Arquitectura Orientada a Servicios

XML Schema

Edwin Salvador

8 de mayo de 2015

Sesión 5

Contenido I

Contenido II

XML Schema

Un XML Schema define la estructura de un documento XML.

```
<?xml version="1.0" encoding="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 hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos
- el orden de los elementos hijos

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos
- el orden de los elementos hijos
- el número de elementos hijos

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos
- el orden de los elementos hijos
- el número de elementos hijos
- si un elemento es vacío o puede incluir texto

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos
- el orden de los elementos hijos
- el número de elementos hijos
- si un elemento es vacío o puede incluir texto
- los tipos de datos para los elementos y atributos

¿Qué hace un XML Schema?

Define:

- elementos que pueden aparecer en un documento
- atributos que pueden aparecer en un documento
- que elementos son hijos
- el orden de los elementos hijos
- el número de elementos hijos
- si un elemento es vacío o puede incluir texto
- los tipos de datos para los elementos y atributos
- los valores por defecto o fijos para elementos y atributos.

Contenido I

Contenido II

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos
- Utilizan sintaxis XML:

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos
- Utilizan sintaxis XML:
  - No se aprende un nuevo lenguaje

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos
- Utilizan sintaxis XML:
  - No se aprende un nuevo lenguaje
  - Se puede utilizar el mismo editor para escribir los Schemas

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos
- Utilizan sintaxis XML:
  - No se aprende un nuevo lenguaje
  - Se puede utilizar el mismo editor para escribir los Schemas
  - Se puede manipular el Schema con el XML DOM (Document Object Model)

¿Por qué utilizar XML Schemas?

- Una alternativa a los DTD basado en XML
- Son más poderosos que los DTD
- También conocidos como XSD (XML Schema Definition)
- Soporta tipos de datos. Facilita:
  - controlar el contenido permitido.
  - validar que los datos sean correctos
  - trabajar con datos de una base de datos
  - definir restricciones a los datos
  - definir formato de datos
  - convertir datos entre diferentes tipos de datos
- Utilizan sintaxis XML:
  - No se aprende un nuevo lenguaje
  - Se puede utilizar el mismo editor para escribir los Schemas
  - Se puede manipular el Schema con el XML DOM (Document Object Model)
  - Se puede transformar el Schema con el XSLT (EXtensible Stylesheet Language Transformations)

Aseguran la comunicación efectiva

- Cuando enviamos o recibimos datos es importante que las dos partes interpreten de igual manera el contenido.

Aseguran la comunicación efectiva

- Cuando enviamos o recibimos datos es importante que las dos partes interpreten de igual manera el contenido.
- XML Schemas no facilitan este requisito

Aseguran la comunicación efectiva

- Cuando enviamos o recibimos datos es importante que las dos partes interpreten de igual manera el contenido.
- XML Schemas no facilitan este requisito
- Una fecha: “03-11-2015” se puede interpretar como 3 de noviembre o como 11 de marzo.

Aseguran la comunicación efectiva

- Cuando enviamos o recibimos datos es importante que las dos partes interpreten de igual manera el contenido.
- XML Schemas no facilitan este requisito
- Una fecha: “03-11-2015” se puede interpretar como 3 de noviembre o como 11 de marzo.
- Con XML Schema lo definimos como

```
<date type="date">2015-03-11</date>
```

Aseguran la comunicación efectiva

- Cuando enviamos o recibimos datos es importante que las dos partes interpreten de igual manera el contenido.
- XML Schemas no facilitan este requisito
- Una fecha: “03-11-2015” se puede interpretar como 3 de noviembre o como 11 de marzo.
- Con XML Schema lo definimos como

```
<date type="date">2015-03-11</date>
```

- El tipo date en XML recibe con el formato “YYYY-MM-DD”.

Son extensibles

- Como todos los documentos XML, los Schemas son extensibles.

Son extensibles

- Como todos los documentos XML, los Schemas son extensibles.
- Podemos:

Son extensibles

- Como todos los documentos XML, los Schemas son extensibles.
- Podemos:
  - Reusar nuestro Schema en otros Schemas

Son extensibles

- Como todos los documentos XML, los Schemas son extensibles.
- Podemos:
  - Reusar nuestro Schema en otros Schemas
  - Crear nuestros tipos de datos derivados de los tipos estándar

Son extensibles

- Como todos los documentos XML, los Schemas son extensibles.
- Podemos:
  - Reusar nuestro Schema en otros Schemas
  - Crear nuestros tipos de datos derivados de los tipos estándar
  - Referirnos a múltiples Schemas en el mismo documento

Bien formado no es suficiente

- ¿Qué es un documento bien formado?

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz
  - toda etiqueta debe tener una de cierre

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz
  - toda etiqueta debe tener una de cierre
  - los elementos son sensibles a mayúsculas y minúsculas

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz
  - toda etiqueta debe tener una de cierre
  - los elementos son sensibles a mayúsculas y minúsculas
  - correctamente anidados

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz
  - toda etiqueta debe tener una de cierre
  - los elementos son sensibles a mayúsculas y minúsculas
  - correctamente anidados
  - atributos entre comillas

Bien formado no es suficiente

- ¿Qué es un documento bien formado? cuando cumple con todas las reglas de sintaxis.
- ¿Cuáles son las reglas de sintaxis de XML?
  - Empezar con la declaración XML
  - una sola raíz
  - toda etiqueta debe tener una de cierre
  - los elementos son sensibles a mayúsculas y minúsculas
  - correctamente anidados
  - atributos entre comillas
  - usar ¿entidades HTML? para caracteres especiales

Contenido I

Contenido II

¿Qué nos dice el XML Schema?

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

¿Quién puede escribir un documento válido para este Schema?

Ejemplo

```
<?xml version="1.0"?>
<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Contenido I

Contenido II

Elemento <schema>

- Es la raíz de todo XML Schema

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

Elemento <schema>

- Es la raíz de todo XML Schema

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

- Puede contener atributos:

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

Atributos de <schema>

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` ¿Qué indica?

Atributos de <schema>

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` ¿Qué indica? Los elementos y tipos de datos utilizados en el Schema vienen del espacio de nombres especificado. Y que esos elementos y tipos de datos deben tener un prefijo **xs:**

Atributos de <schema>

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` ¿Qué indica? Los elementos y tipos de datos utilizados en el Schema vienen del espacio de nombres especificado. Y que esos elementos y tipos de datos deben tener un prefijo **xs:**
- `targetNamespace="http://www.w3schools.com"` indica que los elementos definidos por este Schema (note, to , from, heading, body) vienen del espacio de nombres especificado.

Atributos de <schema>

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` ¿Qué indica? Los elementos y tipos de datos utilizados en el Schema vienen del espacio de nombres especificado. Y que esos elementos y tipos de datos deben tener un prefijo **xs:**
- `targetNamespace="http://www.w3schools.com"` indica que los elementos definidos por este Schema (note, to , from, heading, body) vienen del espacio de nombres especificado.
- `xmlns="http://www.w3schools.com"` indica el espacio de nombres por defecto.

Atributos de <schema>

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"` ¿Qué indica? Los elementos y tipos de datos utilizados en el Schema vienen del espacio de nombres especificado. Y que esos elementos y tipos de datos deben tener un prefijo **xs:**
- `targetNamespace="http://www.w3schools.com"` indica que los elementos definidos por este Schema (note, to , from, heading, body) vienen del espacio de nombres especificado.
- `xmlns="http://www.w3schools.com"` indica el espacio de nombres por defecto.
- `elementFormDefault="qualified"` indica que cualquier elemento utilizado en un documento XML que sea instancia de este Schema tiene que ser calificado.

Contenido I

Contenido II

Referenciar a un Schema desde un documento

```
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- `xmlns="http://www.w3schools.com"` el espacio de nombres por defecto. Todos los elementos utilizados en el documento están declarados in el espacio de nombre especificado.

Referenciar a un Schema desde un documento

```
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- `xmlns="http://www.w3schools.com"` el espacio de nombres por defecto. Todos los elementos utilizados en el documento están declarados in el espacio de nombre especificado.
- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` La instancia de XML Schema.

Referenciar a un Schema desde un documento

```
<?xml version="1.0"?>
<note xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

- `xmlns="http://www.w3schools.com"` el espacio de nombres por defecto. Todos los elementos utilizados en el documento están declarados in el espacio de nombre especificado.
- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` La instancia de XML Schema.
- `xsi:schemaLocation="http://www.w3schools.com note.xsd"` Dos parámetros separados por espacio. Primero indica el espacio de nombres. Segundo indica la localización del XML Schema para ese espacio de nombres.

Contenido I

Contenido II

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting
  - xs:decimal

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting
  - xs:decimal
  - xs:integer

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting
  - xs:decimal
  - xs:integer
  - xs:boolean

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date

Elementos simples

- Pueden contener solo “texto” (booleanos, string, date, un tipo personalizado que hayamos definido, etc).
- No pueden contener otros elementos ni atributos
- Se pueden definir restricciones a los tipos de datos que puede contener.
- Se puede definir un patrón que debe cumplir los datos.

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

- Los posibles tipos son:
  - xs:sting
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

Ejemplo

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

sus definiciones son:

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

Valores por defecto y preestablecidos

El valor por defecto es asignado cuando no se le asigna ningún valor al elemento.

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

Un valor preestablecido es asignado automáticamente y no puede ser cambiado

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

Contenido I

Contenido II

Atributos en XSD

- Los elementos simples no pueden tener atributos, si un elemento tiene atributos se lo considera un tipo complejo.

Atributos en XSD

- Los elementos simples no pueden tener atributos, si un elemento tiene atributos se lo considera un tipo complejo.
- Los atributos son declarados similar a los elementos simples.

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

Atributos en XSD

- Los elementos simples no pueden tener atributos, si un elemento tiene atributos se lo considera un tipo complejo.
- Los atributos son declarados similar a los elementos simples.

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

- Los tipos son los mismos que los elementos simples.

Atributos en XSD

- Los elementos simples no pueden tener atributos, si un elemento tiene atributos se lo considera un tipo complejo.
- Los atributos son declarados similar a los elementos simples.

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

- Los tipos son los mismos que los elementos simples.
- Ejemplo:

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

Atributos en XSD

- Los elementos simples no pueden tener atributos, si un elemento tiene atributos se lo considera un tipo complejo.
- Los atributos son declarados similar a los elementos simples.

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

- Los tipos son los mismos que los elementos simples.
- Ejemplo:

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

- Definición:

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

Valor por defecto y preestablecidos

Similar a los elementos simples:
Por defecto:

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

Preestablecido:

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

Atributos opcionales y obligatorios

Si un atributo es obligatorio se lo debe declarar así:

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

Contenido I

Contenido II

Restricciones de contenido

- Si un elemento o atributo tiene un tipo de datos, automáticamente se establecen restricciones en su contenido.

Restricciones de contenido

- Si un elemento o atributo tiene un tipo de datos, automáticamente se establecen restricciones en su contenido.
- Si un elemento es de tipo "`xs:date`" y contiene un string "Hola mundo", este elemento no será válido.

Restricciones de contenido

- Si un elemento o atributo tiene un tipo de datos, automáticamente se establecen restricciones en su contenido.
- Si un elemento es de tipo "`xs:date`" y contiene un string "Hola mundo", este elemento no será válido.
- Se pueden establecer restricciones adicionales personalizadas.

Restricciones en los valores

El valor de “age” no puede ser menor que 0 ni 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>
```

Restricciones en un grupo de valores

Podemos listar los valores aceptados en un elemento con la restricción "enumeration":

```
<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>
```

Enumeration

Una versión más práctica del ejemplo anterior es:

```
<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>
```

Así podemos utilizar El tipo "`cartype`" en otros elementos porque no es parte del elemento "`car`".

Restricciones en una serie de valores

Podemos definir patrones de valores aceptados utilizando expresiones regulares:

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

El elemento solo acepta **UNA** letra **MINÚSCULA** de la **a** a la **z**.

Ejemplo

```
<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>
```

El elemento solo acepta **TRES** letras **MAYÚSCULAS** de la **A** a la **Z**.

Ejemplo

```
<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>
```

TRES letras MAYÚSCULAS o MINÚSCULAS de la **a** a la **z**.

Ejemplo

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

¿Qué valores son válidos para el elemento? .

Ejemplo

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

¿Qué valores son válidos para el elemento? Una de las letras **x**, **y** o **z**.

Ejemplo

```
<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>
```

¿Qué valores son válidos para el elemento?

Ejemplo

```
<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>
```

¿Qué valores son válidos para el elemento? Cinco números seguidos del 0 al 9.

Ejemplo

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

¿Qué valores son válidos para el elemento?

Ejemplo

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

¿Qué valores son válidos para el elemento? cero o más ocurrencias de letras minúsculas

Ejemplo

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

¿Qué valores son válidos para el elemento?

Ejemplo

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

¿Qué valores son válidos para el elemento? uno o más pares de letras alternadas entre minúsculas y mayúsculas. Ej: “hOla” pero no “HOLA”, ni “Hola”, ni “hola”.

Ejemplo

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

¿Qué valores son válidos para el elemento?

Ejemplo

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

¿Qué valores son válidos para el elemento? solo “male” o “female”.

Ejemplo

```
<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>
```

¿Qué valores son válidos para el elemento?

Ejemplo

```
<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>
```

¿Qué valores son válidos para el elemento? exactamente **8 caracteres** consecutivos en **minúsculas** o **mayúsculas** desde la **a** a la **z** o del **0** al **9**

Restricciones de espacios en blanco

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

El procesador XML **NO REMOVERÁ** los espacios en blanco `<xs:whiteSpace value="preserve/>`.

Restricciones de espacios en blanco

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

El procesador XML **REEMPLAZARÁ** los espacios en blanco (nuevas líneas, tabs, espacios) con espacios.

Restricciones de espacios en blanco

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

El procesador XML **REMOVERÁ** todos los espacios en blanco (nuevas líneas, tabs, espacios) redundantes y los reemplazará con un solo espacio.

Restricciones de longitud

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

Exactamente 8 caracteres.

Restricciones de longitud

```
<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>
```

Mínimo 5 y máximo 8 caracteres.

Otras restricciones

Restricción	**Descripción**
enumeration	lista de valores aceptados
fractionDigits	máximo número de decimales permitido. ≥ 0
length	Número exacto de caracteres permitido. ≥ 0
maxExclusive	Equivalente a $<$
maxInclusive	Equivalente a $\leq$
maxLength	Número máximo de números o items permitido. ≥ 0
minExclusive	Equivalente a $>$
minInclusive	Equivalente a $\geq$
minLength	Número mínimo de caracteres permitido. ≥ 0
pattern	Define una secuencia de caracteres aceptada.
totalDigits	Número exacto de números permitidos. ≥ 0
whiteSpace	Cómo serán manejados los espacios en blanco.

Contenido I

Contenido II

- Ejemplo definición de Schema

4 Deber

5 Ejercicios

Elementos complejos

- Contienen otros elementos y atributos.

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:
  - vacios

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

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:
  - vacios

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

  - los que contienen solamente otros elementos

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

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:
  - vacios

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

  - los que contienen solamente otros elementos

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

  - los que contienen solo texto

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

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:
  - vacios

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

  - los que contienen solamente otros elementos

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

  - los que contienen solo texto

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

  - los que contienen otros elementos y texto

```
<description>
  It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

Elementos complejos

- Contienen otros elementos y atributos.
- Cuatro tipos:
  - vacios

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

  - los que contienen solamente otros elementos

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

  - los que contienen solo texto

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

  - los que contienen otros elementos y texto

```
<description>
  It happened on <date lang="norwegian">03.03.99</date> ....
</description>
```

- Todos pueden contener atributos

Definiendo elementos complejos

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

Podemos definir el elemento employee de dos maneras:

```
<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>
```

- Solamente el elemento employee puede utilizar el “complexType”.

Definiendo elementos complejos

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

Podemos definir el elemento employee de dos maneras:

```
<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>
```

- Solamente el elemento employee puede utilizar el “complexType”.
- <sequence> indica que los elementos hijos **deben** aparecer en el orden declarado.

Otra manera

Definiendo elementos complejos

Otra manera de declarar el elemento `employee`:

```
<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>
```

De esta manera muchos elementos pueden hacer referencia al mismo tipo complejo

Otra manera (Ejemplo)

Definiendo elementos complejos

```
<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>
```

Elementos complejos basados en otros

```
<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>
```

Contenido I

Contenido II

Elementos vacíos

Elementos sin contenido.

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

Definimos un elemento pero no lo declaramos:

```
<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>
```

O

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

Otra manera de declarar un elemento vacío

¿De que otra manera podríamos declarar el elemento anterior?

Otra manera de declarar un elemento vacío

¿De que otra manera podríamos declarar el elemento anterior?

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

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

Contenido I

Contenido II

Tipos complejos que contienen solo elementos

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

Se lo puede definir así:

```
<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>
```

Tipos complejos que contienen solo elementos

Al elemento persona también se lo puede definir así:

```
<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>
```

Contenido I

Contenido II

Elementos complejos que contienen solo texto

Solo contenido simple (texto y atributos). Utilizamos el elemento `simpleContent`. Se debe definir una extensión **O** una restricción.

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

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

La extensión o restricción expande o limita el tipo simple de base para el elemento.

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

Ejemplo

Definiendo un tipo elemento complejo solo texto

```
<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>
```

Ejemplo

Definiendo un tipo elemento complejo solo texto

¿De que otra manera podemos definir el ejemplo anterior?

Ejemplo

Definiendo un tipo elemento complejo solo texto

¿De que otra manera podemos definir el ejemplo anterior?

-
```
<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>
```

Contenido I

Contenido II

Contenido mixto

```
<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>
```

El atributo `mixed` tiene que ser `true` para permitir que el texto entre los elementos se despliegue.

Contenido mixto

- ¿De que otra manera podemos definir el ejemplo anterior?

Contenido mixto

- ¿De que otra manera podemos definir el ejemplo anterior?
-

```
<xs:element name="letter" type="lettertype"/>

<xs:complexType name="lettertype" 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>
```

Contenido I

Contenido II

Indicadores

- Controlan como los elementos son utilizados en los documentos.

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:
    - maxOccurs

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:
    - maxOccurs
    - minOccurs

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:
    - maxOccurs
    - minOccurs
  - De grupo:

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:
    - maxOccurs
    - minOccurs
  - De grupo:
    - group (nombre)

Indicadores

- Controlan como los elementos son utilizados en los documentos.
- Siete diferentes indicadores:
  - De orden:
    - all
    - choice
    - sequence
  - De ocurrencia:
    - maxOccurs
    - minOccurs
  - De grupo:
    - group (nombre)
    - attributeGroup (nombre)

all

De orden

Cualquier orden

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

Con este indicador, el indicador `minOccurs` puede ser 0 o 1 y el `maxOccurs` pueder ser solamente 1

choice

De orden

Solamente uno de los elementos puede aparecer

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

sequence

De orden

Los elementos deben aparecer en el orden especificado

```
<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>
```

maxOccurs

De ocurrencia

Máximo de ocurrencias de un elemento:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

maxOccurs="unbounded" sin limite de ocurrencias

minOccurs

De ocurrencia

Mínimo de ocurrencias de un elemento:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

group

De grupos

Define un grupo de elementos

```
<xs:group name="groupname">
...
</xs:group>
```

Se debe definir como `all`, `choice`, `sequence` dentro de la declaración `group`.

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>
```

Referencias de grupos

Se pueden hacer referencia a los grupos en otras definiciones:

```
<xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
    <xs:element name="birthday" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="country" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

attributeGroup

De grupos

Define un grupo de atributos:

```
<xs:attributeGroup name="groupname">
...
</xs:attributeGroup>
```

Ejemplo:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>
```

Referencia a grupos de atributos

Se pueden referencia los grupos de atributos en otras declaraciones:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

Contenido I

Contenido II

<any>

- Nos permite extender el documento XML con elementos no especificados en el Schema.

<any>

- Nos permite extender el documento XML con elementos no especificados en el Schema.
- Ejemplo: podemos declarar el contenido de “persona” dentro del Schema “family.xsd”:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

<any> Ejemplo

- Si tenemos otro Schema llamado “children.xsd”:

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

<xs:element name="children">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="childname" type="xs:string"
      maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

<any> Ejemplo

- Podemos crear un documento “family.xml” que utiliza los dos Schemas anteriores “family.xsd” y “children.xsd”:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns="http://www.microsoft.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.microsoft.com family.xsd
http://www.w3schools.com children.xsd">
<person>
  <firstname>Hege</firstname>
  <lastname>Refsnes</lastname>
  <children>
    <childname>Cecilie</childname>
  </children>
</person>
<person>
  <firstname>Stale</firstname>
  <lastname>Refsnes</lastname>
</person>
</persons>
```

En este caso estamos utilizando un elemento `children` dentro de `person` a pesar de que `person` no ha sido declarado de esa manera.

<anyAttribute>

- Nos permite extender el documento XML con atributos no especificados en el 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:anyAttribute/>
  </xs:complexType>
</xs:element>
```

Contenido I

Contenido II

Sustitución de elementos

- Podemos sustituir un elemento por otro.

Sustitución de elementos

- Podemos sustituir un elemento por otro.
- Es útil cuando tenemos usuarios de diferentes países y deseamos brindar la posibilidad de que elijan los nombres de elementos en el documento XML en sus propios idiomas.

Sustitución de elementos

- Podemos sustituir un elemento por otro.
- Es útil cuando tenemos usuarios de diferentes países y deseamos brindar la posibilidad de que elijan los nombres de elementos en el documento XML en sus propios idiomas.
- Para esto definimos un `substitutionGroup` en el Schema XML.

Sustitución de elementos

- Podemos sustituir un elemento por otro.
- Es útil cuando tenemos usuarios de diferentes países y deseamos brindar la posibilidad de que elijan los nombres de elementos en el documento XML en sus propios idiomas.
- Para esto definimos un `substitutionGroup` en el Schema XML.
- Declaramos un elemento de cabecera (`name`) y luego los otros elementos que pueden sustituirlo:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

Sustitución de elementos

- Podemos sustituir un elemento por otro.
- Es útil cuando tenemos usuarios de diferentes países y deseamos brindar la posibilidad de que elijan los nombres de elementos en el documento XML en sus propios idiomas.
- Para esto definimos un `substitutionGroup` en el Schema XML.
- Declaramos un elemento de cabecera (`name`) y luego los otros elementos que pueden sustituirlo:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

- El tipo de elementos sustitutos debe ser el mismo o derivados del tipo del elemento de cabecera.

Sustitución de elementos

- Podemos sustituir un elemento por otro.
- Es útil cuando tenemos usuarios de diferentes países y deseamos brindar la posibilidad de que elijan los nombres de elementos en el documento XML en sus propios idiomas.
- Para esto definimos un `substitutionGroup` en el Schema XML.
- Declaramos un elemento de cabecera (`name`) y luego los otros elementos que pueden sustituirlo:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

- El tipo de elementos sustitutos debe ser el mismo o derivados del tipo del elemento de cabecera.
- Todos lo elementos del grupo de sustitución deben ser declarados globales (hijo directos del elemento Schema).

Ejemplo

Sustitución de elementos

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

Ejemplo

Sustitución de elementos

Un documento válido para el Schema anterior sería:

```
<customer>
  <name>John Smith</name>
</customer>
```

O este:

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

Bloquear sustitución

Podemos utilizar el atributo `block` para evitar que un elemento sea sustituido:

```
<xs:element name="name" type="xs:string" block="substitution"/>
<xs:element name="navn" substitutionGroup="name"/>

<xs:complexType name="custinfo">
  <xs:sequence>
    <xs:element ref="name"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="customer" type="custinfo" block="substitution"/>
<xs:element name="kunde" substitutionGroup="customer"/>
```

Ejemplo

Sustitución de elementos

¿Sería este documento válido para el Schema anterior?

```
<customer>
  <name>John Smith</name>
</customer>
```

¿Y este?

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

Ejemplo

Sustitución de elementos

¿Sería este documento válido para el Schema anterior? Si

```
<customer>
  <name>John Smith</name>
</customer>
```

¿Y este?

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

Ejemplo

Sustitución de elementos

¿Sería este documento válido para el Schema anterior? Si

```
<customer>
  <name>John Smith</name>
</customer>
```

¿Y este? No

```
<kunde>
  <navn>John Smith</navn>
</kunde>
```

Contenido I

Contenido II

Ejemplo definición de Schema

Definamos un XML Schema (family.xsd) para el siguiente documento XML que permita ingresar un número ilimitado de elementos `persons` y un máximo de 5 `child_name`:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="family.xsd">
<person>
  <full_name>Hege Refsnes</full_name>
  <child_name>Cecilie</child_name>
</person>
<person>
  <full_name>Tove Refsnes</full_name>
  <child_name>Hege</child_name>
  <child_name>Stale</child_name>
  <child_name>Jim</child_name>
  <child_name>Borge</child_name>
</person>
<person>
  <full_name>Stale Refsnes</full_name>
</person>
```

Ejemplo

Definición de un XML Schema

- Tenemos un elemento raíz `persons`

Ejemplo

Definición de un XML Schema

- Tenemos un elemento raíz `persons`
- La raíz contiene elementos `full_name` y `child_name` (puede estar incluido 0 hasta 5 veces).

Ejemplo

Definición de un XML Schema

- Tenemos un elemento raíz `persons`
- La raíz contiene elementos `full_name` y `child_name` (puede estar incluido 0 hasta 5 veces).
- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` no indica que el documento debe ser validado contra un XML Schema

Ejemplo

Definición de un XML Schema

- Tenemos un elemento raíz `persons`
- La raíz contiene elementos `full_name` y `child_name` (puede estar incluido 0 hasta 5 veces).
- `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"` no indica que el documento debe ser validado contra un XML Schema
- `xsi:noNamespaceSchemaLocation="family.xsd"` no dice el Schema que utilizaremos para validar. Donde está el Schema (en este caso estará en el mismo directorio que el XML).

Ejemplo

Definimos el documento:

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

`xmlns:xs="http://www.w3.org/2001/XMLSchema"` nos indica el espacio de nombre del lenguaje de definición de Schema (**URI por defecto**).

Ejemplo

Definimos el elemento `persons`

```
<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Ejemplo

Definimos el elemento `person` que es de tipo complejo

```
<xs:element name="person" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      ...
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Ejemplo

Definimos los elementos `full_name` y `child_name`

```
<xs:element name="full_name" type="xs:string"/>
<xs:element name="child_name" type="xs:string" minOccurs="0" maxOccurs="5"/
```

El elemento `child_name` puede ocurrir de 0 a 5 veces.

Solución completa

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xs:element name="persons">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="person" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="full_name" type="xs:string"/>
            <xs:element name="child_name" type="xs:string"
            minOccurs="0" maxOccurs="5"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Contenido I

Contenido II

Deber

Seguir el tutorial:
http://www.w3schools.com/schema/schema_dtypes_string.asp
Leer los capítulos “Data Types” y “Schema References”.
La siguiente clase haremos ejercicios referentes a estos capítulos.

Contenido I

Contenido II

Ejercicio 1

Definición de un Schema

Escribir un Schema para el siguiente documento:

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Ejercicio 2

<anyAttribute>

- Escribir un Schema llamado “family.xsd” con el contenido anterior.
- Otro Schema “attribute.xsd” que contenga un atributo `gender` que sea un `string` y solo acepte los valores `male` o `female`
- Escribir un documento XML que utilice componentes de los dos Schemas y extienda el elemento `person` con el atributo `gender`.