llamado *descripción*que contendrá datos de tipo *string*. El hecho de que se use el prefijo **xs** (es decir *xs:string*) es para indicar que es un tipo XML y por lo tanto usa el prefijo designado al espacio de nombres de XMLSchema. Los tipos básicos son:

**textos**

* **string**. Representan textos con cualquier contenido excepto los símbolos **<**,**> &**y las comillas para los que se usará la entidad correspondiente.

**lógicos**

* **boolean.**Sólo puede contener los valores verdadero o falso, escritos como **true** o **false**, o como **1** (verdadero) o **0** (falso)

**números**

* **integer. Permite utilizar números enteros.**
* **float**. Permite utilizar números decimales en formato de coma flotante de precisión simple. El separador decimal es el punto. Ejemplos de valores:*1, 2 3.4, -123,-124.76, 1E+3*(significa 1000), *1.45E+4* (significa 14500),   
  *1.45E-4* (significa 0,000145, es decir 1,45 · 10-4), *INF* (infinito), *-INF* (menos infinito), *NaN*(significa *Not a Number*, no es número, para números no válidos)
* **double**. Números decimales en formato de coma flotante de precisión doble. Es decir el mismo tipo de números pero con mejor precisión.
* **decimal**. Representa números decimales en coma fija. Ocupan más internamente pero se representan de forma exacta. No admite los valores **INF**, **NaN** ni tampoco el formato científico.
* **hexBinary**. Representa números binarios codificados en notación hexadecimal.
* **base64Binary**. Representa números binarios (usando base 64).

**fechas**

Se basan en la especificación **ISO 8601**.

* **duration**. Representa duraciones de tiempo en formato **ISO 8601** (sección 5.3)**P**n**Y**n**M**n**DT**hmsDonde todos los signos *n* son números. Ejemplos de valores:
  + *P1Y*, significa *Un año*.
  + *P1Y2M,*significa *un año y dos meses.*
  + *P1Y2M3D,*significa *un año y dos meses y tres días*
  + *P3D,*significa *tres días*
  + *P1Y2M3DT12H30M40.5S,*significa *un año, dos meses, tres días, doce horas treinta minutos y cuarenta segundosy medio*
  + *PT12H30M40.5S,*doce horas treinta minutos y cuarenta segundosy medio

Como se observa la **P** es obligatoria y la **T**sirve para separar los valores de fecha de los valores hora.

* **dateTime**. Representa fechas según el formato **ISO 8601** (sección 5.4). El formato es **yyyy-mm-ddThh:mm:ss** , por ejemplo *1998-07-12T16:30:00.000* (12 de julio de 1998 a las 16:30). La **T** es obligatoria para separar la fecha de la hora.
* **time**. Representa horas en el formato *hh:mm:ss*
* **date**. Representa fecha en formato *yyyy-mm-dd*
* **gYearMonth**. Representa un mes y un año en formato *yyyy-mm*
* **gYear**. Representa un año usando cuatro cifras.
* **gMonthDay**. Representa un mes y un día en formato *--mm-dd*
* **gDay**. Representa un día. Hay que hacerlo indicando tres guiones por delante (por ejemplo *---12*)
* **gMonth**. Representa un mes en formato *--mm*,por ejemplo *--05*

**especiales**

* **anyURI**. Representa una dirección URI. Por ejemplo [http://www.jorgesanchez.net](http://www.jorgesanchez.net/). La URI puede ser absoluta o relativa.
* **QName**. Nombre cualificado. Representa un nombre XML válido para identificar nombres incluyendo el prefijo de espacio de nombres. Por ejemplo *doc:cabecera*
* **Notation**. Representa notaciones de estilo **NOTATION** XML 1.0 segunda edición. Sólo se debe utilizar para crear tipos de datos derivados de éste
* **anyType**. No restringe el contenido en modo alguno.

**Tipos de datos derivados**

Son datos que se han definido a partir de los anteriores, pero forman parte de XMLSchema, es decir que en la práctica se usan igual que los anteriores (al igual que en los primitivos, cuando se usan en un esquema hay que añadir el prefijo del espacio de nombres del esquema, por ejemplo *xs:normalizedString*).

**textos**

* **normalizedString**. Se basa en el tipo **string**. Texto donde los caracteres de retorno de línea, tabulador y retorno de carro se convierten a espacios antes de procesar el esquema.
* **token**. Se basa en el anterior. Textos en los que no hay más de un espacio en blanco seguido, ni tabuladores ni saltos de línea; en todos esos casos se convierte el texto a un único espacio.
* **language**. Texto que contiene el nombre de un lenguaje según lo definido en la especificación oficial **RFC 1766.** Son los posibles valores de los atributos **xml:lang** de XML 1.0 que coinciden con el formato normalizado de lenguajes habitual en las páginas web; por ejemplo el español se codifica con *es*(a veces con*es-ES*), el catalán *ca*, el gallego *gl*, el portugués *pt*, el euskera *eu*, el inglés *en*, alemán *de* y francés *fr*.
* **Name.** Sólo admite nombres compatibles con la forma de poner nombres de **XML.**Admite los dos puntos pero para manejar nombres con prefijo (nombres cualificados) el tipo idóneo es**QName.**
* **NCName.**Nombres, basado en **Name**, pero sin admitir los dos puntos de los prefijos de espacios de nombres.

**números**

* **integer.**Basado en **number***.*Representa números enteros tanto positivos como negativos
* **nonPositiveInteger***.*Basado en **integer***.* Representa números enteros que no son positivos (es decir cero y negativos).
* **negativeInteger***.*Basado en **integer**. Representa números enteros negativos (no vale el cero).
* **nonNegativeInteger***.*Basado en **integer**. Representa números enteros que no son negativos (es decir cero y positivos).
* **positiveInteger***.*Representa números enteros positivos puros (no vale el cero).
* **long***.*Basado en **integer***.*Representa números enteros de alto rango (de *-9223372036854775898* a *9223372036854775897*). Los integer representan números aún más largos
* **unsignedLong***.*Basado en **long***.*Representa números enteros de alto rango pero usando sólo los positivos y el cero.
* **int***.*Representa números enteros de medio rango (de –2147483648 a 2147483647).
* **unsignedInt***.*Representa números enteros de medio rango pero usando sólo los positivos y el cero.
* **short***.*Representa números enteros de bajo rango (de *–32768*a *32767*).
* **unsignedShort***.* Representa números enteros de medio rango pero usando sólo la parte positiva, de cero a *65535*.
* **byte***.*Representa números enteros pequeños (de*–128*a *127*).
* **unsignedByte***.*Representa números enteros pequeños positivos, de cero a 255.

**Tipos equivalentes a los existentes en el lenguaje DTD**

Están basados en los tipos XML 1.0 y sólo pueden utilizarse en atributos (para mantener la compatibilidad con DTD).

* **ID, IDREF e IDREFS**. Equivalente a los atributos del mismo tipo de XML 1.0 (tienen el mismo significado que en las DTD). Derivan de **NCName**
* **ENTITY, ENTITIES**. Equivalente a los atributos XML 1.0 del mismo nombre. Derivan de **NCName**,
* **NMTOKEN, NMTOKENS**. Permiten indicar textos compatibles con los nombres XML. Derivan de **NCName**.
* **NOTATION**. Es un tipo pensado para hacer anotaciones, su funcionamiento es peculiar y no está pensado para ser usado como tipo básico, sino como base para crear tipos personales.

**Uso de tipos de datos en componentes de XMLSchema**

Para que un determinado componente del esquema (como un elemento o un atributo) use uno de los tipos, se dispone del atributo **type** al que se le indicaría el tipo de datos.

Ejemplo:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**  **xmlns:doc=**”http://www.jorgesanchez.net/doc”  **targetNamespace=**”http://www.jorgesanchez.net/doc”**>**  **<xs:element name=**”documento” **type=**”xs:NCName”**/>**  **<xs:element name=**”contenido” **type=**”xs:string” **/>**  **<xs:element name=**”precio” **type=**”xs:decimal” **/>** **</xs:schema>** |

En el ejemplo se trata de las primeras líneas de un código XMLSchema donde se definen tres elementos, cada uno de los cuales con un tipo distinto. Observar el uso del prefijo *xs* asignado al espacio de nombres de **XMLSchema** antes del nombre del tipo de datos.

**[3.7.5] Definir tipos simples personales**

La sintaxis general es:

|  |
| --- |
| **<**xs:**simpleType** **name=”**nombre**”>**  ...definición del tipo....  </xs:**simpleType**> |

**Definir tipos por unión**

Se trata de utilizar dentro del tipo de datos una etiqueta llamada **union** que permite unir las definiciones de dos tipos de datos. Por ejemplo:

|  |
| --- |
| **<xs:simpleType name=**”gMonthC”**>  <xs:union memberTypes=**”xs:gMonth xs:gMonthDay” **/> </xs:simpleType>** |

Cuando a cualquier elemento del esquema se le asigne el tipo *gMonthC*, se podrán especificar datos en formato *gMonth* y en formato *gMonthDay*.

**[3.7.6] Establecer restricciones para tipos simples**

Permiten establecer reglas complejas que deben de cumplir los datos. En este caso dentro de la etiqueta **simpleType** se indica una etiqueta **restriction**, dentro de la cual se establecen las posibles restricciones. Sintaxis:

|  |
| --- |
| **<**xs:**simpleType** **name=”**nombre**”>**  <xs:**restriction base=”**tipo**”**>  ...definición de la restricción....  <xs:**restriction**>  </xs:**simpleType**> |

El atributo **base** sirve para indicar en qué tipo nos basamos al definir la restricción (es decir de qué tipo estamos creando este derivado). El apartado restriction pueden tener numerosas etiquetas que permiten establecer las restricciones deseadas al tipo.

Las etiquetas interiores a **restriction** disponen de un atributo llamado **fixed**quesólo puede valer verdadero (**true**) o falso (**false**). En caso de que sea verdadero, ningún tipo derivado del definido puede modificar la propiedad establecida; es decir, si establecemos **minLength**(tamaño mínimo) con valor *ocho* (propiedad **value**) y **fixed=”true”**, ningún tipo derivado del definido podrá definir que el tamaño mínimo sea inferior a 8 caracteres. Es un atributo de uso opcional.

Las posibles restricciones que se pueden establecer son:

* **Tamaños de texto.**Indica tamaños máximos y mínimos que debe de tener el texto. Ejemplo:
  + **minLength**. Indica el mínimo número de caracteres. Eso lo hace mediante el atributo **value,**en el que se indica un número con el tamaño mínimo que deseamos.
  + **maxLength**. Indica un tamaño máximo de caracteres o de dígitos numéricos. Usa el mismo atributo **value**.
  + **length**. Indica un tamaño fijo de caracteres para el tipo. Es decir si indicados *length*con *value=”9”* el texto deberá tener exactamente nueve caracteres.

Ejemplo:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”  **xmlns:doc=**”http://www.jorgesanchez.net/doc”  **targetNamespace=**”http://www.jorgesanchez.net/doc”**>**  **<xs:simpleType name=**”nombresTipo”**>**  **<xs:restriction base=**”xs:normalizedString”**>**  **<xs:maxLength value=**”15” **/>**  **<xs:minLength** **value=**”4” **/>**  **</xs:restriction>**  **</xs:simpleType>**  **<xs:element name=**”título” **type=**”doc:nombresTipo” **/>** |

* En el ejemplo, se define el tipo persona *nombresTipo* que permite textos entre 4 y 15 caracteres. El hecho de declararle como derivado de **normalizedString** permite restringir que no considere en los textos más de un espacio seguido ni los tabuladores ni saltos de línea (de otro modo casi siempre se superaría el mínimo de cuatro, quizá incluso fuera mejor derivar de *NCNames* que es aún más restrictivo).
* **Dígitos máximos.**Parecido al anterior pero trabajando con números. Indica las posibles cifras que puede tener el número.
  + **totalDigits.**Número máximo de dígitos del número, incluyendo los decimales. El atributo value indica el número máximo deseado
  + **fractionDigits.**Máximo número de decimales que puede tener el número.

|  |
| --- |
| **<xs:simpleType name=**”tipo1”**>  <xs:restriction** **base=**”xs:decimal”**>**  **<xs:totalDigits value=**”6” **/>  <xs:fractionDigits** **value=**”2” />  **</xs:restriction>** **</xs:simpleType> <xs:element** **name=**”para” **type=**”tipo1” **/>** |

* En el ejemplo, los elementos que utilicen el *tipo1* definido, podrán escribiré números de hasta seis cifras (incluidos
* **Máximos y mínimos numéricos**. Restringe valores numéricos asignando topes a los mismos. Sirve para números y para valores de tiempo o duración. Se hace con:
  + **minExclusive**. Establece un valor mínimo. El valor debe ser mayor que el establecido por la etiqueta a través del atributo **value**.
  + **maxExclusive**. Establece un valor máximo. El valor debe ser menor que el establecido por la etiqueta a través del atributo **value**.
  + **minInclusive**. Establece un valor mínimo. El valor debe ser mayor o igual que el establecido por la etiqueta a través del atributo **value**.
  + **maxInclusive**. Establece un valor máximo. El valor debe ser menor o igual que el establecido por la etiqueta a través del atributo **value**.
* **Espacios en blanco.**Sirve para indicar la política de manejo de espacios en blanco en los textos. La etiqueta que lo controla es whiteSpace y tiene tres posibles valores para el atributo value:
  + **preserve.**No modificar espacios en blanco, ni tabuladores ni saltos de línea. Es decir se les tendrá en cuenta.
  + **replace.** Cada doble espacio o tabulador o salto de línea se cambia por un espacio (al estilo del tipo predefinido **normalizedString)**
  + **collapse.**Como el anterior, pero además elimina los espacios a izquierda y derecha. Es muy útil para usar en combinación con las propiedades de tamaño de texto vistas anteriormente.

Ejemplo (archivo *prueba.xsd*):

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**  **<xs:simpleType name=**”tipo1”**>  <xs:restriction base=**”xs:string”**>  <xs:whiteSpace value=**”collapse” **/>  </xs:restriction>  </xs:simpleType>  <xs:element name=**”prueba” **type=**”tipo1” **/>  </xs:schema>** |

En el archivo anterior, se define un elemento llamado prueba que tiene un tipo que colapsa los espacios en blanco y así en un documento XML que aplique este esquema, por ejemplo:

|  |
| --- |
| **<?xml version=**”1.0” **encoding=**”UTF-8”**?>** **<prueba xmlns:xs=**  ”http://www.w3.org/2001/XMLSchema-instance”  **xs:noNamespaceSchemaLocation=**”prueba.xsd”**>**  Este es el texto   que deseo     probar **</prueba>** |

Cuando mostremos el resultado en un programa que aplique el esquema (por ejemplo en las últimas versiones de **Internet Explorer, Firefox, Safari**o**Chrome**), tendremos el contenido de la etiqueta prueba en esta forma:

|  |
| --- |
| Este es el texto que deseo probar |

* **Enumeraciones**. Limitan el contenido a una lista de valores. Se realizan con una sucesión de etiquetas **enumeration**. Estas etiquetas se pueden combinar con las propiedades anteriores (aunque no tiene sentido) y no poseen atributo **fixed**. De modo que si un tipo deriva de una enumeración, podrá volver a enumerar para indicar valores válidos y estas deberán estar en la enumeración anterior; una vez más podremos restringir más, pero nunca menos.

Ejemplo de enumeración:

|  |
| --- |
| **<xs:simpleType name=**”sexoTipo”**>**  **<xs:restriction base=**”xs:NCName”**>**  **<xs:enumeration value=**”Hombre”**/>**  **<xs:enumeration value=**”Mujer”**/>**  **</xs:restriction>**  **</xs:simpleType>** |

Mediante este tipo sólo podremos elegir como valores *Hombre* o *Mujer*.

* **Plantillas** (*pattern*). Permite establecer **expresiones regulares;** es decir, un texto con símbolos especiales que permiten establecer expresiones que ha de cumplir el contenido. Las expresiones regulares son bien conocidas por casi todos los programadores y dan una potencia increíble para establcer restricciones de texto avanzadas, lo que las hace muy utilizadas.

Las plantillas se manejan con etiquetas **pattern** a las que, en el atributo **value**, se indica un texto con símbolos especiales que especifica la expresión a cumplir. Los símbolos que se pueden utilizar son:

| **Símbolo** | **Significado** |
| --- | --- |
| *texto tal cual* | Hace que sólo se pueda escribir ese texto. Por ejemplo si se indica *“Hombre”*, la restricción será escribir como valor posible exactamente el texto *Hombre*. |
| **[***xyz***]** | Permite elegir entre los caracteres*x, y*o *z* |
| **[^***xyz***]** | Prohíbe usar cualquiera de los caracteres entre corchetes |
| **[***a-z***]** | Vale cualquier carácter de la *a,*a la *z*. |
| **^** | Inicio de línea |
| **$** | Final de línea |
| **+** | Repite acepta el carácter precedente una o más veces |
| **?** | Acepta el carácter precedente 0 o más veces |
| **\*** | Acepta el carácter precedente una o más veces |
| **{***n***}** | Acepta exactamente *n* repeticiones del carácter precedente. |
| **{***n,***}** | Acepta al menos *n* repeticiones del carácter precedente. |
| **{***n,o***}** | Acepta entre *n* y *o* repeticiones del carácter precedente. |
| \*s* | Permite indicar los caracteres especiales. Por ejemplo \^ representa el carácter circunflejo ^ para que sea tomado como texto y no como código especial. |

Por ejemplo la validación para un dato tipo DNI (8 cifras y un número), sería:

|  |
| --- |
| **<xs:simpleType name=**”dniTipo”**>  <xs:restriction base=**”xs:string”**>  <xs:whiteSpace value=**”collapse” **/>  <xs:pattern value=**”[0-9]{8}[A-Z]” **/>  </xs:restriction> </xs:simpleType>** |

* **enumeraciones**. Las realiza una etiqueta llamada **enumeration** que sirve para indicar los posibles valores que puede tomar un componente. Por ejemplo:

|  |
| --- |
| **<xs:simpleType name=**”diasSemanaTipo”**>  <xs:restriction base=**”xs:string”**>  <xs:whiteSpace value=**”collapse” **/>  <xs:enumeration value=**”Lunes” **/>  <xs:enumeration value=**”Martes” **/>  <xs:enumeration value=**”Miércoles” **/>  <xs:enumeration value=**”Jueves” **/>  <xs:enumeration value=**”Viernes” **/>  <xs:enumeration value=**”Sábado” **/>  <xs:enumeration value=**”Domingo” **/>  </xs:restriction> </xs:simpleType>** |

**[3.7.7] Definir tipos simples por lista**

Las listas permiten que un componente tenga como contenido una determinada lista de valores. la construcción de listas indica dos pasos:

**[1]**Crear un tipo simple de datos cuyo contenido es una etiqueta *list*, la cual posee el atributo **itemType** para indicar el tipo de elementos de la lista. Simplemente con ello e podría establecer una lista (valores separados por espacios) de valores pertenecientes al tipo indicado.

**[2]**Crear el tipo ya definitivo de datos que contendrá una etiqueta **restriction** a la cual como tipo base se indica el tipo simple de datos relacionado con la lista. EN la restricción se pueden indicar estas etiquetas usando el atributo **value**:

* **length**. Indica que la lista tendrá un número exacto de valores.
* **minLength.**Indica que la lista tendrá un número mínimo de valores
* **maxLength**. Indica que la lista tendrá un número máximo de valores
* **enumeration**. Posibles valores que puede tener la lista
* **whiteSpace**. Gestión de los espacios en blanco en cada elemento de la lista.
* **pattern**. Expresión regular que debe cumplir cada elemento de la lista.

**[3]**Es decir, son las etiquetas ya conocidas, pero que ahora se refieren a la lista. Ejemplo:

|  |
| --- |
| **<xs:simpleType name=**”listaDecimales”**>  <xs:list itemType=**”xs:decimal” **/> </xs:simpleType> <xs:simpleType name=**”listaNotas”**>  <xs:restriction base=**”listaDecimales”**>  <xs:minLength value=**”3” **/>  <xs:maxLength value=**”6” **/>  </xs:restriction> </xs:simpleType>** |

**[4]**Primero se define un tipo de lista (*listaDecimales*) simplemente indicando que será una lista cuyos valores serán números decimales. Después se concreta la lista (*listaNotas*), de modo que ahora se indica que la lista constará de tres a seis números (en el XML cada número irá separado por espacios)

**[3.7.8] Definiciones locales de tipos**

Como se comentó anteriormente, los componentes de XML Schema pueden definirse en modo local o global. En modo global cualquier otro componente le puede utilizar, en modo local sólo aquel que contiene la definición. Los datos simples se pueden crear, por ejemplo, en un elemento y así se aplican sólo a dicho elemento.

Ejemplo:

|  |
| --- |
| **<xs:element** **name=**”elemento1”**>**  **<xs:simpleType>  <xs:restriction base=**”xs:string”**>  <xs:whiteSpace value=**”collapse” **/>  <xs:maxLength value=**”16” **/>  </xs:restriction>  </xs:simpleType> </xs:element>** |

El tipo simple declarado ni siquiera tiene nombre, ya que no se utilizará fuera del elemento *elemento1*.

**[3.7.9] Definición de atributos en XML Schema**

Los atributos se definen parecido a los elementos. Su sintaxis de definición es:

|  |
| --- |
| **<xs:attribute**  **name=”**nombre del elemento**”**  **type=”**tipo global de datos**”**  **ref=”**declaración del elemento global**”**  **form=”**cualificación**”** <!--qualified o unqualified  **id=”**identificador**”**  **default=”**valor por defecto**”**  **fixed=”**valor fijo**”**  **use=**”uso” <!-- prohibited, optional o required  **>** |

Los atributos sólo pueden asignarse a los elementos mediante tipos complejos de datos. El uso hace referencia a si al atributo no se le puede asignar ningún valor en el documento XML (valor **prohibited**), si su uso es opcional (valor **optional**) o si es obligatorio darle valor.

**[3.7.10] Tipos compuestos**

Los tipos de datos del apartado anterior sólo sirven para indicar contenidos simples (que contienen sólo información en el interior) de elementos o bien para indicar posibles valores a los atributos. Los tipos compuestos permiten definir contenidos más complejos. Puesto que lo normal es que los elementos de un documento XML puedan contener otros elementos y por supuesto atributos, es lógico que la mayoría de elementos indiquen mediante tipos compuestos su contenido. Los datos simples son apropiados para indicar el tipo de contenido de los atributos o bien para indicar el contenido de los elementos simples (lo que en DTD serían elementos sólo con contenido **# PCDATA**).

Al igual que los datos simples, los compuestos pueden ser globales o locales. En el caso de ser locales no se indica un nombre (atributo **name**) y entonces sólo se podrán utilizar para el elemento en el que se definieron. Los globales se pueden utilizar para distintos elementos y por lo tanto requieren que se indique su nombre.

Los tipos compuestos se definen con la etiqueta **complexType**. En esa etiqueta podemos utilizar diferentes modelos de definición de contenidos.

En realidad los elementos XML desde el punto de vista de la sintaxis XML Schema pueden tener cuatro tipos de contenido:

* **Contenido simple**(*Simple Content*). Sólo admite texto en su interior y no otros elementos (son los elementos definidos en DTD como **(#PCDATA)**)
* **Vacíos (***Empty***)**. No pueden contener ni más elementos dentro ni texto.
* **Contenido compuesto** (*Complex Content*). Pueden contener otros elementos, pero no **PCDATA**.
* **Contenido mixto (***Mixed Content***)**. Pueden contener tanto texto como más elementos.

La sintaxis de la etiqueta **complexType** admite señalar contenidos simples y compuestos. Para los vacíos y mixtos hay que utilizar (se verá más adelante) atributos especiales en la etiqueta **element**. Sin indicar nada especial, **compexType** parte de que estamos definiendo contenidos complejos (es decir, elementos que contienen más elementos).

Los elementos pueden contener atributos, más adelante se indica la forma de incorporarlos.

**Elementos vacíos**

Para indicar que un elemento es vacío basta con no indicar valores e indicar el nombre del elemento sin indicar tipo de datos alguno.

Ejemplo:

|  |
| --- |
| **<xs:element name=**”casado” **>  <xs:complexType>  <xs:attribute name=**”valor” **type=**”xs:string” **/>  </xs:complexType> </xs:element>** |

En este caso el elemento caso sólo dispone de un atributo llamado valor, no será posible meter ningún contenido en la etiqueta de casado. Si no deseamos atributos (aunque es muy extraño), entonces simplemente no habrá etiquetas **attribute** (pero sí todas las demás).

**Definición de contenidos con texto**

Se trata de usar la etiqueta **<xs:element** al estilo que se ha usado en los ejemplos. Es decir se indica el nombre y el tipo de datos e incluso se pueden indicar atributos dentro del apartado **complexType**, pero nunca se ponen elementos dentro de este elemento y así sólo se admitirá por contenido el texto (tipo **PCDATA**).

Ejemplo:

|  |
| --- |
| **<xs:element name=**”nombre” **type=**”xs:string” **/>** |

**Definición de contenidos con texto y atributos**

En este caso se indica que el elemento posee contenido simple en la etiqueta **complexType**dentro de la cual se indican los atributos, mientras que es la etiqueta del contenido simple (**simpleContent**) la que poseerá el tipo de datos para el contenido del elemento.

Hay dos formas de indicar contenido simple: por extensión (mediante etiqueta **extension**) y por restricción (etiqueta **restriction**).

Los atributos se deben indicar en el apartado **extension** que es el encargado de indicar los tipos .

Ejemplo:

|  |
| --- |
| **<xs:element name=**”documento”**>  <xs:complexType>  <xs:simpleContent>  <xs:extension base=**”xs:string”**>  <xs:attribute name=**”idioma”  **type=**”xs:string” **/>  </xs:extension>  </xs:simpleContent>  </xs:complexType> </xs:element>** |

En el ejemplo, documento es un elemento de tipo string (texto) que contiene un atributo llamado idioma (también string). Es un poco enrevesado, pero es necesario hacerlo así.

**[3.7.11] Definición de contenidos compuestos**

Como se ha comentado antes, los contenidos compuestos se refieren a los elementos que contienen otros elementos (pero nunca texto libre). Hay tres posibles tipos de elementos a contener: **secuencias**, **elecciones** y contenidos libres (**all**). Además se pueden incorporar atributos.

1. [**Secuencias**](#secuencias)
2. [**Elecciones**](#elecciones)
3. [**Etiqueta all**](#etiquetaAll)
4. [**Mezcla de elementos**](#mezclaElementos)
5. [**Añadir atributos**](#añadirAtributos)

**Secuencias**

Dentro de un elemento es habitual indicar su contenido como una secuencia de elementos. Esto se permite con la etiqueta **sequence**, dentro de la cual se añaden etiquetas **element** para indicar los elementos que entran en la secuencia.

Ejemplo:

|  |
| --- |
| <**xs:simpleType name=**”emailT”**>**  **<xs:restriction** **base=**”xs:string”**>**  **<xs:pattern value=**”[A-Za-z]{3,}@.{3,}” **/>**  **</xs:restriction> </xs:simpleType> <xs:element** **name=**”email”**>**  **<xs:complexType >**  **<xs:sequence>  <xs:element** **name=**”remite” **type=**”emailT” **/>**  **<xs:element** **name=**”para” **type=**”emailT” **minOccurs=**”1” **maxOccurs=**”unbounded” />  **<xs:element** **name=**”CC” **type=**”emailT” **minOccurs=**”0” **maxOccurs=**”unbounded” **/>**  **<xs:element** **name=**”CCO” **type=**”emailT” **minOccurs=**”0” **maxOccurs=**”unbounded” />  **</xs:sequence>**  **</xs:complexType>  </xs:element>** |

En el ejemplo, el elemento email está compuesto de cuatro elementos. El *remite* (que tiene obligatoriamente que aparecer una vez), el *para* que aparecerá al menos una vez y que puede aparecer tantas veces como se desee y los apartados opcionales *CC* y *CCO* que pueden aparecer repetidos.

La etiqueta **sequence** posee los atributos **minOccurs** y **maxOccurs** para indicar que el bloque de la secuencia se puede repetir.

**Elecciones**

Sirven para permitir elegir uno de entre varios elementos. Su funcionamiento es el mismo que en las secuencias, pero en este caso se utiliza una etiqueta llamada **choice**.

Ejemplo:

|  |
| --- |
| **<xs:element name=**”identificación”**>  <xs:complexType>  <xs:choice>  <xs:element name=**”firma” **type=**”xs:NCName” **/>**  **<xs:element name=**”código” **type=**”xs:NCName” **/>  </xs:choice>  </xs:complexType>**  **</xs:element>** |

En el ejemplo, el elemento *identificación*, consta de dos posibles elementos *firma* y *código* de los que sólo se podrá incluir uno.

La etiqueta **choice** también posee los atributos **minOccurs** y **maxOccurs**.

**Etiqueta all**

Se trata de una posibilidad similar a **choice** y **sequence** que se utiliza de la misma forma y que tiene como diferencia principal que los elementos que contiene pueden aparecer cero o una vez y además en el orden que quieran.

Ejemplo:

**<xs:element name=**”identificación”**>**  
 **<xs:complexType>**  
 **<xs:all>**  
 **<xs:element name=**”firma” **type=**”xs:NCName”**/>**  
 **<xs:element name=**”código” **type=**”xs:NCName” **/>**  
 **</xs:all>**  
 **</xs:complexType>**

**</xs:element>**

En este caso la *firma*y el *código* pueden aparecer o no, aparecer los dos e incluso el orden será indiferente. Es una etiqueta muy potente que ahorra mucho trabajo.

Esta etiqueta tiene los atributos **minOccurs** y **maxOccurs**, pero sólo se puede indicar como valores cero o uno.

**Mezcla de elementos**

A veces los contenidos de un documento XML son extremadamente complejos y por eso se permite en los esquema colocar etiquetas **choice** dentro de etiquetas **sequence** y viceversa. Y lo mismo ocurre con las etiquetas **all**. Estas posibilidades permiten crear cualquier tipo de esquema por complejo que resulte.

Ejemplo:

|  |
| --- |
| **<xs:element name=**”correo”**>  <xs:complexType>  <xs:sequence>  <xs:element name=**”remite” **type=**”xs:string”**/>  <xs:element name**=”para” **type=**”xs:string” **minOccurs=**”1” **maxOccurs=**”unbounded”**/>  <xs:choice>  <xs:element name=**”cc” **type=**”xs:string” **minOccurs=**”1” **maxOccurs=**”unbounded” **/>  <xs:element name=**”cco” **type=**”xs:string” **minOccurs=**”1” **maxOccurs=**”unbounded” **/>  </xs:choice>  </xs:sequence>  </xs:complexType> </xs:element>** |

En el ejemplo, el elemento *correo*consta de tres elementos: *remite, para* y un tercero que puede ser *cc* o *cco*.

**Añadir atributos**

En los apartados **complexType**, los atributos del elemento se definen al final del apartado **complexType** (justo antes de cerrarle). Ejemplo:

|  |
| --- |
| **<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:attribute name=**”sexo”**>  <xs:simpleType>  <xs:restriction base=**”xs:string”**>  <xs:enumeration value=**”Hombre” **/>  <xs:enumeration value=**”Mujer” **/>  </xs:restriction>  </xs:simpleType>  </xs:attribute>  <xs:attribute name=**”fechaNacimiento”  **type=**”xs:date” **use=**”required” **/>  </xs:complexType> </xs:element>** |

Las personas contienen dos elementos en secuencia (*nombre*y *apellidos*) y dos atributos: uno opcional (*sexo*) que sólo pueden tomar los valores *Hombre* o *Mujer* y uno obligatorio para la fecha de nacimiento

**[3.7.12] Contenidos mixtos**

Es el caso más complejo. Se trata de elementos que contienen otros elementos y además texto (e incluso atributos). Para permitir esta posibilidad hay que marcar el atributo **mixed** de la etiqueta **complexType** a **true**.

Ejemplo:

|  |
| --- |
| **<xs:element name=**”documento”**>  <xs:complexType mixed=**”true”**>  <xs:choice>  <xs:element name=**”negrita” **type=**”xs:string”  **minOccurs=**”0” **maxOccurs=**”unbounded”**/>   </xs:choice>  <xs:attribute name=**”lenguaje” **type=**”xs:language” **/>  </xs:complexType> </xs:element>** |

Lo malo es que no se puede controlar el tipo de datos del texto interior. Los elementos se controlan completamente, pero el texto no.

**[3.7.13] Uso de elementos y atributos globales**

Se trata de definir un elemento o un atributo para ser reutilizado en diferentes partes del documento. La forma de utilizarlos es:

* Definirlos en la zona global (es decir, no definirlos dentro de ningún otro componente)
* En el caso de los elementos no se pueden indicar los atributos **minOccurs** y **maxOccurs**para indicar la cardinalidad, porque sólo tiene sentido dentro de otro elemento.
* En el caso de los atributos no se puede utilizar el atributo **use**, que indica la obligatoriedad de uso del atributo.
* Una vez definidos, donde se quieran reutilizar se define el elemento o atributo dentro del componente en el que se quiere colocar y no se le da nombre, sino que se usa el atributo **ref** para indicar el nombre del elemento o atributo global.

Ejemplo:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?> **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>  <xs:element name=**”email”**>  <xs:simpleType>  <xs:restriction base=**”xs:string”**>  <xs:pattern value=**”[A-Za-z]{3,}@.{3,}” **/>**  **</xs:pattern>**  **</xs:restriction>  </xs:simpleType>  </xs:element>   <xs:element name=**”trabajador”**>  <xs:complexType>  <xs:sequence>  <xs:element name=**”nombre” **type=**”xs:string” **/>  <xs:element name=**”apellidos” **type=**”xs:string” **/>  <xs:element ref=”email” minOccurs=**”1”  **maxOccurs=**”unbounded” **/>  </xs:sequence>  </xs:complexType>  </xs:element> </xs:schema>** |

En este caso se ha definido un elemento global llamado email que representa una dirección de email (dentro se define un tipo simple para restringir la forma en la que se debería rellenar un email) y que después el elemento *trabajador* la reutiliza para indicar los correos electrónicos de un trabajador. Otros elementos podrían reutilizar esta misma definición.

**[3.7.14] Grupos de elementos**

La etiqueta **group** permite realizar grupos de elementos y eso permite organizarse mejor a la hora de crear un esquema. Dentro de cada grupo podemos utilizar etiquetas **sequence, choice**y **all** de la misma forma que la vista anteriormente y así después utilizar el grupo en la forma deseada. Muchas veces los grupos se definen de forma global y así se pueden utilizar en distintos elementos; pero es posible definirles localmente (suele tener menos interés hacerlo en local).

Los grupos cuando se definen de forma global requieren indicar un nombre para ellos (si se definen de forma local no). Cuando un elemento desea incorporar un grupo global, utiliza la etiqueta **group** y con el atributo **ref** indicaría el nombre del elemento global definido anteriormente. Pueden contener los atributos **minOccurs**, **maxOccurs** para indicar las veces que puede repetirse el grupo en el elemento que le contiene. Ejemplo:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?> **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>**  **<xs:group name=**”seccionesCorreo”**>**  **<xs:sequence>**  **<xs:element name=**”remite” **type=**”xs:string”**/>**  **<xs:element name=**”para” **type=**”xs:string”  **minOccurs=**”1” **maxOccurs=**”unbounded”**/>**  **<xs:choice>**  **<xs:element name=**”cc” **type=**”xs:string”  **minOccurs=**”1” **maxOccurs=**”unbounded” **/>**  **<xs:element name=**”cco” **type=**”xs:string”  **minOccurs=**”1” **maxOccurs=**”unbounded” **/>**  **</xs:choice>**  **</xs:sequence>**  **</xs:group>**   **<xs:element name=**”correo”**>**  **<xs:complexType>**  **<xs:sequence>**  **<xs:group ref=**”seccionesCorreo” **/>**  **<xs:element name=**”contenido” **type=**”xs:string” **/>**  **</xs:sequence>**  **</xs:complexType>**  **</xs:element>** **</xs:schema>** |

**[3.7.15] Grupos de atributos**

La idea es la misma que con los grupos de elementos: facilitar la escritura del código del esquema. Se trata de definir (lo lógico es de forma global) una serie de atributos que utilizan diferentes elementos y así mejorar el mantenimiento del esquema XML.

Los grupos de atributos se declaran con **attributeGroup**, al definir se utiliza el nombre y al usarle en un elemento se hace referencia a dicho nombre mediante la etiqueta **ref**.

Ejemplo:

|  |
| --- |
| **<xs:schema xmlns:xs=**”http://www.w3.org/2001/XMLSchema”**>  <xs:attributeGroup name=**”infoDoc”**>  <xs:attribute name=**”lenguaje” **type=**”xs:language” **use=**”required” **/>  <xs:attribute name=**”tamaño”**>  <xs:simpleType>  <xs:restriction base=**”xs:positiveInteger”**>  <xs:minExclusive value=**”20” **/>  <xs:maxExclusive value=**”200” **/>  </xs:restriction>  </xs:simpleType>  </xs:attribute>**  **<xs:attribute name=”título” type=”xs:string”**  **use=”required”/>**  **</xs:attributeGroup>    <xs:element name=**”documento”**>  <xs:complexType>  <xs:simpleContent>  <xs:extension base=”**xs:string”**>  <xs:attributeGroup ref=**”infoDoc” **/>  </xs:extension>  </xs:simpleContent>  </xs:complexType>  </xs:element>    <xs:element name=**”documentoRef”**>  <xs:complexType>  <xs:attributeGroup ref=**”infoDoc” **/>  <xs:attribute name=**”referencia”  **type=**”xs:positiveInteger” **use=**”required” **/>  </xs:complexType>  </xs:element>** |
| **<xs:element name=**”documentos”**>  <xs:complexType>  <xs:choice minOccurs=**”1” **maxOccurs=**”unbounded”**>  <xs:element ref=**”documento” **/>  <xs:element ref=**”documentoRef” **/>  </xs:choice>  </xs:complexType>  </xs:element> </xs:schema>** |

Se define en el ejemplo un grupo de atributos globales que se llama *infoDoc* y que está formado por tres atributos (*lenguaje, tamaño*y*título)*que se definen completamente en el grupo. Después dos elementos (**documento** y **documentoRef**) utilizan el grupo de atributos como si fuera un solo atributo. De modo que un XML basado en este esquema y válido podría ser:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?> **<documentos   xmlns:xs=**”http://www.w3.org/2001/XMLSchema-instance” **xs:noNamespaceSchemaLocation=**”correo.xsd” **>  <documento lenguaje=**”es” **tamaño=**”123”  **título=**”Informe 32” **/>  <documento lenguaje=**”es” **tamaño=**”34”  **título=**”Alta 34” **/>  <documentoRef lenguaje=**”es” **referencia=**”3234”  **título=**”Informe 56” **/> </documentos>** |