Learning XML: A Beginner's Guide

By Cooper Smitherman

Contents

Defining XML

XML, or eXtensible Markup Language, allows you to define, store, and share structured information. XML is a self-descriptive language, which means that users can create meaningful tag names to carry and structure information web-based documents.

Why Should You Use XML?

XML is not designed to perform operations like other programming languages. Instead, XML annotates existing information so that software may store, format, publish, and update existing documents.

What are the Pros of XML?

- Allows users to customize a document's labeling and organizational system without changing the content within the document.
- Improves efficiency for storing, publishing, and updating information.
- Readable, precise, unambiguous, and flexible.

What are the Cons of XML?

- Requires structure definitions for new tags and elements.
- Requires entity references, or substitutes, for multiple characters. For example, an apostrophe (‘) must be written as **'** and an ampersand (**&**) must be written as **&**).
- Uses more storage because of redundancy.

Glossary

Child Element: An element that is contained or nested within another element (i.e., the parent element). For example, the parent element “newspaper” could contain child elements such as “headline” or “locallistings.”

Document Type Definition (DTD): Essentially a glossary for an XML document that defines the document’s element attributes and structure. The DTD does not contain any content, is not written in XML, and does not allow for easy reusability or perfect validation.

Element: Contains the document’s content and constitutes the informational hierarchy. Each element is an instance of a type of content and includes the opening and closing tags, attributes, text, and other elements. For example, an element called “rule” could contain an instance such as “Don’t run, jump, or shout.”

Parent Element: An element that contains other elements or is situated directly above another element within the information hierarchy. For example, the element “song” could be the parent element for the “releasedate” element.

Prolog: The first part of a document that provides external information about the XML version, character set, DTD, and stylesheet to build a properly formatted document. The Prolog ensures the document adheres to XML rules and is correctly displayed to users.

Root Element: Located at the top of the document’s hierarchy, the root element contains every other element nested within it. Naturally, the root element is the only required part of the document because it’s the first, or highest, level of content.

Schema: Like a DTD in that it takes extra steps to describe structures for documents. However, unlike a DTD, a Schema is written in XML, offers improved validation, and allows more flexibility in grouping at the cost of readability and efficiency.

Tags: Label and group elements by category and type for the program to format, transform, store, and publish documents. Additionally, XML tags allow documents to update themselves as information changes or new information is added.

Tree: The informational hierarchy of an XML document resembles a tree because all information is contained within one root element and progresses from higher to lower levels. All elements branch from this one root element and may contain other elements or be nested within other elements, .

Valid XML: Follows general XML syntax rules, and the document structure conforms to the pre-existing DTD or Schema and other guidelines provided in the Prolog. Valid XML is well-formed and adheres to a DTD or schema.

Well-Formed XML: Follows general XML syntax rules. For example, all content must be defined, properly nested, and possess a beginning and end tag; further, beginning and end tags must match each other, a root element must be present, and entity references are used when appropriate.

Introducing XML Syntax

Why start with a prolog?

The prolog, or XML declarations, draws external information about the XML version, character set, DTD, and stylesheet to display the document correctly.

Prolog Syntax Rules	Examples
• Always type the prolog at the top of the document. You can often copy and paste the prolog across documents.	**Correct**: <?xml version="1.0" encoding="UTF-8"?> <first> **Incorrect**: <first> **<?xml version="1.0" encoding="UTF-8"?>** (The prolog is incorrectly not placed at the top of the document.)
• Type **"1.0"** (recommended) or "**1.1"** after **version=** to reflect the document's XML version. • Type **"UTF-8"** (recommended) or **"UTF-16"** after **encoding=**. • Type **"yes"** or **"no"** after **standalone=** to reflect whether the document draws from an external DTD.	**Correct**: <?xml version="1.0" encoding="UTF-8" standalone="no"?> **Incorrect**: <?xml version="**2**.0" encoding="UT**f**-8"?> (The xml version is incorrectly listed as "2.0" (which doesn't exist), and the "f" is not capitalized.)
• Type **<?xml** at the beginning of the prolog. Type **?>** at the end of the prolog.	**Correct**: <?xml version="1.0" encoding="UTF-8"?> **Incorrect**: **<xml** version="1.0" encoding="UTF-8"**>** (The question mark is missing at the beginning and the end of the prolog.)

How do you write tags?

Tags label and group the document's information so the program may easily read, publish, and update documents. The writer customizes tags to represent the information contained within each tag. For example, if someone wrote the tag **<Dishes>**, the information contained within this tag could likely be "Mug" or "Plate."

Tags Syntax Rules	Examples
Type all tags between **<** and **>**.	**Correct**: <song> **Incorrect**: <album (**<** is placed after the tag name instead of **>**.)
Never use spaces in tag names.	**Correct:** <BusinessName> **Incorrect:** <Business Name> (A space is placed between "Business" and "Name.")
Type **/** after **<** in the closing tag.	**Correct**: <MenuItem>Cheeseburger</MenuItem> **Incorrect**:<FilmTitle>Django Unchained<FilmTitle> (The **/** is missing in the closing tag.)
Use the same case for opening and closing tags. Tags can be lowercase or uppercase as long as opening tags match their closing tags.	**Correct**: <greeting>Hello!</greeting> **Incorrect**: <Greeting>How's it going?</greeting> (The opening tag is written in uppercase, and the closing tag doesn't match because it's written in lowercase.)

How do you organize elements?

Elements are the document's content and constitute the informational hierarchy. Elements include the opening and closing tags, as well as everything in between these tags. For example, the **<VideoGame>Fallout 4</VideoGame** element is labeled as **VideoGame** with **Fallout 4** as the text.

Elements Syntax Rules	Example
Always type a root element at the top of a document or immediately after the prolog; a root element is the only required part of a document.	**Correct**: <?xml version="1.0" encoding="UTF-8**"**?> <total> **Incorrect**: **<total>** <?xml version="1.0" encoding="UTF-8"?> (The root element is incorrectly placed before the prolog.)
Properly nest all elements. Nesting, or placing one element inside another, contributes to XML's hierarchical structure. If you open an element inside another element, the first element must also be closed before the second element.	**Correct**: <poem><line>Two roads diverged in a yellow wood, </line></poem> **Incorrect**: <team><name>San Francisco 49ers**</team>** **</name>** (The **<team>** tag is incorrectly closed before the **<name>** element, which must be closed first because it was opened in the **<team>** element.)
Add the element's attribute in the opening Tag. Type the attribute name after a space. Then, type **="Text"** or **='Text'** before **>**.	**Correct**: <cat name="Oakley"> **Incorrect**: <dog name=**Kelso**> (The **Kelso** attribute has no quotation marks around it.)

How do you write text correctly?

The text in each element is what will be displayed to the user. Because of XML's specific syntax requirements, there are a couple of important caveats to writing text in XML.

Text Syntax Rules	Examples
Because you must use characters such as **<** and **>** as markup for XML, these characters must be written with entity references, or substitutes, within the document's text. Consult this list of entity references for these characters: 1. Instead of **<**, type **<** 2. Instead of **>**, type **>** 3. Instead of **&**, type **&** 4. Instead of **'**, type **'** 5. Instead of **"**, type **"**	**Correct**: <money>profit > $5,000</money> **Incorrect**: <characters>Saul **&** Kim</characters> (The ampersand is not written as its entity reference, which means the document will be parsed incorrectly.)
Type only the desired amount of spaces in all text because XML will display all white spaces.	**Correct**: <wish>One million dollars</wish> **Incorrect**: <groceries>Apples, Oranges, and **Grapes** </groceries> (There are extra spaces between **and** and **Grapes**, which means the program will display unnecessary white space on the document when it's viewed by the user.)

Activity 1

Try to correct the syntax errors in the following XML document that organizes an ice cream parlor's menu by flavor, price, and color.

```
<?xml version= "1" encoding="UtF-8?>
<Ice Cream Menu>
  <IceCream<
    <Flavor>Banana Blitz<flavor>
    <Price>$3.55</Price>
    <Color>Yellow</Color>
  </Icecream>
  <IceCream>
    >Flavor>Belgian Chocolate</Flavor >
    <Price>$4.00</Price>
    <Color>Brown<Color>
  <IceCream>
    <Flavor>Strawberry Supreme<Flavor>
    <Price>$3.05</price>
    <Color>Pink</ Color>
  </IceCream>
</IceCreamMenu>
```

Activity 1 Answer

```
<?xml version="1.0" encoding="UTF-8"?>
<IceCreamMenu>
  <IceCream>
    <Flavor>Banana Blitz</Flavor>
    <Price>$3.55</Price>
    <Color>Yellow</Color>
  </IceCream>
  <IceCream>
    <Flavor>Belgian Chocolate</Flavor>
    <Price>$4.00</Price>
    <Color>Brown</Color>
  </IceCream>
  <IceCream>
    <Flavor>Strawberry Supreme</Flavor>
    <Price>$3.05</Price>
    <Color>Pink</Color>
  </IceCream>
</IceCreamMenu>
```

The appropriate changes are highlighted in blue.

Validating Documents

To manually validate a document, you can ensure the document's structure and content adhere to XML syntax rules and, if applicable, internal/external DTD or schema rules.

If a document adheres to general XML syntax, then it is considered **well-formed**. If a document adheres to general XML syntax and a pre-existing DTD or Schema and other guidelines provided in the Prolog, then it is considered **valid**. A document can be well-formed and valid, but a document cannot be valid and not well-formed.

While you can manually validate a document, you can also validate a document's syntax, a document against an internal DTD, or a schema in isolation using an online XML Validator.

Activity 2

Create an XML document that organizes the Miami Heat, Chicago Bulls, and Indiana Pacers into the NBA's Eastern Conference. Further, organize the information from the following table, which provides each team's number of Finals wins, number of MVPs, and number of head coaches.

NBA TEAM INFORMATION

	Miami Heat	Chicago Bulls	Indiana Pacers
Finals Wins	3	6	0
Number of MVPs	2	6	0
Number of Head Coaches	6	24	16

To write or edit an XML document, you can use one of these Authoring Tools or others:

- Microsoft Notepad
- Oxygen XML Editor
- Adobe FrameMaker

Activity 2 Potential Answer

```
<?xml version="1.0" encoding="UTF-8"?>
<EasternConference>
  <NBATeam>
    <Name>Miami Heat</Name>
    <Finals>3</Finals>
    <MVPs>2</MVPs>
    <HeadCoaches>6</HeadCoaches>
  </NBATeam>
  <NBATeam>
    <Name>Chicago Bulls</Name>
    <Finals>6</Finals>
    <MVPs>6</MVPs>
    <HeadCoaches>24</HeadCoaches>
  </NBATeam>
  <NBATeam>
    <Name>Indiana Pacers</Name>
    <Finals>0</Finals>
    <MVPs>0</MVPs>
    <HeadCoaches>16</HeadCoaches>
  </NBATeam>
</EasternConference>
```

This answer places every team within the “EasternConference” parent element. Each “NBATeam” element lists the team name and the team’s number of Finals wins, MVPs, and head coaches.

DTD Basics

Occasionally, you will need to use unique element attributes and structures. If so, you will need to define these attributes and structures by writing an internal or an external DTD. Luckily, DTDs often already exist and will be provided to you.

An internal DTD is contained within the same file as the document, which is ideal for one-off documents. The internal DTD is opened and closed after the Prolog (but before the document itself).

An external DTD is written and defines a document's rules in a separate file, which is ideal for creating complex, reusable structures to be used for a wide range of documents. The external DTD, therefore, is better suited for creating consistency within larger projects rather than individual, isolated documents.

What are the Pros & Cons of DTDs?

Pros	Cons
Usually, DTDs are created for you. As long as you know the DTD structure, you'll generally be able to proceed with ease.	Difficult to change because they are not written in XML.
Can validate document structures.	Can only somewhat validate that elements are unique because DTDs are written to validate structures and not individual elements. For example, a DTD would validate a **<Flower name="Lily">** element if the **<Flower>** element is defined in the DTD. However, if two **<Flower name="Lily">** elements were present, the DTD would not detect error of the repeated elements.
Efficiently defines a document's structure.	Difficult to read and maintain if they are quite large.

Introducing Internal DTD Syntax

Internal DTD Syntax Rules	Example
Always open the internal DTD after the Prolog and close it after defining each element in the document. Open the internal DTD as follows: **<!DOCTYPE RootElement [** Close the internal DTD as follows: **]>**	**Correct**: <?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE Collection [**Incorrect**: **<DOCTYPE Collection [** <?xml version="1.0" encoding="UTF-8"?> (The exclamation point is missing from the internal DTD declaration, which is incorrectly placed before the prolog.)
Define each parent element in hierarchical order. Place any child elements within parentheses. Ensure that elements are consistently named. Define parent elements as follows: **<!ELEMENT ParentElement (ChildElement1, ChildElement2)>**	**Correct**: <!ELEMENT Album (Song)> <!ELEMENT Song (Writer, TrackNumber, Length)> **Incorrect**: <!**element Can** (Vegetable, ExpirationDate, Brand)> <!**element** Pantry (**Can**)> (The **<!ELEMENT** declaration is incorrectly written in lowercase, and the **Can** child element is incorrectly listed before its **Pantry** parent element.)
Define elements that do not have a child element as follows: **<!ELEMENT ChildElement (#PCDATA)>** *#PCDATA essentially means each element contains regular text.	**Correct**: <!ELEMENT Price (#PCDATA)> <!ELEMENT Name (#PCDATA)> <!ELEMENT SideItem (#PCDATA)> **Incorrect**: <!ELEMENT Name (**#pcdata**)> <!ELEMENT Age (**#pcdata**)> <!ELEMENT VocalRange (**#pcdata**)> (The **#PCDATA** designation is incorrectly written in lowercase for every child element.)

Schema Basics

You can also define a document's elements and structures by writing an external schema. A schema offers many of the same capabilities as a DTD and improves on DTDs in many ways, but schemas still pose some limitations.

What are the Pros & Cons of Schemas?

Pros	Cons
Easier to edit because they are written in XML.	More dense and redundant, leading to less readability (especially for beginners).
Offers improved validation and ensures elements are unique.	Takes up more file space and may require more technological power/resources to function.
Flexibility in editing across documents.	May appear more like programming than an outline, which likely doesn't appeal as much to writers (a considerably large XML user base).

Activity 3

Create an internal DTD for the following XML file that organizes a collection of Lego helmets by character, pieces, and franchise.

```
<?xml version="1.0" encoding="UTF-8"?>
<LegoCollection>
  <Helmet>
    <Character>Spiderman</Character>
    <Pieces>450</Pieces>
    <Franchise>Marvel</Franchise>
  </Helmet>
  <Helmet>
    <Character>Shadow the Hedgehog</Character>
    <Pieces>500</Pieces>
    <Franchise>Sega</Franchise>
  </Helmet>
</LegoCollection>
```

Activity 3 Potential Answer

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE LegoCollection [
    <!ELEMENT LegoCollection (Helmet)>
    <!ELEMENT Helmet (Character, Pieces, Franchise)>

    <!ELEMENT Character (#PCDATA)>
    <!ELEMENT Pieces (#PCDATA)>
    <!ELEMENT Franchise (#PCDATA)>
]>
```

This answer uses appropriate DTD syntax and demonstrates the hierarchical structure of the XML file.