

# XSD

The XML Schema language is also referred to as **XML Schema Definition**.

An XML Schema describes the structure of an XML document.

An XML Schema Definition describes and validates the elements, attributes and data types of XML document instances that refer to it.

XSD Example:

```
<?xml version="1.0"?>
<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>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

1. The elements and attributes that can appear in a document;
2. The number of (and order of) child elements;
3. Data types for elements and attributes;
4. Default and fixed values for elements and attributes.

Regarding the support for data types:

- Easy to describe allowed data content;
- Easy to validate correctness of data;
- Easy to define data facets (constraints or restrictions);
- Easy to define data patterns (data formats);
- Easy to convert from different data types.

Extensibility:

- One schema can be reused at another schema via import or include;
- You can create your own custom data types based on standard data types;
- Refer to multiple schemas in the same document.

Security:

- When being sent from a sender to a receiver, the data can be described in a way that it is understood by both;
- The validation against an XSD, added to the well formation of the XML document itself, can catch or reduce the risk of passing data errors on a message, automatically, using software.

Characteristics:

- XSD can be a representation – archetype or blueprint – to all XML document instances you might have
- XSD can work as a standard for different groups of people to interchange data among them
- XSD makes it possible for interchanged data to be validated against
- XSD is easy to edit, parse, manipulate ([DOM](#)) and transform ([XSLT](#)), the same way as with XML documents instances, and with the same related technologies.

## XSD Schema

The basic syntax of a XSD is as follows

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

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

</xs:schema>
```

### XSD <schema> Element

The **<schema>** element is the root element of every XML 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="https://www.w3schools.com"
xmlns="https://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="https://www.w3schools.com"
```

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

This fragment ...

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

... indicates that the default namespace is "https://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.

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>

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

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

The following fragment...

```
xmlns="https://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 "https://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, separated by a space:

1. The first value is the namespace to use;
2. The second value is the location of the XML schema to use for that namespace.

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

## XSD Simple Elements

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

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"/>
```

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.

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

### 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 Element Substitution

With XML Schemas, one element can substitute another element.

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>
```

## XSD <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>
```

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

  <xs:element name="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="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
https://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 Complex Elements

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

There are four kinds of complex elements:

1. Empty elements;
2. Elements that contain only other elements;
3. Elements that contain only text;
4. Elements that contain both other elements and text.

**Note:** Each of these elements may contain attributes as well.

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 type on an existing complex type 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>
```



## Complex Empty Elements

An empty complex element cannot have contents, only attributes.

```
<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 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>
```

## Complex Text-Only Elements

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

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:

<pre>&lt;xs:element name="somename"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:extension base="basetype"&gt;         ....       &lt;/xs:extension&gt;     &lt;/xs:simpleContent&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>	<pre>&lt;xs:element name="somename"&gt;   &lt;xs:complexType&gt;     &lt;xs:simpleContent&gt;       &lt;xs:restriction base="basetype"&gt;         ....       &lt;/xs:restriction&gt;     &lt;/xs:simpleContent&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt;</pre>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**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>
```

## Complex Types with Mixed Content

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

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 Data Types

## XSD String

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.

## XSD Date and Time

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

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>
```

Restrictions that can be used with Date data types:

- enumeration

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

### 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

### 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    <start>09:30:10.5</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    or    <startdate>2002-05-30T09:30:10.5
</startdate>                    </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>
```

## XSD Numeric

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="price" type="xs:decimal"/>
```

An element in your document might look like this:

```
<price>999.50</price>
```

```
<price>+999.5450</price>
```

```
<price>-999.5230</price>
```

```
<price>0</price>
```

```
<price>14</price>
```

## 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="price" type="xs:integer"/>
```

An element in your document might look like this:

```
<price>999</price>
```

```
<price>+999</price>
```

```
<price>-999</price>
```

```
<price>0</price>
```

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

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

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.

### 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:

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

**Note:** Legal values for boolean are true or 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:

1. base64Binary (Base64-encoded binary data);
2. 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="https://www.w3schools.com/images/smiley.gif" />
```

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

## XSD Attributes

Simple elements can't 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.

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

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"/>
```

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

### 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 "eyecolor" attribute. In this case we can do so, even if the author of the schema above never declared any "eyecolor" attribute.

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

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

```

<xs:attribute name="eyecolor">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="blue|brown|green|grey"/>
    </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="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
https://www.w3schools.com attribute.xsd">

  <person eyecolor="green">
    <firstname>Hege</firstname>
    <lastname>Refsnes</lastname>
  </person>

  <person eyecolor="blue">
    <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 Restrictions and Facets

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

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>
```

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 "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>

```

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 uppercase 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 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>

```

```
</xs:restriction>
</xs:simpleType>
</xs:element>
```

### Restrictions for Data Types

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 Indicators

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

There are seven indicators:

Order indicators	Occurrence indicators	Group indicators
<ul style="list-style-type: none"><li>- All</li><li>- Choice</li><li>- Sequence</li></ul>	<ul style="list-style-type: none"><li>- maxOccurs</li><li>- minOccurs</li></ul>	<ul style="list-style-type: none"><li>- Group name</li><li>- attributeGroup name</li></ul>

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>
```