XML (eXtensible Markup Language) is a metalanguage (a language used to describe other languages) for defining vocabularies (custom markup languages)

XML-based vocabularies (such as XHTML) let you describe documents in a meaningful way

text-based and consist of markup (encoded descriptions of a document’s logical structure) and content (document text not interpreted as markup). Markup is evidenced via tags (angle bracket-delimited syntactic constructs) and each tag has a name. Furthermore, some tags have

attributes (name-value pairs)

The key difference between XML and HTML is that XML invites you to

create your own vocabularies with its own tags and rules, whereas HTML gives you a single precreated vocabulary with its own fixed set of tags and rules.

XML declaration

XML declaration is special markup telling an XML parser that the document is XML, not required, and nothing appears before it

Its format minimized: <?xml version="1.0"?>

nonoptional version attribute identifies the version of the XML specification to which the document conforms

XML supports Unicode

The document’s characters are encoded into bytes for storage or transmission, and the encoding is specified via the XML declaration’s optional encoding attribute. One common encoding is UTF-8 which is a variable-length encoding of the Unicode character set. UTF-8 is a

strict superset of ASCII, which means that pure ASCII text files are also UTF-8 documents.

The final attribute that can appear in the XML declaration is

standalone

.

This optional attribute, which is only relevant with DTDs (discussed later), determines if there are external markup declarations that affect the information passed from an XML processor(a parser) to the application.

Element:

XML declaration>hierarchical(tree) structure of elements

Element is a portion of the document delimited by a start tag (such as <name>) and an end tag(such as </name>), or is an empty-element tag(a standalone tag whose name ends with a forward slash (/), such as <break/>). Start tags and end tags surround content and possibly other markup whereas empty-element tags don’t surround anything

the structure of an XML document is anchored in a root element(the topmost element), XML allows choosing root element

Elements can contain child elements, content, or mixed content(a combination of child elements and content)

Attribute

An XML element’s start tag can contain one or more attributes

Attributes provide additional details about elements and can be optional

Character reference and CDATA section

Certain characters cannot appear literally in the content that appears

between a start tag and an end tag or within an attribute value

Solution: replace the literal character with a character reference, which is a code that represents the character. Character references are classified as numeric character references or character entity references:

+A numeric character reference refers to a character via its Unicode code point and adheres to the format &#nnnn; (not restricted to four positions) or &#xhhhh; (not restricted to four positions), where nnnn provides a decimal representation of the code point and hhhh provides a hexadecimal representation

Although XML mandates that the x in &#xhhhh;be lowercase, it’s flexible in that the leading zero is optional in either format and in allowing you to specify an uppercase or lowercase letter for each h.

+A character entity reference refers to a character via the name of an entity(aliased data) that specifies the desired character as its replacement text. Character entity references are predefined by XML and have the format &name; , in which Name is the entity’s name. XML predefines five character entity references: &lt; (<), &gt; (>), &amp; (&), &apos; ('), and &qout; (").

To save you from tedium and potential errors (from replacing each literal

<(start of tag) and &(start of entity) character with its <lt and &amp predefined character entity reference, XML provides an alternative in the form of a CDATA (character data) section

A

CDATA section is a section of literal HTML or XML markup and content surrounded by the <![CDATA[ prefix and the ]]> suffix. You don’t need to specify predefined character entity references within a CDATA section

Namespace

It’s common to create XML documents that combine features from different

XML languages.

Namespaces are used to prevent name conflicts when elements and other XML language features appear. Without namespaces, an XML parser couldn’t distinguish between same-named elements or other language features that mean different things

A Namespace is a Uniform Resource Identifier (URI)-based container that helps differentiate XML vocabularies by providing a unique context for its contained identifiers. The namespace URI is associated with a “namespace prefix”(an alias for the URI) by specifying, typically in an XML document’s root element, either the *xmlns* attribute by itself (which signifies the default namespace) or the *xmlns*:Prefix attribute (which signifies the namespace identified as prefix), and assigning the URI to this attribute.

When Prefix is specified, the prefix and a colon character are prepended to the name of each element tag that belongs to that namespace

A tag’s attributes don’t need to be prefixed when those attributes belong

to the element. However, a prefix is required for attributes belonging to other namespaces.

When multiple namespaces are involved, it can be convenient to specify one of these namespaces as the default namespace to reduce the tedium in entering namespace prefixes

Comments and Processing Instructions

XML documents can contain comments, which are character sequences beginning with <!-- and ending with -->.

Comments are used to clarify portions of a document. They can appear anywhere after the XML declaration except within tags, cannot be nested, cannot contain a double hyphen (--)

XML also permits processing instructions to be present. A processing instruction is an instruction that’s made available to the application parsing the document. The instruction begins with <? and ends with ?>. The <? prefix is followed by a name known as the target.This name typically identifies the application to which the processing instruction is intended. The rest of the processing instruction contains text in a format appropriate to the application.

Well-Formed Documents

XML is a much stricter language. To make XML documents easier

to parse, XML mandates that XML documents follow certain rules

+All elements must either have start and end tags or consist of empty-element tags.

+Tags must be nested correctly.

+All attribute values must be quoted. Either single quotes (') or double quotes ( ") are permissible (although double quotes are the more commonly specified quotes). It’s an error to omit these quotes.

+Empty elements must be properly formatted

+Be careful with case.

XML is a case-sensitive language in which tags differing in case (such as <author>and <Author>) are considered different. It’s an error to mix start and end tags of different cases,

XML parsers that are aware of namespaces enforce two additional rules:

+Each element and attribute name must not include more

than one colon character.

+No entity names, processing instruction targets, or notation names (discussed later) can contain colons

An XML document that conforms to these rules is well formed. The document has a logical and clean appearance and is much easier to process. XML parsers will only parse well-formed XML documents.

Valid document

Aside from being well-formed, XML document need to be valid

A valid document adheres to constraints. Some XML parsers perform validation, whereas other parsers don’t because validating parsers are harder to write. A parser that performs validation compares an XML document to a grammar document.Any deviation from the grammar document is reported as an error to the application—the XML document isn’t valid. The application may choose to fix the error or reject the XML document. Unlike well-formedness errors, validity errors aren’t necessarily fatal and the parser can continue to parse the XML document.

Grammar documents are written in a special language. Two commonly used grammar languages are Document Type Definition and XML Schema.

+Document Type Definition (DTD)

the oldest grammar language for specifying an XML document’s grammar. DTD grammar documents (known as DTDs) are written in accordance to a strict syntax that states what elements may be present and in what parts of a document, and also what is contained within elements (child elements, content, or mixed content) and what attributes may be specified

A DTD-based validating XML parser requires that a document include a document type declaration identifying the DTD that specifies the document’s grammar before it will validate the document.

You can also declare notations and general and parameter entities within DTDs. A notation is an arbitrary piece of data that typically describes the format of unparsed binary data, and typically has the form <!NOTATION name SYSTEM uri>, where Name identifies the notation and uri identifies some kind of plug-in that can process the data on behalf of the application that’s parsing the XML document

It’s also common to use notations to specify binary data types via media types

General entities are entities referenced from inside an XML document via *general entity references*, syntactic constructs of the form &name;.

General entities: are classified as internal or external. An internal general entity is a general entity whose value is stored in the DTD, and has the form <!ENTITY name value>, where name identifies the entity and value specifies its value

An external general entity is a general entity whose value is stored outside the DTD. The value might be textual data (such as an XML document) or it might be binary data (such as a JPEG image). External general entities are classified as external parsed general entities and external unparsed general entities.

An external parsed general entity references an external file that stores the entity’s textual data, which is subject to being inserted into a document and parsed by a validating parser when a general entity reference is specified in the document, and which has the form <!ENTITY Name SYSTEM uri>, where name identifies the entity and Uri identifies the external file

An external unparsed general entity references an external file that stores the entity’s binary data and has the form <!ENTITY Name SYSTEM Uri NDATA nname>, where Name identifies the entity, Uri locates the external file, and NDATA identifies the notation declaration named nname. The notation typically identifies a plug-in for processing the binary data or the Internet media type of this data.

Parameter entities are entities referenced from inside a DTD via parameter entity references, syntactic constructs of the form *%name*;. They’re useful for eliminating repetitive content from element declarations

Parameter entities are classified as internal or external. An

internal parameter entity is a parameter entity whose value is stored in the DTD and has the form <!ENTITY %name value>, where Name identifies the entity and Values pecifies its value.

An external parameter entity is a parameter entity whose value is stored outside the DTD. It has the form <!ENTITY % name SYSTEM uri>, where name identifies the entity and uri locates the external file

XML schema

XML Schema

is a grammar language for declaring the structure, content,

and semantics(meaning) of an XML document. This language’s grammar documents are known as schemas that are themselves XML documents. Schemas must conform to the XML Schema DTD

XML Schema provides

an object-oriented approach to declaring an XML document’s grammar. This grammar language provides a much larger set of primitive types than DTD’s CDATA and PCDATA types.

XML Schema provides restriction(reducing the set of permitted values through constraints), list(allowing a sequence of values), and union(allowing a choice of values from several types) derivation methods for creating new simple types from these primitive types

Familiar with XML schema: identify all of its elements and attributes-> classify the elements according to XML Schema’s content model, which specifies the types of child elements and text nodes that can be included in an element An element is considered to be empty when the element has no child elements or text nodes, simple when only text nodes are accepted, complex when only child elements are accepted, and Mixed when child elements and text nodes are accepted

XML Schema classifies elements having a simple content model and no attributes as simple types. Furthermore, it classifies elements having a simple content model and attributes, or elements from other content models as complex types. Furthermore, XML Schema classifies attributes as simple types because they only contain text values—attributes don’t have child elements.

XML Schema requires that each element have a name and (unlike DTD)

be associated with a type, which identifies the kind of data stored in the

element.

This attribute element declares an attribute named qty. I chose unsigned Intas this attribute’s type because quantities are nonnegative values. Furthermore, I specified [as the default value for when qty isn’t specified— attribute elements default to declaring optional attributes .

The xmlns attribute identifies as the document’s default namespace. Unprefixed elements and their unprefixed attributes belong to this namespace. The xmlns:xsiattribute associates the conventional xsi

(XML Schema Instance) prefix with the standard namespace. The only item in the document that’s prefixed with xsi:is schemaLocation

.

The schemaLocation attribute is used to locate the schema. This attribute’s value can be multiple pairs of space-separated values, but is specified as a single pair of such values in this example. The first value ( ) identifies the target namespace for the schema, and the second value (recipe.xsd) identifies the location of the schema within

this namespace.

If an XML document declares a namespace (xmlns default or

xmlns:prefix), that namespace must be made available to the schema so that a validating parser can resolve all references to elements and other schema components for that namespace. You also need to mention which namespace the schema describes, and you do so by including the

targetNamespace attribute on the Schema element. For example, suppose your recipe document declares a default XML namespace, as follows