Introduction to XML Schema

XML Schema is an XML-based alternative to DTD.

An XML schema describes the structure of an XML document.

The XML Schema language is also referred to as XML Schema Definition (XSD).

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML / XHTML
- XML and XML Namespaces
- A basic understanding of DTD

If you want to study these subjects first, find the tutorials on our Home page.

What is an XML Schema?

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

---

XML Schemas are the Successors of DTDs

We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs. Here are some reasons:

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why Use XML Schemas?

XML Schemas are much more powerful than DTDs.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

With support for data types:

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to work with data from a database
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

Some benefits of that XML Schemas are written in XML:

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas Secure Data Communication

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.

With XML Schemas, the sender can describe the data in a way that the receiver will understand.

A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.

However, an XML element with a data type like this:

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

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are Extensible

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

Well-Formed is not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

XSD How To?

XML documents can have a reference to a DTD or to an XML Schema.

A Simple XML Document

Look at this simple XML document called "note.xml":

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

A DTD File

The following example is a DTD file called "note.dtd" that defines the elements of the XML document above ("note.xml"):

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

The first line defines the note element to have four child elements: "to, from, heading, body".

Line 2-5 defines the to, from, heading, body elements to be of type "#PCDATA".

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?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: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>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to a DTD

This XML document has a reference to a DTD:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?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>
```

XSD - The <schema> Element

Previous Next

The <schema> element is the root element of every XML Schema.

The <schema> Element

The <schema> element is the root element of every XML Schema:

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

The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?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>
```

The following fragment:

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

indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs:**

This fragment:

```
targetNamespace="http://www.w3schools.com"
```

indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

This fragment:

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

indicates that the default namespace is "http://www.w3schools.com".

This fragment:

```
elementFormDefault="qualified"
```

indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

Referencing a Schema in an XML Document

This XML document has a reference to an XML Schema:

```
<?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>
```

The following fragment:

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

specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the "http://www.w3schools.com" namespace.

Once you have the XML Schema Instance namespace available:

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

you can use the schemaLocation attribute. This attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

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

XSD Simple Elements

XML Schemas define the elements of your XML files.

A simple element is an XML element that contains only text. It cannot contain any other elements or attributes.

What is a Simple Element?

A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.

However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element

The syntax for defining a simple element is:

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

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

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

Example

Here are some XML elements:

```
<lastname>Refsnes</lastname>
<age>36</age>
```

```
<dateborn>1970-03-27</dateborn>
```

And here are the corresponding simple element definitions:

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

Default and Fixed Values for Simple Elements

Simple elements may have a default value OR a fixed value specified.

A default value is automatically assigned to the element when no other value is specified.

In the following example the default value is "red":

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

A fixed value is also automatically assigned to the element, and you cannot specify another value.

In the following example the fixed value is "red":

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

XSD Attributes

Previous Next

All attributes are declared as simple types.

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type. But the attribute itself is always declared as a simple type.

How to Define an Attribute?

The syntax for defining an attribute is:

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

where xxx is the name of the attribute and yyy specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

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

Example

Here is an XML element with an attribute:

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

And here is the corresponding attribute definition:

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

Default and Fixed Values for Attributes

Attributes may have a default value OR a fixed value specified.

A default value is automatically assigned to the attribute when no other value is specified.

In the following example the default value is "EN":

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

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.

In the following example the fixed value is "EN":

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

Optional and Required Attributes

Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

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

Restrictions on Content

When an XML element or attribute has a data type defined, it puts restrictions on the element's or attribute's content.

If an XML element is of type "xs:date" and contains a string like "Hello World", the element will not validate.

With XML Schemas, you can also add your own restrictions to your XML elements and attributes. These restrictions are called facets. You can read more about facets in the next chapter.

XSD Restrictions / Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

The following example defines an element called "age" with a restriction. The value of age cannot be lower than 0 or greater than 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>
```

Restrictions on a Set of Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

The example below defines an element called "car" with a restriction. The only acceptable values are: 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>
```

The example above could also have been written like this:

```
<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>
```

Note: In this case the type "carType" can be used by other elements because it is not a part of the "car" element.

Restrictions on a Series of Values

To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.

The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

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

The next example defines an element called "initials" with a restriction. The only acceptable value is THREE of the UPPERCASE letters from a to 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>
```

The next example also defines an element called "initials" with a restriction. The only acceptable value is THREE of the LOWERCASE OR UPPERCASE letters from a to 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>
```

The next example defines an element called "choice" with a restriction. The only acceptable value is ONE of the following letters: x, y, OR z:

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

The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 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>
```

Other Restrictions on a Series of Values

The example below defines an element called "letter" with a restriction. The acceptable value is zero or more occurrences of lowercase letters from a to z:

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

The next example also defines an element called "letter" with a restriction. The acceptable value is one or more pairs of letters, each pair consisting of a lower case letter followed by an upper case letter. For example, "sToP" will be validated by this pattern, but not "Stop" or "STOP" or "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>
```

The next example defines an element called "gender" with a restriction. The only acceptable value is male OR female:

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

The next example defines an element called "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 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>
```

Restrictions on Whitespace Characters

To specify how whitespace characters should be handled, we would use the whiteSpace constraint.

This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

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

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "replace", which means that the XML processor WILL REPLACE all white space characters (line feeds, tabs, spaces, and carriage returns) with spaces:

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

This example also defines an element called "address" with a restriction. The whiteSpace constraint is set to "collapse", which means that the XML processor WILL REMOVE all white space characters (line feeds, tabs, spaces, carriage returns are replaced with spaces, leading and trailing spaces are removed, and multiple spaces are reduced to a single space):

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

Restrictions on Length

To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.

This example defines an element called "password" with a restriction. The value must be exactly eight characters:

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

This example defines another element called "password" with a restriction. The value must be minimum five characters and maximum eight characters:

```
<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>
```

Restrictions for Datatypes

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
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 Complex Elements

Previous Next

A complex element contains other elements and/or attributes.

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex XML element, "product", which is empty:

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

A complex XML element, "employee", which contains only other elements:

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

A complex XML element, "food", which contains only text:

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

A complex XML element, "description", which contains both elements and text:

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

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

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

We can define a complex element in an XML Schema two different ways:

1. The "employee" element can be declared directly by naming the element, like this:

```
<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>
```

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared. You will learn more about indicators in the XSD Indicators chapter.

2. The "employee" element can have a type attribute that refers to the name of the complex type to use:

```
<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>
```

If you use the method described above, several elements can refer to the same complex type, like this:

```
<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>
```

You can also base a complex element on an existing complex element and add some elements, like this:

```
<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 Complex Empty Elements

An empty complex element cannot have contents, only attributes.

Complex Empty Elements

An empty XML element:

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

The "product" element above has no content at all. To define a type with no content, we must define a type that allows only elements in its content, but we do not actually declare any elements, like this:

```
<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>
```

In the example above, we define a complex type with a complex content. The complexContent element signals that we intend to restrict or extend the content model of a complex type, and the restriction of integer declares one attribute but does not introduce any element content.

However, it is possible to declare the "product" element more compactly, like this:

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

Or you can give the complexType element a name, and let the "product" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

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

XSD Complex Type - Elements Only

An "elements-only" complex type contains an element that contains only other elements.

Complex Types Containing Elements Only

An XML element, "person", that contains only other elements:

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

You can define the "person" element in a schema, like this:

```
<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>
```

Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

Or you can give the complexType element a name, and let the "person" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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 Complex Text-Only Elements

A complex text-only element can contain text and attributes.

Complex Text-Only Elements

This type contains only simple content (text and attributes), therefore we add a simpleContent element around the content. When using simple content, you must define an extension OR a restriction within the simpleContent element, like this:

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

OR

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

Tip: Use the extension/restriction element to expand or to limit the base simple type for the element.

Here is an example of an XML element, "shoesize", that contains text-only:

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

The following example declares a complexType, "shoesize". The content is defined as an integer value, and the "shoesize" element also contains an attribute named "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>
```

We could also give the complexType element a name, and let the "shoesize" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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 Complex Types With Mixed Content

Previous Next

A mixed complex type element can contain attributes, elements, and text.

Complex Types with Mixed Content

An XML element, "letter", that contains both text and other elements:

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

The following schema declares the "letter" element:

```
<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>
```

Note: To enable character data to appear between the child-elements of "letter", the mixed attribute must be set to "true". The <xs:sequence> tag means that the elements defined (name, orderid and shipdate) must appear in that order inside a "letter" element.

We could also give the complexType element a name, and let the "letter" element have a type attribute that refers to the name of the complexType (if you use this method, several elements can refer to the same complex type):

```
<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>
```

XSD Complex Types Indicators

We can control HOW elements are to be used in documents with indicators.

Indicators

There are seven indicators:

Order indicators:

- All
- Choice
- Sequence

Occurrence indicators:

- maxOccurs
- minOccurs

Group indicators:

- Group name
- attributeGroup name

Order Indicators

Order indicators are used to define the order of the elements.

All Indicator

The <all> indicator specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<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>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

Choice Indicator

The <choice> indicator specifies that either one child element or another can occur:

```
<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 Indicator

The <sequence> indicator specifies that the child elements must appear in a specific order:

```
<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>
```

Occurrence Indicators

Occurrence indicators are used to define how often an element can occur.

Note: For all "Order" and "Group" indicators (any, all, choice, sequence, group name, and group reference) the default value for maxOccurs and minOccurs is 1.

maxOccurs Indicator

The <maxOccurs> indicator specifies the maximum number of times an element can occur:

```
<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>
```

The example above indicates that the "child_name" element can occur a minimum of one time (the default value for minOccurs is 1) and a maximum of ten times in the "person" element.

minOccurs Indicator

The <minOccurs> indicator specifies the minimum number of times an element can occur:

```
<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>
```

The example above indicates that the "child_name" element can occur a minimum of zero times and a maximum of ten times in the "person" element.

Tip: To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement:

A working example:

An XML file called "Myfamily.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
</persons>
```

The XML file above contains a root element named "persons". Inside this root element we have defined three "person" elements. Each "person" element must contain a "full_name" element and it can contain up to five "child_name" elements.

Here is the schema file "family.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

Group Indicators

Group indicators are used to define related sets of elements.

Element Groups

Element groups are defined with the group declaration, like this:

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

You must define an all, choice, or sequence element inside the group declaration. The following example defines a group named "persongroup", that defines a group of elements that must occur in an exact sequence:

```
<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>
```

After you have defined a group, you can reference it in another definition, like this:

```
<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>
```

Attribute Groups

Attribute groups are defined with the attributeGroup declaration, like this:

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

The following example defines an attribute group named "personattrgroup":

```
<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>
```

After you have defined an attribute group, you can reference it in another definition, like this:

```
<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>
```

XSD The <any> Element

Previous Next

The <any> element enables us to extend the XML document with elements not specified by the schema!

The <any> Element

The <any> element enables us to extend the XML document with elements not specified by the schema.

The following example is a fragment from an XML schema called "family.xsd". It shows a declaration for the "person" element. By using the <any> element we can extend (after <lastname>) the content of "person" with any element:

```
<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>
```

Now we want to extend the "person" element with a "children" element. In this case we can do so, even if the author of the schema above never declared any "children" element.

Look at this schema file, called "children.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

The XML file below (called "Myfamily.xml"), uses components from two different schemas; "family.xsd" and "children.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

The XML file above is valid because the schema "family.xsd" allows us to extend the "person" element with an optional element after the "lastname" element.

The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

XSD The <anyAttribute> Element

Previous Next

The <anyAttribute> element enables us to extend the XML document with attributes not specified by the schema!

The <anyAttribute> Element

The <anyAttribute> element enables us to extend the XML document with attributes not specified by the schema.

The following example is a fragment from an XML schema called "family.xsd". It shows a declaration for the "person" element. By using the <anyAttribute> element we can add any number of attributes to the "person" element:

```
<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>
```

Now we want to extend the "person" element with a "gender" attribute. In this case we can do so, even if the author of the schema above never declared any "gender" attribute.

Look at this schema file, called "attribute.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:attribute name="gender">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="male|female"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:schema>
```

The XML file below (called "Myfamily.xml"), uses components from two different schemas; "family.xsd" and "attribute.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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 attribute.xsd">
<person gender="female">
<firstname>Hege</firstname>
<lastname>Refsnes</lastname>
</person>
<person gender="male">
<firstname>Stale</firstname>
<lastname>Refsnes</lastname>
</person>
</persons>
```

The XML file above is valid because the schema "family.xsd" allows us to add an attribute to the "person" element.

The <any> and <anyAttribute> elements are used to make EXTENSIBLE documents! They allow documents to contain additional elements that are not declared in the main XML schema.

XSD Element Substitution

Previous Next

With XML Schemas, one element can substitute another element.

Element Substitution

Let's say that we have users from two different countries: England and Norway. We would like the ability to let the user choose whether he or she would like to use the Norwegian element names or the English element names in the XML document.

To solve this problem, we could define a **substitutionGroup** in the XML schema. First, we declare a head element and then we declare the other elements which state that they are substitutable for the head element.

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

In the example above, the "name" element is the head element and the "navn" element is substitutable for "name".

Look at this fragment of an XML schema:

```
<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"/>
```

A valid XML document (according to the schema above) could look like this:

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

or like this:

```
<kunde>
```

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

Blocking Element Substitution

To prevent other elements from substituting with a specified element, use the block attribute:

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

Look at this fragment of an XML schema:

```
<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"/>
```

A valid XML document (according to the schema above) looks like this:

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

BUT THIS IS NO LONGER VALID:

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

Using substitutionGroup

The type of the substitutable elements must be the same as, or derived from, the type of the head element. If the type of the substitutable element is the same as the type of the head element you will not have to specify the type of the substitutable element.

Note that all elements in the substitutionGroup (the head element and the substitutable elements) must be declared as global elements, otherwise it will not work!

What are Global Elements?

Global elements are elements that are immediate children of the "schema" element! Local elements are elements nested within other elements.

An XSD Example

This chapter will demonstrate how to write an XML Schema. You will also learn that a schema can be written in different ways.

An XML Document

Let's have a look at this XML document called "shiporder.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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>
```

The XML document above consists of a root element, "shiporder", that contains a required attribute called "orderid". The "shiporder" element contains three different child elements: "orderperson", "shipto" and "item". The "item" element appears twice, and it contains a "title", an optional "note" element, a "quantity", and a "price" element.

The line above: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" tells the XML parser that this document should be validated against a schema. The line: xsi:noNamespaceSchemaLocation="shiporder.xsd" specifies WHERE the schema resides (here it is in the same folder as "shiporder.xml").

Create an XML Schema

Now we want to create a schema for the XML document above.

We start by opening a new file that we will call "shiporder.xsd". To create the schema we could simply follow the structure in the XML document and define each element as we find it. We will start with the standard XML declaration followed by the xs:schema element that defines a schema:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
...


</xs:schema>
```

In the schema above we use the standard namespace (xs), and the URI associated with this namespace is the Schema language definition, which has the standard value of http://www.w3.org/2001/XMLSchema.

Next, we have to define the "shiporder" element. This element has an attribute and it contains other elements, therefore we consider it as a complex type. The child elements of the "shiporder" element is surrounded by a xs:sequence element that defines an ordered sequence of sub elements:

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

Then we have to define the "orderperson" element as a simple type (because it does not contain any attributes or other elements). The type (xs:string) is prefixed with the namespace prefix associated with XML Schema that indicates a predefined schema data type:

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

Next, we have to define two elements that are of the complex type: "shipto" and "item". We start by defining the "shipto" element:

```
<xs:element name="shipto">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="address" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="country" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

With schemas we can define the number of possible occurrences for an element with the maxOccurs and minOccurs attributes. maxOccurs specifies the maximum number of occurrences for an element and minOccurs specifies the minimum number of occurrences for an element. The default value for both maxOccurs and minOccurs is 1!

Now we can define the "item" element. This element can appear multiple times inside a "shiporder" element. This is specified by setting the maxOccurs attribute of the "item" element to "unbounded" which means that there can be as many occurrences of the "item" element as the author wishes. Notice that the "note" element is optional. We have specified this by setting the minOccurs attribute to zero:

```
<xs:element name="item" maxOccurs="unbounded">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="title" type="xs:string"/>
   <xs:element name="note" type="xs:string" minOccurs="0"/>
   <xs:element name="quantity" type="xs:positiveInteger"/>
   <xs:element name="price" type="xs:decimal"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

We can now declare the attribute of the "shiporder" element. Since this is a required attribute we specify use="required".

Note: The attribute declarations must always come last:

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

Here is the complete listing of the schema file called "shiporder.xsd":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="orderperson" type="xs:string"/>
   <xs:element name="shipto">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="orderid" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Divide the Schema

The previous design method is very simple, but can be difficult to read and maintain when documents are complex.

The next design method is based on defining all elements and attributes first, and then referring to them using the ref attribute.

Here is the new design of the schema file ("shiporder.xsd"):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>
<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>
<!-- definition of complex elements -->
<xs:element name="shipto">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="name"/>
   <xs:element ref="address"/>
   <xs:element ref="city"/>
   <xs:element ref="country"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="item">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="title"/>
   <xs:element ref="note" minOccurs="0"/>
   <xs:element ref="quantity"/>
   <xs:element ref="price"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="orderperson"/>
   <xs:element ref="shipto"/>
   <xs:element ref="item" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute ref="orderid" use="required"/>
 </xs:complexType>
</xs:element>
</xs:schema>
```

Using Named Types

The third design method defines classes or types, that enables us to reuse element definitions. This is done by naming the simpleTypes and complexTypes elements, and then point to them through the type attribute of the element.

Here is the third design of the schema file ("shiporder.xsd"):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:simpleType name="stringtype">
 <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="inttype">
 <xs:restriction base="xs:positiveInteger"/>
</xs:simpleType>
<xs:simpleType name="dectype">
 <xs:restriction base="xs:decimal"/>
</xs:simpleType>
<xs:simpleType name="orderidtype">
 <xs:restriction base="xs:string">
  <xs:pattern value="[0-9]{6}"/>
 </xs:restriction>
</xs:simpleType>
<xs:complexType name="shiptotype">
 <xs:sequence>
  <xs:element name="name" type="stringtype"/>
  <xs:element name="address" type="stringtype"/>
  <xs:element name="city" type="stringtype"/>
  <xs:element name="country" type="stringtype"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="itemtype">
 <xs:sequence>
  <xs:element name="title" type="stringtype"/>
  <xs:element name="note" type="stringtype" minOccurs="0"/>
  <xs:element name="quantity" type="inttype"/>
  <xs:element name="price" type="dectype"/>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="shipordertype">
 <xs:sequence>
  <xs:element name="orderperson" type="stringtype"/>
  <xs:element name="shipto" type="shiptotype"/>
  <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
 </xs:sequence>
 <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>
<xs:element name="shiporder" type="shipordertype"/>
</xs:schema>
```

The restriction element indicates that the datatype is derived from a W3C XML Schema namespace datatype. So, the following fragment means that the value of the element or attribute must be a string value:

```
<xs:restriction base="xs:string">
```

The restriction element is more often used to apply restrictions to elements. Look at the following lines from the schema above:

```
<xs:simpleType name="orderidtype">
 <xs:restriction base="xs:string">
  <xs:pattern value="[0-9]{6}"/>
 </xs:restriction>
</xs:simpleType>
```

This indicates that the value of the element or attribute must be a string, it must be exactly six characters in a row, and those characters must be a number from 0 to 9.

XSD String Data Types

String data types are used for values that contains character strings.

String Data Type

The string data type can contain characters, line feeds, carriage returns, and tab characters.

The following is an example of a string declaration in a schema:

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

An element in your document might look like this:

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

Or it might look like this:

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

Note: The XML processor will not modify the value if you use the string data type.

NormalizedString Data Type

The normalizedString data type is derived from the String data type.

The normalizedString data type also contains characters, but the XML processor will remove line feeds, carriage returns, and tab characters.

The following is an example of a normalizedString declaration in a schema:

```
<xs:element name="customer" type="xs:normalizedString"/>
```

An element in your document might look like this:

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

Or it might look like this:

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

Note: In the example above the XML processor will replace the tabs with spaces.

Token Data Type

The token data type is also derived from the String data type.

The token data type also contains characters, but the XML processor will remove line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces.

The following is an example of a token declaration in a schema:

```
<xs:element name="customer" type="xs:token"/>
```

An element in your document might look like this:

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

Or it might look like this:

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

Note: In the example above the XML processor will remove the tabs.

String Data Types

Note that all of the data types below derive from the String data type (except for string itself)!

Name	Description
ENTITIES	
ENTITY	
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
IDREFS	
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NCName	
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
NMTOKENS	
normalizedString	A string that does not contain line feeds, carriage returns, or tabs
QName	
string	A string
token	A string that does not contain line feeds, carriage returns, tabs, leading or

	trailing spaces, or multiple spaces

Restrictions on String Data Types

Restrictions that can be used with String data types:

- enumeration
- length
- maxLength
- minLength
- pattern (NMTOKENS, IDREFS, and ENTITIES cannot use this constraint)
- whiteSpace

XSD Date and Time Data Types

Previous Next

Date and time data types are used for values that contain date and time.

Date Data Type

The date data type is used to specify a date.

The date is specified in the following form "YYYY-MM-DD" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day

Note: All components are required!

The following is an example of a date declaration in a schema:

```
<xs:element name="start" type="xs:date"/>
```

An element in your document might look like this:

```
<start>2002-09-24</start>
```

Time Zones

To specify a time zone, you can either enter a date in UTC time by adding a "Z" behind the date - like this:

```
<start>2002-09-24Z</start>
```

or you can specify an offset from the UTC time by adding a positive or negative time behind the date - like this:

```
<start>2002-09-24-06:00</start>
or
<start>2002-09-24+06:00</start>
```

Time Data Type

The time data type is used to specify a time.

The time is specified in the following form "hh:mm:ss" where:

- hh indicates the hour
- mm indicates the minute
- ss indicates the second

Note: All components are required!

The following is an example of a time declaration in a schema:

```
<xs:element name="start" type="xs:time"/>
```

An element in your document might look like this:

```
<start>09:00:00</start>
```

Or it might look like this:

```
<start>09:30:10.5</start>
```

Time Zones

To specify a time zone, you can either enter a time in UTC time by adding a "Z" behind the time - like this:

```
<start>09:30:10Z</start>
```

or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<start>09:30:10-06:00</start>
or
<start>09:30:10+06:00</start>
```

DateTime Data Type

The dateTime data type is used to specify a date and a time.

The dateTime is specified in the following form "YYYY-MM-DDThh:mm:ss" where:

- YYYY indicates the year
- MM indicates the month
- DD indicates the day
- T indicates the start of the required time section
- hh indicates the hour
- mm indicates the minute
- ss indicates the second

Note: All components are required!

The following is an example of a dateTime declaration in a schema:

```
<xs:element name="startdate" type="xs:dateTime"/>
```

An element in your document might look like this:

```
<startdate>2002-05-30T09:00:00</startdate>
```

Or it might look like this:

```
<startdate>2002-05-30T09:30:10.5</startdate>
```

Time Zones

To specify a time zone, you can either enter a dateTime in UTC time by adding a "Z" behind the time - like this:

```
<startdate>2002-05-30T09:30:10Z</startdate>
```

or you can specify an offset from the UTC time by adding a positive or negative time behind the time - like this:

```
<startdate>2002-05-30T09:30:10-06:00</startdate>
or
<startdate>2002-05-30T09:30:10+06:00</startdate>
```

Duration Data Type

The duration data type is used to specify a time interval.

The time interval is specified in the following form "PnYnMnDTnHnMnS" where:

- P indicates the period (required)
- nY indicates the number of years
- nM indicates the number of months
- nD indicates the number of days
- T indicates the start of a time section (required if you are going to specify hours, minutes, or seconds)
- nH indicates the number of hours

- nM indicates the number of minutes
- nS indicates the number of seconds

The following is an example of a duration declaration in a schema:

```
<xs:element name="period" type="xs:duration"/>
```

An element in your document might look like this:

```
<period>P5Y</period>
```

The example above indicates a period of five years.

Or it might look like this:

```
<period>P5Y2M10D</period>
```

The example above indicates a period of five years, two months, and 10 days.

Or it might look like this:

```
<period>P5Y2M10DT15H</period>
```

The example above indicates a period of five years, two months, 10 days, and 15 hours.

Or it might look like this:

```
<period>PT15H</period>
```

The example above indicates a period of 15 hours.

Negative Duration

To specify a negative duration, enter a minus sign before the P:

```
<period>-P10D</period>
```

The example above indicates a period of minus 10 days.

Date and Time Data Types

Name	Description
date	Defines a date value
dateTime	Defines a date and time value
duration	Defines a time interval
gDay	Defines a part of a date - the day (DD)
gMonth	Defines a part of a date - the month (MM)
gMonthDay	Defines a part of a date - the month and day (MM-DD)
gYear	Defines a part of a date - the year (YYYY)
gYearMonth	Defines a part of a date - the year and month (YYYY-MM)

time	Defines a time value

Restrictions on Date Data Types

Restrictions that can be used with Date data types:

- enumeration
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- whiteSpace

XSD Numeric Data Types

Previous Next

Decimal data types are used for numeric values.

Decimal Data Type

The decimal data type is used to specify a numeric value.

The following is an example of a decimal declaration in a schema:

```
<xs:element name="prize" type="xs:decimal"/>
```

An element in your document might look like this:

```
<prize>999.50</prize>
```

Or it might look like this:

```
<prize>+999.5450</prize>
```

Or it might look like this:

```
<prize>-999.5230</prize>
```

Or it might look like this:

```
<prize>0</prize>
```

Or it might look like this:

```
<prize>14</prize>
```

Note: The maximum number of decimal digits you can specify is 18.

Integer Data Type

The integer data type is used to specify a numeric value without a fractional component.

The following is an example of an integer declaration in a schema:

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

An element in your document might look like this:

```
<prize>999</prize>
```

Or it might look like this:

```
<prize>+999</prize>
```

Or it might look like this:

```
<prize>-999</prize>
```

Or it might look like this:

```
<prize>0</prize>
```

Numeric Data Types

Note that all of the data types below derive from the Decimal data type (except for decimal itself)!

Name	Description
byte	A signed 8-bit integer
decimal	A decimal value
int	A signed 32-bit integer
integer	An integer value
long	A signed 64-bit integer
negativeInteger	An integer containing only negative values (.., -2, -1.)
nonNegativeInteger	An integer containing only non-negative values (0, 1, 2, ..)
nonPositiveInteger	An integer containing only non-positive values (.., -2, -1, 0)
positiveInteger	An integer containing only positive values (1, 2, ..)
short	A signed 16-bit integer
unsignedLong	An unsigned 64-bit integer
unsignedInt	An unsigned 32-bit integer
unsignedShort	An unsigned 16-bit integer
unsignedByte	An unsigned 8-bit integer

Restrictions on Numeric Data Types

Restrictions that can be used with Numeric data types:

- enumeration
- fractionDigits
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- totalDigits
- whiteSpace

XSD Miscellaneous Data Types

Previous Next

Other miscellaneous data types are boolean, base64Binary, hexBinary, float, double, anyURI, QName, and NOTATION.

Boolean Data Type

The boolean data type is used to specify a true or false value.

The following is an example of a boolean declaration in a schema:

```
<xs:attribute name="disabled" type="xs:boolean"/>
```

An element in your document might look like this:

```
<prize disabled="true">999</prize>
```

Note: Legal values for boolean are true, false, 1 (which indicates true), and 0 (which indicates false).

Binary Data Types

Binary data types are used to express binary-formatted data.

We have two binary data types:

- base64Binary (Base64-encoded binary data)
- hexBinary (hexadecimal-encoded binary data)

The following is an example of a hexBinary declaration in a schema:

```
<xs:element name="blobsrc" type="xs:hexBinary"/>
```

AnyURI Data Type

The anyURI data type is used to specify a URI.

The following is an example of an anyURI declaration in a schema:

```
<xs:attribute name="src" type="xs:anyURI"/>
```

An element in your document might look like this:

```
<pic src="http://www.w3schools.com/images/smiley.gif" />
```

Note: If a URI has spaces, replace them with %20.

Miscellaneous Data Types

Name	Description
anyURI	
base64Binary	
boolean	
double	
float	
hexBinary	
NOTATION	
QName	

Restrictions on Miscellaneous Data Types

Restrictions that can be used with the other data types:

- enumeration (a Boolean data type cannot use this constraint)
- length (a Boolean data type cannot use this constraint)
- maxLength (a Boolean data type cannot use this constraint)
- minLength (a Boolean data type cannot use this constraint)
- pattern
- whiteSpace

You Have Learned XML Schema, Now What?

Previous Next

XML Schema Summary

This tutorial has taught you how to describe the structure of an XML document.

You have learned how to use an XML Schema is to define the legal elements of an XML document, just like a DTD. We think that very soon XML Schemas will be used in most Web applications as a replacement for DTDs.

You have also learned that the XML Schema language is very rich. Unlike DTDs, it supports data types and namespaces.

For more information on XML Schema, please look at our XML Schema reference.