DTD Tutorial

[❮ Previous](https://www.w3schools.com/xml/xquery_functions.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_building.asp)

What is a DTD?

A DTD is a Document Type Definition.

A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

With a DTD, independent groups of people can agree on a standard DTD for interchanging data.

An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

XML document with an internal DTD

<?xml version="1.0"?>  
<!DOCTYPE note [  
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>  
]>  
<note>  
<to>Tove</to>  
<from>Jani</from>  
<heading>Reminder</heading>  
<body>Don't forget me this weekend</body>  
</note>

[View XML file »](https://www.w3schools.com/xml/note_in_dtd.xml)

In the XML file, select "view source" to view the DTD.

The DTD above is interpreted like this:

* **!DOCTYPE note** defines that the root element of this document is note
* **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
* **!ELEMENT to** defines the to element to be of type "#PCDATA"
* **!ELEMENT from** defines the from element to be of type "#PCDATA"
* **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
* **!ELEMENT body** defines the body element to be of type "#PCDATA"

An External DTD Declaration

If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

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

[View XML file »](https://www.w3schools.com/xml/note_ex_dtd.xml)

And here is the file "note.dtd", which contains the DTD:

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

DTD - XML Building Blocks

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_intro.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_elements.asp)

The main building blocks of both XML and HTML documents are elements.

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

* Elements
* Attributes
* Entities
* PCDATA
* CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

<body>some text</body>  
  
<message>some text</message>

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

<img src="computer.gif" />

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.

Most of you know the HTML entity: "&nbsp;". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

|  |  |
| --- | --- |
| **Entity References** | **Character** |
| &lt; | < |
| &gt; | > |
| &amp; | & |
| &quot; | " |
| &apos; | ' |

PCDATA

PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

**PCDATA is text that WILL be parsed by a parser**. **The text will be examined by the parser for entities and markup**.

Tags inside the text will be treated as markup and entities will be expanded.

However, parsed character data should not contain any &, <, or > characters; these need to be represented by the &amp; &lt; and &gt; entities, respectively.

CDATA

CDATA means character data.

**CDATA is text that will NOT be parsed by a parser**. Tags inside the text will NOT be treated as markup and entities will not be expanded.

DTD - Elements

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_building.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_attributes.asp)

In a DTD, elements are declared with an ELEMENT declaration.

Declaring Elements

In a DTD, XML elements are declared with the following syntax:

<!ELEMENT element-name category>  
or  
<!ELEMENT element-name (element-content)>

Empty Elements

Empty elements are declared with the category keyword EMPTY:

<!ELEMENT element-name EMPTY>  
  
Example:  
  
<!ELEMENT br EMPTY>  
  
XML example:  
  
<br />

Elements with Parsed Character Data

Elements with only parsed character data are declared with #PCDATA inside parentheses:

<!ELEMENT element-name (#PCDATA)>  
  
Example:  
  
<!ELEMENT from (#PCDATA)>

Elements with any Contents

Elements declared with the category keyword ANY, can contain any combination of parsable data:

<!ELEMENT element-name ANY>  
  
Example:  
  
<!ELEMENT note ANY>

Elements with Children (sequences)

Elements with one or more children are declared with the name of the children elements inside parentheses:

<!ELEMENT element-name (child1)>  
or  
<!ELEMENT element-name (child1,child2,...)>  
  
Example:  
  
<!ELEMENT note (to,from,heading,body)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the "note" element is:

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

Declaring Only One Occurrence of an Element

<!ELEMENT element-name (child-name)>  
  
Example:  
  
<!ELEMENT note (message)>

The example above declares that the child element "message" must occur once, and only once inside the "note" element.

Declaring Minimum One Occurrence of an Element

<!ELEMENT element-name (child-name+)>  
  
Example:  
  
<!ELEMENT note (message+)>

The + sign in the example above declares that the child element "message" must occur one or more times inside the "note" element.

Declaring Zero or More Occurrences of an Element

<!ELEMENT element-name (child-name\*)>  
  
Example:  
  
<!ELEMENT note (message\*)>

The \* sign in the example above declares that the child element "message" can occur zero or more times inside the "note" element.

Declaring Zero or One Occurrences of an Element

<!ELEMENT element-name (child-name?)>  
  
Example:  
  
<!ELEMENT note (message?)>

The ? sign in the example above declares that the child element "message" can occur zero or one time inside the "note" element.

Declaring either/or Content

<!ELEMENT note (to,from,header,(message|body))>

The example above declares that the "note" element must contain a "to" element, a "from" element, a "header" element, and either a "message" or a "body" element.

Declaring Mixed Content

<!ELEMENT note (#PCDATA|to|from|header|message)\*>

The example above declares that the "note" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "message" elements.

DTD - Attributes

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_elements.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_el_vs_attr.asp)

In a DTD, attributes are declared with an ATTLIST declaration.

Declaring Attributes

An attribute declaration has the following syntax:

<!ATTLIST element-name attribute-name attribute-type attribute-value>  
  
DTD example:  
  
<!ATTLIST payment type CDATA "check">  
  
XML example:  
  
<payment type="check" />

The **attribute-type** can be one of the following:

|  |  |
| --- | --- |
| **Type** | **Description** |
| CDATA | The value is character data |
| (*en1*|*en2*|..) | The value must be one from an enumerated list |
| ID | The value is a unique id |
| IDREF | The value is the id of another element |
| IDREFS | The value is a list of other ids |
| NMTOKEN | The value is a valid XML name |
| NMTOKENS | The value is a list of valid XML names |
| ENTITY | The value is an entity |
| ENTITIES | The value is a list of entities |
| NOTATION | The value is a name of a notation |
| xml: | The value is a predefined xml value |

The **attribute-value** can be one of the following:

|  |  |
| --- | --- |
| **Value** | **Explanation** |
| *value* | The default value of the attribute |
| #REQUIRED | The attribute is required |
| #IMPLIED | The attribute is optional |
| #FIXED *value* | The attribute value is fixed |

A Default Attribute Value

DTD:  
<!ELEMENT square EMPTY>  
<!ATTLIST square width CDATA "0">  
  
Valid XML:  
<square width="100" />

In the example above, the "square" element is defined to be an empty element with a "width" attribute of  type CDATA. If no width is specified, it has a default value of 0.

#REQUIRED

Syntax

<!ATTLIST element-name attribute-name attribute-type #REQUIRED>

Example

DTD:  
<!ATTLIST person number CDATA #REQUIRED>  
  
Valid XML:  
<person number="5677" />  
  
Invalid XML:  
<person />

Use the #REQUIRED keyword if you don't have an option for a default value, but still want to force the attribute to be present.

#IMPLIED

Syntax

<!ATTLIST element-name attribute-name attribute-type #IMPLIED>

Example

DTD:  
<!ATTLIST contact fax CDATA #IMPLIED>  
  
Valid XML:  
<contact fax="555-667788" />  
  
Valid XML:  
<contact />

Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

#FIXED

Syntax

<!ATTLIST element-name attribute-name attribute-type #FIXED "value">

Example

DTD:  
<!ATTLIST sender company CDATA #FIXED "Microsoft">  
  
Valid XML:  
<sender company="Microsoft" />  
  
Invalid XML:  
<sender company="W3Schools" />

Use the #FIXED keyword when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

Enumerated Attribute Values

Syntax

<!ATTLIST element-name attribute-name (en1|en2|..) default-value>

Example

DTD:  
<!ATTLIST payment type (check|cash) "cash">  
  
XML example:  
<payment type="check" />  
or  
<payment type="cash" />

Use enumerated attribute values when you want the attribute value to be one of a fixed set of legal values.

XML Elements vs. Attributes

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_attributes.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_entities.asp)

In XML, there are no rules about when to use attributes, and when to use child elements.

Use of Elements vs. Attributes

Data can be stored in child elements or in attributes.

Take a look at these examples:

<person sex="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>

<person>  
  <sex>female</sex>  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>

In the first example sex is an attribute. In the last, sex is a child element. Both examples provide the same information.

There are no rules about when to use attributes, and when to use child elements. My experience is that attributes are handy in HTML, but in XML you should try to avoid them. Use child elements if the information feels like data.

My Favorite Way

**I like to store data in child elements.**

The following three XML documents contain exactly the same information:

A date attribute is used in the first example:

<note date="12/11/2002">  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>

A date element is used in the second example:

<note>  
  <date>12/11/2002</date>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>

An expanded date element is used in the third: (THIS IS MY FAVORITE):

<note>  
  <date>  
    <day>12</day>  
    <month>11</month>  
    <year>2002</year>  
  </date>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>

Avoid using attributes?

Should you avoid using attributes?

Some of the problems with attributes are:

* attributes cannot contain multiple values (child elements can)
* attributes are not easily expandable (for future changes)
* attributes cannot describe structures (child elements can)
* attributes are more difficult to manipulate by program code
* attribute values are not easy to test against a DTD

If you use attributes as containers for data, you end up with documents that are difficult to read and maintain. Try to use **elements** to describe data. Use attributes only to provide information that is not relevant to the data.

Don't end up like this (this is not how XML should be used):

<note day="12" month="11" year="2002"  
to="Tove" from="Jani" heading="Reminder"  
body="Don't forget me this weekend!">  
</note>

An Exception to my Attribute Rule

Rules always have exceptions.

My rule about attributes has one exception:

Sometimes I assign ID references to elements. These ID references can be used to access XML elements in much the same way as the NAME or ID attributes in HTML. This example demonstrates this:

<messages>  
<note id="p501">  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>  
  
<note id="p502">  
  <to>Jani</to>  
  <from>Tove</from>  
  <heading>Re: Reminder</heading>  
  <body>I will not!</body>  
</note>  
</messages>

The ID in these examples is just a counter, or a unique identifier, to identify the different notes in the XML file, and not a part of the note data.

What I am trying to say here is that metadata (data about data) should be stored as attributes, and that data itself should be stored as elements.

DTD - Entities

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_el_vs_attr.asp)[Next ❯](https://www.w3schools.com/xml/xml_dtd_examples.asp)

Entities are used to define shortcuts to special characters.

Entities can be declared internal or external.

An Internal Entity Declaration

Syntax

<!ENTITY entity-name "entity-value">

Example

DTD Example:  
  
<!ENTITY writer "Donald Duck.">  
<!ENTITY copyright "Copyright W3Schools.">  
  
XML example:  
  
<author>&writer;&copyright;</author>

**Note:** An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration

Syntax

<!ENTITY entity-name SYSTEM "URI/URL">

Example

DTD Example:  
  
<!ENTITY writer SYSTEM "https://www.w3schools.com/entities.dtd">  
<!ENTITY copyright SYSTEM "https://www.w3schools.com/entities.dtd">  
  
XML example:  
  
<author>&writer;&copyright;</author>

DTD Examples

[❮ Previous](https://www.w3schools.com/xml/xml_dtd_entities.asp)[Next ❯](https://www.w3schools.com/xml/schema_intro.asp)

TV Schedule DTD

<!DOCTYPE TVSCHEDULE [  
  
<!ELEMENT TVSCHEDULE (CHANNEL+)>  
<!ELEMENT CHANNEL (BANNER,DAY+)>  
<!ELEMENT BANNER (#PCDATA)>  
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>  
<!ELEMENT HOLIDAY (#PCDATA)>  
<!ELEMENT DATE (#PCDATA)>  
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>  
<!ELEMENT TIME (#PCDATA)>  
<!ELEMENT TITLE (#PCDATA)>   
<!ELEMENT DESCRIPTION (#PCDATA)>  
  
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  
<!ATTLIST TITLE RATING CDATA #IMPLIED>  
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  
]>

Newspaper Article DTD

<!DOCTYPE NEWSPAPER [  
  
<!ELEMENT NEWSPAPER (ARTICLE+)>  
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  
<!ELEMENT HEADLINE (#PCDATA)>  
<!ELEMENT BYLINE (#PCDATA)>  
<!ELEMENT LEAD (#PCDATA)>  
<!ELEMENT BODY (#PCDATA)>  
<!ELEMENT NOTES (#PCDATA)>  
  
<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>  
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  
  
<!ENTITY NEWSPAPER "Vervet Logic Times">  
<!ENTITY PUBLISHER "Vervet Logic Press">  
<!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">  
  
]>

Product Catalog DTD

<!DOCTYPE CATALOG [  
  
<!ENTITY AUTHOR "John Doe">  
<!ENTITY COMPANY "JD Power Tools, Inc.">  
<!ENTITY EMAIL "jd@jd-tools.com">  
  
<!ELEMENT CATALOG (PRODUCT+)>  
  
<!ELEMENT PRODUCT  
(SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>  
<!ATTLIST PRODUCT  
NAME CDATA #IMPLIED  
CATEGORY (HandTool|Table|Shop-Professional) "HandTool"  
PARTNUM CDATA #IMPLIED  
PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"  
INVENTORY (InStock|Backordered|Discontinued) "InStock">  
  
<!ELEMENT SPECIFICATIONS (#PCDATA)>  
<!ATTLIST SPECIFICATIONS  
WEIGHT CDATA #IMPLIED  
POWER CDATA #IMPLIED>  
  
<!ELEMENT OPTIONS (#PCDATA)>  
<!ATTLIST OPTIONS  
FINISH (Metal|Polished|Matte) "Matte"  
ADAPTER (Included|Optional|NotApplicable) "Included"  
CASE (HardShell|Soft|NotApplicable) "HardShell">  
  
<!ELEMENT PRICE (#PCDATA)>  
<!ATTLIST PRICE  
MSRP CDATA #IMPLIED  
WHOLESALE CDATA #IMPLIED  
STREET CDATA #IMPLIED  
SHIPPING CDATA #IMPLIED>  
  
<!ELEMENT NOTES (#PCDATA)>  
  
]>

What is an XML Schema?

An XML Schema describes the structure of an XML document.

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

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:

* the elements and attributes that can appear in a document
* the number of (and order of) child elements
* data types for elements and attributes
* default and fixed values for elements and attributes

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.

Many of these XML standards are defined by XML Schemas.

XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the 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 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.

* 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 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

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".

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.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

# XSD How To?

[❮ Previous](https://www.w3schools.com/xml/schema_intro.asp)[Next ❯](https://www.w3schools.com/xml/schema_schema.asp)

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="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>

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

# XSD - The <schema> Element

[❮ Previous](https://www.w3schools.com/xml/schema_howto.asp)[Next ❯](https://www.w3schools.com/xml/schema_simple.asp)

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="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.

## Referencing a Schema in an XML Document

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. 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="https://www.w3schools.com note.xsd"

# XSD Simple Elements

[❮ Previous](https://www.w3schools.com/xml/schema_schema.asp)[Next ❯](https://www.w3schools.com/xml/schema_simple_attributes.asp)

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](https://www.w3schools.com/xml/schema_simple.asp)[Next ❯](https://www.w3schools.com/xml/schema_facets.asp)

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

[❮ Previous](https://www.w3schools.com/xml/schema_simple_attributes.asp)[Next ❯](https://www.w3schools.com/xml/schema_complex.asp)

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 handle |

# XSD Complex Elements

[❮ Previous](https://www.w3schools.com/xml/schema_facets.asp)[Next ❯](https://www.w3schools.com/xml/schema_complex_empty.asp)

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>