XML markup:

<message>

<text>Hello, world!</text>

</message>

Syntax rule to write XML doc:

<?xml version = "1.0"?>

<contact-info>

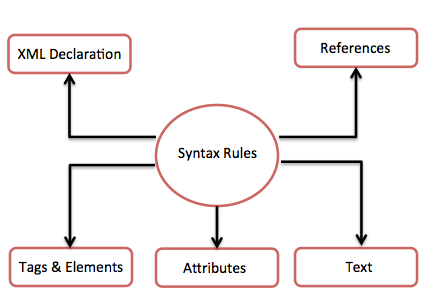
<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

</contact-info>

The syntax rules to write different types of markup and text in an XML document.



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

*version* is the XML version and *encoding* specifies the character encoding used in the document.

This chapter covers XML declaration in detail. **XML declaration** contains details that prepare an XML processor to parse the XML document. It is optional, but when used, it must appear in the first line of the XML document.

Syntax

Following syntax shows XML declaration −

<?xml

version = "version\_number"

encoding = "encoding\_declaration"

standalone = "standalone\_status"

?>

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote. Following table shows the above syntax in detail −

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Parameter\_value** | **Parameter\_description** |
| Version | 1.0 | Specifies the version of the XML standard used. |
| Encoding | UTF-8, UTF-16, ISO-10646-UCS-2, ISO-10646-UCS-4, ISO-8859-1 to ISO-8859-9, ISO-2022-JP, Shift\_JIS, EUC-JP | It defines the character encoding used in the document. UTF-8 is the default encoding used. |
| Standalone | yes or no | It informs the parser whether the document relies on the information from an external source, such as external document type definition (DTD), for its content. The default value is set to *no*. Setting it to *yes* tells the processor there are no external declarations required for parsing the document. |

Rules

An XML declaration should abide with the following rules −

* If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
* If the XML declaration is included, it must contain version number attribute.
* The Parameter names and values are case-sensitive.
* The names are always in lower case.
* The order of placing the parameters is important. The correct order is: *version, encoding and standalone.*
* Either single or double quotes may be used.
* The XML declaration has no closing tag i.e. **</?xml>**

XML Declaration Examples

Following are few examples of XML declarations −

XML declaration with no parameters −

<?xml >

XML declaration with version definition −

<?xml version = "1.0">

XML declaration with all parameters defined −

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

XML declaration with all parameters defined in single quotes −

<?xml version = '1.0' encoding = 'iso-8859-1' standalone = 'no' ?>

**XML attributes**. Attributes are part of XML elements. An element can have multiple unique attributes. Attribute gives more information about XML elements. To be more precise, they define properties of elements. An XML attribute is always a name-value pair.

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

<!DOCTYPE garden [

<!ELEMENT garden (plants)\*>

<!ELEMENT plants (#PCDATA)>

<!ATTLIST plants category CDATA #REQUIRED>

]>

<garden>

<plants category = "flowers" />

<plants category = "shrubs">

</plants>

</garden>

Following is the syntax for CDATA section −

<![CDATA[

characters with markup

]]>

The above syntax is composed of three sections −

* **CDATA Start section** − CDATA begins with the nine-character delimiter **<![CDATA[**
* **CDATA End section** − CDATA section ends with **]]>** delimiter.
* **CData section** − Characters between these two enclosures are interpreted as characters, and not as markup. This section may contain markup characters (<, >, and &), but they are ignored by the XML processor.

Example

The following markup code shows an example of CDATA. Here, each character written inside the CDATA section is ignored by the parser.

<script>

<![CDATA[

<message> Welcome to TutorialsPoint </message>

]] >

</script >

In the above syntax, everything between <message> and </message> is treated as character data and not as markup.

CDATA Rules

The given rules are required to be followed for XML CDATA −

* CDATA cannot contain the string "]]>" anywhere in the XML document.
* Nesting is not allowed in CDATA section.

This chapter describes the **Processing Instructions (PIs)**. As defined by the XML 1.0 Recommendation,

"Processing instructions (PIs) allow documents to contain instructions for applications. PIs are not part of the character data of the document, but MUST be passed through to the application.

Processing instructions (PIs) can be used to pass information to applications. PIs can appear anywhere in the document outside the markup. They can appear in the prolog, including the document type definition (DTD), in textual content, or after the document.

Syntax

Following is the syntax of PI −

<?target instructions?>

Where

* **target** − Identifies the application to which the instruction is directed.
* **instruction** − A character that describes the information for the application to process.

A PI starts with a special tag **<?** and ends with **?>**. Processing of the contents ends immediately after the string **?>** is encountered.

Example

PIs are rarely used. They are mostly used to link XML document to a style sheet. Following is an example −

<?xml-stylesheet href = "tutorialspointstyle.css" type = "text/css"?>

Here, the *target* is *xml-stylesheet*. *href="tutorialspointstyle.css"* and *type="text/css"* are *data* or *instructions* the target application will use at the time of processing the given XML document.

In this case, a browser recognizes the target by indicating that the XML should be transformed before being shown; the first attribute states that the type of the transform is XSL and the second attribute points to its location.

Processing Instructions Rules

A PI can contain any data except the combination **?>**, which is interpreted as the closing delimiter. Here are two examples of valid PIs −

<?welcome to pg = 10 of tutorials point?>

<?welcome?>

**Validation** is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration(DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are −

* Well-formed XML document
* Valid XML document

Well-formed XML Document

An XML document is said to be **well-formed** if it adheres to the following rules −

* Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
* It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
* Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
* It must have only one attribute in a start tag, which needs to be quoted.
* **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Example

Following is an example of a well-formed XML document −

<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>

<!DOCTYPE address

[

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

]>

<address>

<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

</address>

The above example is said to be well-formed as −

* It defines the type of document. Here, the document type is **element** type.
* It includes a root element named as **address**.
* Each of the child elements among name, company and phone is enclosed in its self explanatory tag.
* Order of the tags is maintained.

Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document. We will study more about DTD in the chapter [XML - DTDs](https://www.tutorialspoint.com/xml/xml_dtds.htm).

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

## Syntax

Basic syntax of a DTD is as follows −

<!DOCTYPE element DTD identifier

[

declaration1

declaration2

........

]>

In the above syntax,

* The **DTD** starts with <!DOCTYPE delimiter.
* An **element** tells the parser to parse the document from the specified root element.
* **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
* **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

## Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

### Syntax

Following is the syntax of internal DTD −

<!DOCTYPE root-element [element-declarations]>

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

### Example

Following is a simple example of internal DTD −

<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>

<!DOCTYPE address [

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

]>

<address>

<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

</address>

Let us go through the above code −

**Start Declaration** − Begin the XML declaration with the following statement.

<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>

**DTD** − Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE −

<!DOCTYPE address [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

**DTD Body** − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone\_no (#PCDATA)>

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

**End Declaration** − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]>**). This effectively ends the definition, and thereafter, the XML document follows immediately.

### Rules

* The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
* Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
* The Name in the document type declaration must match the element type of the root element.

## External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

### Syntax

Following is the syntax for external DTD −

<!DOCTYPE root-element SYSTEM "file-name">

where *file-name* is the file with *.dtd* extension.

### Example

The following example shows external DTD usage −

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

<!DOCTYPE address SYSTEM "address.dtd">

<address>

<name>Tanmay Patil</name>

<company>TutorialsPoint</company>

<phone>(011) 123-4567</phone>

</address>

The content of the DTD file **address.dtd** is as shown −

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

## Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

### System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

<!DOCTYPE name SYSTEM "address.dtd" [...]>

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

### Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows −

<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers, or FPIs**.

XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

## Syntax

You need to declare a schema in your XML document as follows −

### Example

The following example shows how to use schema −

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

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

<xs:element name = "contact">

<xs:complexType>

<xs:sequence>

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

<xs:element name = "company" type = "xs:string" />

<xs:element name = "phone" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

## Elements

As we saw in the [XML - Elements](https://www.tutorialspoint.com/xml/xml_elements.htm) chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows −

<xs:element name = "x" type = "y"/>

## Definition Types

You can define XML schema elements in the following ways −

### Simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example −

<xs:element name = "phone\_number" type = "xs:int" />

### Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example −

<xs:element name = "Address">

<xs:complexType>

<xs:sequence>

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

<xs:element name = "company" type = "xs:string" />

<xs:element name = "phone" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

In the above example, *Address* element consists of child elements. This is a container for other **<xs:element>** definitions, that allows to build a simple hierarchy of elements in the XML document.

### Global Types

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

<xs:element name = "AddressType">

<xs:complexType>

<xs:sequence>

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

<xs:element name = "company" type = "xs:string" />

</xs:sequence>

</xs:complexType>

</xs:element>

Now let us use this type in our example as follows −

<xs:element name = "Address1">

<xs:complexType>

<xs:sequence>

<xs:element name = "address" type = "AddressType" />

<xs:element name = "phone1" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

<xs:element name = "Address2">

<xs:complexType>

<xs:sequence>

<xs:element name = "address" type = "AddressType" />

<xs:element name = "phone2" type = "xs:int" />

</xs:sequence>

</xs:complexType>

</xs:element>

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

## Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below −

<xs:attribute name = "x" type = "y"/>

An XML document is always descriptive. The tree structure is often referred to as **XML Tree** and plays an important role to describe any XML document easily.

The tree structure contains root (parent) elements, child elements and so on. By using tree structure, you can get to know all succeeding branches and sub-branches starting from the root. The parsing starts at the root, then moves down the first branch to an element, take the first branch from there, and so on to the leaf nodes.

Example

Following example demonstrates simple XML tree structure −

<?xml version = "1.0"?>

<Company>

<Employee>

<FirstName>Tanmay</FirstName>

<LastName>Patil</LastName>

<ContactNo>1234567890</ContactNo>

<Email>tanmaypatil@xyz.com</Email>

<Address>

<City>Bangalore</City>

<State>Karnataka</State>

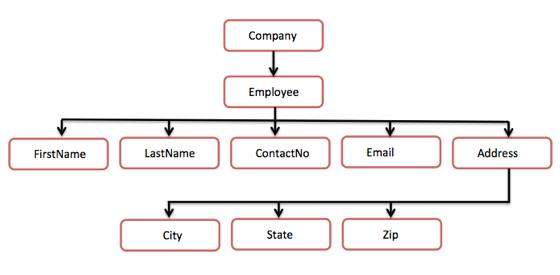
<Zip>560212</Zip>

</Address>

</Employee>

</Company>

Following tree structure represents the above XML document −



In the above diagram, there is a root element named as <company>. Inside that, there is one more element <Employee>. Inside the employee element, there are five branches named <FirstName>, <LastName>, <ContactNo>, <Email>, and <Address>. Inside the <Address> element, there are three sub-branches, named <City> <State> and <Zip>.

# Displaying XML Using XSLT

**XSLT** stands for **E**xtensible **S**tylesheet **L**anguage **T**ransformation.

* XSLT is used to transform XML document from one form to another form.
* XSLT uses Xpath to perform matching of nodes to perform these transformation .
* The result of applying XSLT to XML document could be an another XML document, HTML, text or any another document from technology prespective.
* The XSL code is written within the XML document with the extension of *(.xsl)*.
* In other words, an XSLT document is a different kind of XML document.

**XML Namespace:** XML Namespaces are the unique names .

* XML Namespace is a mechanism by which element or attribute is assigned to a group.
* XML Namespace is used to avoid the name conflicts in the XML document.
* XML Namespace is recommended by W3C.

**XML Namespace Declaration:**  
It is declared using reserved attribute such as the attribute is *xmlns* or it can begin with *xmlns:*

* **Syntax:**

<element xmlns:name = "URL">

where

* + Namespace starts with the *xmlns.*
  + The word *name* is the namespace prefix.
  + the *URL* is the namespace identifier.
* **Example:**  
  Consider the following xml document named Table.xml :-

filter\_none

brightness\_4

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/css" href="rule.css"?>   <tables>    <table>     <tr>      <td>Apple</td>      <td>Banana</td>     </tr>    </table>    <table>     <height>100</height>     <width>150</width>    </table>   </tables> |

In the above code, there would be a name conflict, both of them contains the same *table* element but the contents of the table element is different .To handle this situation, the concept of XML Namespace is used.

* **Example:**  
  Consider the same XML document to resolve name conflict:

filter\_none

brightness\_4

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/css" href="rule.css"?>   <tables>    <m:table xmlns:m=""[http://www.google.co.in](http://www.google.co.in/)">     <m:tr>      <m:td>Apple</m:td>      <m:td>Banana</m:td>     </m:tr>    </m:table>    <n:table xmlns:m=""[http://www.yahoo.co.in](http://www.yahoo.co.in/)">     <n:height>100</n:height>     <n:width>150</n:width>    </n:table>   </tables> |

**Xpath:**

* Xpath is an important component of XSLT standard.
* Xpath is used to traverse the element and attributes of an XML document.
* Xpath uses different types of expression to retrieve relevant information from the XML document.
* Xpath contains a library of standard functions.  
  Example:
  + bookstore/book[1] => Fetches details of first child of bookstore element.
  + bookstore/book[last()] => Fetches details of last child of bookstore element.

**Templates:**

* An XSL stylesheet contains one or more set of rules that are called templates.
* A template contains rules that are applied when the specific element is matched.
* An XSLT document has the following things:
  + The root element of the stylesheet.
  + A file of extension .xsl .
  + The syntax of XSLT i.e what is allowed and what is not allowed.
  + The standard namespace whose URL is *http://www.w3.org/1999/XSL/Transform*.

**Example:**  
In this example, creating the XML file that contains the information about five students and displaying the XML file using XSLT.

* **XML file:**  
  Creating Students.xml as:

filter\_none

brightness\_4

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl "href="Rule.xsl" ?>   <student>    <s>     <name> Divyank Singh Sikarwar </name>     <branch> CSE</branch>     <age>18</age>     <city> Agra </city>    </s>    <s>     <name> Aniket Chauhan </name>     <branch> CSE</branch>     <age> 20</age>     <city> Shahjahanpur </city>    </s>    <s>     <name> Simran Agarwal</name>     <branch> CSE</branch>     <age> 23</age>     <city> Buland Shar</city>    </s>    <s>     <name> Abhay Chauhan</name>     <branch> CSE</branch>     <age> 17</age>     <city> Shahjahanpur</city>    </s>    <s>     <name> Himanshu Bhatia</name>     <branch> IT</branch>     <age> 25</age>     <city> Indore</city>    </s>   </student> |

In the above example, Students.xml is created and linking it with Rule.xsl which contains the corresponding XSL style sheet rules.

* **XSLT Code:**

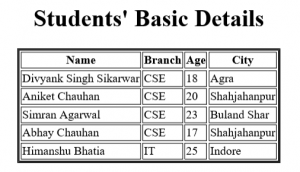
Creating Rule.xsl as:

filter\_none

brightness\_4

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <xsl:stylesheet version="1.0"    xmlns:xsl="<http://www.w3.org/1999/XSL/Transform>">  <xsl:template match="/">   <html>   <body>    <h1 align="center">Students' Basic Details</h1>     <table border="3" align="center" >     <tr>      <th>Name</th>      <th>Branch</th>      <th>Age</th>      <th>City</th>     </tr>      <xsl:for-each select="student/s">     <tr>      <td><xsl:value-of select="name"/></td>      <td><xsl:value-of select="branch"/></td>      <td><xsl:value-of select="age"/></td>      <td><xsl:value-of select="city"/></td>     </tr>      </xsl:for-each>      </table>  </body>  </html>  </xsl:template>  </xsl:stylesheet> |

* **Output :**



XSL

Before learning XSLT, we should first understand XSL which stands for E**X**tensible **S**tylesheet **L**anguage. It is similar to XML as CSS is to HTML.

Need for XSL

In case of HTML document, tags are predefined such as table, div, and span; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

Following are the main parts of XSL −

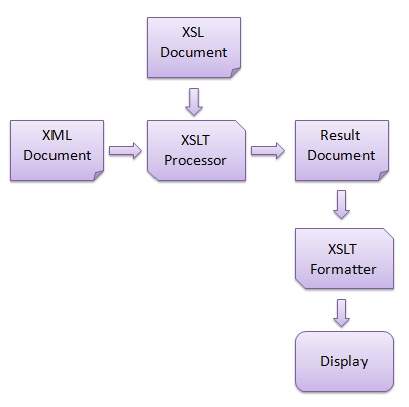
* **XSLT** − used to transform XML document into various other types of document.
* **XPath** − used to navigate XML document.
* **XSL-FO** − used to format XML document.

What is XSLT

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

How XSLT Works

An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.



Advantages

Here are the advantages of using XSLT −

* Independent of programming. Transformations are written in a separate xsl file which is again an XML document.
* Output can be altered by simply modifying the transformations in xsl file. No need to change any code. So Web designers can edit the stylesheet and can see the change in the output quickly.

Let’s suppose we have the following sample XML file, students.xml, which is required to be transformed into a well-formatted HTML document.

**students.xml**

<?xml version = "1.0"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

We need to define an XSLT style sheet document for the above XML document to meet the following criteria −

* Page should have a title **Students**.
* Page should have a table of student details.
* Columns should have following headers: Roll No, First Name, Last Name, Nick Name, Marks
* Table must contain details of the students accordingly.

Step 1: Create XSLT document

Create an XSLT document to meet the above requirements, name it as students.xsl and save it in the same location where students.xml lies.

**students.xsl**

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

<!-- xsl stylesheet declaration with xsl namespace:

Namespace tells the xlst processor about which

element is to be processed and which is used for output purpose only

-->

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<!-- xsl template declaration:

template tells the xlst processor about the section of xml

document which is to be formatted. It takes an XPath expression.

In our case, it is matching document root element and will

tell processor to process the entire document with this template.

-->

<xsl:template match = "/">

<!-- HTML tags

Used for formatting purpose. Processor will skip them and browser

will simply render them.

-->

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<!-- for-each processing instruction

Looks for each element matching the XPath expression

-->

<xsl:for-each select="class/student">

<tr>

<td>

<!-- value-of processing instruction

process the value of the element matching the XPath expression

-->

<xsl:value-of select = "@rollno"/>

</td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Step 2: Link the XSLT Document to the XML Document

Update student.xml document with the following xml-stylesheet tag. Set href value to students.xsl

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

...

</class>

Step 3: View the XML Document in Internet Explorer

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

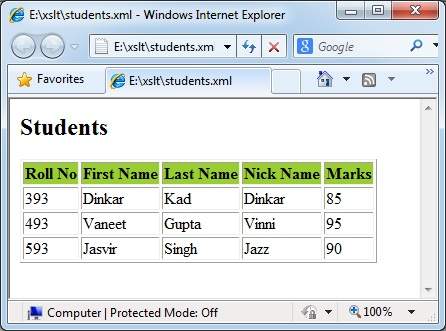
<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

Output



# XSLT <template>

Advertisements

Ad by Valueimpression

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_syntax.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_valueof.htm)

<xsl:template> defines a way to reuse templates in order to generate the desired output for nodes of a particular type/context.

## Declaration

Following is the syntax declaration of **<xsl:template>** element.

<xsl:template

name = Qname

match = Pattern

priority = number

mode = QName >

</xsl:template>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **name**  Name of the element on which template is to be applied. |
| 2 | **match**  Pattern which signifies the element(s) on which template is to be applied. |
| 3 | **priority**  Priority number of a template. Matching template with low priority is not considered in from in front of high priority template. |
| 4 | **mode**  Allows element to be processed multiple times to produce a different result each time. |

## Elements

|  |  |
| --- | --- |
| **Number of occurrences** | **Unlimited** |
| **Parent elements** | xsl:stylesheet, xsl:transform |
| **Child elements** | xsl:apply-imports,xsl:apply-templates,xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:number, xsl:param, xsl:processing-instruction, xsl:text, xsl:value-of, xsl:variable, output elements |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks>.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "class/student">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

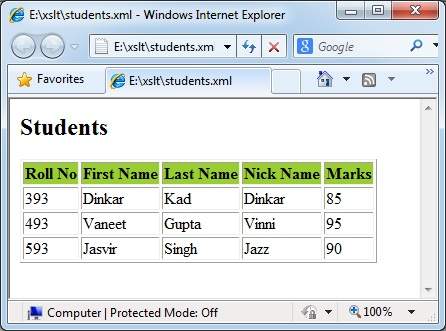
</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Verify the output



[Previous Page](https://www.tutorialspoint.com/xslt/xslt_template.htm) [Print Page](javascript:printPage();)

# XSLT <value-of>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_template.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_foreach.htm)

<xsl:value-of> tag puts the value of the selected node as per XPath expression, as text.

## Declaration

Following is the syntax declaration of **<xsl:value-of>** element.

<xsl:value-of

select = Expression

disable-output-escaping = "yes" | "no" >

</xsl:value-of>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **Select**  XPath Expression to be evaluated in current context. |
| 2 | **disable-outputescaping**  Default-"no". If "yes", output text will not escape xml characters from text. |

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processing instruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements |
| **Child elements** | None |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks>.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "class/student">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Verify the output

# Formatted Output

# XSLT <for-each>

Advertisements

Ad by Valueimpression

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_valueof.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_sort.htm)

<xsl:for-each> tag applies a template repeatedly for each node.

## Declaration

Following is the syntax declaration of **<xsl:for-each>** element

<xsl:for-each

select = Expression >

</xsl:for-each>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **Select**  XPath Expression to be evaluated in current context to determine the set of nodes to be iterated. |

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:foreach, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processinginstruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements. |
| **Child elements** | xsl:apply-imports, xsl:apply-templates, xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:number, xsl:processing-instruction, xsl:sort, xsl:text, xsl:value-of, xsl:variable. |
| Demo Example This example creates a table of <student> element with its attribute rollno and its child <firstname>,<lastname><nickname> and <marks> by iterating over each student.  **students.xml**  <?xml version = "1.0"?>  <?xml-stylesheet type = "text/xsl" href = "students.xsl"?>  <class>  <student rollno = "393">  <firstname>Dinkar</firstname>  <lastname>Kad</lastname>  <nickname>Dinkar</nickname>  <marks>85</marks>  </student>  <student rollno = "493">  <firstname>Vaneet</firstname>  <lastname>Gupta</lastname>  <nickname>Vinni</nickname>  <marks>95</marks>  </student>  <student rollno = "593">  <firstname>Jasvir</firstname>  <lastname>Singh</lastname>  <nickname>Jazz</nickname>  <marks>90</marks>  </student>  </class>  **students.xsl**  <?xml version = "1.0" encoding = "UTF-8"?>  <xsl:stylesheet version = "1.0"  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">  <xsl:template match = "/">  <html>  <body>  <h2>Students</h2>  <table border = "1">  <tr bgcolor = "#9acd32">  <th>Roll No</th>  <th>First Name</th>  <th>Last Name</th>  <th>Nick Name</th>  <th>Marks</th>  </tr>    <xsl:for-each select = "class/student">    <tr>  <td><xsl:value-of select = "@rollno"/></td>  <td><xsl:value-of select = "firstname"/></td>  <td><xsl:value-of select = "lastname"/></td>  <td><xsl:value-of select = "nickname"/></td>  <td><xsl:value-of select = "marks"/></td>  </tr>  </xsl:for-each>    </table>  </body>  </html>  </xsl:template>  </xsl:stylesheet> Output Formatted Output |  |
|  |  |

# XSLT <sort>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_foreach.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_if.htm)

<xsl:sort> tag specifies a sort criteria on the nodes.

## Declaration

Following is the syntax declaration of **<xsl:sort>** element.

<xsl:sort

select = string-expression

lang = { nmtoken }

data-type = { "text" | "number" | QName }

order = { "ascending" | "descending" }

case-order = { "upper-first" | "lower-first" } >

</xsl:sort>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **select**  Sorting key of the node. |
| 2 | **lang**  Language alphabet used to determine sort order. |
| 3 | **data-type**  Data type of the text. |
| 4 | **order**  Sorting order. Default is "ascending". |
| 5 | **case-order**  Sorting order of string by capitalization. Default is "upper-first". |

## Elements

|  |  |
| --- | --- |
| **Number of occurrences** | Unlimited |
| **Parent elements** | xsl:apply-templates, xsl:for-each |
| **Child elements** | None |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student sort them by first name.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "class/student">

<xsl:sort select = "firstname"/>

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Output

# Formatted Sort Output

# XSLT <if>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_sort.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_choose.htm)

<xsl:if> tag specifies a conditional test against the content of nodes.

## Declaration

Following is the syntax declaration of **<xsl:if>** element.

<xsl:if

test = boolean-expression >

</xsl:if>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **test**  The condition in the xml data to test. |

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processing-instruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements |
| **Child elements** | xsl:apply-templates, xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:for-each, xsl:if, xsl:processing-instruction, xsl:text, xsl:value-of, xsl:variable, output elements |

# XSLT <choose>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_if.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_key.htm)

<xsl:choose> tag specifies a multiple conditional tests against the content of nodes in conjunction with the <xsl:otherwise> and <xsl:when> elements.

## Declaration

Following is the syntax declaration of **<xsl:choose>** element.

<xsl:choose >

</xsl:choose>

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:for-each, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processing-instruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements |
| **Child elements** | xsl:otherwise, xsl:when |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student. It checks and then prints the grade details.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

<th>Grade</th>

</tr>

<xsl:for-each select = "class/student">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

<td>

<xsl:choose>

<xsl:when test = "marks > 90">

High

</xsl:when>

<xsl:when test = "marks > 85">

Medium

</xsl:when>

<xsl:otherwise>

Low

</xsl:otherwise>

</xsl:choose>

</td>

</tr>

</xsl:for-each>

</table>

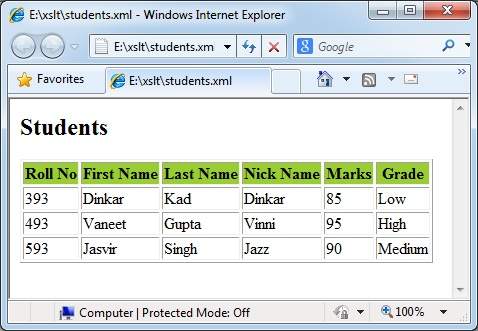
</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Output



# XSLT <key>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_choose.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_message.htm)

<xsl:key> tag element specifies a named name-value pair assigned to a specific element in an XML document. This key is used with the key() function in XPath expressions to access the assigned elements in an XML document.

## Declaration

Following is the syntax declaration of **<xsl:key>** element.

<xsl:key

name = QName

match = Pattern

use = Expression >

</xsl:key>

## Attributes

|  |  |
| --- | --- |
| **S.No** | **Name & Description** |
| 1 | **Name**  Name of the key to be used. |
| 2 | **Match**  Patterns used to identify a node that holds this key. |
| 3 | **Use**  XPath expression to identify the value of the nodes of xml document. |

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | Unlimited |
| **Parent elements** | xsl:stylesheet |
| **Child elements** | None |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student. It checks key as firstname to be one of the student's name and then prints the student details.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:key name = "firstname-search" match = "student" use = "firstname"/>

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "key('firstname-search', 'Dinkar')">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

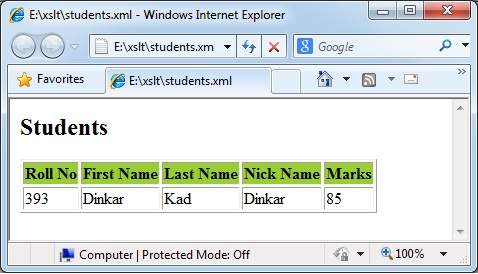
</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Output



# XSLT <message>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_key.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_apply_template.htm)

<message> tag element helps to debug an XSLT processing. It is similar to javascript alerts. <xsl:> tag buffers a message to XSLT processor which terminates the processing and sends a message to the caller application to display the error message.

## Declaration

Following is the syntax declaration of **<xsl:message>** element.

<xsl:message

terminate = "yes" | "no" >

</xsl:message>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **terminate**  It specifies whether the transformation should terminate upon executing this instruction or not. Default is "yes". |

## Elements

|  |  |
| --- | --- |
| **Number of Occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:foreach, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processinginstruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements |
| **Child elements** | xsl:apply-templates, xsl:attribute, xsl:call-template, xsl:choose, xsl:comment, xsl:copy, xsl:copy-of, xsl:element, xsl:for-each, xsl:if, xsl:processing-instruction, xsl:text, xsl:value-of, xsl:variable, output elements |

## Demo Example

This example creates a table of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student. It checks key as firstname to be present and then prints the student details, otherwise displays an error message.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname></firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "class/student">

<xsl:if test = "firstname = ''">

<xsl:message terminate = "yes">A first name field is empty.

</xsl:message>

</xsl:if>

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

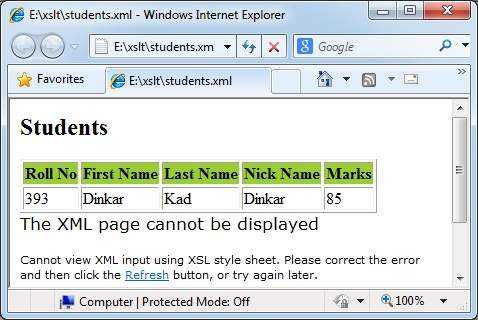
</body>

</html>

</xsl:template>

</xsl:stylesheet>

## Output



# XSLT <apply-template>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_message.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_import_stylesheet.htm)

<xsl:apply-template> tag signals the XSLT processor to find the appropriate template to apply, based on the type and context of each selected node.

## Declaration

Following is the syntax declaration of **<xsl:apply-template>** element.

<xsl:apply-template

select = Expression

mode = QName >

</xsl:apply-template>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1 | **select**  Used to process nodes selected by an XPath expression, instead of processing all the children. |
| 2 | **mode**  Allows an element as specified by its Qualified Names to be processed multiple times, each time producing a different result. |

## Elements

|  |  |
| --- | --- |
| **Number of occurrences** | **Unlimited** |
| **Parent elements** | xsl:attribute, xsl:comment, xsl:copy, xsl:element, xsl:fallback, xsl:foreach, xsl:if, xsl:message, xsl:otherwise, xsl:param, xsl:processinginstruction, xsl:template, xsl:variable, xsl:when, xsl:with-param, output elements |
| **Child elements** | xsl:sort, xsl:with-param |

## Demo Example

This example creates a list of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<xsl:apply-templates select = "class/student" />

</body>

</html>

</xsl:template>

<xsl:template match = "class/student">

<xsl:apply-templates select = "@rollno" />

<xsl:apply-templates select = "firstname" />

<xsl:apply-templates select = "lastname" />

<xsl:apply-templates select = "nickname" />

<xsl:apply-templates select = "marks" />

<br />

</xsl:template>

<xsl:template match = "@rollno">

<span style = "font-size = 22px;">

<xsl:value-of select = "." />

</span>

<br />

</xsl:template>

<xsl:template match = "firstname">

First Name:<span style = "color:blue;">

<xsl:value-of select = "." />

</span>

<br />

</xsl:template>

<xsl:template match = "lastname">

Last Name:<span style = "color:green;">

<xsl:value-of select = "." />

</span>

<br />

</xsl:template>

<xsl:template match = "nickname">

Nick Name:<span style = "color:red;">

<xsl:value-of select = "." />

</span>

<br />

</xsl:template>

<xsl:template match = "marks">

Marks:<span style = "color:gray;">

<xsl:value-of select = "." />

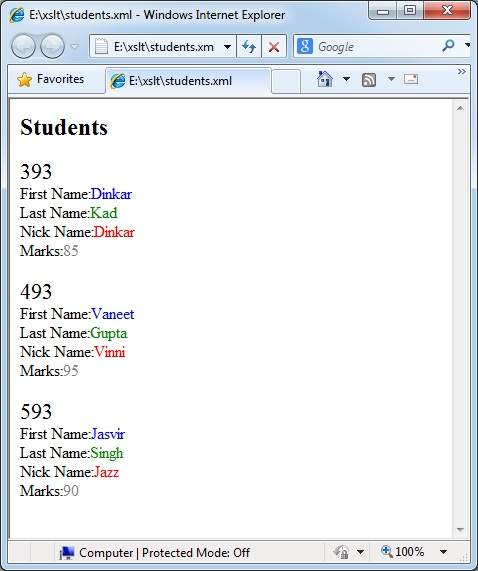
</span>

<br />

</xsl:template>

</xsl:stylesheet>

## Output



# XSLT <import>

Advertisements

[Previous Page](https://www.tutorialspoint.com/xslt/xslt_apply_template.htm)

[Next Page](https://www.tutorialspoint.com/xslt/xslt_quick_guide.htm)

<xsl:import> tag imports the contents of one stylesheet into another. Importing a style sheet has higher precedence over imported stylesheet.

## Declaration

Following is the syntax declaration of **<xsl:import>** element.

<xsl:import href = "uri">

</xsl:import>

## Attributes

|  |  |
| --- | --- |
| **Sr.No** | **Name & Description** |
| 1. | **href**  used to pass the path of xslt stylesheet to be imported  . |

## Elements

|  |  |
| --- | --- |
| **Number of occurrences** | Unlimited |
| **Parent elements** | xsl:stylesheet, xsl:transform |
| **Child elements** | none |

## Demo Example

This example creates a list of <student> element with its attribute **rollno** and its child <firstname>, <lastname>, <nickname>, and <marks> by iterating over each student. Here we have created two xsl stylesheets where students\_imports.xsl stylesheet imports students.xsl and students.xml is linked to students\_imports.xsl.

**students.xml**

<?xml version = "1.0"?>

<?xml-stylesheet type = "text/xsl" href = "students\_imports.xsl"?>

<class>

<student rollno = "393">

<firstname>Dinkar</firstname>

<lastname>Kad</lastname>

<nickname>Dinkar</nickname>

<marks>85</marks>

</student>

<student rollno = "493">

<firstname>Vaneet</firstname>

<lastname>Gupta</lastname>

<nickname>Vinni</nickname>

<marks>95</marks>

</student>

<student rollno = "593">

<firstname>Jasvir</firstname>

<lastname>Singh</lastname>

<nickname>Jazz</nickname>

<marks>90</marks>

</student>

</class>

**students.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:template match = "/">

<html>

<body>

<h2>Students</h2>

<table border = "1">

<tr bgcolor = "#9acd32">

<th>Roll No</th>

<th>First Name</th>

<th>Last Name</th>

<th>Nick Name</th>

<th>Marks</th>

</tr>

<xsl:for-each select = "class/student">

<tr>

<td><xsl:value-of select = "@rollno"/></td>

<td><xsl:value-of select = "firstname"/></td>

<td><xsl:value-of select = "lastname"/></td>

<td><xsl:value-of select = "nickname"/></td>

<td><xsl:value-of select = "marks"/></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

**students\_imports.xsl**

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

<xsl:stylesheet version = "1.0"

xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

<xsl:import href = "students.xsl"/>

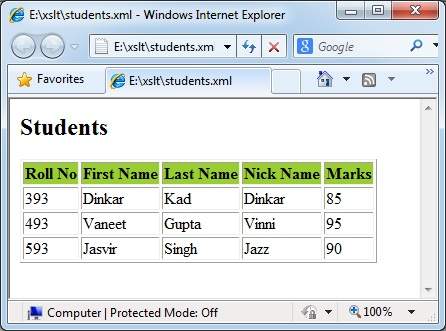
<xsl:template match = "/">

<xsl:apply-imports/>

</xsl:template>

</xsl:stylesheet>

## Output



# XML - Processors

Advertisements

[Previous Page](https://www.tutorialspoint.com/xml/xml_parsers.htm)

[Next Page](https://www.tutorialspoint.com/xml/xml_quick_guide.htm)

When a software program reads an XML document and takes actions accordingly, this is called *processing* the XML. Any program that can read and process XML documents is known as an *XML processor*. An XML processor reads the XML file and turns it into in-memory structures that the rest of the program can access.

The most fundamental XML processor reads an XML document and converts it into an internal representation for other programs or subroutines to use. This is called a *parser*, and it is an important component of every XML processing program.

Processor involves processing the instructions, that can be studied in the chapter [Processing Instruction](https://www.tutorialspoint.com/xml/xml_processing.htm).

## Types

XML processors are classified as **validating** or **non-validating** types, depending on whether or not they check XML documents for validity. A processor that discovers a validity error must be able to report it, but may continue with normal processing.

**A few validating parsers are** − xml4c (IBM, in C++), xml4j (IBM, in Java), MSXML (Microsoft, in Java), TclXML (TCL), xmlproc (Python), XML::Parser (Perl), Java Project X (Sun, in Java).

**A few non-validating parsers are** − OpenXML (Java), Lark (Java), xp (Java), AElfred (Java), expat (C), XParse (JavaScript), xmllib (Python).

# XML Parser

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

All major browsers have a built-in XML parser to access and manipulate XML.

## XML Parser

The [XML DOM (Document Object Model)](https://www.w3schools.com/xml/dom_intro.asp) defines the properties and methods for accessing and editing XML.

However, before an XML document can be accessed, it must be loaded into an XML DOM object.

All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

## Parsing a Text String

This example parses a text string into an XML DOM object, and extracts the info from it with JavaScript:

### Example

<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
var text, parser, xmlDoc;  
  
text = "<bookstore><book>" +  
"<title>Everyday Italian</title>" +  
"<author>Giada De Laurentiis</author>" +  
"<year>2005</year>" +  
"</book></bookstore>";  
  
parser = new DOMParser();  
xmlDoc = parser.parseFromString(text,"text/xml");  
  
document.getElementById("demo").innerHTML =  
xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/xml/tryit.asp?filename=try_dom_loadxmltext)

## Example Explained

A text string is defined:

text = "<bookstore><book>" +  
"<title>Everyday Italian</title>" +  
"<author>Giada De Laurentiis</author>" +  
"<year>2005</year>" +  
"</book></bookstore>";

An XML DOM parser is created:

parser = new DOMParser();

The parser creates a new XML DOM object using the text string:

xmlDoc = parser.parseFromString(text,"text/xml");

## Old Versions of Internet Explorer

Old versions of Internet Explorer (IE5, IE6, IE7, IE8) do not support the DOMParser object.

To handle older versions of Internet Explorer, check if the browser supports the DOMParser object, or else create an ActiveXObject:

### Example

if (window.DOMParser) {  
  // code for modern browsers  
  parser = new DOMParser();  
  xmlDoc = parser.parseFromString(text,"text/xml");  
} else {  
  // code for old IE browsers  
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");  
  xmlDoc.async = false;  
  xmlDoc.loadXML(text);  
}

[Try it Yourself »](https://www.w3schools.com/xml/tryit.asp?filename=try_dom_loadxmltext_ie)

## The XMLHttpRequest Object

The [XMLHttpRequest Object](https://www.w3schools.com/xml/xml_http.asp) has a built in XML Parser.

The **responseText**property returns the response as a string.

The **responseXML** property returns the response as an XML DOM object.

If you want to use the response as an XML DOM object, you can use the responseXML property.

### Example

Request the file [cd\_catalog.xml](https://www.w3schools.com/xml/cd_catalog.xml) and use the response as an XML DOM object:

xmlDoc = xmlhttp.responseXML;  
txt = "";  
x = xmlDoc.getElementsByTagName("ARTIST");  
for (i = 0; i < x.length; i++) {  
    txt += x[i].childNodes[0].nodeValue + "<br>";  
}  
document.getElementById("demo").innerHTML = txt;

# XML DOM

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

### XML DOM



## What is the DOM?

The DOM defines a standard for accessing and manipulating documents:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.

The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.

Understanding the DOM is a must for anyone working with HTML or XML.

## The HTML DOM

All HTML elements can be accessed through the HTML DOM.

This example changes the value of an HTML element with id="demo":

### Example

<h1 id="demo">This is a Heading</h1>  
  
<button type="button"  
onclick="document.getElementById('demo').innerHTML = 'Hello World!'">Click Me!  
</button>

[Try it Yourself »](https://www.w3schools.com/xml/tryit.asp?filename=tryxml_change_html)

You can learn a lot more about the HTML DOM in our [JavaScript tutorial](https://www.w3schools.com/js/js_htmldom.asp).

## The XML DOM

All XML elements can be accessed through the XML DOM.

### Books.xml

<?xml version="1.0" encoding="UTF-8"?>  
<bookstore>  
  
  <book category="cooking">  
    <title lang="en">Everyday Italian</title>  
    <author>Giada De Laurentiis</author>  
    <year>2005</year>  
    <price>30.00</price>  
  </book>  
  
  <book category="children">  
    <title lang="en">Harry Potter</title>  
    <author>J K. Rowling</author>  
    <year>2005</year>  
    <price>29.99</price>  
  </book>  
  
</bookstore>

This code retrieves the text value of the first <title> element in an XML document:

### Example

txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;

The XML DOM is a standard for how to get, change, add, and delete XML elements.

This example loads a text string into an XML DOM object, and extracts the info from it with JavaScript:

### Example

<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
var text, parser, xmlDoc;  
  
text = "<bookstore><book>" +  
"<title>Everyday Italian</title>" +  
"<author>Giada De Laurentiis</author>" +  
"<year>2005</year>" +  
"</book></bookstore>";  
  
parser = new DOMParser();  
xmlDoc = parser.parseFromString(text,"text/xml");  
  
document.getElementById("demo").innerHTML =  
**xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue**;  
</script>  
  
</body>  
</html>

[Try it Yourself »](https://www.w3schools.com/xml/tryit.asp?filename=try_dom_loadxmltext)

# XML and XPath

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

## What is XPath?

XPath is a major element in the XSLT standard.

XPath can be used to navigate through elements and attributes in an XML document.

|  |  |
| --- | --- |
| XPath | * XPath is a syntax for defining parts of an XML document * XPath uses path expressions to navigate in XML documents * XPath contains a library of standard functions * XPath is a major element in XSLT and in XQuery * XPath is a W3C recommendation |

## XPath Path Expressions

XPath uses path expressions to select nodes or node-sets in an XML document. These path expressions look very much like the expressions you see when you work with a traditional computer file system.

XPath expressions can be used in JavaScript, Java, XML Schema, PHP, Python, C and C++, and lots of other languages.

## XPath is Used in XSLT

XPath is a major element in the XSLT standard.

With XPath knowledge you will be able to take great advantage of XSL.

## XPath Example

We will use the following XML document:

<?xml version="1.0" encoding="UTF-8"?>  
  
<bookstore>  
  
<book category="cooking">  
  <title lang="en">Everyday Italian</title>  
  <author>Giada De Laurentiis</author>  
  <year>2005</year>  
  <price>30.00</price>  
</book>  
  
<book category="children">  
  <title lang="en">Harry Potter</title>  
  <author>J K. Rowling</author>  
  <year>2005</year>  
  <price>29.99</price>  
</book>  
  
<book category="web">  
  <title lang="en">XQuery Kick Start</title>  
  <author>James McGovern</author>  
  <author>Per Bothner</author>  
  <author>Kurt Cagle</author>  
  <author>James Linn</author>  
  <author>Vaidyanathan Nagarajan</author>  
  <year>2003</year>  
  <price>49.99</price>  
</book>  
  
<book category="web">  
  <title lang="en">Learning XML</title>  
  <author>Erik T. Ray</author>  
  <year>2003</year>  
  <price>39.95</price>  
</book>  
  
</bookstore>

In the table below we have listed some XPath expressions and the result of the expressions:

|  |  |
| --- | --- |
| **XPath Expression** | **Result** |
| /bookstore/book[1] | Selects the first book element that is the child of the bookstore element |
| /bookstore/book[last()] | Selects the last book element that is the child of the bookstore element |
| /bookstore/book[last()-1] | Selects the last but one book element that is the child of the bookstore element |
| /bookstore/book[position()<3] | Selects the first two book elements that are children of the bookstore element |
| //title[@lang] | Selects all the title elements that have an attribute named lang |
| //title[@lang='en'] | Selects all the title elements that have a "lang" attribute with a value of "en" |
| /bookstore/book[price>35.00] | Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00 |
| /bookstore/book[price>35.00]/title | Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00 |

## XPath Tutorial

You will learn a lot more about XPath in our [XPath Tutorial](https://www.w3schools.com/xml/xpath_intro.asp).

# XML Namespaces

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

XML Namespaces provide a method to avoid element name conflicts.

## Name Conflicts

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.

This XML carries HTML table information:

<table>  
  <tr>  
    <td>Apples</td>  
    <td>Bananas</td>  
  </tr>  
</table>

This XML carries information about a table (a piece of furniture):

<table>  
  <name>African Coffee Table</name>  
  <width>80</width>  
  <length>120</length>  
</table>

If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.

A user or an XML application will not know how to handle these differences.

## Solving the Name Conflict Using a Prefix

Name conflicts in XML can easily be avoided using a name prefix.

This XML carries information about an HTML table, and a piece of furniture:

<h:table>  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>  
  
<f:table>  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>

In the example above, there will be no conflict because the two <table> elements have different names.

## XML Namespaces - The xmlns Attribute

When using prefixes in XML, a **namespace** for the prefix must be defined.

The namespace can be defined by an **xmlns** attribute in the start tag of an element.

The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

<root>  
  
<h:table xmlns:h="http://www.w3.org/TR/html4/">  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>  
  
<f:table xmlns:f="https://www.w3schools.com/furniture">  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>  
  
</root>

In the example above:

The xmlns attribute in the first <table> element gives the h: prefix a qualified namespace.

The xmlns attribute in the second <table> element gives the f: prefix a qualified namespace.

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Namespaces can also be declared in the XML root element:

<root xmlns:h="http://www.w3.org/TR/html4/"  
xmlns:f="https://www.w3schools.com/furniture">  
  
<h:table>  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>  
  
<f:table>  
  <f:name>African Coffee Table</f:name>  
  <f:width>80</f:width>  
  <f:length>120</f:length>  
</f:table>  
  
</root>

**Note:** The namespace URI is not used by the parser to look up information.

The purpose of using an URI is to give the namespace a unique name.

However, companies often use the namespace as a pointer to a web page containing namespace information.

## Uniform Resource Identifier (URI)

A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.

The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Uniform Resource Name** (URN).

## Default Namespaces

Defining a default namespace for an element saves us from using prefixes in all the child elements. It has the following syntax:

xmlns="*namespaceURI*"

This XML carries HTML table information:

<table xmlns="http://www.w3.org/TR/html4/">  
  <tr>  
    <td>Apples</td>  
    <td>Bananas</td>  
  </tr>  
</table>

This XML carries information about a piece of furniture:

<table xmlns="https://www.w3schools.com/furniture">  
  <name>African Coffee Table</name>  
  <width>80</width>  
  <length>120</length>  
</table>

## Namespaces in Real Use

XSLT is a language that can be used to transform XML documents into other formats.

The XML document below, is a document used to transform XML into HTML.

The namespace "http://www.w3.org/1999/XSL/Transform" identifies XSLT elements inside an HTML document:

<?xml version="1.0" encoding="UTF-8**"**?>  
  
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
  
<xsl:template match="/">  
<html>  
<body>  
  <h2>My CD Collection</h2>  
  <table border="1">  
    <tr>  
      <th style="text-align:left">Title</th>  
      <th style="text-align:left">Artist</th>  
    </tr>  
    <xsl:for-each select="catalog/cd">  
    <tr>  
      <td><xsl:value-of select="title"/></td>  
      <td><xsl:value-of select="artist"/></td>  
    </tr>  
    </xsl:for-each>  
  </table>  
</body>  
</html>  
</xsl:template>  
  
</xsl:stylesheet>

If you want to learn more about XSLT, please read our [XSLT Tutorial](https://www.w3schools.com/xml/xsl_intro.asp).

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

# XML Web Services

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

Web services are web application components.

Web services can be published, found, and used on the Web.

This tutorial introduces WSDL, SOAP, RDF, and RSS.

## WSDL

* WSDL stands for Web Services Description Language
* WSDL is an XML-based language for describing Web services.
* WSDL is a W3C recommendation

## SOAP

* SOAP stands for Simple Object Access Protocol
* SOAP is an XML based protocol for accessing Web Services.
* SOAP is based on XML
* SOAP is a W3C recommendation

## RDF

* RDF stands for Resource Description Framework
* RDF is a framework for describing resources on the web
* RDF is written in XML
* RDF is a W3C Recommendation

## RSS

* RSS stands for Really Simple Syndication
* RSS allows you to syndicate your site content
* RSS defines an easy way to share and view headlines and content
* RSS files can be automatically updated
* RSS allows personalized views for different sites
* RSS is written in XML

## What You Should Already Know

Before you study web services you should have a basic understanding of XML and XML Namespaces.

If you want to study these subjects first, please read our [XML Tutorial](https://www.w3schools.com/xml/default.asp).

# XML WSDL

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

* WSDL stands for Web Services Description Language
* WSDL is used to describe web services
* WSDL is written in XML
* WSDL is a W3C recommendation from 26. June 2007

## WSDL Documents

An WSDL document describes a web service. It specifies the location of the service, and the methods of the service, using these major elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| <types> | Defines the (XML Schema) data types used by the web service |
| <message> | Defines the data elements for each operation |
| <portType> | Describes the operations that can be performed and the messages involved. |
| <binding> | Defines the protocol and data format for each port type |

The main structure of a WSDL document looks like this:

<definitions>  
  
<types>  
  data type definitions........  
</types>  
  
<message>  
  definition of the data being communicated....  
</message>  
  
<portType>  
  set of operations......  
</portType>  
  
<binding>  
  protocol and data format specification....  
</binding>  
  
</definitions>

## WSDL Example

This is a simplified fraction of a WSDL document:

<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>  
  
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>  
  
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>

In this example the**<portType>** element defines "glossaryTerms" as the name of a **port**, and "getTerm" as the name of an **operation**.

The "getTerm" operation has an **input message** called "getTermRequest" and an **output message** called "getTermResponse".

The**<message>** elements define the **parts** of each message and the associated data types.

## The <portType> Element

The <portType> element defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.

The request-response type is the most common operation type, but WSDL defines four types:

|  |  |
| --- | --- |
| **Type** | **Definition** |
| One-way | The operation can receive a message but will not return a response |
| Request-response | The operation can receive a request and will return a response |
| Solicit-response | The operation can send a request and will wait for a response |
| Notification | The operation can send a message but will not wait for a response |

## WSDL One-Way Operation

A one-way operation example:

<message name="newTermValues">  
  <part name="term" type="xs:string"/>  
  <part name="value" type="xs:string"/>  
</message>  
  
<portType name="glossaryTerms">  
  <operation name="setTerm">  
    <input name="newTerm" message="newTermValues"/>  
  </operation>  
</portType >

In the example above, the portType "glossaryTerms" defines a one-way operation called "setTerm".

The "setTerm" operation allows input of new glossary terms messages using a "newTermValues" message with the input parameters "term" and "value". However, no output is defined for the operation.

## WSDL Request-Response Operation

A request-response operation example:

<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>  
  
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>  
  
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>

In the example above, the portType "glossaryTerms" defines a request-response operation called "getTerm".

The "getTerm" operation requires an input message called "getTermRequest" with a parameter called "term", and will return an output message called "getTermResponse" with a parameter called "value".

## WSDL Binding to SOAP

WSDL bindings defines the message format and protocol details for a web service.

A request-response operation example:

<message name="getTermRequest">  
  <part name="term" type="xs:string"/>  
</message>  
  
<message name="getTermResponse">  
  <part name="value" type="xs:string"/>  
</message>  
  
<portType name="glossaryTerms">  
  <operation name="getTerm">  
    <input message="getTermRequest"/>  
    <output message="getTermResponse"/>  
  </operation>  
</portType>  
  
<binding type="glossaryTerms" name="b1">  
   <soap:binding style="document"  
   transport="http://schemas.xmlsoap.org/soap/http" />  
   <operation>  
     <soap:operation soapAction="http://example.com/getTerm"/>  
     <input><soap:body use="literal"/></input>  
     <output><soap:body use="literal"/></output>  
  </operation>  
</binding>

The**binding** element has two attributes - name and type.

The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "glossaryTerms" port.

The **soap:binding** element has two attributes - style and transport.

The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use. In this case we use HTTP.

The**operation** element defines each operation that the portType exposes.

For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded. In this case we use "literal".

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

# XML Soap

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

* SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
* SOAP is an application communication protocol
* SOAP is a format for sending and receiving messages
* SOAP is platform independent
* SOAP is based on XML
* SOAP is a W3C recommendation

## Why SOAP?

It is important for web applications to be able to communicate over the Internet.

The best way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

## SOAP Building Blocks

A SOAP message is an ordinary XML document containing the following elements:

* An Envelope element that identifies the XML document as a SOAP message
* A Header element that contains header information
* A Body element that contains call and response information
* A Fault element containing errors and status information

All the elements above are declared in the default namespace for the SOAP envelope:

<http://www.w3.org/2003/05/soap-envelope/>

and the default namespace for SOAP encoding and data types is:

<http://www.w3.org/2003/05/soap-encoding>

## Syntax Rules

Here are some important syntax rules:

* A SOAP message MUST be encoded using XML
* A SOAP message MUST use the SOAP Envelope namespace
* A SOAP message must NOT contain a DTD reference
* A SOAP message must NOT contain XML Processing Instructions

## Skeleton SOAP Message

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Header>  
...  
</soap:Header>  
  
<soap:Body>  
...  
  <soap:Fault>  
  ...  
  </soap:Fault>  
</soap:Body>  
  
</soap:Envelope>

## The SOAP Envelope Element

The required SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message.

### Example

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  ...  
  Message information goes here  
  ...  
</soap:Envelope>

## The xmlns:soap Namespace

Notice the xmlns:soap namespace in the example above. It should always have the value of: "http://www.w3.org/2003/05/soap-envelope/".

The namespace defines the Envelope as a SOAP Envelope.

If a different namespace is used, the application generates an error and discards the message.

## The encodingStyle Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.

A SOAP message has no default encoding.

#### Syntax

soap:encodingStyle="*URI*"

#### Example

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  ...  
  Message information goes here  
  ...  
</soap:Envelope>

## The SOAP Header Element

The optional SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message.

If the Header element is present, it must be the first child element of the Envelope element.

**Note:** All immediate child elements of the Header element must be namespace-qualified.

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Header>  
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"  
  soap:mustUnderstand="1">234  
  </m:Trans>  
</soap:Header>  
...  
...  
</soap:Envelope>

The example above contains a header with a "Trans" element, a "mustUnderstand" attribute with a value of 1, and a value of 234.

SOAP defines three attributes in the default namespace. These attributes are: mustUnderstand, actor, and encodingStyle.

The attributes defined in the SOAP Header defines how a recipient should process the SOAP message.

## The mustUnderstand Attribute

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.

If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

### Syntax

soap:mustUnderstand="0|1"

### Example

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Header>  
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"  
  soap:mustUnderstand="1">234  
  </m:Trans>  
</soap:Header>  
...  
...  
</soap:Envelope>

## The actor Attribute

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path.

The SOAP actor attribute is used to address the Header element to a specific endpoint.

### Syntax

soap:actor="*URI*"

### Example

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Header>  
  <m:Trans xmlns:m="https://www.w3schools.com/transaction/"  
  soap:actor="https://www.w3schools.com/code/">234  
  </m:Trans>  
</soap:Header>  
...  
...  
</soap:Envelope>

## The encodingStyle Attribute

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.

A SOAP message has no default encoding.

### Syntax

soap:encodingStyle="*URI*"

## The SOAP Body Element

The required SOAP Body element contains the actual SOAP message intended for the ultimate endpoint of the message.

Immediate child elements of the SOAP Body element may be namespace-qualified.

### Example

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Body>  
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">  
    <m:Item>Apples</m:Item>  
  </m:GetPrice>  
</soap:Body>  
  
</soap:Envelope>

The example above requests the price of apples. Note that the m:GetPrice and the Item elements above are application-specific elements. They are not a part of the SOAP namespace.

A SOAP response could look something like this:

<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Body>  
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">  
    <m:Price>1.90</m:Price>  
  </m:GetPriceResponse>  
</soap:Body>  
  
</soap:Envelope>

## The SOAP Fault Element

The optional SOAP Fault element is used to indicate error messages.

The SOAP Fault element holds errors and status information for a SOAP message.

If a Fault element is present, it must appear as a child element of the Body element. A Fault element can only appear once in a SOAP message.

The SOAP Fault element has the following sub elements:

|  |  |
| --- | --- |
| **Sub Element** | **Description** |
| <faultcode> | A code for identifying the fault |
| <faultstring> | A human readable explanation of the fault |
| <faultactor> | Information about who caused the fault to happen |
| <detail> | Holds application specific error information related to the Body element |

### SOAP Fault Codes

The faultcode values defined below must be used in the faultcode element when describing faults:

|  |  |
| --- | --- |
| **Error** | **Description** |
| VersionMismatch | Found an invalid namespace for the SOAP Envelope element |
| MustUnderstand | An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood |
| Client | The message was incorrectly formed or contained incorrect information |
| Server | There was a problem with the server so the message could not proceed |

## The HTTP Protocol

HTTP communicates over TCP/IP. An HTTP client connects to an HTTP server using TCP. After establishing a connection, the client can send an HTTP request message to the server:

POST /item HTTP/1.1  
Host: 189.123.255.239  
Content-Type: text/plain  
Content-Length: 200

The server then processes the request and sends an HTTP response back to the client. The response contains a status code that indicates the status of the request:

200 OK  
Content-Type: text/plain  
Content-Length: 200

In the example above, the server returned a status code of 200. This is the standard success code for HTTP.

If the server could not decode the request, it could have returned something like this:

400 Bad Request  
Content-Length: 0

## SOAP Binding

The SOAP specification defines the structure of the SOAP messages, not how they are exchanged. This gap is filled by what is called "SOAP Bindings". SOAP bindings are mechanisms which allow SOAP messages to be effectively exchanged using a transport protocol.

Most SOAP implementations provide bindings for common transport protocols, such as HTTP or SMTP.

HTTP is synchronous and widely used. A SOAP HTTP request specifies at least two HTTP headers: Content-Type and Content-Length.

SMTP is asynchronous and is used in last resort or particular cases.

Java implementations of SOAP usually provide a specific binding for the JMS (Java Messaging System) protocol.

## Content-Type

The Content-Type header for a SOAP request and response defines the MIME type for the message and the character encoding (optional) used for the XML body of the request or response.

#### Syntax

Content-Type: MIMEType; charset=character-encoding

#### Example

POST /item HTTP/1.1  
Content-Type: application/soap+xml; charset=utf-8

## Content-Length

The Content-Length header for a SOAP request and response specifies the number of bytes in the body of the request or response.

#### Syntax

Content-Length: bytes

#### Example

POST /item HTTP/1.1  
Content-Type: application/soap+xml; charset=utf-8  
Content-Length: 250

## A SOAP Example

In the example below, a GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response. The namespace for the function is defined in "http://www.example.org/stock".

### A SOAP request:

POST /InStock HTTP/1.1  
Host: www.example.org  
Content-Type: application/soap+xml; charset=utf-8  
Content-Length: nnn  
  
<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Body xmlns:m="http://www.example.org/stock">  
  <m:GetStockPrice>  
    <m:StockName>IBM</m:StockName>  
  </m:GetStockPrice>  
</soap:Body>  
  
</soap:Envelope>

### The SOAP response:

HTTP/1.1 200 OK  
Content-Type: application/soap+xml; charset=utf-8  
Content-Length: nnn  
  
<?xml version="1.0"?>  
  
<soap:Envelope  
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"  
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">  
  
<soap:Body xmlns:m="http://www.example.org/stock">  
  <m:GetStockPriceResponse>  
    <m:Price>34.5</m:Price>  
  </m:GetStockPriceResponse>  
</soap:Body>  
  
</soap:Envelope>

# XML RDF

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

### RDF Document Example

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:si="https://www.w3schools.com/rdf/">  
  
<rdf:Description rdf:about="https://www.w3schools.com">  
  <si:title>W3Schools</si:title>  
  <si:author>Jan Egil Refsnes</si:author>  
</rdf:Description>  
  
</rdf:RDF>

## What is RDF?

* RDF stands for **R**esource **D**escription **F**ramework
* RDF is a framework for describing resources on the web
* RDF is designed to be read and understood by computers
* RDF is not designed for being displayed to people
* RDF is written in XML
* RDF is a part of the W3C's Semantic Web Activity
* RDF is a W3C Recommendation from 10. February 2004

## RDF - Examples of Use

* Describing properties for shopping items, such as price and availability
* Describing time schedules for web events
* Describing information about web pages (content, author, created and modified date)
* Describing content and rating for web pictures
* Describing content for search engines
* Describing electronic libraries

## RDF is Designed to be Read by Computers

RDF was designed to provide a common way to describe information so it can be read and understood by computer applications.

RDF descriptions are not designed to be displayed on the web.

## RDF is Written in XML

RDF documents are written in XML. The XML language used by RDF is called RDF/XML.

By using XML, RDF information can easily be exchanged between different types of computers using different types of operating systems and application languages.

## RDF and "The Semantic Web"

The RDF language is a part of the W3C's Semantic Web Activity. W3C's "Semantic Web Vision" is a future where:

* Web information has exact meaning
* Web information can be understood and processed by computers
* Computers can integrate information from the web

RDF uses Web identifiers (URIs) to identify resources.

RDF describes resources with properties and property values.

## RDF Resource, Property, and Property Value

RDF identifies things using Web identifiers (URIs), and describes resources with properties and property values.

Explanation of Resource, Property, and Property value:

* A **Resource** is anything that can have a URI, such as "https://www.w3schools.com/rdf"
* A **Property** is a Resource that has a name, such as "author" or "homepage"
* A **Property value** is the value of a Property, such as "Jan Egil Refsnes" or "https://www.w3schools.com" (note that a property value can be another resource)

The following RDF document could describe the resource "https://www.w3schools.com/rdf":

<?xml version="1.0"?>  
  
<RDF>  
  <Description about="https://www.w3schools.com/rdf">  
    <author>Jan Egil Refsnes</author>  
    <homepage>https://www.w3schools.com</homepage>  
  </Description>  
</RDF>

The example above is simplified. Namespaces are omitted.

## RDF Statements

The combination of a Resource, a Property, and a Property value forms a **Statement** (known as the **subject, predicate and object** of a Statement).

Let's look at some example statements to get a better understanding:

Statement: "The author of https://www.w3schools.com/rdf is Jan Egil Refsnes".

* The subject of the statement above is: https://www.w3schools.com/rdf
* The predicate is: author
* The object is: Jan Egil Refsnes

Statement: "The homepage of https://www.w3schools.com/rdf is https://www.w3schools.com".

* The subject of the statement above is: https://www.w3schools.com/rdf
* The predicate is: homepage
* The object is: https://www.w3schools.com

## RDF Example

Here are two records from a CD-list:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Title** | **Artist** | **Country** | **Company** | **Price** | **Year** |
| Empire Burlesque | Bob Dylan | USA | Columbia | 10.90 | 1985 |
| Hide your heart | Bonnie Tyler | UK | CBS Records | 9.90 | 1988 |

Below is a few lines from an RDF document:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">  
  <cd:artist>Bob Dylan</cd:artist>  
  <cd:country>USA</cd:country>  
  <cd:company>Columbia</cd:company>  
  <cd:price>10.90</cd:price>  
  <cd:year>1985</cd:year>  
</rdf:Description>  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Hide your heart">  
  <cd:artist>Bonnie Tyler</cd:artist>  
  <cd:country>UK</cd:country>  
  <cd:company>CBS Records</cd:company>  
  <cd:price>9.90</cd:price>  
  <cd:year>1988</cd:year>  
</rdf:Description>  
.  
.  
.  
</rdf:RDF>

The first line of the RDF document is the XML declaration. The XML declaration is followed by the root element of RDF documents: **<rdf:RDF>**.

The **xmlns:rdf** namespace, specifies that elements with the rdf prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#".

The **xmlns:cd** namespace, specifies that elements with the cd prefix are from the namespace "http://www.recshop.fake/cd#".

The **<rdf:Description>** element contains the description of the resource identified by the **rdf:about** attribute.

The elements:**<cd:artist>, <cd:country>, <cd:company>,** etc. are properties of the resource.

## RDF Online Validator

[W3C's RDF Validation Service](http://www.w3.org/RDF/Validator/) is useful when learning RDF. Here you can experiment with RDF files.

The online RDF Validator parses your RDF document, checks your syntax, and generates tabular and graphical views of your RDF document.

Copy and paste the example below into W3C's RDF validator:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:si="https://www.w3schools.com/rdf/">  
<rdf:Description rdf:about="https://www.w3schools.com">  
  <si:title>W3Schools.com</si:title>  
  <si:author>Jan Egil Refsnes</si:author>  
</rdf:Description>  
</rdf:RDF>

When you parse the example above, [the result will look something like this](https://www.w3schools.com/xml/RDFvalidationresult.htm).

## RDF Elements

The main elements of RDF are the root element, <RDF>, and the <Description> element, which identifies a resource.

## The <rdf:RDF> Element

<rdf:RDF> is the root element of an RDF document. It defines the XML document to be an RDF document. It also contains a reference to the RDF namespace:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">  
  ...*Description goes here...*  
</rdf:RDF>

## The <rdf:Description> Element

The <rdf:Description> element identifies a resource with the about attribute.

The <rdf:Description> element contains elements that describe the resource:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">  
  <cd:artist>Bob Dylan</cd:artist>  
  <cd:country>USA</cd:country>  
  <cd:company>Columbia</cd:company>  
  <cd:price>10.90</cd:price>  
  <cd:year>1985</cd:year>  
</rdf:Description>  
  
</rdf:RDF>

The elements, artist, country, company, price, and year, are defined in the http://www.recshop.fake/cd# namespace. This namespace is outside RDF (and not a part of RDF). RDF defines only the framework. The elements, artist, country, company, price, and year, must be defined by someone else (company, organization, person, etc).

## Properties as Attributes

The property elements can also be defined as attributes (instead of elements):

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Empire Burlesque"  
cd:artist="Bob Dylan" cd:country="USA"  
cd:company="Columbia" cd:price="10.90"  
cd:year="1985" />  
  
</rdf:RDF>

## Properties as Resources

The property elements can also be defined as resources:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">  
  <cd:artist rdf:resource="http://www.recshop.fake/cd/dylan" />  
  ...  
  ...  
</rdf:Description>  
  
</rdf:RDF>

In the example above, the property artist does not have a value, but a reference to a resource containing information about the artist.

## RDF Containers

RDF containers are used to describe group of things.

The following RDF elements are used to describe groups: <Bag>, <Seq>, and <Alt>.

## The <rdf:Bag> Element

The <rdf:Bag> element is used to describe a list of values that do not have to be in a specific order.

The <rdf:Bag> element may contain duplicate values.

### Example

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Beatles">  
  <cd:artist>  
    <rdf:Bag>  
      <rdf:li>John</rdf:li>  
      <rdf:li>Paul</rdf:li>  
      <rdf:li>George</rdf:li>  
      <rdf:li>Ringo</rdf:li>  
    </rdf:Bag>  
  </cd:artist>  
</rdf:Description>  
  
</rdf:RDF>

## The <rdf:Seq> Element

The <rdf:Seq> element is used to describe an ordered list of values (For example, in alphabetical order).

The <rdf:Seq> element may contain duplicate values.

### Example

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Beatles">  
  <cd:artist>  
    <rdf:Seq>  
      <rdf:li>George</rdf:li>  
      <rdf:li>John</rdf:li>  
      <rdf:li>Paul</rdf:li>  
      <rdf:li>Ringo</rdf:li>  
    </rdf:Seq>  
  </cd:artist>  
</rdf:Description>  
  
</rdf:RDF>

## The <rdf:Alt> Element

The <rdf:Alt> element is used to describe a list of alternative values (the user can select only one of the values).

### Example

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://www.recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://www.recshop.fake/cd/Beatles">  
  <cd:format>  
    <rdf:Alt>  
      <rdf:li>CD</rdf:li>  
      <rdf:li>Record</rdf:li>  
      <rdf:li>Tape</rdf:li>  
    </rdf:Alt>  
  </cd:format>  
</rdf:Description>  
  
</rdf:RDF>

## RDF Terms

In the examples above we have talked about "list of values" when describing the container elements. In RDF these "list of values" are called members.

So, we have the following:

* A container is a resource that contains things
* The contained things are called members (not list of values)

## RDF Collections

RDF collections describe groups that can ONLY contain the specified members.

## The rdf:parseType="Collection" Attribute

As seen in the previous chapter, a container says that the containing resources are members - it does not say that other members are not allowed.

RDF collections are used to describe groups that can ONLY contain the specified members.

A collection is described by the attribute rdf:parseType="Collection".

### Example

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:cd="http://recshop.fake/cd#">  
  
<rdf:Description  
rdf:about="http://recshop.fake/cd/Beatles">  
  <cd:artist rdf:parseType="Collection">  
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/George"/>  
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/John"/>  
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Paul"/>  
    <rdf:Description rdf:about="http://recshop.fake/cd/Beatles/Ringo"/>  
  </cd:artist>  
</rdf:Description>  
  
</rdf:RDF>

## RDF Schema and Application Classes

RDF Schema (RDFS) is an extension to RDF.

RDF describes resources with classes, properties, and values.

In addition, RDF also needs a way to define application-specific classes and properties. Application-specific classes and properties must be defined using extensions to RDF.

One such extension is RDF Schema.

## RDF Schema (RDFS)

RDF Schema does not provide actual application-specific classes and properties.

Instead RDF Schema provides the framework to describe application-specific classes and properties.

Classes in RDF Schema are much like classes in object oriented programming languages. This allows resources to be defined as instances of classes, and subclasses of classes.

## RDFS Example

The following example demonstrates some of the RDFS facilities:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
xml:base="http://www.animals.fake/animals#">  
  
<rdf:Description rdf:ID="animal">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
</rdf:Description>  
  
<rdf:Description rdf:ID="horse">  
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>  
  <rdfs:subClassOf rdf:resource="#animal"/>  
</rdf:Description>  
  
</rdf:RDF>

In the example above, the resource "horse" is a subclass of the class "animal".

## Example Abbreviated

Since an RDFS class is an RDF resource we can abbreviate the example above by using rdfs:Class instead of rdf:Description, and drop the rdf:type information:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
xml:base="http://www.animals.fake/animals#">  
  
<rdfs:Class rdf:ID="animal" />  
  
<rdfs:Class rdf:ID="horse">  
  <rdfs:subClassOf rdf:resource="#animal"/>  
</rdfs:Class>  
  
</rdf:RDF>

That's it!

## The Dublin Core

The Dublin Core Metadata Initiative (DCMI) has created some predefined properties for describing documents.

RDF is metadata (data about data). RDF is used to describe information resources.

The Dublin Core is a set of predefined properties for describing documents.

The first Dublin Core properties were defined at the **Metadata Workshop in Dublin, Ohio** in 1995 and is currently maintained by the [Dublin Core Metadata Initiative](http://dublincore.org/).

|  |  |
| --- | --- |
| **Property** | **Definition** |
| Contributor | An entity responsible for making contributions to the content of the resource |
| Coverage | The extent or scope of the content of the resource |
| Creator | An entity primarily responsible for making the content of the resource |
| Format | The physical or digital manifestation of the resource |
| Date | A date of an event in the lifecycle of the resource |
| Description | An account of the content of the resource |
| Identifier | An unambiguous reference to the resource within a given context |
| Language | A language of the intellectual content of the resource |
| Publisher | An entity responsible for making the resource available |
| Relation | A reference to a related resource |
| Rights | Information about rights held in and over the resource |
| Source | A Reference to a resource from which the present resource is derived |
| Subject | A topic of the content of the resource |
| Title | A name given to the resource |
| Type | The nature or genre of the content of the resource |

A quick look at the table above indicates that RDF is ideal for representing Dublin Core information.

## RDF Example

The following example demonstrates the use of some of the Dublin Core properties in an RDF document:

<?xml version="1.0"?>  
  
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:dc= "http://purl.org/dc/elements/1.1/">  
  
<rdf:Description rdf:about="https://www.w3schools.com">  
  <dc:description>W3Schools - Free tutorials</dc:description>  
  <dc:publisher>Refsnes Data as</dc:publisher>  
  <dc:date>2008-09-01</dc:date>  
  <dc:type>Web Development</dc:type>  
  <dc:format>text/html</dc:format>  
  <dc:language>en</dc:language>  
</rdf:Description>  
  
</rdf:RDF>

## RDF Reference

The RDF namespace (xmlns:rdf) is: [http://www.w3.org/1999/02/22-rdf-syntax-ns#](https://www.w3schools.com/xml/rdf-syntax-ns.xml)

The RDFS namespace (xmlns:rdfs ) is: [http://www.w3.org/2000/01/rdf-schema#](https://www.w3schools.com/xml/rdf-schema.xml)

The recommended file extension for RDF files is **.rdf**. However, the extension **.xml** is often used to provide compatibility with old xml parsers.

The MIME type should be **"application/rdf+xml"**.

## RDFS / RDF Classes

|  |  |  |
| --- | --- | --- |
| **Element** | **Class of** | **Subclass of** |
| rdfs:Class | All classes |  |
| rdfs:Datatype | Data types | Class |
| rdfs:Resource | All resources | Class |
| rdfs:Container | Containers | Resource |
| rdfs:Literal | Literal values (text and numbers) | Resource |
| rdf:List | Lists | Resource |
| rdf:Property | Properties | Resource |
| rdf:Statement | Statements | Resource |
| rdf:Alt | Containers of alternatives | Container |
| rdf:Bag | Unordered containers | Container |
| rdf:Seq | Ordered containers | Container |
| rdfs:ContainerMembershipProperty | Container membership properties | Property |
| rdf:XMLLiteral | XML literal values | Literal |

## RDFS / RDF Properties

|  |  |  |  |
| --- | --- | --- | --- |
| **Element** | **Domain** | **Range** | **Description** |
| rdfs:domain | Property | Class | The domain of the resource |
| rdfs:range | Property | Class | The range of the resource |
| rdfs:subPropertyOf | Property | Property | The property is a sub property of a property |
| rdfs:subClassOf | Class | Class | The resource is a subclass of a class |
| rdfs:comment | Resource | Literal | The human readable description of the resource |
| rdfs:label | Resource | Literal | The human readable label (name)  of the resource |
| rdfs:isDefinedBy | Resource | Resource | The definition of the resource |
| rdfs:seeAlso | Resource | Resource | The additional information about the resource |
| rdfs:member | Resource | Resource | The member of the resource |
| rdf:first | List | Resource |  |
| rdf:rest | List | List |  |
| rdf:subject | Statement | Resource | The subject of the resource in an RDF Statement |
| rdf:predicate | Statement | Resource | The predicate of the resource in an RDF Statement |
| rdf:object | Statement | Resource | The object of the resource in an RDF Statement |
| rdf:value | Resource | Resource | The property used for values |
| rdf:type | Resource | Class | The resource is an instance of a class |

## RDF Attributes

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| rdf:about | Defines the resource being described |
| rdf:Description | Container for the description of a resource |
| rdf:resource | Defines a resource to identify a property |
| rdf:datatype | Defines the data type of an element |
| rdf:ID | Defines the ID of an element |
| rdf:li | Defines a list |
| rdf:\_*n* | Defines a node |
| rdf:nodeID | Defines the ID of an element node |
| rdf:parseType | Defines how an element should be parsed |
| rdf:RDF | The root of an RDF document |
| xml:base | Defines the XML base |
| xml:lang | Defines the language of the element content |

# XML RSS

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

With RSS it is possible to distribute up-to-date web content from one web site to thousands of other web sites around the world.

RSS allows fast browsing for news and updates.

### RSS Document Example

<?xml version="1.0" encoding="UTF-8" ?>  
<rss version="2.0">  
  
<channel>  
  <title>W3Schools Home Page</title>  
  <link>https://www.w3schools.com</link>  
  <description>Free web building tutorials</description>  
  <item>  
    <title>RSS Tutorial</title>  
    <link>https://www.w3schools.com/xml/xml\_rss.asp</link>  
    <description>New RSS tutorial on W3Schools</description>  
  </item>  
  <item>  
    <title>XML Tutorial</title>  
    <link>https://www.w3schools.com/xml</link>  
    <description>New XML tutorial on W3Schools</description>  
  </item>  
</channel>  
  
</rss>

## What is RSS?

* RSS stands for **R**eally **S**imple **S**yndication
* RSS allows you to syndicate your site content
* RSS defines an easy way to share and view headlines and content
* RSS files can be automatically updated
* RSS allows personalized views for different sites
* RSS is written in XML

## Why use RSS?

RSS was designed to show selected data.

Without RSS, users will have to check your site daily for new updates. This may be too time-consuming for many users. With an RSS feed (RSS is often called a News feed or RSS feed) they can check your site faster using an RSS aggregator (a site or program that gathers and sorts out RSS feeds).

Since RSS data is small and fast-loading, it can easily be used with services like cell phones or PDA's.

Web-rings with similar information can easily share data on their web sites to make them better and more useful.

## Who Should use RSS?

Webmasters who seldom update their web sites do not need RSS!

RSS is useful for web sites that are updated frequently, like:

* News sites - Lists news with title, date and descriptions
* Companies - Lists news and new products
* Calendars - Lists upcoming events and important days
* Site changes - Lists changed pages or new pages

## Benefits of RSS

Here are some benefits of using RSS:

**Choose your news**  
With RSS you can choose to view the news you want, the news that interest you and are relevant to your work.

**Remove unwanted information**  
With RSS you can (finally) separate wanted information from unwanted information (spam)!

**Increase your site traffic**  
With RSS you can create your own news channel, and publish it to the Internet!

## The History of RSS

* 1997 - Dave Winer at UserLand develops scriptingNews. RSS was born
* 1999 - Netscape develops RSS 0.90 (which supported scriptingNews)
* 1999 - Dave Winer develops scriptingNews 2.0b1 (which included RSS 0.90 features)
* 1999 - Netscape develops RSS 0.91 (which included most features from scriptingNews 2.0b1)
* 1999 - UserLand gets rid of scriptingNews and uses only RSS 0.91
* 1999 - Netscape stops their RSS development
* 2000 - UserLand releases the official RSS 0.91 specification
* 2000 - O'Reilly develops RSS 1.0. This format uses RDF and namespaces.
* 2000 - Dave Winer at UserLand develops RSS 0.92
* 2002 - Dave Winer develops RSS 2.0 after leaving UserLand
* 2003 - The official RSS 2.0 specification is released

RSS 1.0 is the only version that was developed using the W3C RDF (Resource Description Framework) standard.

The idea behind RDF was to help create a Semantic Web. However, this does not matter too much for ordinary users, but by using web standards it will be easier for persons and applications to exchange data.

## What RSS Version Should I Use?

RSS 0.91 and RSS 2.0 are easier to understand than RSS 1.0. Our tutorial is based on RSS 2.0.

The syntax rules of RSS 2.0 are very simple and very strict.

## Is RSS a Web Standard?

There is no official standard for RSS.

* About 50 % of all RSS feeds use RSS 0.91
* About 25 % use RSS 1.0
* The last 25 % is split between RSS 0.9x versions and RSS 2.0

## How RSS Works

RSS is used to share content between websites.

With RSS, you register your content with companies called aggregators.

So, to be a part of it: First, create an RSS document and save it with an .xml extension. Then, upload the file to your website. Next, register with an RSS aggregator. Each day the aggregator searches the registered websites for RSS documents, verifies the link, and displays information about the feed so clients can link to documents that interests them.

**Tip:** Read our RSS Publishing chapter to view free RSS aggregation services.

## RSS Example

RSS documents use a self-describing and simple syntax.

Here is a simple RSS document:

<?xml version="1.0" encoding="UTF-8" ?>  
<rss version="2.0">  
  
<channel>  
  <title>W3Schools Home Page</title>  
  <link>https://www.w3schools.com</link>  
  <description>Free web building tutorials</description>  
  <item>  
    <title>RSS Tutorial</title>  
    <link>https://www.w3schools.com/xml/xml\_rss.asp</link>  
    <description>New RSS tutorial on W3Schools</description>  
  </item>  
  <item>  
    <title>XML Tutorial</title>  
    <link>https://www.w3schools.com/xml</link>  
    <description>New XML tutorial on W3Schools</description>  
  </item>  
</channel>  
  
</rss>

The first line in the document - the XML declaration - defines the XML version and the character encoding used in the document. In this case the document conforms to the 1.0 specification of XML and uses the UTF-8 character set.

The next line is the RSS declaration which identifies that this is an RSS document (in this case, RSS version 2.0).

The next line contains the <channel> element. This element is used to describe the RSS feed.

The <channel> element has three required child elements:

* <title> - Defines the title of the channel (e.g. W3Schools Home Page)
* <link> - Defines the hyperlink to the channel (e.g. https://www.w3schools.com)
* <description> - Describes the channel (e.g. Free web building tutorials)

Each <channel> element can have one or more <item> elements.

Each <item> element defines an article or "story" in the RSS feed.

The <item> element has three required child elements:

* <title> - Defines the title of the item (e.g. RSS Tutorial)
* <link> - Defines the hyperlink to the item (e.g. https://www.w3schools.com/xml/xml\_rss.asp)
* <description> - Describes the item (e.g. New RSS tutorial on W3Schools)

Finally, the two last lines close the <channel> and <rss> elements.

## Comments in RSS

The syntax for writing comments in RSS is similar to that of HTML:

<!-- This is an RSS comment -->

## RSS is Written in XML

Because RSS is XML, keep in mind that:

* All elements must have a closing tag
* Elements are case sensitive
* Elements must be properly nested
* Attribute values must always be quoted

## The RSS <channel> Element

The RSS <channel> element describes the RSS feed.

Look at the following RSS document:

<?xml version="1.0" encoding="UTF-8" ?>  
<rss version="2.0">  
  
<channel>  
  <title>W3Schools Home Page</title>  
  <link>https://www.w3schools.com</link>  
  <description>Free web building tutorials</description>  
  <item>  
    <title>RSS Tutorial</title>  
    <link>https://www.w3schools.com/xml/xml\_rss.asp</link>  
    <description>New RSS tutorial on W3Schools</description>  
  </item>  
</channel>  
  
</rss>

As mentioned before, the <channel> element describes the RSS feed, and has three required child elements:

* <title> - Defines the title of the channel (e.g. W3Schools Home Page)
* <link> - Defines the hyperlink to the channel (e.g. https://www.w3schools.com)
* <description> - Describes the channel (e.g. Free web building tutorials)

The <channel> element usually contains one or more <item> elements. Each <item> element defines an article or "story" in the RSS feed.

Furthermore, there are several optional child elements of <channel>. We will explain the most important ones below.

## The <category> Element

The <category> child element is used to specify a category for your feed.

The <category> element makes it possible for RSS aggregators to group sites based on category.

The category for the RSS document above could be:

<category>Web development</category>

## The <copyright> Element

The <copyright> child element notifies about copyrighted material.

The copyright for the RSS document above could be:

<copyright>2006 Refsnes Data as. All rights reserved.</copyright>

## The <image> Element

The <image> child element allows an image to be displayed when aggregators present a feed.

The <image> element has three required child elements:

* <url> - Defines the URL to the image
* <title> - Defines the text to display if the image could not be shown
* <link> - Defines the hyperlink to the website that offers the channel

The image for the RSS document above could be:

<image>  
  <url>https://www.w3schools.com/images/logo.gif</url>  
  <title>W3Schools.com</title>  
  <link>https://www.w3schools.com</link>  
</image>

## The <language> Element

The <language> child element is used to specify the language used to write your document.

The <language> element makes it possible for RSS aggregators to group sites based on language.

The language for the RSS document above could be:

<language>en-us</language>

## The <item> Element

Each <item> element defines an article or "story" in an RSS feed.

Look at the following RSS document:

<?xml version="1.0" encoding="UTF-8" ?>  
<rss version="2.0">  
  
<channel>  
  <title>W3Schools Home Page</title>  
  <link>https://www.w3schools.com</link>  
  <description>Free web building tutorials</description>  
  <item>  
    <title>RSS Tutorial</title>  
    <link>https://www.w3schools.com/xml/xml\_rss.asp</link>  
    <description>New RSS tutorial on W3Schools</description>  
  </item>  
</channel>  
  
</rss>

As mentioned before, each <item> element defines an article or "story" in the RSS feed.

The <item> element has three required child elements:

* <title> - Defines the title of the item (e.g. RSS Tutorial)
* <link> - Defines the hyperlink to the item (e.g. https://www.w3schools.com/xml/xml\_rss.asp)
* <description> - Describes the item (e.g. New RSS tutorial on W3Schools)

Furthermore, there are several optional child elements of <item>. We will explain the most important ones below.

## The <author> Element

The <author> child element is used to specify the e-mail address of the author of an item.

**Note:** To prevent spam e-mails, some developers do not include the <author> element.

The author of the item in the RSS document above could be:

<author>hege@refsnesdata.no</author>

## The <comments> Element

The <comments> child element allows an item to link to comments about that item.

A comment of the item in the RSS document above could be:

<comments>https://www.w3schools.com/comments</comments>

## The <enclosure> Element

The <enclosure> child element allows a media-file to be included with an item.

The <enclosure> element has three required attributes:

* url - Defines the URL to the media file
* length - Defines the length (in bytes) of the media file
* type - Defines the type of media file

A media-file included in the item in the RSS document above could be:

<enclosure url="https://www.w3schools.com/xml/rss.mp3"  
length="5000" type="audio/mpeg" />

## Get Your RSS Feed Up On The Web

Having an RSS document is not useful if other people cannot reach it.

Now it's time to get your RSS file up on the web. Here are the steps:

1. Name your RSS file. Notice that the file must have an .xml extension.

2. Validate your RSS file (a good validator can be found at [http://www.feedvalidator.org](http://www.feedvalidator.org/)).

3. Upload the RSS file to your web directory on your web server.

4. Copy the little orange RSS Logo or XML Logo button to your web directory.

5. Put the little orange "RSS" or "XML" button on the page where you will offer RSS to the world (e.g. on your home page). Then add a link to the button that links to the RSS file. The code will look something like this:  
<a href="https://www.w3schools.com/xml/myfirstrss.xml">  
<img src="https://www.w3schools.com/xml/pic\_rss.gif" width="36" height="14">  
</a>.

6. Submit your RSS feed to the RSS Feed Directories (you can Google or Yahoo for "RSS Feed Directories"). Note! The URL to your feed is not your home page, it is the URL to your feed, like "https://www.w3schools.com/xml/myfirstrss.xml". Here is a free RSS aggregation service:

* [Newsisfree](http://www.newsisfree.com/): [Register here](http://www.newsisfree.com/user/new/)

7. Register your feed with the major search engines:

* Google -<http://www.google.com/submityourcontent/website-owner>
* Bing - <http://www.bing.com/toolbox/submit-site-url>

8. Update your feed - After registering your RSS feed, you must make sure that you update your content frequently and that your RSS feed is constantly available.

## Can I Manage my RSS Feed Myself?

The best way to ensure your RSS feed works the way you want, is to manage it yourself.

However, this can be very time consuming, especially for pages with lot of updates.

An alternative is to use a third-party automated RSS.

## Automated RSS

For users who only need an RSS feed for their personal website, some of the most popular blog (Web Log) managers that offer built-in RSS services are:

* [Wordpress](http://wordpress.org/)
* [Blogger](http://www.blogger.com/)
* [Radio](http://radio.userland.com/)

## RSS Readers

An RSS Reader is used to read RSS Feeds!

RSS readers are available for many different devices and OS.

There are a lot of different RSS readers. Some work as web services, and some are limited to windows (or Mac, PDA or UNIX):

* [QuiteRSS](http://quiterss.org/en) - FREE! QuiteRSS is an open-source, cross-platform RSS/Atom news reader. It is versatile, and has a full set of options. QuietRSS has a rich set of social sharing options (Email/Twitter/Facebook/.../Pocket/Etc). QuietRSS is fast starting, and navigation is quick
* [FeedReader](http://feedreader.com/) - FREE! Simple, straightforward feed reader that easily handles large number of feeds. Has the essential options (not a lot of confusing ones). Does not require Java. Import or export OPML files. Option to open links in an external browser
* [RssReader](http://www.rssreader.com/) - FREE! Windows-based RSS reader. Supports RSS versions 0.9x, 1.0 and 2.0 and Atom 0.1, 0.2 and 0.3
* [blogbotrss](http://sourceforge.net/projects/blogbotrss/) - FREE! An RSS reader plug-in for Internet Explorer and Microsoft Outlook

**Tip:** Most browsers have a built-in RSS Reader. If you go to a web site that offers RSS feeds, you will see an RSS icon  in the address bar, or toolbar. Click on the icon to view a list of the different feeds. Choose the feed you want to read.

## I have an RSS Reader. Now what?

Click on the little RSS Logo or XML Logo button next to the RSS feed you want to read. Copy The URL you get in the browser window and paste it in your RSS reader.

## RSS Examples

These examples demonstrate RSS using our RSS reader to view the results.

**RSS <channel> Element**

[Setting the required channel elements (<title>, <link>, and <description>)](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_title_link_description)  
[Specify a category for the RSS](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_category)  
[Specify the program used to generate the RSS](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_generator)  
[Add an image to the RSS](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_image)  
[Specify the language of the RSS](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_language)  
[Add an text input field to the RSS](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_textinput)  
[Specify days that the RSS should not be updated](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_skipdays)

**RSS <item> element**

[Setting the required item elements (<title>, <link>, and <description>)](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_title_link_description_item)  
[Add a link to comments about the RSS item](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_comments)  
[Add a media file to the RSS item](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_enclosure)  
[Specify a unique identifier for the item](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_guid)  
[Specify the publication date for the RSS item](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_pubdate_item)  
[Specify a third-party source for the RSS item](https://www.w3schools.com/xml/tryrss.asp?filename=rss_ex_source)

## RSS Reference

## RSS <channel> Element

The links in the "Element" column point to more information about each specific element.

|  |  |
| --- | --- |
| **Element** | **Description** |
| [<category>](https://www.w3schools.com/xml/rss_tag_category_channel.asp) | Optional. Defines one or more categories for the feed |
| [<cloud>](https://www.w3schools.com/xml/rss_tag_cloud.asp) | Optional. Register processes to be notified immediately of updates of the feed |
| [<copyright>](https://www.w3schools.com/xml/rss_tag_copyright.asp) | Optional. Notifies about copyrighted material |
| [<description>](https://www.w3schools.com/xml/rss_tag_title_link_description_channel.asp) | Required. Describes the channel |
| [<docs>](https://www.w3schools.com/xml/rss_tag_docs.asp) | Optional. Specifies a URL to the documentation of the format used in the feed |
| [<generator>](https://www.w3schools.com/xml/rss_tag_generator.asp) | Optional. Specifies the program used to generate the feed |
| [<image>](https://www.w3schools.com/xml/rss_tag_image.asp) | Optional. Allows an image to be displayed when aggregators present a feed |
| [<language>](https://www.w3schools.com/xml/rss_tag_language.asp) | Optional. Specifies the language the feed is written in |
| [<lastBuildDate>](https://www.w3schools.com/xml/rss_tag_lastbuilddate.asp) | Optional. Defines the last-modified date of the content of the feed |
| [<link>](https://www.w3schools.com/xml/rss_tag_title_link_description_channel.asp) | Required. Defines the hyperlink to the channel |
| [<managingEditor>](https://www.w3schools.com/xml/rss_tag_managingeditor.asp) | Optional. Defines the e-mail address to the editor of the content of the feed |
| [<pubDate>](https://www.w3schools.com/xml/rss_tag_pubdate.asp) | Optional. Defines the last publication date for the content of the feed |
| <rating> | Optional. The PICS rating of the feed |
| [<skipDays>](https://www.w3schools.com/xml/rss_tag_skipDays.asp) | Optional. Specifies the days where aggregators should skip updating the feed |
| [<skipHours>](https://www.w3schools.com/xml/rss_tag_skipHours.asp) | Optional. Specifies the hours where aggregators should skip updating the feed |
| [<textInput>](https://www.w3schools.com/xml/rss_tag_textinput.asp) | Optional. Specifies a text input field that should be displayed with the feed |
| [<title>](https://www.w3schools.com/xml/rss_tag_title_link_description_channel.asp) | Required. Defines the title of the channel |
| [<ttl>](https://www.w3schools.com/xml/rss_tag_ttl.asp) | Optional. Specifies the number of minutes the feed can stay cached before refreshing it from the source |
| [<webMaster>](https://www.w3schools.com/xml/rss_tag_webmaster.asp) | Optional. Defines the e-mail address to the webmaster of the feed |

## RSS <item> Element

|  |  |
| --- | --- |
| **Element** | **Description** |
| [<author>](https://www.w3schools.com/xml/rss_tag_author.asp) | Optional. Specifies the e-mail address to the author of the item |
| [<category>](https://www.w3schools.com/xml/rss_tag_category_item.asp) | Optional. Defines one or more categories the item belongs to |
| [<comments>](https://www.w3schools.com/xml/rss_tag_comments.asp) | Optional. Allows an item to link to comments about that item |
| [<description>](https://www.w3schools.com/xml/rss_tag_title_link_description_item.asp) | Required. Describes the item |
| [<enclosure>](https://www.w3schools.com/xml/rss_tag_enclosure.asp) | Optional. Allows a media file to be included with the item |
| [<guid>](https://www.w3schools.com/xml/rss_tag_guid.asp) | Optional. Defines a unique identifier for the item |
| [<link>](https://www.w3schools.com/xml/rss_tag_title_link_description_item.asp) | Required. Defines the hyperlink to the item |
| [<pubDate>](https://www.w3schools.com/xml/rss_tag_pubdate_item.asp) | Optional. Defines the last-publication date for the item |
| [<source>](https://www.w3schools.com/xml/rss_tag_source.asp) | Optional. Specifies a third-party source for the item |
| [<title>](https://www.w3schools.com/xml/rss_tag_title_link_description_item.asp) | Required. Defines the title of the item |