



# Week 4 Lecture : XML Elements, Attributes, and Namespaces

Screencast of Recorded Lecture (2022):



- [https://sjeccd-edu.zoom.us/rec/play/\\_XgP4IMpQN3WbbePkqiWaXsMd\\_owepnZHQ4m7g4J\\_4evnVr2SgYvY\\_4qy-ciZTyOgsq12Tf11JJKuezJS.BFnHGQ91rpWCSP79](https://sjeccd-edu.zoom.us/rec/play/_XgP4IMpQN3WbbePkqiWaXsMd_owepnZHQ4m7g4J_4evnVr2SgYvY_4qy-ciZTyOgsq12Tf11JJKuezJS.BFnHGQ91rpWCSP79)  (https://sjeccd-edu.zoom.us/rec/play/\_XgP4IMpQN3WbbePkqiWaXsMd\_owepnZHQ4m7g4J\_4evnVr2SgYvY\_4qy-ciZTyOgsq12Tf11JJKuezJS.BFnHGQ91rpWCSP79)

Screencast of Recorded Lecture (2021):

- [https://sjeccd-edu.zoom.us/rec/play/RKjYqWD3evVh-kgjx2o5JPlkq\\_9Dr-eyRCNqs0SQjbiCNOAKJ\\_zsH0NGotUD1vQYqgZvxJtAUtIGphp4.ft3sFm1hzxvvkaOO](https://sjeccd-edu.zoom.us/rec/play/RKjYqWD3evVh-kgjx2o5JPlkq_9Dr-eyRCNqs0SQjbiCNOAKJ_zsH0NGotUD1vQYqgZvxJtAUtIGphp4.ft3sFm1hzxvvkaOO)  (https://sjeccd-edu.zoom.us/rec/play/RKjYqWD3evVh-kgjx2o5JPlkq\_9Dr-eyRCNqs0SQjbiCNOAKJ\_zsH0NGotUD1vQYqgZvxJtAUtIGphp4.ft3sFm1hzxvvkaOO)

---

Further reading:

- Beginning XML -- Chapter 3: XML Namespaces  
<https://learning.oreilly.com/library/view/beginning-xml-5th/9781118239483/xhtmll/Chapter03.html>  (https://learning.oreilly.com/library/view/beginning-xml-5th/9781118239483/xhtmll/Chapter03.html)
- XML Visual Quickstart Guide -- Chapter 12: XML Namespaces  
<https://learning.oreilly.com/library/view/xml-visual-quickstart/9780321602589/ch13.html>  (https://learning.oreilly.com/library/view/xml-visual-quickstart/9780321602589/ch13.html)

Now, let's talk some more about Elements and Attributes.

XML elements are **extensible**. This means that XML documents can be **extended** to contain more or additional information.

Look at this example:

```
<note>
<to>Joe</to>
<from>Tom</from>
<body>Don't forget your book</body>
</note>
```

Imagine that we now create an application to extract the `<to>`, `<from>`, and `<body>` elements from the XML document to produce this output:

### Message

**To: Joe**

**From: Tom**

**Don't forget your book**

OK, now suppose that some additional information is added to the document, such as:

```
<note>
<date>2016-09-14</date>
<to>Joe</to>
<from>Tom</from>
<body>Don't forget your book</body>
</note>
```

Do you think the extracting application will now crash?

No, the application will still be able to find the `<to>`, `<from>`, and `<body>` elements in the XML document and produce the same output. Therefore, XML documents are *Extensible*. That is -- XML documents can have more information placed on it than they originally contained; they can be *extended*.

## XML Elements have *Relationships*. Elements are related as *parents and children*.

To really understand XML terminology, you need to know how relationships between XML elements are named, and how element content is built.

Given the following description of a text book:

## XML by Example

### Introduction to XML

- What is XML
- XML Examples

### XML Syntax

- Elements
- Attributes

An XML document that describes the book could look like this:

```
<book>
<title>XML by Example</title>
<chapter>Introduction to XML
  <paragraph>What is XML</paragraph>
  <paragraph>XML Examples</paragraph>
</chapter>
<chapter>XML Syntax
  <paragraph>Elements</paragraph>
  <paragraph>Attributes</paragraph>
</chapter>
</book>
```

*book* is the **root** element; *title* and *chapter* are **child** elements of *book*.

*book* is the **parent** element of *title* and *chapter*. *title* and *chapter* are **siblings**, or **sister elements**, because they have the same parent.

*paragraph* is the child of *chapter*, which is the child of *book*.

## Elements have Content, and elements can have different content types.

An XML element is everything from, and including, the element's start tag to, and including, the element's end tag. An element can have element content, mixed content, simple content, or empty content. An element can also have attributes, more on this later.

In the book example above, *book* has element content, because it contains other elements. *chapter* has mixed content because it contains both text and other elements. *paragraph* has simple content, or text content, because it contains only text.

## Element Naming

XML elements **must** follow these naming rules:

- Names **can** contain letters, numbers, and other characters
- Names **must not** start with a number or punctuation character
- Names **must not** start with the letters xml, or XML, or Xml, etc.
- Names **must not** contain spaces

Be careful when you invent element names and be sure to follow the above rules. Any name can be used -- no words are reserved -- but try to make element names descriptive. Using an underscore separator can be a useful technique, such as `<first_name>`. Some XML programmers prefer to use "camel case" -- that is, the second word is capitalized (so it looks like a 1-humped camel). `<firstName>` is an example of camel case. Another choice is to capitalize each word. But whichever method you use, you must always remember to close the tag, either by using a closing tag or by making it an empty element (and the closing tag needs to match the naming convention used in the starting tag).

`<FirstName> </FirstName>` is OK.

`<FirstName />` is OK.

`<FirstName> </firstName>` is NOT ok. Nor is `<firstname></first_name>`

Element names can be as long as you like, but short and simple is the best rule to follow.

## XML elements can have attributes.

Attribute values must always be enclosed in quotes, but either single or double quotes can be used. For a person's gender, the person tag could be written like this:

`<person gender="female">` or like this: `<person gender='female'>`.

Note that if you start an attribute with a double quote, you need to end it with a double quote; if you start it with a single quote, you need to end it with a single quote. Don't mix & match! It's been my experience that most programmers in the US tend to use double quotes, whereas in lots of the rest of the world, programmers tend to use single quotes. It's really a personal choice.

Data can be stored in child elements or in attributes, such as:

```
<person gender="female">
```

```
<first_name>Mary</first_name>
<last_name>Jones</last_name>
</person>
```

OR

```
<person>
  <gender>female</gender>
  <first_name>Mary</first_name>
  <last_name>Jones</last_name>
</person>
```

In the first example above **gender** is an attribute. In the last example above **gender** is a child element. Both examples provide the same information. There are no rules about when to use attributes and when to use child elements. I generally prefer using child elements, but it is your choice.

Here are some of the problems with attributes:

- attributes cannot contain multiple values (child elements can)
- attributes are not easily expandable for future changes
- attributes cannot describe structures (child elements can)
- attributes are more difficult to manipulate by an application
- attribute values are not easy to test against a Document Type Definition (DTD), which is used to define the legal elements of an XML document. We'll talk more about this later on.

## Now, let's talk about XML Namespaces

XML allows you to invent your own markup tags for your own purpose and this opens the possibility that many documents contain the same tag name but have different meaning. XML documents are often combined and this allows the possibility of element name duplication. XML Namespaces provide a method to avoid element name conflicts. Since element names in XML are not fixed, very often a name conflict will occur when two different documents use the same names describing two different types of elements.

Look at these examples:


This document carries information in a table (a table of data):

```
<table>
  <fruit>Apples</fruit>
  <fruit>Pears</fruit>
</table>
```

This document carries information about a table (a table as in the furniture):

```
<table>
<name>Coffee Table</name>
<length>60</length>
</table>
```

If these two XML documents were added together, there would be an element name conflict because **both** documents contain a `<table>` element with different content and meaning.

Namespaces were developed as a way to avoid this name collision by linking a collection of names with a URI (Uniform Resource Identifier). A URI is the method in which resources on the Web are named, the most widespread form being URLs (Uniform Resource Locator) or Web address, such as, <http://www.anyplace.com>  <http://www.anyplace.com/>. The URI doesn't have to point to any particular file. The URI that defines a namespace is purely formal. Its only purpose is to group and disambiguate element and attribute names in the document.

Namespaces are declared by adding an `xmlns` attribute to the outermost element in which you want to use the namespace. The syntax for explicitly declaring a namespace is:

```
<myelement xmlns:prefix="URI">
</myelement>
```

In the above example:

- `xmlns` is a reserved attribute used to declare namespaces.
- `prefix` is the optional alias for the namespace.
- `URI` is a unique string used to identify a namespace.

Given the following XML structure to describe a dinner plate:

```
<dish>
<type>china</type>
<shape>round</shape>
</dish>
```

Let's declare and use a namespace, such as `xmlns:pf="http://www.pfaltzgraff.com"`:

```
<pf:dish xmlns:pf="http://www.pfaltzgraff.com">  
<pf:type>china</pf:type>  
<pf:shape>round</pf:round>  
</pf:dish>
```

When you declare a namespace, **the element in which you define it dictates the namespace's scope.**

A namespace can be declared at the root level of a document or it can be declared inside any element of the XML structure. However, a namespace is visible to only the element in which it is declared and all of the child elements of that container element.

In addition to binding a prefix to a namespace, you can declare a default namespace as well. A default namespace applies a namespace to the element in which it is declared and all of its child elements and does so without adding a prefix before the element name. To declare a default namespace, add a `xmlns` attribute to your element rather than a `xmlns:prefix` attribute, as follows:

```
<dish xmlns="http://www.pfaltzgaff.com">  
<type>china</type>  
<shape>round</shape>  
</dish>
```

Well, I think that is enough for now.