Schema
XSD
Xml Schema Definition

Qué es xml Schema?

Un Schema XML describe la estructura de un documento XML.

El lenguaje Schema XML también es conocido como XML Schema Definition (XSD).

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Qué es xml Schema?

EL propósito de un Schema XML es definir los bloques de construcción de un documento XML:

- **Los elementos y atributos que pueden aparecer en un documento.**
- **El número y orden de los elementos hijo.**
- **Los tipos de datos para los elementos y atributos.**
- **Valores fijos y por defecto para los elementos y los atributos.**

Por qué aprender xml Schema?

En el mundo XML, cientos de formatos estandarizados XML son usados diariamente.

Muchos de estos estándares son definidos mediante XML Schemas.

XML Schema está basado en XML, (es más potente), y es una alternativa al uso de DTD.

XML Schemas soportan tipos de datos

Una de las grandes fortalezas de Schemas es que tiene soporte para tipos de datos.

- Es más fácil describir el contenido permitido del documento.
- Es más fácil validar la exactitud de los datos.
- Es más fácil definir facetas de datos, (restricciones).
- Es más fácil definir patrones de datos, (formatos de datos).
- Es más fácil convertir datos entre diferentes tipos de datos.

XML Schemas usan la sintaxis XML

Otra gran ventaja de los esquemas XML es que están escritos en XML.

- No hay que aprender un nuevo lenguaje.
- Se puede utilizar un editor XML para editar los archivos Schema.
- Se puede utilizar tu analizador XML para analizar tus archivos de Schema.
- Se puede manipular un esquema con XML DOM.
- Se puede transformar un esquema con XSLT.
- Los esquemas XML son extensibles porque están escritos en XML.

Con una definición de esquema extensible se puede:

- Reutilizar el Schema en otros Schemas.
- Crear tus propios tipos de datos derivados de los tipos estándar.
- Hacer referencia a varios Schemas en el mismo documento.

Esquemas XML Comunicación segura de datos

Al enviar datos de un remitente a un receptor, es fundamental que ambas partes tengan las mismas "normas" sobre el contenido.

Con los esquemas XML, el remitente debe describir los datos de una manera que el receptor los entienda.

Una fecha como: "03-11-2004" se interpretará, en algunos países, como el 3 de noviembre y en otros países como el 11 de marzo.

Sin embargo, un elemento XML con un tipo de datos como este:

```
<date type="date">2004-03-11</date>
```

garantiza una comprensión mutua del contenido, ya que el tipo de datos XML "fecha (date)" requiere el formato "AAAA-MM-DD".

Bien formado no es suficiente

Un documento XML bien formado es un documento que se ajusta a las reglas de sintaxis XML, como:

- debe comenzar con la declaración XML (prólogo).
- debe tener un elemento raíz único.
- las etiquetas de inicio deben tener etiquetas finales coincidentes.
- los elementos distinguen entre mayúsculas y minúsculas.
- todos los elementos deben estar cerrados.
- todos los elementos deben estar correctamente anidados.
- todos los valores de los atributos deben ir entre comillas.
- Las entidades deben usarse para caracteres especiales.
- Incluso si los documentos están bien formados, aún pueden contener errores, y esos errores pueden generar graves consecuencias.

XSD How To?

Los documentos XML pueden tener una referencia a un DTD o a un XML Schema.

Un documento simple XML

```
<?xml version="1.0" enconding=”UTF-8” ?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Un fichero DTD

El siguiente ejemplo es un fichero de DTD llamado "note.dtd" que define los elementos del documento XML de arriba ("note.xml"):

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Un XML Schema

El siguiente ejemplo es un Schema XML llamado "note.xsd" que define los elementos del documento XML "note.xml":

```
<?xml version="1.0" enconding=”UTF-8” ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

EL elemento note es un tipo complejo porque contiene otros elementos. Los otros elementos (to, from, heading, body) son de tipo simple porque no contienen otros elementos. Aprenderemos más sobre ello más adelante.

XSD How To?

Referencia a un DTD

```
<?xml version="1.0" enconding=”UTF-8” ?>

<!DOCTYPE note SYSTEM “note.dtd">

<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Referencia a un XML Schema

Este documento tiene una referencia a un XML Schema:

```
<?xml version="1.0" enconding=”UTF-8” ?>

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XSD - The <schema> Element

El elemento <schema> es el elemento raíz de todo XML Schema.

The <schema> Element

```
<?xml version="1.0" encoding=”UTF-8” ?>

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

El elemento <schema> puede contener algunos atributos. Una declaración de Schema suele tener el siguiente aspecto:

```
<?xml version="1.0" encoding=”UTF-8” ?>

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

XSD – El elemento <schema>

El siguiente trozo:

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

indica que los elementos y los tipos de datos usados en el Schema vienen del espacio de nombres "http://www.w3.org/2001/XMLSchema".

También específica que los elementos y tipos de datos vienen del espacio de nombres (namespace) "http://www.w3.org/2001/XMLSchema" deberán ir prefijados con el prefijo xs:

Este trozo: targetNamespace="https://www.w3schools.com" indica que los elementos definidos por este Schema (note, to, from, heading, body.) vienen del espacio de nombres "https://www.w3schools.com".

Este trozo: xmlns="https://www.w3schools.com" indica que el espacio de nombres por defecto es "https://www.w3schools.com".

Este trozo: elementFormDefault="qualified" indica que cualquier elemento utilizado por el documento de instancia XML declarado en este esquema, debe llevar la notación con prefijo del espacio de nombres.

XSD How To? Referenciando un Schema en un documento XML

Este documento XML tiene una referencia a un Schema XML:

```
<?xml version="1.0" encoding=”UTF-8” ?>

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

<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

El siguiente fragmento:

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

Especifica la declaración del espacio de nombres por defecto. Esta declaración le dice al validador del Schema que todos los elementos usados en este documento XML están declarados en el espacio de nombres "https://www.w3schools.com".

XSD How To?

Referenciando un Schema en un documento XML

Una vez que tengamos disponible el espacio de nombres de la instancia de Schema Xml:

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

Podremos usar el atributo schemaLocation. Este atributo tiene dos valores, separados por un espacio. El primer valor es el espacio de nombres a utilizar. El segundo valor es donde se encuentra el Schema XML para ese espacio de nombres:

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

XSD Elementos Simples

XML Schemas define los elementos de nuestros ficheros.

Un elemento simple es un elemento xml que solo puede contener texto. No puede contener otros elementos o atributos.

Que es un elemento Simple?

Sin embargo, la restricción de "sólo texto" es bastante engañosa.

El texto puede ser de muchos tipos diferentes. Puede ser uno de los tipos incluidos en la definición del esquema XML (booleano, cadena, fecha, etc.) o puede ser un tipo personalizado definido por nosotros.

Se pueden añadir restricciones (facetas) a un tipo de datos para limitar su contenido, o podemos exigir a los datos que casen con un patrón específico.

Definiendo un Elemento Simple

La sintaxis para definir un elemento simple es:

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

Donde xxx es el nombre del elemento e yyy es el tipo de datos del elemento.

XML Schema tiene muchos tipos de datos. Los más comunes son:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

Definiendo un elemento simple

Ejemplo. Algunos elementos XML:

```
<lastname>Basanta</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

Están son las definiciones de los elementos:

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
```

Valores Fijos y por defecto para elementos simples

Los elementos simples pueden tener un valor por defecto o un valor fijo especificado.

- Un valor por defecto es asignado automáticamente a un elemento cuando ningún valor es especificado.

En el siguiente ejemplo el valor por defecto es "red":

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

- Un valor fijo es también asignado a un elemento, y no podemos asignarle otro valor.

En el siguiente ejemplo el valor fijo es "red":

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

XSD Atributos

Todo atributo es declarado como de tipo simple.

Qué es un Atributo?

Los elementos simples no pueden tener atributos por lo que, si un elemento tiene atributos, se considera como de tipo complejo. Pero el atributo por sí mismo es declarado siempre como de tipo simple.

Como definir un atributo?

La sintaxis para definir un atributo es:

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

Donde xxx es el nombre del atributo e yyy especifica el tipo de dato del atributo.

Ejemplo. Aquí está un elemento XML con un atributo:

```
<lastname lang="EN">Smith</lastname>
```

Y aquí está la correspondiente definición del atributo:

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

XML Schema. Tipos de datos más comunes:

xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time

Atributo opcional y obligatorio

Los atributos son opcionales por defecto.

Para especificar que un atributo es obligatorio, se usa el atributo “use”:

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

<xs:attribute name="lang" type="xs:string" use="optional"/>
```

Restricciones sobre el Contenido

Cuando un elemento o atributo xml tiene un tipo de dato definido, esto le impone restricciones al contenido del elemento o atributo.

Si un elemento xml es del tipo "xs:date" y contiene una cadena como "Hello World", el elemento no será validado.

Con los Schemas XML, podrás añadir tus propias restricciones a los elementos y atributos. Estas restricciones son llamadas facetas, (facets). Más adelante se explicará.

XSD Restricciones/Facets

Las restricciones son usadas para definir valores aceptables para los elementos o atributos. A las restricciones sobre elementos XML se les llama facets, (facetas, caras, aspectos).

Restricciones sobre valores

Los siguientes ejemplos definen un elemento llamado "age" con una restricción. El valor de age no puede ser menor que 0 o mayor que 120:

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

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

Restricciones sobre un conjunto de valores

Si queremos limitar el contenido de un elemento xml a un conjunto de valores aceptados, usaremos la restricción de enumeración.

El siguiente ejemplo define un elemento llamado "car" con una restricción. Los únicos valores aceptados son: Audi, Golf, BMW:

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

Restricciones sobre un conjunto de valores

El ejemplo anterior también podría haber sido escrito:

```
<xs:element name="car" type="carType"/>

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

Nota: En este caso el tipo "carType" puede ser usado por otros elementos porque no es parte del elemento car.

Restricciones sobre series de valores

Para limitar el contenido de un elemento xml para definir una serie de números o letras que pueden ser usados, usaremos la restricción patrón, (pattern).

El ejemplo de abajo define un elemento llamado "letter" con una restricción. El único valor aceptable es UNA de la LETRAS MINÚSCULAS de la a a la z:

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

Restricciones sobre series de valores

El siguiente ejemplo define un elemento llamado “initials" con una restricción. El único valor aceptado es 3 letras MAYÚSCULAS de la a la z:

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones sobre una series de valores

Elemento llamado "initials" con la restricción: El único valor aceptado es 3 letras MINÚSCULAS o MAYÚSCULAS de la a a la z:

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

Restricciones sobre una series de valores

Definir un elemento llamado "choice" con la restricción: El único valor aceptado es 1 de las siguientes letras: x, y, ó z:

```
<xs:element name="choice">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[xyz]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones sobre una series de valores

Elemento de nombre "prodid" con la restricción de que el único valor aceptado es un número de 5 dígitos y cada dígito en el rango 0-9:

```
<xs:element name="prodid">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Otras restricciones sobre una serie de valores

Elemento de nombre "letter" con la restricción de que el valor aceptado es 0 o más ocurrencias de letras minúsculas de la a a la z:

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

Otras restricciones sobre una serie de valores

Definir elemento llamado "letter" con la restricción de que el único valor aceptable es uno o más pares de letras, cada par consistente en una letra minúscula seguida por una letra mayúscula. Por ejemplo, "sToP" será valida por este patrón, pero no serán validos: "Stop" o "STOP" o "stop":

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Otras restricciones sobre una serie de valores

Definir elemento "gender" con la restricción: El único valor aceptable es macho o hembra:

```
<xs:element name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Otras restricciones sobre una serie de valores

Define elemento "password" con la restricción: Habrá exactamente 8 caracteres seguidos y esos caracteres serán minúsculas o mayúsculas de la a a la z, o un número del 0 al 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restricciones sobre caracteres Whitespace

Para especificar como los caracteres en blanco serán tratados, usaremos la restricción de Espacio en Blanco.

Definir elemento "address" con la restricción: restricción Espacio en Blanco se pondrá a "preserve", lo que significa que el procesador XML no borrará ningún carácter espacio en blanco:

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

Restricciones sobre caracteres Whitespace

Definir elemento "address" con la restricción: Restricción WhiteSpace puesta a "replace", lo que significa que el procesador XML reemplazará todos los caracteres espacio en blanco, (avances de línea, tabulaciones, espacios y retornos de carro) por espacios:

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

Restricciones sobre caracteres Whitespace

Define elemento "address" con la restricción: restricción Espacio en Blanco se pone a "collapse", lo que significa que el procesador XML borrará todos los caracteres espacio en blanco (line feeds, tabs, spaces, carriage returns y reemplazará con espacios, espacios iniciales y finales son borrados, y múltiples espacios son reducidos a un espacio):

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

Restricciones sobre longitud

Para limitar la longitud del valor de un valor en un elemento, podremos usar las restricciones: length, maxLength, y minLength.

Define elemento "password" con la restricción: El valor debe de ser exactamente 8 caracteres.

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

Restricciones sobre longitud

Define otro elemento "password" con la restricción: El valor tendrá mínimo 5 caracteres y máximo 8 caracteres.

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

Restricciones para tipos de datos

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero

Restrictions for Datatypes

minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

XSD Elementos complejos

Un elemento complejo contiene otros elementos y/o atributos.

Qué es un elemento complejo?

Hay 4 tipos de elementos complejos:

Elementos vacíos, (empty).
Elementos que contienen otros elementos.
Elementos que contienen solo texto.
Elementos que contienen ambas cosas, (otros elementos y texto).

Nota: Cada uno de estos elementos puede contener atributos también!

Ejemplos de elementos complejos

- Un elemento complejo "product", que es vacío (empty):

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

- Un elemento complejo "employee", que contiene solo otros elementos:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

- Un elemento complejo "food", que contiene solo texto:

```
<food type="dessert">Ice cream</food>
```

- Un elemento complejo "description", que contiene ambos: elementos y texto:

```
<description>
    Sucedió en <date lang="norwegian">03.03.99</date> ....
</description>
```

Como definir un elemento complejo?

Supongamos el elemento complejo "employee", que contiene solo oros elementos:

```
<employee>
   <firstname>John</firstname>
   <lastname>Smith</lastname>
</employee>
```

Como definir un elemento complejo?

Hay 2 formas de definir un elemento complejo en un Schema:

1. El elemento "employee" puede ser declarado directamente poniéndole un nombre:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Si se usa este método, solo el elemento “employee" podrá usar el tipo complejo especificado. Fijarse en que los elementos hijo, "firstname" y "lastname", están rodeados por el indicador de <sequence>. Lo cual significa que los elementos hijo deberán aparecer en el mismo orden a como están declarados.

Como definir un elemento complejo?

2. El elemento "employee" puede tener un tipo que haga referencia al nombre del tipo complejo a usar:

```
<xs:element name="employee" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Como definir un elemento complejo?

Si se utiliza este último método, varios elementos podrán hacer referencia al mismo tipo complejo:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="student" type="personinfo"/>
<xs:element name="member" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Como definir un elemento complejo?

También se podrá usar este método y añadir algunos elementos a mayores a los definidos en el tipo:

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XSD Elementos Complejos

Un elemento complejo no puede tener contenido, solamente atributos.

Elementos complejos vacíos

```
<product prodid="1345" />
```

"product" no tiene contenido. Para definir un tipo sin contenido, definiremos un tipo que permita elementos en su contenido, (complexType), pero no contendrá ningún elemento:

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Elementos complejos vacíos

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Se define un tipo complejo con contenido complejo. Donde el elemento de contenido complejo, (complexContent), señala que lo que se pretende es restringir o extender el modelo de contenido de un tipo complejo, y la restricción de número entero declara un atributo pero no introduce, (añade), ningún contenido al elemento.

Elementos complejos vacíos

Se puede definir el elemento "product" así:

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

O podríamos darle al elemento de complexType un nombre, y dejar al elemento "product" tener un tipo atributo que haga referencia al nombre del tipo complexType, (si se usa este método, varios elementos podrán hacer referencia al mismo tipo complejo):

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

XSD Solo elementos

Un tipo complejo "solo-elementos" contiene un elemento que contiene solamente otros elementos.

Tipos complejos conteniendo solo-elementos

Un elemento Xml "person" que contiene solo otros elementos:

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Tipos complejos conteniendo solo-elementos

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Podremos definir al elemento "person" en un schema:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

La etiqueta <xs:sequence>. Significa que los elementos definidos, ("firstname" y "lastname"), deberan aparecer en ese orden dentro de un elemento "person".

Tipos complejos conteniendo solo-elementos

O también podríamos darle al elemento de tipo complejo un nombre, y permitir al elemento "person" tener un tipo atributo que haga referencia al nombre del element complexType, (si usamos este método, varios elementos podrán hacer referencia al mismo tipo complejo:

```
<xs:element name="person" type="persontype"/>

<xs:complexType name="persontype">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

XSD Elementos solo-texto

Un elemento complejo solo-texto puede contener texto y atributos.

Elementos complejos solo-texto

Este tipo contiene solo contenido simple, (texto y atributos), de forma que añadimos un elemento de contenido simple alrededor del contenido. Cuando usamos contenido simple, debemos definir una extensión o una restricción dentro del elemento de contenido simple:

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Tip: Usar el elemento extension / restriction para expandir o para limitar el tipo simple base para el elemento.

Elementos complejos solo-texto

O:

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Tip: Usar el elemento extension / restriction para expandir o para limitar el tipo simple base para el elemento.

Elementos complejos solo-texto

Ejemplo de elemento que contiene solo-texto:

```
<shoesize country="france">35</shoesize>
```

Se declara un tipo complejo, "shoesize". El contenido es definido como un valor entero, y el elemento "shoesize" también contiene un atributo de nombre "country":

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Elementos complejos solo-texto

También podremos darel un nombre al elemento complexType, y permitir al elemento "shoesize" tener un atributo tipo que haga referencia al nombre del complexType (si este método es usado, varios elementos podrán hacer referencia al mismo tipo complejo):

```
<xs:element name="shoesize" type="shoetype"/>

<xs:complexType name="shoetype">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="country" type="xs:string" />
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

XSD Mixed Content

Un elemento XML "letter", que contiene ambas cosas: texto y otros elementos:

```
<letter>
  Dear Mr. <name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```