XML Interview Questions and Answers

**What is XML?**XML is the Extensible Markup Language. It improves the functionality of the Web by letting you identify your information in a more accurate, flexible, and adaptable way.   
It is extensible because it is not a fixed format like HTML (which is a single, predefined markup language). Instead, XML is actually a metalanguage—a language for describing other languages—which lets you design your own markup languages for limitless different types of documents. XML can do this because it's written in SGML, the international standard metalanguage for text document markup (ISO 8879).

**What is a markup language?**   
A markup language is a set of words and symbols for describing the identity of pieces of a document (for example ‘this is a paragraph’, ‘this is a heading’, ‘this is a list’, ‘this is the caption of this figure’, etc). Programs can use this with a style sheet to create output for screen, print, audio, video, Braille, etc.   
Some markup languages (e.g. those used in word processors) only describe appearances (‘this is italics’, ‘this is bold’), but this method can only be used for display, and is not normally re-usable for anything else.

**Where should I use XML?**   
Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.   
Despite early attempts, browsers never allowed other SGML, only HTML (although there were plugins), and they allowed it (even encouraged it) to be corrupted or broken, which held development back for over a decade by making it impossible to program for it reliably. XML fixes that by making it compulsory to stick to the rules, and by making the rules much simpler than SGML.   
*But XML is not just for Web pages: in fact it's very rarely used for Web pages on its own because browsers still don't provide reliable support for formatting and transforming it.*

*Common uses for XML include:*Information identification   
because you can define your own markup, you can define meaningful names for all your information items.

Information storage   
because XML is portable and non-proprietary, it can be used to store textual information across any platform. Because it is backed by an international standard, it will remain accessible and processable as a data format.

Information structure   
XML can therefore be used to store and identify any kind of (hierarchical) information structure, especially for long, deep, or complex document sets or data sources, making it ideal for an information-management back-end to serving the Web. This is its most common Web application, with a transformation system to serve it as HTML until such time as browsers are able to handle XML consistently.

Publishing   
The original goal of XML as defined in the quotation at the start of this section. Combining the three previous topics (identity, storage, structure) means it is possible to get all the benefits of robust document management and control (with XML) and publish to the Web (as HTML) as well as to paper (as PDF) and to other formats (e.g. Braille, Audio, etc) from a single source document by using the appropriate style sheets. Messaging and data transfer   
XML is also very heavily used for enclosing or encapsulating information in order to pass it between different computing systems which would otherwise be unable to communicate. By providing a lingua franca for data identity and structure, it provides a common envelope for inter-process communication (messaging).

Web services   
Building on all of these, as well as its use in browsers, machine-processable data can be exchanged between consenting systems, where before it was only comprehensible by humans (HTML). Weather services, e-commerce sites, blog newsfeeds, AJAX sites, and thousands of other data-exchange services use XML for data management and transmission, and the web browser for display and interaction.

**Why is XML such an important development?**

It removes two constraints which were holding back Web developments:  
1. dependence on a single, inflexible document type (HTML) which was being much abused for tasks it was never designed for;  
2. the complexity of full SGML, whose syntax allows many powerful but hard-to-program options.  
XML allows the flexible development of user-defined document types. It provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the Web; and it removes the more complex options of SGML, making it easier to program for.

**Describe the role that XSL can play when dynamically generating HTML pages from a relational database.**   
Even if candidates have never participated in a project involving this type of architecture, they should recognize it as one of the common uses of XML. Querying a database and then formatting the result set so that it can be validated as an XML document allows developers to translate the data into an HTML table using XSLT rules. Consequently, the format of the resulting HTML table can be modified without changing the database query or application code since the document rendering logic is isolated to the XSLT rules.

**What is SGML?**   
*SGML is the Standard Generalized Markup Language (ISO 8879:1986), the international standard for defining descriptions of the structure of different types of electronic document.* There is an SGML FAQ from David Megginson at http://math.albany.edu:8800/hm/sgml/cts-faq.htmlFAQ; and Robin Cover's SGML Web pages are at http://www.oasis-open.org/cover/general.html. For a little light relief, try Joe English's ‘Not the SGML FAQ’ at http://www.flightlab.com/~joe/sgml/faq-not.txtFAQ.   
SGML is very large, powerful, and complex. It has been in heavy industrial and commercial use for nearly two decades, and there is a significant body of expertise and software to go with it.   
*XML is a lightweight cut-down version of SGML which keeps enough of its functionality to make it useful but removes all the optional features which made SGML too complex to program for in a Web environment.*

**Aren't XML, SGML, and HTML all the same thing?**   
Not quite; SGML is the *mother tongue*, and has been used for describing thousands of different document types in many fields of human activity, from transcriptions of ancient Irish manuscripts to the technical documentation for stealth bombers, and from patients' clinical records to musical notation. SGML is very large and complex, however, and probably overkill for most common office desktop applications.   
XML is an abbreviated version of SGML, to make it easier to use over the Web, easier for you to define your own document types, and easier for programmers to write programs to handle them. It omits all the complex and less-used options of SGML in return for the benefits of being easier to write applications for, easier to understand, and more suited to delivery and interoperability over the Web. But it is still SGML, and XML files may still be processed in the same way as any other SGML file (see the question on XML software).   
HTML is just one of many SGML or XML applications—the one most frequently used on the Web.   
Technical readers may find it more useful to think of XML as being SGML-- rather than HTML++.

**Who is responsible for XML?**   
XML is a project of the World Wide Web Consortium (W3C), and the development of the specification is supervised by an XML Working Group. A Special Interest Group of co-opted contributors and experts from various fields contributed comments and reviews by email.   
XML is a public format: it is not a proprietary development of any company, although the membership of the WG and the SIG represented companies as well as research and academic institutions. The v1.0 specification was accepted by the W3C as a Recommendation on Feb 10, 1998.

**Why is XML such an important development?**   
It removes two constraints which were holding back Web developments:   
1. dependence on a single, inflexible document type (HTML) which was being much abused for tasks it was never designed for;  
2. the complexity of full question A.4, SGML, whose syntax allows many powerful but hard-to-program options.   
XML allows the flexible development of user-defined document types. It provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the Web; and it removes the more complex options of SGML, making it easier to program for.

**Give a few examples of types of applications that can benefit from using XML.**There are literally thousands of applications that can benefit from XML technologies. The point of this question is not to have the candidate rattle off a laundry list of projects that they have worked on, but, rather, to allow the candidate to explain the rationale for choosing XML by citing a few real world examples. For instance, one appropriate answer is that XML allows content management systems to store documents independently of their format, which thereby reduces data redundancy. Another answer relates to B2B exchanges or supply chain management systems. In these instances, XML provides a mechanism for multiple companies to exchange data according to an agreed upon set of rules. A third common response involves wireless applications that require WML to render data on hand held devices.

**What is DOM and how does it relate to XML?**The Document Object Model (DOM) is an interface specification maintained by the W3C DOM Workgroup that defines an application independent mechanism to access, parse, or update XML data. In simple terms it is a hierarchical model that allows developers to manipulate XML documents easily Any developer that has worked extensively with XML should be able to discuss the concept and use of DOM objects freely. Additionally, it is not unreasonable to expect advanced candidates to thoroughly understand its internal workings and be able to explain how DOM differs from an event-based interface like SAX.

**What is SOAP and how does it relate to XML?**The Simple Object Access Protocol (SOAP) uses XML to define a protocol for the exchange of information in distributed computing environments. SOAP consists of three components: an envelope, a set of encoding rules, and a convention for representing remote procedure calls. Unless experience with SOAP is a direct requirement for the open position, knowing the specifics of the protocol, or how it can be used in conjunction with HTTP, is not as important as identifying it as a natural application of XML

**Why not just carry on extending HTML?**   
HTML was already overburdened with dozens of interesting but incompatible inventions from different manufacturers, because it provides only one way of describing your information.   
XML allows groups of people or organizations to question C.13, create their own customized markup applications for exchanging information in their domain (music, chemistry, electronics, hill-walking, finance, surfing, petroleum geology, linguistics, cooking, knitting, stellar cartography, history, engineering, rabbit-keeping, question C.19, mathematics, genealogy, etc).   
HTML is now well beyond the limit of its usefulness as a way of describing information, and while it will continue to play an important role for the content it currently represents, many new applications require a more robust and flexible infrastructure.

**Why should I use XML?**Here are a few reasons for using XML (in no particular order). Not all of these will apply to your own requirements, and you may have additional reasons not mentioned here (if so, please let the editor of the FAQ know!).   
\* XML can be used to describe and identify information accurately and unambiguously, in a way that computers can be programmed to ‘understand’ (well, at least manipulate as if they could understand).  
\* XML allows documents which are all the same type to be created consistently and without structural errors, because it provides a standardised way of describing, controlling, or allowing/disallowing particular types of document structure. [Note that this has absolutely nothing whatever to do with formatting, appearance, or the actual text content of your documents, only the structure of them.]  
\* XML provides a robust and durable format for information storage and transmission. Robust because it is based on a proven standard, and can thus be tested and verified; durable because it uses plain-text file formats which will outlast proprietary binary ones.  
\* XML provides a common syntax for messaging systems for the exchange of information between applications. Previously, each messaging system had its own format and all were different, which made inter-system messaging unnecessarily messy, complex, and expensive. If everyone uses the same syntax it makes writing these systems much faster and more reliable.  
\* XML is free. Not just free of charge (free as in beer) but free of legal encumbrances (free as in speech). It doesn't belong to anyone, so it can't be hijacked or pirated. And you don't have to pay a fee to use it (you can of course choose to use commercial software to deal with it, for lots of good reasons, but you don't pay for XML itself).  
\* XML information can be manipulated programmatically (under machine control), so XML documents can be pieced together from disparate sources, or taken apart and re-used in different ways. They can be converted into almost any other format with no loss of information.  
\* XML lets you separate form from content. Your XML file contains your document information (text, data) and identifies its structure: your formatting and other processing needs are identified separately in a stylesheet or processing system. The two are combined at output time to apply the required formatting to the text or data identified by its structure (location, position, rank, order, or whatever).

**Can you walk us through the steps necessary to parse XML documents?** Superficially, this is a fairly basic question. However, the point is not to determine whether candidates understand the concept of a parser but rather have them walk through the process of parsing XML documents step-by-step. Determining whether a non-validating or validating parser is needed, choosing the appropriate parser, and handling errors are all important aspects to this process that should be included in the candidate's response.

**Give some examples of XML DTDs or schemas that you have worked with.**   
Although XML does not require data to be validated against a DTD, many of the benefits of using the technology are derived from being able to validate XML documents against business or technical architecture rules. Polling for the list of DTDs that developers have worked with provides insight to their general exposure to the technology. The ideal candidate will have knowledge of several of the commonly used DTDs such as FpML, DocBook, HRML, and RDF, as well as experience designing a custom DTD for a particular project where no standard existed.

**Using XSLT, how would you extract a specific attribute from an element in an XML document?**   
Successful candidates should recognize this as one of the most basic applications of XSLT. If they are not able to construct a reply similar to the example below, they should at least be able to identify the components necessary for this operation: xsl:template to match the appropriate XML element, xsl:value-of to select the attribute value, and the optional xsl:apply-templates to continue processing the document.  
  
Extract Attributes from XML Data   
Example 1.  
<xsl:template match="element-name">  
Attribute Value:  
<xsl:value-of select="@attribute"/>  
<xsl:apply-templates/>  
</xsl:template>

**When constructing an XML DTD, how do you create an external entity reference in an attribute value?**   
Every interview session should have at least one trick question. Although possible when using SGML, XML DTDs don't support defining external entity references in attribute values. It's more important for the candidate to respond to this question in a logical way than than the candidate know the somewhat obscure answer.

**How would you build a search engine for large volumes of XML data?**   
The way candidates answer this question may provide insight into their view of XML data. For those who view XML primarily as a way to denote structure for text files, a common answer is to build a full-text search and handle the data similarly to the way Internet portals handle HTML pages. Others consider XML as a standard way of transferring structured data between disparate systems. These candidates often describe some scheme of importing XML into a relational or object database and relying on the database's engine for searching. Lastly, candidates that have worked with vendors specializing in this area often say that the best way the handle this situation is to use a third party software package optimized for XML data.

**What is the difference between XML and C or C++ or Java?**C and C++ (and other languages like FORTRAN, or Pascal, or Visual Basic, or Java or hundreds more) are programming languages with which you specify calculations, actions, and decisions to be carried out in order:

mod curconfig[if left(date,6) = "01-Apr",   
t.put "April googlel!",   
f.put days('31102005','DDMMYYYY') -  
days(sdate,'DDMMYYYY')  
" more shopping days to Samhain"];

XML is a markup specification language with which you can design ways of describing information (text or data), usually for storage, transmission, or processing by a program. It says nothing about what you should do with the data (although your choice of element names may hint at what they are for):

<part num="DA42" models="LS AR DF HG KJ"   
update="2001-11-22">  
<name>Camshaft end bearing retention circlip</name>  
<image drawing="RR98-dh37" type="SVG" x="476"   
y="226"/> <maker id="RQ778">Ringtown Fasteners Ltd</maker>  
<notes>Angle-nosed insertion tool <tool   
id="GH25"/> is required for the removal   
and replacement of this part.</notes>  
</part>

On its own, an SGML or XML file (including HTML) doesn't do anything. It's a data format which just sits there until you run a program which does something with it.

**Does XML replace HTML?**   
No. XML itself does not replace HTML. Instead, it provides an alternative which allows you to define your own set of markup elements. HTML is expected to remain in common use for some time to come, and the current version of HTML is in XML syntax. XML is designed to make the writing of DTDs much simpler than with full SGML. (See the question on DTDs for what one is and why you might want one.)

**Do I have to know HTML or SGML before I learn XML?**   
No, although it's useful because a lot of XML terminology and practice derives from two decades' experience of SGML.   
Be aware that ‘knowing HTML’ is not the same as ‘understanding SGML’. Although HTML was written as an SGML application, browsers ignore most of it (which is why so many useful things don't work), so just because something is done a certain way in HTML browsers does not mean it's correct, least of all in XML.

**What does an XML document actually look like (inside)?**   
The basic structure of XML is similar to other applications of SGML, including HTML. The basic components can be seen in the following examples. An XML document starts with a Prolog:   
1. The XML Declaration  
  
which specifies that this is an XML document;  
2. Optionally a Document Type Declaration  
  
which identifies the type of document and says where the Document Type Description (DTD) is stored;  
The Prolog is followed by the document instance:  
1. A root element, which is the outermost (top level) element (start-tag plus end-tag) which encloses everything else: in the examples below the root elements are conversation and titlepage;  
2. A structured mix of descriptive or prescriptive elements enclosing the character data content (text), and optionally any attributes (‘name=value’ pairs) inside some start-tags.  
XML documents can be very simple, with straightforward nested markup of your own design:  
<?xml version="1.0" standalone="yes"?>  
<conversation><br>  
<greeting>Hello, world!</greeting>  
<response>Stop the planet, I want to get   
off!</response>  
</conversation>  
Or they can be more complicated, with a Schema or question C.11, Document Type Description (DTD) or internal subset (local DTD changes in [square brackets]), and an arbitrarily complex nested structure:  
<?xml version="1.0" encoding="iso-8859-1"?>  
<!DOCTYPE titlepage   
SYSTEM "http://www.google.bar/dtds/typo.dtd"   
[<!ENTITY % active.links "INCLUDE">]>  
<titlepage id="BG12273624">  
<white-space type="vertical" amount="36"/>  
<title font="Baskerville" alignment="centered"  
size="24/30">Hello, world!</title>  
<white-space type="vertical" amount="12"/>  
<!-- In some copies the following   
decoration is hand-colored, presumably   
by the author -->  
<image location="http://www.google.bar/fleuron.eps"   
type="URI" alignment="centered"/>  
<white-space type="vertical" amount="24"/>  
<author font="Baskerville" size="18/22"   
style="italic">Vitam capias</author>  
<white-space type="vertical" role="filler"/>  
</titlepage>  
  
Or they can be anywhere between: a lot will depend on how you want to define your document type (or whose you use) and what it will be used for. Database-generated or program-generated XML documents used in e-commerce is usually unformatted (not for human reading) and may use very long names or values, with multiple redundancy and sometimes no character data content at all, just values in attributes:  
<?xml version="1.0"?> <ORDER-UPDATE AUTHMD5="4baf7d7cff5faa3ce67acf66ccda8248"  
ORDER-UPDATE-ISSUE="193E22C2-EAF3-11D9-9736-CAFC705A30B3"  
ORDER-UPDATE-DATE="2005-07-01T15:34:22.46" ORDER-UPDATE-DESTINATION="6B197E02-EAF3-11D9-85D5-997710D9978F"  
ORDER-UPDATE-ORDERNO="8316ADEA-EAF3-11D9-9955-D289ECBC99F3">  
<ORDER-UPDATE-DELTA-MODIFICATION-DETAIL ORDER-UPDATE-ID="BAC352437484">  
<ORDER-UPDATE-DELTA-MODIFICATION-VALUE ORDER-UPDATE-ITEM="56"  
ORDER-UPDATE-QUANTITY="2000"/>  
</ORDER-UPDATE-DELTA-MODIFICATION-DETAIL>  
</ORDER-UPDATE>

**How does XML handle white-space in my documents?**All white-space, including linebreaks, TAB characters, and normal spaces, even between ‘structural’ elements where no text can ever appear, is passed by the parser unchanged to the application (browser, formatter, viewer, converter, etc), identifying the context in which the white-space was found (element content, data content, or mixed content, if this information is available to the parser, eg from a DTD or Schema). This means it is the application's responsibility to decide what to do with such space, not the parser's:  
\* insignificant white-space between structural elements (space which occurs where only element content is allowed, ie between other elements, where text data never occurs) will get passed to the application (in SGML this white-space gets suppressed, which is why you can put all that extra space in HTML documents and not worry about it)  
\* significant white-space (space which occurs within elements which can contain text and markup mixed together, usually mixed content or PCDATA) will still get passed to the application exactly as under SGML. It is the application's responsibility to handle it correctly.  
The parser must inform the application that white-space has occurred in element content, if it can detect it. (Users of SGML will recognize that this information is not in the ESIS, but it is in the Grove.)  
  
<chapter>   
<title>   
My title for  
Chapter 1.   
</title>   
<para>   
text   
</para>   
</chapter>  
  
In the example above, the application will receive all the pretty-printing linebreaks, TABs, and spaces between the elements as well as those embedded in the chapter title. It is the function of the application, not the parser, to decide which type of white-space to discard and which to retain. Many XML applications have configurable options to allow programmers or users to control how such white-space is handled.

**Which parts of an XML document are case-sensitive?**   
All of it, both markup and text. This is significantly different from HTML and most other SGML applications. It was done to allow markup in non-Latin-alphabet languages, and to obviate problems with case-folding in writing systems which are caseless.  
\* Element type names are case-sensitive: you must follow whatever combination of upper- or lower-case you use to define them (either by first usage or in a DTD or Schema). So you can't say <BODY>…</body>: upper- and lower-case must match; thus <Img/>, <IMG/>, and <img/> are three different element types;  
\* For well-formed XML documents with no DTD, the first occurrence of an element type name defines the casing;  
\* Attribute names are also case-sensitive, for example the two width attributes in <PIC width="7in"/> and <PIC WIDTH="6in"/> (if they occurred in the same file) are separate attributes, because of the different case of width and WIDTH;  
\* Attribute values are also case-sensitive. CDATA values (eg Url="MyFile.SGML") always have been, but NAME types (ID and IDREF attributes, and token list attributes) are now case-sensitive as well;  
\* All general and parameter entity names (eg Á), and your data content (text), are case-sensitive as always.

**How can I make my existing HTML files work in XML?**Either convert them to conform to some new document type (with or without a DTD or Schema) and write a stylesheet to go with them; or edit them to conform to XHTML.   
It is necessary to convert existing HTML files because XML does not permit end-tag minimisation (missing   
, etc), unquoted attribute values, and a number of other SGML shortcuts which have been normal in most HTML DTDs. However, many HTML authoring tools already produce almost (but not quite) well-formed XML.   
You may be able to convert HTML to XHTML using the Dave Raggett's HTML Tidy program, which can clean up some of the formatting mess left behind by inadequate HTML editors, and even separate out some of the formatting to a stylesheet, but there is usually still some hand-editing to do.

**Is there an XML version of HTML?**Yes, the W3C recommends using XHTML which is ‘a reformulation of HTML 4 in XML 1.0’. This specification defines HTML as an XML application, and provides three DTDs corresponding to the ones defined by HTML 4.\* (Strict, Transitional, and Frameset).   
The semantics of the elements and their attributes are as defined in the W3C Recommendation for HTML 4. These semantics provide the foundation for future extensibility of XHTML. Compatibility with existing HTML browsers is possible by following a small set of guidelines (see the W3C site).

**If XML is just a subset of SGML, can I use XML files directly with existing SGML tools?**   
Yes, provided you use up-to-date SGML software which knows about the WebSGML Adaptations TC to ISO 8879 (the features needed to support XML, such as the variant form for EMPTY elements; some aspects of the SGML Declaration such as NAMECASE GENERAL NO; multiple attribute token list declarations, etc).   
An alternative is to use an SGML DTD to let you create a fully-normalised SGML file, but one which does not use empty elements; and then remove the DocType Declaration so it becomes a well-formed DTDless XML file. Most SGML tools now handle XML files well, and provide an option switch between the two standards.

**Can XML use non-Latin characters?**Yes, the XML Specification explicitly says XML uses ISO 10646, the international standard character repertoire which covers most known languages. Unicode is an identical repertoire, and the two standards track each other. The spec says (2.2): ‘All XML processors must accept the UTF-8 and UTF-16 encodings of ISO 10646…’. There is a Unicode FAQ at http://www.unicode.org/faq/FAQ.   
UTF-8 is an encoding of Unicode into 8-bit characters: the first 128 are the same as ASCII, and higher-order characters are used to encode anything else from Unicode into sequences of between 2 and 6 bytes. UTF-8 in its single-octet form is therefore the same as ISO 646 IRV (ASCII), so you can continue to use ASCII for English or other languages using the Latin alphabet without diacritics. Note that UTF-8 is incompatible with ISO 8859-1 (ISO Latin-1) after code point 127 decimal (the end of ASCII).   
UTF-16 is an encoding of Unicode into 16-bit characters, which lets it represent 16 planes. UTF-16 is incompatible with ASCII because it uses two 8-bit bytes per character (four bytes above U+FFFF).

**What's a Document Type Definition (DTD) and where do I get one?**   
A DTD is a description in XML Declaration Syntax of a particular type or class of document. It sets out what names are to be used for the different types of element, where they may occur, and how they all fit together. (A question C.16, Schema does the same thing in XML Document Syntax, and allows more extensive data-checking.)   
For example, if you want a document type to be able to describe Lists which contain Items, the relevant part of your DTD might contain something like this:   
<!ELEMENT List (Item)+>   
<!ELEMENT Item (#PCDATA)>  
  
This defines a list as an element type containing one or more items (that's the plus sign); and it defines items as element types containing just plain text (Parsed Character Data or PCDATA). Validators read the DTD before they read your document so that they can identify where every element type ought to come and how each relates to the other, so that applications which need to know this in advance (most editors, search engines, navigators, and databases) can set themselves up correctly. The example above lets you create lists like:  
  
<List>  
<Item>Chocolate</Item>  
<Item>Music</Item>  
<Item>Surfingv</Item>  
</List>   
  
(The indentation in the example is just for legibility while editing: it is not required by XML.)   
A DTD provides applications with advance notice of what names and structures can be used in a particular document type. Using a DTD and a validating editor means you can be certain that all documents of that particular type will be constructed and named in a consistent and conformant manner.   
DTDs are not required for processing the tip in question Bwell-formed documents, but they are needed if you want to take advantage of XML's special attribute types like the built-in ID/IDREF cross-reference mechanism; or the use of default attribute values; or references to external non-XML files (‘Notations’); or if you simply want a check on document validity before processing.   
There are thousands of DTDs already in existence in all kinds of areas (see the SGML/XML Web pages for pointers). Many of them can be downloaded and used freely; or you can write your own (see the question on creating your own DTD. Old SGML DTDs need to be converted to XML for use with XML systems: read the question on converting SGML DTDs to XML, but most popular SGML DTDs are already available in XML form.   
The alternatives to a DTD are various forms of question C.16, Schema. These provide more extensive validation features than DTDs, including character data content validation.

No, it lets you make up names for your own element types. If you think tags and elements are the same thing you are already in considerable trouble: read the rest of this question carefully.

**How do I create my own document type?**   
Document types usually need a formal description, either a DTD or a Schema. Whilst it is possible to process well-formed XML documents without any such description, trying to create them without one is asking for trouble. A DTD or Schema is used with an XML editor or API interface to guide and control the construction of the document, making sure the right elements go in the right places.   
Creating your own document type therefore begins with an analysis of the class of documents you want to describe: reports, invoices, letters, configuration files, credit-card verification requests, or whatever. Once you have the structure correct, you write code to express this formally, using DTD or Schema syntax.

**How do I write my own DTD?**You need to use the XML Declaration Syntax (very simple: declaration keywords begin with   
<!ELEMENT Shopping-List (Item)+>  
<!ELEMENT Item (#PCDATA)>  
  
It says that there shall be an element called Shopping-List and that it shall contain elements called Item: there must be at least one Item (that's the plus sign) but there may be more than one. It also says that the Item element may contain only parsed character data (PCDATA, ie text: no further markup).   
Because there is no other element which contains Shopping-List, that element is assumed to be the ‘root’ element, which encloses everything else in the document. You can now use it to create an XML file: give your editor the declarations:   
<?xml version="1.0"?>   
<!DOCTYPE Shopping-List SYSTEM "shoplist.dtd">   
  
(assuming you put the DTD in that file). Now your editor will let you create files according to the pattern:   
<Shopping-List>  
  
<Item>Chocolate</Item>  
<Item>Sugar</Item>  
<Item>Butter</Item>  
</Shopping-List>  
  
It is possible to develop complex and powerful DTDs of great subtlety, but for any significant use you should learn more about document systems analysis and document type design. See for example Developing SGML DTDs: From Text to Model to Markup (Maler and el Andaloussi, 1995): this was written for SGML but perhaps 95% of it applies to XML as well, as XML is much simpler than full SGML—see the list of restrictions which shows what has been cut out.   
  
Warning  
Incidentally, a DTD file never has a DOCTYPE Declaration in it: that only occurs in an XML document instance (it's what references the DTD). And a DTD file also never has an XML Declaration at the top either. Unfortunately there is still software around which inserts one or both of these.

**Can a root element type be explicitly declared in the DTD?**No. This is done in the document's Document Type Declaration, not in the DTD.

**I keep hearing about alternatives to DTDs. What's a Schema?**   
The W3C XML Schema recommendation provides a means of specifying formal data typing and validation of element content in terms of data types, so that document type designers can provide criteria for checking the data content of elements as well as the markup itself. Schemas are written in XML Document Syntax, like XML documents are, avoiding the need for processing software to be able to read XML Declaration Syntax (used for DTDs).   
There is a separate Schema FAQ at http://www.schemavalid.comFAQ. The term ‘vocabulary’ is sometimes used to refer to DTDs and Schemas together. Schemas are aimed at e-commerce, data control, and database-style applications where character data content requires validation and where stricter data control is needed than is possible with DTDs; or where strong data typing is required. They are usually unnecessary for traditional text document publishing applications.   
Unlike DTDs, Schemas cannot be specified in an XML Document Type Declaration. They can be specified in a Namespace, where Schema-aware software should pick it up, but this is optional:  
  
<invoice id="abc123"  
xmlns="http://example.org/ns/books/"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://acme.wilycoyote.org/xsd/invoice.xsd">  
...  
</invoice>   
  
More commonly, you specify the Schema in your processing software, which should record separately which Schema is used by which XML document instance.   
In contrast to the complexity of the W3C Schema model, Relax NG is a lightweight, easy-to-use XML schema language devised by James Clark (see http://relaxng.org/) with development hosted by OASIS. It allows similar richness of expression and the use of XML as its syntax, but it provides an additional, simplified, syntax which is easier to use for those accustomed to DTDs.

**How do I get XML into or out of a database?**   
Ask your database manufacturer: they all provide XML import and export modules to connect XML applications with databases. In some trivial cases there will be a 1:1 match between field names in the database table and element type names in the XML Schema or DTD, but in most cases some programming will be required to establish the desired match. This can usually be stored as a procedure so that subsequent uses are simply commands or calls with the relevant parameters.   
In less trivial, but still simple, cases, you could export by writing a report routine that formats the output as an XML document, and you could import by writing an XSLT transformation that formatted the XML data as a load file.

**Can I encode mathematics using XML ?**  
Yes, if the document type you use provides for math, and your users' browsers are capable of rendering it. The mathematics-using community has developed the MathML Recommendation at the W3C, which is a native XML application suitable for embedding in other DTDs and Schemas.   
It is also possible to make XML fragments from other DTDs, such as ISO 12083 Math, or OpenMath, or one of your own making. Browsers which display math embedded in SGML existed for many years (eg DynaText, Panorama, Multidoc Pro), and mainstream browsers are now rendering MathML. David Carlisle has produced a set of stylesheets for rendering MathML in browsers. It is also possible to use XSLT to convert XML math markup to LATEX for print (PDF) rendering, or to use XSL:FO.   
Please note that XML is not itself a programming language, so concepts such as arithmetic and if-statements (if-then-else logic) are not meaningful in XML documents.

**How will XML affect my document links?**The linking abilities of XML systems are potentially much more powerful than those of HTML, so you'll be able to do much more with them. Existing href-style links will remain usable, but the new linking technology is based on the lessons learned in the development of other standards involving hypertext, such as TEI and HyTime, which let you manage bidirectional and multi-way links, as well as links to a whole element or span of text (within your own or other documents) rather than to a single point. These features have been available to SGML users for many years, so there is considerable experience and expertise available in using them. Currently only Mozilla Firefox implements XLink.   
The XML Linking Specification (XLink) and the XML Extended Pointer Specification (XPointer) documents contain the details. An XLink can be either a URI or a TEI-style Extended Pointer (XPointer), or both. A URI on its own is assumed to be a resource; if an XPointer follows it, it is assumed to be a sub-resource of that URI; an XPointer on its own is assumed to apply to the current document (all exactly as with HTML).   
An XLink may use one of #, ?, or |. The # and ? mean the same as in HTML applications; the | means the sub-resource can be found by applying the link to the resource, but the method of doing this is left to the application. An XPointer can only follow a #.   
The TEI Extended Pointer Notation (EPN) is much more powerful than the fragment address on the end of some URIs, as it allows you to specify the location of a link end using the structure of the document as well as (or in addition to) known, fixed points like IDs. For example, the linked second occurrence of the word ‘XPointer’ two paragraphs back could be referred to with the URI (shown here with linebreaks and spaces for clarity: in practice it would of course be all one long string):  
  
http://xml.silmaril.ie/faq.xml#ID(hypertext)  
.child(1,#element,'answer')  
.child(2,#element,'para')  
.child(1,#element,'link')  
This means the first link element within the second paragraph within the answer in the element whose ID is hypertext (this question). Count the objects from the start of this question (which has the ID hypertext) in the XML source:   
1. the first child object is the element containing the question ();  
2. the second child object is the answer (the element);  
3. within this element go to the second paragraph;  
4. find the first link element.  
Eve Maler explained the relationship of XLink and XPointer as follows:  
XLink governs how you insert links into your XML document, where the link might point to anything (eg a GIF file); XPointer governs the fragment identifier that can go on a URL when you're linking to an XML document, from anywhere (eg from an HTML file).  
[Or indeed from an XML file, a URI in a mail message, etc…Ed.]  
David Megginson has produced an xpointer function for Emacs/psgml which will deduce an XPointer for any location in an XML document. XML Spy has a similar function.

**How does XML handle metadata?**Because XML lets you define your own markup languages, you can make full use of the extended hypertext features of XML (see the question on Links) to store or link to metadata in any format (eg using ISO 11179, as a Topic Maps Published Subject, with Dublin Core, Warwick Framework, or with Resource Description Framework (RDF), or even Platform for Internet Content Selection (PICS)).   
There are no predefined elements in XML, because it is an architecture, not an application, so it is not part of XML's job to specify how or if authors should or should not implement metadata. You are therefore free to use any suitable method. Browser makers may also have their own architectural recommendations or methods to propose.

**Can I use JavaScript, ActiveX, etc in XML files?**   
This will depend on what facilities your users' browsers implement. XML is about describing information; scripting languages and languages for embedded functionality are software which enables the information to be manipulated at the user's end, so these languages do not normally have any place in an XML file itself, but in stylesheets like XSL and CSS where they can be added to generated HTML.   
XML itself provides a way to define the markup needed to implement scripting languages: as a neutral standard it neither encourages not discourages their use, and does not favour one language over another, so it is possible to use XML markup to store the program code, from where it can be retrieved by (for example) XSLT and re-expressed in a HTML script element.   
Server-side script embedding, like PHP or ASP, can be used with the relevant server to modify the XML code on the fly, as the document is served, just as they can with HTML. Authors should be aware, however, that embedding server-side scripting may mean the file as stored is not valid XML: it only becomes valid when processed and served, so care must be taken when using validating editors or other software to handle or manage such files. A better solution may be to use an XML serving solution like Cocoon, AxKit, or PropelX.

**Can I use Java to create or manage XML files?**   
Yes, any programming language can be used to output data from any source in XML format. There is a growing number of front-ends and back-ends for programming environments and data management environments to automate this. Java is just the most popular one at the moment.   
There is a large body of middleware (APIs) written in Java and other languages for managing data either in XML or with XML input or output.

**How do I execute or run an XML file?**   
You can't and you don't. XML itself is not a programming language, so XML files don't ‘run’ or ‘execute’. XML is a markup specification language and XML files are just data: they sit there until you run a program which displays them (like a browser) or does some work with them (like a converter which writes the data in another format, or a database which reads the data), or modifies them (like an editor).   
If you want to view or display an XML file, open it with an XML editor or an question B.3, XML browser.   
The water is muddied by XSL (both XSLT and XSL:FO) which use XML syntax to implement a declarative programming language. In these cases it is arguable that you can ‘execute’ XML code, by running a processing application like Saxon, which compiles the directives specified in XSLT files into Java bytecode to process XML.

**How do I control formatting and appearance?**In HTML, default styling was built into the browsers because the tagset of HTML was predefined and hardwired into browsers. In XML, where you can define your own tagset, browsers cannot possibly be expected to guess or know in advance what names you are going to use and what they will mean, so you need a stylesheet if you want to display formatted text.   
Browsers which read XML will accept and use a CSS stylesheet at a minimum, but you can also use the more powerful XSLT stylesheet language to transform your XML into HTML—which browsers, of course, already know how to display (and that HTML can still use a CSS stylesheet). This way you get all the document management benefits of using XML, but you don't have to worry about your readers needing XML smarts in their browsers.

**How do I use graphics in XML?**Graphics have traditionally just been links which happen to have a picture file at the end rather than another piece of text. They can therefore be implemented in any way supported by the XLink and XPointer specifications (see question C.18, ‘How will XML affect my document links?’), including using similar syntax to existing HTML images. They can also be referenced using XML's built-in NOTATION and ENTITY mechanism in a similar way to standard SGML, as external unparsed entities.   
However, the SVG specification (see the tip below, by Peter Murray-Rust) lets you use XML markup to draw vector graphics objects directly in your XML file. This provides enormous power for the inclusion of portable graphics, especially interactive or animated sequences, and it is now slowly becoming supported in browsers.   
The XML linking specifications for external images give you much better control over the traversal and activation of links, so an author can specify, for example, whether or not to have an image appear when the page is loaded, or on a click from the user, or in a separate window, without having to resort to scripting.   
XML itself doesn't predicate or restrict graphic file formats: GIF, JPG, TIFF, PNG, CGM, EPS, and SVG at a minimum would seem to make sense; however, vector formats (EPS, SVG) are normally essential for non-photographic images (diagrams).   
You cannot embed a raw binary graphics file (or any other binary [non-text] data) directly into an XML file because any bytes happening to resemble markup would get misinterpreted: you must refer to it by linking (see below). It is, however, possible to include a text-encoded transformation of a binary file as a CDATA Marked Section, using something like UUencode with the markup characters ], & and > removed from the map so that they could not occur as an erroneous CDATA termination sequence and be misinterpreted. You could even use simple hexadecimal encoding as used in PostScript. For vector graphics, however, the solution is to use SVG (see the tip below, by Peter Murray-Rust).   
Sound files are binary objects in the same way that external graphics are, so they can only be referenced externally (using the same techniques as for graphics). Music files written in MusiXML or an XML variant of SMDL could however be embedded in the same way as for SVG.   
The point about using entities to manage your graphics is that you can keep the list of entity declarations separate from the rest of the document, so you can re-use the names if an image is needed more than once, but only store the physical file specification in a single place. This is available only when using a DTD, not a Schema.

**How do I include one XML file in another?**   
This works exactly the same as for SGML. First you declare the entity you want to include, and then you reference it by name:  
<?xml version="1.0"?>  
<!DOCTYPE novel SYSTEM "/dtd/novel.dtd" [  
<!ENTITY chap1 SYSTEM "mydocs/chapter1.xml">  
<!ENTITY chap2 SYSTEM "mydocs/chapter2.xml">  
<!ENTITY chap3 SYSTEM "mydocs/chapter3.xml">  
<!ENTITY chap4 SYSTEM "mydocs/chapter4.xml">  
<!ENTITY chap5 SYSTEM "mydocs/chapter5.xml">  
]>  
<novel>  
<header>  
...blah blah...  
</header>  
&chap1;   
&chap2;  
&chap3;  
&chap4;   
&chap5;   
</novel>  
  
The difference between this method and the one used for including a DTD fragment (see question D.15, ‘How do I include one DTD (or fragment) in another?’) is that this uses an external general (file) entity which is referenced in the same way as for a character entity (with an ampersand).   
The one thing to make sure of is that the included file must not have an XML or DOCTYPE Declaration on it. If you've been using one for editing the fragment, remove it before using the file in this way. Yes, this is a pain in the butt, but if you have lots of inclusions like this, write a script to strip off the declaration (and paste it back on again for editing).

**What is parsing and how do I do it in XML**   
Parsing is the act of splitting up information into its component parts (schools used to teach this in language classes until the teaching profession collectively caught the anti-grammar disease).   
‘Mary feeds Spot’ parses as  
1. Subject = Mary, proper noun, nominative case  
2. Verb = feeds, transitive, third person singular, present tense  
3. Object = Spot, proper noun, accusative case  
In computing, a parser is a program (or a piece of code or API that you can reference inside your own programs) which analyses files to identify the component parts. All applications that read input have a parser of some kind, otherwise they'd never be able to figure out what the information means. Microsoft Word contains a parser which runs when you open a .doc file and checks that it can identify all the hidden codes. Give it a corrupted file and you'll get an error message.   
XML applications are just the same: they contain a parser which reads XML and identifies the function of each the pieces of the document, and it then makes that information available in memory to the rest of the program.   
While reading an XML file, a parser checks the syntax (pointy brackets, matching quotes, etc) for well-formedness, and reports any violations (reportable errors). The XML Specification lists what these are.   
Validation is another stage beyond parsing. As the component parts of the program are identified, a validating parser can compare them with the pattern laid down by a DTD or a Schema, to check that they conform. In the process, default values and datatypes (if specified) can be added to the in-memory result of the validation that the validating parser gives to the application.  
  
<person corpid="abc123" birth="1960-02-31" gender="female"> <name> <forename>Judy</forename> <surname>O'Grady</surname> </name> </person>   
The example above parses as: 1. Element person identified with Attribute corpid containing abc123 and Attribute birth containing 1960-02-31 and Attribute gender containing female containing ...  
2. Element name containing ...  
3. Element forename containing text ‘Judy’ followed by ...  
4. Element surname containing text ‘O'Grady’  
(and lots of other stuff too).  
As well as built-in parsers, there are also stand-alone parser-validators, which read an XML file and tell you if they find an error (like missing angle-brackets or quotes, or misplaced markup). This is essential for testing files in isolation before doing something else with them, especially if they have been created by hand without an XML editor, or by an API which may be too deeply embedded elsewhere to allow easy testing.

**When should I use a CDATA Marked Section?**You should almost never need to use CDATA Sections. The CDATA mechanism was designed to let an author quote fragments of text containing markup characters (the open-angle-bracket and the ampersand), for example when documenting XML (this FAQ uses CDATA Sections quite a lot, for obvious reasons). A CDATA Section turns off markup recognition for the duration of the section (it gets turned on again only by the closing sequence of double end-square-brackets and a close-angle-bracket).   
Consequently, nothing in a CDATA section can ever be recognised as anything to do with markup: it's just a string of opaque characters, and if you use an XML transformation language like XSLT, any markup characters in it will get turned into their character entity equivalent.   
If you try, for example, to use:  
some text with <![CDATA[markup]]> in it.  
in the expectation that the embedded markup would remain untouched, it won't: it will just output  
some text with <em>markup</em> in it.  
In other words, CDATA Sections cannot preserve the embedded markup as markup. Normally this is exactly what you want because this technique was designed to let people do things like write documentation about markup. It was not designed to allow the passing of little chunks of (possibly invalid) unparsed HTML embedded inside your own XML through to a subsequent process—because that would risk invalidating the output.   
As a result you cannot expect to keep markup untouched simply because it looked as if it was safely ‘hidden’ inside a CDATA section: it can't be used as a magic shield to preserve HTML markup for future use as markup, only as characters.

**How can I handle embedded HTML in my XML**   
Apart from using CDATA Sections, there are two common occasions when people want to handle embedded HTML inside an XML element:  
1. when they have received (possibly poorly-designed) XML from somewhere else which they must find a way to handle;  
2. when they have an application which has been explicitly designed to store a string of characters containing < and & character entity references with the objective of turning them back into markup in a later process (eg FreeMind, Atom).  
Generally, you want to avoid this kind of trick, as it usually indicates that the document structure and design has been insufficiently thought out. However, there are occasions when it becomes unavoidable, so if you really need or want to use embedded HTML markup inside XML, and have it processable later as markup, there are a couple of techniques you may be able to use:  
\* Provide templates for the handling of that markup in your XSLT transformation or whatever software you use which simply replicates what was there, eg  
<xsl:template match="b">  
<b>  
<xsl:apply-templates/>  
</b>  
</xsl:template/>  
\* Use XSLT's ‘deep copy’ instruction, which outputs nested well-formed markup verbatim, eg  
<xsl:template match="ol">  
<xsl:copy-of select="."/>  
</xsl:template/>  
\* As a last resort, use the disable-output-escaping attribute on the xsl:text element of XSL[T] which is available in some processors, eg  
<xsl:text disable-output-escaping="yes"><![CDATA[<b>Now!</b>]]></xsl:text>  
\* Some processors (eg JX) are now providing their own equivalents for disabling output escaping. Their proponents claim it is ‘highly desirable’ or ‘what most people want’, but it still needs to be treated with care to prevent unwanted (possibly dangerous) arbitrary code from being passed untouched through your system. It also adds another dependency to your software.  
For more details of using these techniques in XSL[T], see the relevant question in the XSL FAQ.

**What are the special characters in XML ?**  
For normal text (not markup), there are no special characters: just make sure your document refers to the correct encoding scheme for the language and/or writing system you want to use, and that your computer correctly stores the file using that encoding scheme. See the question on non-Latin characters for a longer explanation.   
If your keyboard will not allow you to type the characters you want, or if you want to use characters outside the limits of the encoding scheme you have chosen, you can use a symbolic notation called ‘entity referencing’. Entity references can either be numeric, using the decimal or hexadecimal Unicode code point for the character (eg if your keyboard has no Euro symbol (€) you can type €); or they can be character, using an established name which you declare in your DTD (eg ) and then use as € in your document. If you are using a Schema, you must use the numeric form for all except the five below because Schemas have no way to make character entity declarations.   
If you use XML with no DTD, then these five character entities are assumed to be predeclared, and you can use them without declaring them:   
&lt;   
The less-than character (<) starts element markup (the first character of a start-tag or an end-tag).  
&amp;  
  
The ampersand character (>) starts entity markup (the first character of a character entity reference).  
&gt;  
The greater-than character (>) ends a start-tag or an end-tag.  
&quot;  
The double-quote character (") can be symbolised with this character entity reference when you need to embed a double-quote inside a string which is already double-quoted.  
'  
The apostrophe or single-quote character (') can be symbolised with this character entity reference when you need to embed a single-quote or apostrophe inside a string which is already single-quoted.  
If you are using a DTD then you must declare all the character entities you need to use (if any), including any of the five above that you plan on using (they cease to be predeclared if you use a DTD). If you are using a Schema, you must use the numeric form for all except the five above because Schemas have no way to make character entity declarations.

**Do I have to change any of my server software to work with XML?**   
The only changes needed are to make sure your server serves up .xml, .css, .dtd, .xsl, and whatever other file types you will use as the correct MIME content (media) types.   
The details of the settings are specified in RFC 3023. Most new versions of Web server software come preset.   
If not, all that is needed is to edit the mime-types file (or its equivalent: as a server operator you already know where to do this, right?) and add or edit the relevant lines for the right media types. In some servers (eg Apache), individual content providers or directory owners may also be able to change the MIME types for specific file types from within their own directories by using directives in a .htaccess file. The media types required are:  
\* text/xml for XML documents which are ‘readable by casual users’;  
\* application/xml for XML documents which are ‘unreadable by casual users’;  
\* text/xml-external-parsed-entity for external parsed entities such as document fragments (eg separate chapters which make up a book) subject to the readability distinction of text/xml;  
\* application/xml-external-parsed-entity for external parsed entities subject to the readability distinction of application/xml;  
\* application/xml-dtd for DTD files and modules, including character entity sets.  
The RFC has further suggestions for the use of the +xml media type suffix for identifying ancillary files such as XSLT (application/xslt+xml).  
If you run scripts generating XHTML which you wish to be treated as XML rather than HTML, they may need to be modified to produce the relevant Document Type Declaration as well as the right media type if your application requires them to be validated.

**I'm trying to understand the XML Spec: why does it have such difficult terminology?**For implementation to succeed, the terminology needs to be precise. Design goal eight of the specification tells us that ‘the design of XML shall be formal and concise’. To describe XML, the specification therefore uses formal language drawn from several fields, specifically those of text engineering, international standards and computer science. This is often confusing to people who are unused to these disciplines because they use well-known English words in a specialised sense which can be very different from their common meanings—for example: grammar, production, token, or terminal.   
The specification does not explain these terms because of the other part of the design goal: the specification should be concise. It doesn't repeat explanations that are available elsewhere: it is assumed you know this and either know the definitions or are capable of finding them. In essence this means that to grok the fullness of the spec, you do need a knowledge of some SGML and computer science, and have some exposure to the language of formal standards.   
Sloppy terminology in specifications causes misunderstandings and makes it hard to implement consistently, so formal standards have to be phrased in formal terminology. This FAQ is not a formal document, and the astute reader will already have noticed it refers to ‘element names’ where ‘element type names’ is more correct; but the former is more widely understood.

**Can I still use server-side inclusions?**Yes, so long as what they generate ends up as part of an XML-conformant file (ie either valid or just well-formed).   
Server-side tag-replacers like shtml, PHP, JSP, ASP, Zope, etc store almost-valid files using comments, Processing Instructions, or non-XML markup, which gets replaced at the point of service by text or XML markup (it is unclear why some of these systems use non-HTML/XML markup). There are also some XML-based preprocessors for formats like XVRL (eXtensible Value Resolution Language) which resolve specialised references to external data and output a normalised XML file.

**Can I (and my authors) still use client-side inclusions?**The same rule applies as for server-side inclusions, so you need to ensure that any embedded code which gets passed to a third-party engine (eg calls to SQL, VB, Java, etc) does not contain any characters which might be misinterpreted as XML markup (ie no angle brackets or ampersands). Either use a CDATA marked section to avoid your XML application parsing the embedded code, or use the standard <, and & character entity references instead.

**How can I include a conditional statement in my XML?**   
You can't: XML isn't a programming language, so you can't say things like   
<google if {DB}="A">bar</google>   
If you need to make an element optional, based on some internal or external criteria, you can do so in a Schema. DTDs have no internal referential mechanism, so it isn't possible to express this kind of conditionality in a DTD at the individual element level.   
It is possible to express presence-or-absence conditionality in a DTD for the whole document, by using parameter entities as switches to include or ignore certain sections of the DTD based on settings either hardwired in the DTD or supplied in the internal subset. Both the TEI and Docbook DTDs use this mechanism to implement modularity.   
Alternatively you can make the element entirely optional in the DTD or Schema, and provide code in your processing software that checks for its presence or absence. This defers the checking until the processing stage: one of the reasons for Schemas is to provide this kind of checking at the time of document creation or editing.

**I have to do an overview of XML for my manager/client/investor/advisor. What should I mention?**   
\* XML is not a markup language. XML is a ‘metalanguage’, that is, it's a language that lets you define your own markup languages (see definition).  
\* XML is a markup language [two (seemingly) contradictory statements one after another is an attention-getting device that I'm fond of], not a programming language. XML is data: is does not ‘do’ anything, it has things done to it.  
\* XML is non-proprietary: your data cannot be held hostage by someone else.  
\* XML allows multi-purposing of your data.  
\* Well-designed XML applications most often separate ‘content’ from ‘presentation’. You should describe what something is rather what something looks like (the exception being data content which never gets presented to humans).  
Saying ‘the data is in XML’ is a relatively useless statement, similar to saying ‘the book is in a natural language’. To be useful, the former needs to specify ‘we have used XML to define our own markup language’ (and say what it is), similar to specifying ‘the book is in French’.  
A classic example of multipurposing and separation that I often use is a pharmaceutical company. They have a large base of data on a particular drug that they need to publish as:  
\* reports to the FDA;  
\* drug information for publishers of drug directories/catalogs;  
\* ‘prescribe me!’ brochures to send to doctors;  
\* little pieces of paper to tuck into the boxes;  
\* labels on the bottles;  
\* two pages of fine print to follow their ad in Reader's Digest;  
\* instructions to the patient that the local pharmacist prints out;  
\* etc.  
Without separation of content and presentation, they need to maintain essentially identical information in 20 places. If they miss a place, people die, lawyers get rich, and the drug company gets poor. With XML (or SGML), they maintain one set of carefully validated information, and write 20 programs to extract and format it for each application. The same 20 programs can now be applied to all the hundreds of drugs that they sell.  
In the Web development area, the biggest thing that XML offers is fixing what is wrong with HTML:  
\* browsers allow non-compliant HTML to be presented;  
\* HTML is restricted to a single set of markup (‘tagset’).  
If you let broken HTML work (be presented), then there is no motivation to fix it. Web pages are therefore tag soup that are useless for further processing. XML specifies that processing must not continue if the XML is non-compliant, so you keep working at it until it complies. This is more work up front, but the result is not a dead-end.  
If you wanted to mark up the names of things: people, places, companies, etc in HTML, you don't have many choices that allow you to distinguish among them. XML allows you to name things as what they are:  
<person>Charles Goldfarb</person> worked   
at <company>IBM</company>  
gives you a flexibility that you don't have with HTML:  
<B>Charles Goldfarb</B> worked at<B>IBM<</B>  
With XML you don't have to shoe-horn your data into markup that restricts your options.

**What is the purpose of XML namespaces?**XML namespaces are designed to provide universally unique names for elements and attributes. This allows people to do a number of things, such as:  
\* Combine fragments from different documents without any naming conflicts. (See example below.)  
\* Write reusable code modules that can be invoked for specific elements and attributes. Universally unique names guarantee that such modules are invoked only for the correct elements and attributes.  
\* Define elements and attributes that can be reused in other schemas or instance documents without fear of name collisions. For example, you might use XHTML elements in a parts catalog to provide part descriptions. Or you might use the nil attribute defined in XML Schemas to indicate a missing value.  
As an example of how XML namespaces are used to resolve naming conflicts in XML documents that contain element types and attributes from multiple XML languages, consider the following two XML documents:  
<?xml version="1.0" ?>  
<Address>  
<Street>Apple 7</Street>  
<City>Color</City>  
<State>State</State>  
<Country>Country</Country>  
<PostalCode>H98d69</PostalCode>  
</Address>  
and:  
<?xml version="1.0" ?>  
<Server>  
<Name>OurWebServer</Name>  
<Address>888.90.67.8</Address>  
</Server>  
  
Each document uses a different XML language and each language defines an Address element type. Each of these Address element types is different -- that is, each has a different content model, a different meaning, and is interpreted by an application in a different way. This is not a problem as long as these element types exist only in separate documents. But what if they are combined in the same document, such as a list of departments, their addresses, and their Web servers? How does an application know which Address element type it is processing?  
One solution is to simply rename one of the Address element types -- for example, we could rename the second element type IPAddress. However, this is not a useful long term solution. One of the hopes of XML is that people will standardize XML languages for various subject areas and write modular code to process those languages. By reusing existing languages and code, people can quickly define new languages and write applications that process them. If we rename the second Address element type to IPAddress, we will break any code that expects the old name.  
A better answer is to assign each language (including its Address element type) to a different namespace. This allows us to continue using the Address name in each language, but to distinguish between the two different element types. The mechanism by which we do this is XML namespaces.  
(Note that by assigning each Address name to an XML namespace, we actually change the name to a two-part name consisting of the name of the XML namespace plus the name Address. This means that any code that recognizes just the name Address will need to be changed to recognize the new two-part name. However, this only needs to be done once, as the two-part name is universally unique.

**What is an XML namespace?**An XML namespace is a collection of element type and attribute names. The collection itself is unimportant -- in fact, a reasonable argument can be made that XML namespaces don't actually exist as physical or conceptual entities . What is important is the name of the XML namespace, which is a URI. This allows XML namespaces to provide a two-part naming system for element types and attributes. The first part of the name is the URI used to identify the XML namespace -- the namespace name. The second part is the element type or attribute name itself -- the local part, also known as the local name. Together, they form the universal name.   
This two-part naming system is the only thing defined by the XML namespaces recommendation.

**Does the XML namespaces recommendation define anything except a two-part naming system for element types and attributes?**No.  
This is a very important point and a source of much confusion, so we will repeat it:  
THE XML NAMESPACES RECOMMENDATION DOES NOT DEFINE ANYTHING EXCEPT A TWO-PART NAMING SYSTEM FOR ELEMENT TYPES AND ATTRIBUTES.  
In particular, they do not provide or define any of the following:  
\* A way to merge two documents that use different DTDs.   
\* A way to associate XML namespaces and schema information.   
\* A way to validate documents that use XML namespaces.   
\* A way to associate element type or attribute declarations in a DTD with an XML namespace.

**What do XML namespaces actually contain?**   
XML namespaces are collections of names, nothing more. That is, they contain the names of element types and attributes, not the elements or attributes themselves. For example, consider the following document.  
<google:A xmlns:google="http://www.google.org/">  
<B google:C="google" D="bar"/>  
</google:A>  
The element type name A and the attribute name C are in the http://www.google.org/ namespace because they are mapped there by the google prefix. The element type name B and the attribute name D are not in any XML namespace because no prefix maps them there. On the other hand, the elements A and B and the attributes C and D are not in any XML namespace, even though they are physically within the scope of the http://www.google.org/ namespace declaration. This is because XML namespaces contain names, not elements or attributes.  
XML namespaces also do not contain the definitions of the element types or attributes. This is an important difference, as many people are tempted to think of an XML namespace as a schema, which it is not.

**Are the names of all element types and attributes in some XML namespace?**No.   
If an element type or attribute name is not specifically declared to be in an XML namespace -- that is, it is unprefixed and (in the case of element type names) there is no default XML namespace -- then that name is not in any XML namespace. If you want, you can think of it as having a null URI as its name, although no "null" XML namespace actually exists. For example, in the following, the element type name B and the attribute names C and E are not in any XML namespace:   
<google:A xmlns:google="http://www.google.org/">  
<B C="bar"/>  
<google:D E="bar"/>  
</google:A>

**Do XML namespaces apply to entity names, notation names, or processing instruction targets?**   
No.  
XML namespaces apply only to element type and attribute names. Furthermore, in an XML document that conforms to the XML namespaces recommendation, entity names, notation names, and processing instruction targets must not contain colons.

**Who can create an XML namespace?**Anybody can create an XML namespace -- all you need to do is assign a URI as its name and decide what element type and attribute names are in it. The URI must be under your control and should not be being used to identify a different XML namespace, such as by a coworker.   
(In practice, most people that create XML namespaces also describe the element types and attributes whose names are in it -- their content models and types, their semantics, and so on. However, this is not part of the process of creating an XML namespace, nor does the XML namespace include or provide a way to discover such information.)

**Do I need to use XML namespaces?**   
Maybe, maybe not.   
If you don't have any naming conflicts in the XML documents you are using today, as is often the case with documents used inside a single organization, then you probably don't need to use XML namespaces. However, if you do have conflicts today, or if you expect conflicts in the future due to distributing your documents outside your organization or bringing outside documents into your organization, then you should probably use XML namespaces.   
Regardless of whether you use XML namespaces in your own documents, it is likely that you will use them in conjunction with some other XML technology, such as XSL, XHTML, or XML Schemas. For example, the following XSLT (XSL Transformations) stylesheet uses XML namespaces to distinguish between element types defined in XSLT and those defined elsewhere:   
<xsl:stylesheet version="1.0"  
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">  
<xsl:template match="Address">  
<!-- The Addresses element type is not part of the XSLT namespace. -->  
<Addresses>  
<xsl:apply-templates/>  
</Addresses>  
</xsl:template>  
</xsl:stylesheet>

**What is the relationship between XML namespaces and the XML 1.0 recommendation?**   
Although the XML 1.0 recommendation anticipated the need for XML namespaces by noting that element type and attribute names should not include colons, it did not actually support XML namespaces. Thus, XML namespaces are layered on top of XML 1.0. In particular, any XML document that uses XML namespaces is a legal XML 1.0 document and can be interpreted as such in the absence of XML namespaces. For example, consider the following document:  
  
<google:A xmlns:google="http://www.google.org/">  
<google:B google:C="bar"/>  
</google:A>  
  
If this document is processed by a namespace-unaware processor, that processor will see two elements whose names are google:A and google:B. The google:A element has an attribute named xmlns:google and the google:B element has an attribute named google:C. On the other hand, a namespace-aware processor will see two elements with universal names {http://www.google.org}A and {http://www.google.org}B. The {http://www.google.org}A does not have any attributes; instead, it has a namespace declaration that maps the google prefix to the URI http://www.google.org. The {http://www.google.org}B element has an attribute named {http://www.google.org}C.  
Needless to say, this has led to a certain amount of confusion. One area of confusion is the relationship between XML namespaces and validating XML documents against DTDs. This occurs because the XML namespaces recommendation did not describe how to use XML namespaces with DTDs. Fortunately, a similar situation does not occur with XML schema languages, as all of these support XML namespaces.   
The other main area of confusion is in recommendations and specifications such as DOM and SAX whose first version predates the XML namespaces recommendation. Although these have since been updated to include XML namespace support, the solutions have not always been pretty due to backwards compatibility requirements. All recommendations in the XML family now support XML namespaces.

**What is the difference between versions 1.0 and 1.1 of the XML namespaces recommendation?**   
There are only two differences between XML namespaces 1.0 and XML namespaces 1.1:   
\* Version 1.1 adds a way to undeclare prefixes. For more information, see question 4.7.  
\* Version 1.1 uses IRIs (Internationalized Resource Identifiers) instead of URIs. Basically, URIs are restricted to a subset of ASCII characters, while IRIs allow much broader use of Unicode characters. For complete details, see section 9 of Namespaces in XML 1.1.  
NOTE: As of this writing (February, 2003), Namespaces in XML 1.1 is still a candidate recommendation and not widely used. PART II: DECLARING AND USING XML NAMESPACES

**How do I declare an XML namespace in an XML document?**   
To declare an XML namespace, you use an attribute whose name has the form:  
xmlns:prefix  
--OR--  
xmlns  
These attributes are often called xmlns attributes and their value is the name of the XML namespace being declared; this is a URI. The first form of the attribute (xmlns:prefix) declares a prefix to be associated with the XML namespace. The second form (xmlns) declares that the specified namespace is the default XML namespace.   
For example, the following declares two XML namespaces, named http://www.google.com/ito/addresses and http://www.google.com/ito/servers. The first declaration associates the addr prefix with the http://www.google.com/ito/addresses namespace and the second declaration states that the http://www.google.com/ito/servers namespace is the default XML namespace.  
<Department  
xmlns:addr="http://www.google.com/ito/addresses"  
xmlns="http://www.google.com/ito/servers">  
NOTE: Technically, xmlns attributes are not attributes at all -- they are XML namespace declarations that just happen to look like attributes. Unfortunately, they are not treated consistently by the various XML recommendations, which means that you must be careful when writing an XML application.   
For example, in the XML Information Set (http://www.w3.org/TR/xml-infoset), xmlns "attributes" do not appear as attribute information items. Instead, they appear as namespace declaration information items. On the other hand, both DOM level 2 and SAX 2.0 treat namespace attributes somewhat ambiguously. In SAX 2.0, an application can instruct the parser to return xmlns "attributes" along with other attributes, or omit them from the list of attributes. Similarly, while DOM level 2 sets namespace information based on xmlns "attributes", it also forces applications to manually add namespace declarations using the same mechanism the application would use to set any other attributes.

**Where can I declare an XML namespace?**You can declare an XML namespace on any element in an XML document. The namespace is in scope for that element and all its descendants unless it is overridden.

**Can I use an attribute default in a DTD to declare an XML namespace?**   
Yes.   
For example, the following uses the FIXED attribute xmlns:google on the A element type to associate the google prefix with the http://www.google.org/ namespace. The effect of this is that both A and B are in the http://www.google.org/ namespace.   
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A (google:B)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT google:B (#PCDATA)>  
]>  
<!-- google prefix declared through default attribute. -->  
<google:A>  
<google:B>abc</google:B>  
</google:A>  
  
IMPORTANT: You should be very careful about placing XML namespace declarations in external entities (external DTDs), as non-validating parsers are not required to read these. For example, suppose the preceding DTD was placed in an external entity (google.dtd) and that the document was processed by a non-validating parser that did not read google.dtd. This would result in a namespace error because the google prefix was never declared:   
<?xml version="1.0" ?>  
<!-- google.dtd might not be read by non-validating parsers. -->  
<!DOCTYPE google:A SYSTEM "google.dtd">  
<!-- google prefix not declared unless google.dtd is read. -->  
<google:A>  
<google:B>abc</google:B>  
</google:A>

**Do the default values of xmlns attributes declared in the DTD apply to the DTD?**   
No.   
Declaring a default value of an xmlns attribute in the DTD does not declare an XML namespace for the DTD. (In fact, no XML namespace declarations apply to DTDs.) Instead, these defaults (declarations) take effect only when the attribute is instantiated on an element. For example:   
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A (google:B)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT google:B (#PCDATA)>  
]>  
<google:A> <========== Namespace declaration takes effect here.  
<google:B>abc</google:B>  
</google:A> <========= Namespace declaration ends here.  
For more information, see question 7.2. (Note that an earlier version of MSXML (the parser used by Internet Explorer) did use fixed xmlns attribute declarations as XML namespace declarations, but that this was removed in MSXML 4.

**How do I override an XML namespace declaration that uses a prefix?**   
To override the prefix used in an XML namespace declaration, you simply declare another XML namespace with the same prefix. For example, in the following, the google prefix is associated with the http://www.google.org/ namespace on the A and B elements and the http://www.bar.org/ namespace on the C and D elements. That is, the names A and B are in the http://www.google.org/ namespace and the names C and D are in the http://www.bar.org/ namespace.  
  
<google:A xmlns:google="http://www.google.org/">  
<google:B>  
<google:C xmlns:google="http://www.bar.org/">  
<google:D>abcd</google:D>  
</google:C>  
</google:B>  
</google:A>  
  
In general, this leads to documents that are confusing to read and should be avoided.

**How do I override a default XML namespace declaration?**   
To override the current default XML namespace, you simply declare another XML namespace as the default. For example, in the following, the default XML namespace is the http://www.google.org/ namespace on the A and B elements and the http://www.bar.org/ namespace on the C and D elements. That is, the names A and B are in the http://www.google.org/ namespace and the names C and D are in the http://www.bar.org/ namespace.  
  
<A xmlns="http://www.google.org/">  
<B>  
<C xmlns="http://www.bar.org/">  
<D>abcd</D>  
</C>  
</B>  
</A>  
  
Using multiple default XML namespaces can lead to documents that are confusing to read and should be done carefully.

**How do I undeclare an XML namespace prefix?**   
In version 1.0 of the XML namespaces recommendation, you cannot "undeclare" an XML namespace prefix. It remains in scope until the end of the element on which it was declared unless it is overridden. Furthermore, trying to undeclare a prefix by redeclaring it with an empty (zero-length) name (URI) results in a namespace error. For example:  
  
<google:A xmlns:google="http://www.google.org/">  
<google:B>  
<google:C xmlns:google=""> <==== This is an error in v1.0, legal in v1.1.  
<google:D>abcd</google:D>  
</google:C>  
</google:B>  
</google:A>  
  
In version 1.1 of the XML namespaces recommendation [currently a candidate recommendation -- February, 2003], you can undeclare an XML namespace prefix by redeclaring it with an empty name. For example, in the above document, the XML namespace declaration xmlns:google="" is legal and removes the mapping from the google prefix to the http://www.google.org URI. Because of this, the use of the google prefix in the google:D element results in a namespace error.

**How do I undeclare the default XML namespace?**To "undeclare" the default XML namespace, you declare a default XML namespace with an empty (zero-length) name (URI). Within the scope of this declaration, unprefixed element type names do not belong to any XML namespace. For example, in the following, the default XML namespace is the http://www.google.org/ for the A and B elements and there is no default XML namespace for the C and D elements. That is, the names A and B are in the http://www.google.org/ namespace and the names C and D are not in any XML namespace.  
  
<A xmlns="http://www.google.org/">  
<B>  
<C xmlns="">  
<D>abcd</D>  
</C>  
</B>  
</A>

**Why are special attributes used to declare XML namespaces?**   
I don't know the answer to this question, but the likely reason is that the hope that they would simplify the process of moving fragments from one document to another document. An early draft of the XML namespaces recommendation proposed using processing instructions to declare XML namespaces. While these were simple to read and process, they weren't easy to move to other documents. Attributes, on the other hand, are intimately attached to the elements being moved.   
Unfortunately, this hasn't worked as well as was hoped. For example, consider the following XML document:  
  
<google:A xmlns:google="http://www.google.org/">  
<google:B>  
<google:C>bar</google:C>  
</google:B>  
</google:A>  
  
Simply using a text editor to cut the fragment headed by the <B> element from one document and paste it into another document results in the loss of namespace information because the namespace declaration is not part of the fragment -- it is on the parent element (<A>) -- and isn't moved.   
Even when this is done programmatically, the situation isn't necessarily any better. For example, suppose an application uses DOM level 2 to "cut" the fragment from the above document and "paste" it into a different document. Although the namespace information is transferred (it is carried by each node), the namespace declaration (xmlns attribute) is not, again because it is not part of the fragment. Thus, the application must manually add the declaration before serializing the document or the new document will be invalid.

**How do different XML technologies treat XML namespace declarations?**   
This depends on the technology -- some treat them as attributes and some treat them as namespace declarations. For example, SAX1 treats them as attributes and SAX2 can treat them as attributes or namespace declarations, depending on how the parser is configured. DOM levels 1 and 2 treat them as attributes, but DOM level 2 also interprets them as namespace declarations. XPath, XSLT, and XML Schemas treat them as namespaces declarations.   
The reason that different technologies treat these differently is that many of these technologies predate XML namespaces. Thus, newer versions of them need to worry both about XML namespaces and backwards compatibility issues.

**How do I use prefixes to refer to element type and attribute names in an XML namespace?**Make sure you have declared the prefix and that it is still in scope . All you need to do then is prefix the local name of an element type or attribute with the prefix and a colon. The result is a qualified name, which the application parses to determine what XML namespace the local name belongs to.   
For example, suppose you have associated the serv prefix with the http://www.our.com/ito/servers namespace and that the declaration is still in scope. In the following, serv:Address refers to the Address name in the http://www.our.com/ito/servers namespace. (Note that the prefix is used on both the start and end tags.)   
<!-- serv refers to the http://www.our.com/ito/servers namespace. -->  
<serv:Address>127.66.67.8</serv:Address>  
  
Now suppose you have associated the xslt prefix with the http://www.w3.org/1999/XSL/Transform namespace. In the following, xslt:version refers to the version name in the http://www.w3.org/1999/XSL/Transform namespace:   
<!-- xslt refers to the http://www.w3.org/1999/XSL/Transform namespace. -->  
<html xslt:version="1.0">

**How do I use the default XML namespace to refer to element type names in an XML namespace?**   
Make sure you have declared the default XML namespace and that that declaration is still in scope . All you need to do then is use the local name of an element type. Even though it is not prefixed, the result is still a qualified name ), which the application parses to determine what XML namespace it belongs to.   
For example, suppose you declared the http://www.w3.org/to/addresses namespace as the default XML namespace and that the declaration is still in scope. In the following, Address refers to the Address name in the http://www.w3.org/to/addresses namespace.  
  
<!-- http://www.w3.org/to/addresses is the default XML namespace. -->  
<Address>123.45.67.8</Address>

**How do I use the default XML namespace to refer to attribute names in an XML namespace?**You can't.   
The default XML namespace only applies to element type names, so you can refer to attribute names that are in an XML namespace only with a prefix. For example, suppose that you declared the http://http://www.w3.org/to/addresses namespace as the default XML namespace. In the following, the type attribute name does not refer to that namespace, although the Address element type name does. That is, the Address element type name is in the http://http://www.fyicneter.com/ito/addresses namespace, but the type attribute name is not in any XML namespace.  
  
<!-- http://http://www.w3.org/to/addresses is the default XML namespace. -->  
<Address type="home">  
  
To understand why this is true, remember that the purpose of XML namespaces is to uniquely identify element and attribute names. Unprefixed attribute names can be uniquely identified based on the element type to which they belong, so there is no need identify them further by including them in an XML namespace. In fact, the only reason for allowing attribute names to be prefixed is so that attributes defined in one XML language can be used in another XML language.

**When should I use the default XML namespace instead of prefixes?**This is purely a matter of choice, although your choice may affect the readability of the document. When elements whose names all belong to a single XML namespace are grouped together, using a default XML namespace might make the document more readable. For example:  
  
<!-- A, B, C, and G are in the http://www.google.org/ namespace. -->  
<A xmlns="http://www.google.org/">  
<B>abcd</B>  
<C>efgh</C>  
<!-- D, E, and F are in the http://www.bar.org/ namespace. -->  
<D xmlns="http://www.bar.org/">  
<E>1234</E>  
<F>5678</F>  
</D>  
<!-- Remember! G is in the http://www.google.org/ namespace. -->  
<G>ijkl</G>  
</A>  
  
When elements whose names are in multiple XML namespaces are interspersed, default XML namespaces definitely make a document more difficult to read and prefixes should be used instead. For example:  
  
<A xmlns="http://www.google.org/">  
<B xmlns="http://www.bar.org/">abcd</B>  
<C xmlns="http://www.google.org/">efgh</C>  
<D xmlns="http://www.bar.org/">  
<E xmlns="http://www.google.org/">1234</E>  
<F xmlns="http://www.bar.org/">5678</F>  
</D>  
<G xmlns="http://www.google.org/">ijkl</G>  
</A>  
  
In some cases, default namespaces can be processed faster than namespace prefixes, but the difference is certain to be negligible in comparison to total processing time.

**What is the scope of an XML namespace declaration?**   
The scope of an XML namespace declaration is that part of an XML document to which the declaration applies. An XML namespace declaration remains in scope for the element on which it is declared and all of its descendants, unless it is overridden or undeclared on one of those descendants.   
For example, in the following, the scope of the declaration of the http://www.google.org/ namespace is the element A and its descendants (B and C). The scope of the declaration of the http://www.bar.org/ namespace is only the element C.   
<google:A xmlns:google="http://www.google.org/">  
<google:B>  
<bar:C xmlns:bar="http://www.bar.org/" />  
</google:B>  
</google:A>

**Does the scope of an XML namespace declaration include the element it is declared on?**   
Yes.   
For example, in the following, the names B and C are in the http://www.bar.org/ namespace, not the http://www.google.org/ namespace. This is because the declaration that associates the google prefix with the http://www.bar.org/ namespace occurs on the B element, overriding the declaration on the A element that associates it with the http://www.google.org/ namespace.   
<google:A xmlns:google="http://www.google.org/">  
<google:B xmlns:google="http://www.bar.org/">  
<google:C>abcd</google:C>  
</google:B>  
</google:A>  
  
Similarly, in the following, the names B and C are in the http://www.bar.org/ namespace, not the http://www.google.org/ namespace because the declaration declaring http://www.bar.org/ as the default XML namespace occurs on the B element, overriding the declaration on the A element.  
  
<A xmlns="http://www.google.org/">  
<B xmlns="http://www.bar.org/">  
<C>abcd</C>  
</B>  
</A>  
  
A final example is that, in the following, the attribute name D is in the http://www.bar.org/ namespace.   
<google:A xmlns:google="http://www.google.org/">  
<google:B google:D="In http://www.bar.org/ namespace"  
xmlns:google="http://www.bar.org/">  
<C>abcd</C>  
</google:B>  
</google:A>  
  
One consequence of XML namespace declarations applying to the elements they occur on is that they actually apply before they appear. Because of this, software that processes qualified names should be particularly careful to scan the attributes of an element for XML namespace declarations before deciding what XML namespace (if any) an element type or attribute name belongs to.

**If an element or attribute is in the scope of an XML namespace declaration, is its name in that namespace?**Not necessarily.   
When an element or attribute is in the scope of an XML namespace declaration, the element or attribute's name is checked to see if it has a prefix that matches the prefix in the declaration. Whether the name is actually in the XML namespace depends on whether the prefix matches. For example, in the following, the element type names A, B, and D and the attribute names C and E are in the scope of the declaration of the http://www.google.org/ namespace. While the names A, B, and C are in that namespace, the names D and E are not.  
  
<google:A xmlns:google="http://www.google.org/">  
<google:B google:C="google" />  
<bar:D bar:E="bar" />  
</google:A>

**What happens when an XML namespace declaration goes out of scope?**When an XML namespace declaration goes out of scope, it simply no longer applies. For example, in the following, the declaration of the http://www.google.org/ namespace does not apply to the C element because this is outside its scope. That is, it is past the end of the B element, on which the http://www.google.org/ namespace was declared.   
<!-- B is in the http://www.google.org/ namespace;  
C is not in any XML namespace. -->  
<A>  
<B xmlns="http://www.google.org/">abcd</B>  
<C>efgh</C>  
</A>  
  
In addition to the declaration no longer applying, any declarations that it overrode come back into scope. For example, in the following, the declaration of the http://www.google.org/ namespace is brought back into scope after the end of the B element. This is because it was overridden on the B element by the declaration of the http://www.bar.org/ namespace.   
<!-- A and C are in the http://www.google.org/ namespace.  
B is in the http://www.bar.org/ namespace. -->  
<A xmlns="http://www.google.org/">  
<B xmlns="http://www.bar.org/">abcd</B>  
<C>efgh</C>  
</A>

**What happens if no XML namespace declaration is in scope?**   
If no XML namespace declaration is in scope, then any prefixed element type or attribute names result in namespace errors. For example, in the following, the names google:A and google:B result in namespace errors.   
<?xml version="1.0" ?>  
<google:A google:B="error" />  
  
In the absence of an XML namespace declaration, unprefixed element type and attribute names do not belong to any XML namespace. For example, in the following, the names A and B are not in any XML namespace.

**Can multiple XML namespace declarations be in scope at the same time?**Yes, as long as they don't use the same prefixes and at most one of them is the default XML namespace. For example, in the following, the http://www.google.org/ and http://www.bar.org/ namespaces are both in scope for all elements:   
<A xmlns:google="http://www.google.org/"  
xmlns:bar="http://www.bar.org/">  
<google:B>abcd</google:B>  
<bar:C>efgh</bar:C>  
</A>  
One consequence of this is that you can place all XML namespace declarations on the root element and they will be in scope for all elements. This is the simplest way to use XML namespaces.

**How can I declare XML namespaces so that all elements and attributes are in their scope?**   
XML namespace declarations that are made on the root element are in scope for all elements and attributes in the document. This means that an easy way to declare XML namespaces is to declare them only on the root element.

**Does the scope of an XML namespace declaration ever include the DTD?**   
No.   
XML namespaces can be declared only on elements and their scope consists only of those elements and their descendants. Thus, the scope can never include the DTD.

**Can I use XML namespaces in DTDs?**   
Yes and no.  
In particular, DTDs can contain qualified names but XML namespace declarations do not apply to DTDs .  
This has a number of consequences. Because XML namespace declarations do not apply to DTDs:  
1. There is no way to determine what XML namespace a prefix in a DTD points to. Which means...  
2. Qualified names in a DTD cannot be mapped to universal names. Which means...  
3. Element type and attribute declarations in a DTD are expressed in terms of qualified names, not universal names. Which means...  
4. Validation cannot be redefined in terms of universal names as might be expected.  
This situation has caused numerous complaints but, as XML namespaces are already a recommendation, is unlikely to change. The long term solution to this problem is an XML schema language: all of the proposed XML schema languages provide a mechanism by which the local name in an element type or attribute declaration can be associated with an XML namespace. This makes it possible to redefine validity in terms of universal names.

**Do XML namespace declarations apply to DTDs?**   
No.   
In particular, an xmlns attribute declared in the DTD with a default is not an XML namespace declaration for the DTD.. (Note that an earlier version of MSXML (the parser used by Internet Explorer) did use such declarations as XML namespace declarations, but that this was removed in MSXML 4

**Can I use qualified names in DTDs?**Yes.   
For example, the following is legal:  
  
<!ELEMENT google:A (google:B)>  
<!ATTLIST google:A  
google:C CDATA #IMPLIED>  
<!ELEMENT google:B (#PCDATA)>  
  
However, because XML namespace declarations do not apply to DTDs , qualified names in the DTD cannot be converted to universal names. As a result, qualified names in the DTD have no special meaning. For example, google:A is just google:A -- it is not A in the XML namespace to which the prefix google is mapped.   
The reason qualified names are allowed in the DTD is so that validation will continue to work.

**Can the content model in an element type declaration contain element types whose names come from other XML namespaces?**   
Yes and no.   
The answer to this question is yes in the sense that a qualified name in a content model can have a different prefix than the qualified name of the element type being declared. For example, the following is legal:  
<!ELEMENT google:A (bar:B, baz:C)>  
The answer to this question is no in the sense that XML namespace declarations do not apply to DTDs so the prefixes used in an element type declaration are technically meaningless. In particular, they do not specify that the name of a certain element type belongs to a certain namespace. Nevertheless, the ability to mix prefixes in this manner is crucial when: a) you have a document whose names come from multiple XML namespaces , and b) you want to construct that document in a way that is both valid and conforms to the XML namespaces recommendation .

**Can the attribute list of an element type contain attributes whose names come from other XML namespaces?**   
Yes and no.   
For example, the following is legal:   
<!ATTLIST google:A   
bar:B CDATA #IMPLIED>

**How can I construct an XML document that is valid and conforms to the XML namespaces recommendation?**In answering this question, it is important to remember that:  
\* Validity is a concept defined in XML 1.0,  
\* XML namespaces are layered on top of XML 1.0 , and  
\* The XML namespaces recommendation does not redefine validity, such as in terms of universal names .  
Thus, validity is the same for a document that uses XML namespaces and one that doesn't. In particular, with respect to validity:  
\* xmlns attributes are treated as attributes, not XML namespace declarations.  
\* Qualified names are treated like other names. For example, in the name google:A, google is not treated as a namespace prefix, the colon is not treated as separating a prefix from a local name, and A is not treated as a local name. The name google:A is treated simply as the name google:A.  
Because of this, XML documents that you might expect to be valid are not. For example, the following document is not valid because the element type name A is not declared in the DTD, in spite of the fact both google:A and A share the universal name {http://www.google.org/}A:  
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A EMPTY>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/"  
xmlns CDATA #FIXED "http://www.google.org/">  
]>  
<A/>  
  
Similarly, the following is not valid because the xmlns attribute is not declared in the DTD:  
  
<?xml version="1.0" ?>  
<!DOCTYPE A [  
<!ELEMENT A EMPTY>  
]>  
<A xmlns="http://www.google.org/" />  
  
Furthermore, documents that you might expect to be invalid are valid. For example, the following document is valid but contains two definitions of the element type with the universal name {http://www.google.org/}A:  
  
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A (bar:A)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT bar:A (#PCDATA)>  
<!ATTLIST bar:A  
xmlns:bar CDATA #FIXED "http://www.google.org/">  
]>  
<google:A>  
<bar:A>abcd</bar:A>  
</google:A>  
  
Finally, validity has nothing to do with correct usage of XML namespaces. For example, the following document is valid but does not conform to the XML namespaces recommendation because the google prefix is never declared:   
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A EMPTY>  
]>  
<google:A />  
  
Therefore, when constructing an XML document that uses XML namespaces, you need to do both of the following if you want the document to be valid:  
\* Declare xmlns attributes in the DTD.  
\* Use the same qualified names in the DTD and the body of the document.  
For example:  
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A (google:B)  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT google:B EMPTY>  
]>  
<google:A>  
<google:B />  
</google:A>  
  
There is no requirement that the same prefix always be used for the same XML namespace. For example, the following is also valid:  
<?xml version="1.0" ?>  
<!DOCTYPE google:A [  
<!ELEMENT google:A (bar:B)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT bar:B EMPTY>  
<!ATTLIST bar:B  
xmlns:bar CDATA #FIXED "http://www.google.org/">  
]>  
<google:A>  
<bar:B />  
</google:A>  
  
However, documents that use multiple prefixes for the same XML namespace or the same prefix for multiple XML namespaces are confusing to read and thus prone to error. They also allow abuses such as defining an element type or attribute with a given universal name more than once, as was seen earlier. Therefore, a better set of guidelines for writing documents that are both valid and conform to the XML namespaces recommendation is:   
\* Declare all xmlns attributes in the DTD.  
\* Use the same qualified names in the DTD and the body of the document.  
\* Use one prefix per XML namespace.  
\* Do not use the same prefix for more than one XML namespace.  
\* Use at most one default XML namespace.  
  
The latter three guidelines guarantee that prefixes are unique. This means that prefixes fulfill the role normally played by namespace names (URIs) -- uniquely identifying an XML namespace -- and that qualified names are equivalent to universal names, so a given universal name is always represented by the same qualified name. Unfortunately, this is contrary to the spirit of prefixes, which were designed for their flexibility. For a slightly better solution.

**How can I allow the prefixes in my document to be different from the prefixes in my DTD?**One of the problems with the solution proposed in question is that it requires the prefixes in the document to match those in the DTD. Fortunately, there is a workaround for this problem, although it does require that a single prefix be used for a particular namespace URI throughout the document. (This is a good practice anyway, so it's not too much of a restriction.) The solution assumes that you are using a DTD that is external to the document, which is common practice.   
To use different prefixes in the external DTD and XML documents, you declare the prefix with a pair of parameter entities in the DTD. You can then override these entities with declarations in the internal DTD in a given XML document. This works because the internal DTD is read before the external DTD and the first definition of a particular entity is the one that is used. The following paragraphs describe how to use a single namespace in your DTD. You will need to modify them somewhat to use multiple namespaces.   
To start with, declare three parameter entities in your DTD:  
  
<!ENTITY % p "" >  
<!ENTITY % s "" >  
<!ENTITY % nsdecl "xmlns%s;" >  
  
The p entity ("p" is short for "prefix") is used in place of the actual prefix in element type and attribute names. The s entity ("s" is short for "suffix") is used in place of the actual prefix in namespace declarations. The nsdecl entity ("nsdecl" is short for "namespace declaration") is used in place of the name of the xmlns attribute in declarations of that attribute.   
Now use the p entity to define parameter entities for each of the names in your namespace. For example, suppose element type names A, B, and C and attribute name D are in your namespace.  
  
<!ENTITY % A "%p;A">  
<!ENTITY % B "%p;B">  
<!ENTITY % C "%p;C">  
<!ENTITY % D "%p;D">  
  
Next, declare your element types and attributes using the "name" entities, not the actual names. For example:  
  
<!ELEMENT %A; ((%B;)\*, %C;)>  
<!ATTLIST %A;  
%nsdecl; CDATA "http://www.google.org/">  
<!ELEMENT %B; EMPTY>  
<!ATTLIST %B;  
%D; NMTOKEN #REQUIRED  
E CDATA #REQUIRED>  
<!ELEMENT %C; (#PCDATA)>  
  
There are several things to notice here.  
\* Attribute D is in a namespace, so it is declared with a "name" entity. Attribute E is not in a namespace, so no entity is used.  
\* The nsdecl entity is used to declare the xmlns attribute. (xmlns attributes must be declared on every element type on which they can occur.) Note that a default value is given for the xmlns attribute.  
\* The reference to element type B in the content model of A is placed inside parentheses. The reason for this is that a modifier -- \* in this case -- is applied to it. Using parentheses is necessary because the replacement values of parameter entities are padded with spaces; directly applying the modifier to the parameter entity reference would result in illegal syntax in the content model.  
For example, suppose the value of the A entity is "google:A", the value of the B entity is "google:B", and the value of the C entity is "google:C". The declaration:  
<!ELEMENT %A; (%B;\*, %C;)>  
would resolve to:  
<!ELEMENT google:A ( google:B \*, google:C )>  
  
This is illegal because the \* modifier must directly follow the reference to the google:B element type. By placing the reference to the B entity in parentheses, the declaration resolves to:  
  
<!ELEMENT google:A (( google:B )\*, google:C )>  
  
This is legal because the \* modifier directly follows the closing parenthesis.  
  
Now let's see how this all works. Suppose our XML document won't use prefixes, but instead wants the default namespace to be the http://www.google.org/ namespace. In this case, no entity declarations are needed in the document. For example, our document might be:  
  
<!DOCTYPE A SYSTEM "http://www.google.org/google.dtd">  
<A>  
<B D="bar" E="baz buz" />  
<B D="boo" E="biz bez" />  
<C>bizbuz</C>  
</A>  
  
This document is valid because the declarations for p, s, and nsdecl in the DTD set p and s to "" and nsdecl to "xmlns". That is, after replacing the p, s, and nsdecl parameter entities, the DTD is as follows. Notice that both the DTD and document use the element type names A, B, and C and the attribute names D and E.  
<!ELEMENT A (( B )\*, C )>  
<!ATTLIST A   
xmlns CDATA "http://www.google.org/">  
<!ELEMENT B EMPTY>  
<!ATTLIST B   
D NMTOKEN #REQUIRED  
E CDATA #REQUIRED>  
<!ELEMENT C (#PCDATA)>  
  
But what if the document wants to use a different prefix, such as google? In this case, the document must override the declarations of the p and s entities in its internal DTD. That is, it must declare these entities so that they use google as a prefix (followed by a colon) and a suffix (preceded by a colon). For example:  
  
<!DOCTYPE google:A SYSTEM "http://www.google.org/google.dtd" [  
<!ENTITY % p "google:">  
<!ENTITY % s ":google">  
]>  
<google:A>  
<google:B google:D="bar" E="baz buz" />  
<google:B google:D="boo" E="biz bez" />  
<google:C>bizbuz</google:C>  
</google:A>  
  
In this case, the internal DTD is read before the external DTD, so the values of the p and s entities from the document are used. Thus, after replacing the p, s, and nsdecl parameter entities, the DTD is as follows. Notice that both the DTD and document use the element type names google:A, google:B, and google:C and the attribute names google:D and E.  
  
<!ELEMENT google:A (( google:B )\*, google:C )>  
<!ATTLIST google:A   
xmlns:google CDATA "http://www.google.org/">  
<!ELEMENT google:B EMPTY>  
<!ATTLIST google:B   
google:D NMTOKEN #REQUIRED  
E CDATA #REQUIRED>  
<!ELEMENT google:C (#PCDATA)>

**How can I validate an XML document that uses XML namespaces?**   
When people ask this question, they usually assume that validity is different for documents that use XML namespaces and documents that don't. In fact, it isn't -- it's the same for both. Thus, there is no difference between validating a document that uses XML namespaces and validating one that doesn't. In either case, you simply use a validating parser or other software that performs validation.

**If I start using XML namespaces, do I need to change my existing DTDs?**Probably. If you want your XML documents to be both valid and conform to the XML namespaces recommendation, you need to declare any xmlns attributes and use the same qualified names in the DTD as in the body of the document.   
If your DTD contains element type and attribute names from a single XML namespace, the easiest thing to do is to use your XML namespace as the default XML namespace. To do this, declare the attribute xmlns (no prefix) for each possible root element type. If you can guarantee that the DTD is always read , set the default value in each xmlns attribute declaration to the URI used as your namespace name. Otherwise, declare your XML namespace as the default XML namespace on the root element of each instance document.   
If your DTD contains element type and attribute names from multiple XML namespaces, you need to choose a single prefix for each XML namespace and use these consistently in qualified names in both the DTD and the body of each document. You also need to declare your xmlns attributes in the DTD and declare your XML namespaces. As in the single XML namespace case, the easiest way to do this is add xmlns attributes to each possible root element type and use default values if possible.

**How do I create documents that use XML namespaces?**The same as you create documents that don't use XML namespaces. If you're currently using Notepad on Windows or emacs on Linux, you can continue using Notepad or emacs. If you're using an XML editor that is not namespace-aware, you can also continue to use that, as qualified names are legal names in XML documents and xmlns attributes are legal attributes. And if you're using an XML editor that is namespace-aware, it will probably provide features such as automatically declaring XML namespaces and keeping track of prefixes and the default XML namespace for you.

**How can I check that a document conforms to the XML namespaces recommendation?**   
Unfortunately, I know of no software that only checks for conformance to the XML namespaces recommendation. It is possible that some namespace-aware validating parsers (such as those from DataChannel (Microsoft), IBM, Oracle, or Sun) check XML namespace conformance as part of parsing and validating. Thus, you might be able to run your document through such parsers as a way of testing conformance.   
Note that writing an application to check conformance to the XML namespaces recommendation is not as easy as it might seem. The problem is that most parsers do not make DTD information available to the application, so it might not be possible to check conformance in the DTD. Also note that writing a SAX 1.0 application that checks conformance in the body of the document (as opposed to the DTD) should be an easy thing to do.

**Can I use the same document with both namespace-aware and namespace-unaware applications?**   
Yes.   
This situation is quite common, such as when a namespace-aware application is built on top of a namespace-unaware parser. Another common situation is when you create an XML document with a namespace-unaware XML editor but process it with a namespace-aware application.   
Using the same document with both namespace-aware and namespace-unaware applications is possible because XML namespaces use XML syntax. That is, an XML document that uses XML namespaces is still an XML document and is recognized as such by namespace-unaware software.   
The only thing you need to be careful about when using the same document with both namespace-aware and namespace-unaware applications is when the namespace-unaware application requires the document to be valid. In this case, you must be careful to construct your document in a way that is both valid and conforms to the XML namespaces recommendation. (It is possible to construct documents that conform to the XML namespaces recommendation but are not valid and vice versa.)

**What software is needed to process XML namespaces?**   
From a document author's perspective, this is generally not a relevant question. Most XML documents are written in a specific XML language and processed by an application that understands that language. If the language uses an XML namespace, then the application will already use that namespace -- there is no need for any special XML namespace software.

**How do I use XML namespaces with Internet Explorer 5.0 and/or the MSXML parser?**   
WARNING! The following applies only to earlier versions of MSXML. It does not apply to MSXML 4, which is the currently shipping version [July, 2002].   
An early version of the MSXML parser, which was shipped as part of Internet Explorer 5.0, required that every XML namespace prefix used in an element type or attribute declaration had to be "declared" in the attribute declaration for that element type. This had to be done with a fixed xmlns attribute declaration. For example, the following was accepted by MSXML and both xmlns:google attributes were required:   
<!ELEMENT google:A (#PCDATA)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT google:B (#PCDATA)>  
<!ATTLIST google:B  
xmlns:google CDATA #FIXED "http://www.google.org/">  
  
MSXML returned an error for the following because the second google prefix was not "declared":  
  
<!ELEMENT google:A (#PCDATA)>  
<!ATTLIST google:A  
xmlns:google CDATA #FIXED "http://www.google.org/">  
<!ELEMENT google:B (#PCDATA)>  
  
The reason for this restriction was so that MSXML could use universal names to match element type and attribute declarations to elements and attributes during validation. Although this would have simplified many of the problems of writing documents that are both valid and conform to the XML namespaces recommendation some users complained about it because it was not part of the XML namespaces recommendation. In response to these complaints, Microsoft removed this restriction in later versions, which are now shipping. Ironically, the idea was later independently derived as a way to resolve the problems of validity and namespaces. However, it has not been implemented by anyone.

**How do applications process documents that use XML namespaces?**Applications process documents that use XML namespaces in almost exactly the same way they process documents that don't use XML namespaces. For example, if a namespace-unaware application adds a new sales order to a database when it encounters a SalesOrder element, the equivalent namespace-aware application does the same. The only difference is that the namespace-aware application:  
\* Might need to check for xmlns attributes and parse qualified names. Whether it does this depends on whether such processing is already done by lower-level software, such as a namespace-aware DOM implementation.  
\* Uses universal (two-part) names instead of local (one-part) names. For example, the namespace-aware application might add a new sales order in response to an {http://www.google.com/ito/sales}SalesOrder element instead of a SalesOrder element.

**How do I use XML namespaces with SAX 1.0?**   
The easiest way to use XML namespaces with SAX 1.0 is to use John Cowan's Namespace SAX Filter (see http://www.ccil.org/~cowan/XML). This is a SAX filter that keeps track of XML namespace declarations, parses qualified names, and returns element type and attribute names as universal names in the form:  
URI^local-name  
For example:  
http://www.google.com/ito/sales^SalesOrder  
Your application can then base its processing on these longer names. For example, the code:  
public void startElement(String elementName, AttributeList attrs)  
throws SAXException  
{  
...  
if (elementName.equals("SalesOrder"))  
{  
// Add new database record.  
}  
...  
}  
might become:  
public void startElement(String elementName, AttributeList attrs)  
throws SAXException  
{  
...  
if (elementName.equals("http://www.google.com/sales^SalesOrder"))  
{  
// Add new database record.  
}  
...  
}  
or:  
public void startElement(String elementName, AttributeList attrs)  
throws SAXException  
{  
...  
// getURI() and getLocalName() are utility functions  
// to parse universal names.  
if (getURI(elementName).equals("http://www.foo.com/ito/sales"))  
{  
if (getLocalName(elementName).equals("SalesOrder"))  
{  
// Add new database record.  
}  
}  
...  
}  
If you do not want to use the Namespace SAX Filter, then you will need to do the following in addition to identifying element types and attributes by their universal names:  
\* In startElement, scan the attributes for XML namespace declarations before doing any other processing. You will need to maintain a table of current prefix-to-URI mappings (including a null prefix for the default XML namespace).  
\* In startElement and endElement, check whether the element type name includes a prefix. If so, use your mappings to map this prefix to a URI. Depending on how your software works, you might also check if the local part of the qualified name includes any colons, which are illegal.  
\* In startElement, check whether attribute names include a prefix. If so, process as in the previous point.

**How do I use XML namespaces with SAX 2.0?**   
SAX 2.0 primarily supports XML namespaces through the following methods: \* startElement and endElement in the ContentHandler interface return namespace names (URIs) and local names as well as qualified names. \* getValue, getType, and getIndex in the Attributes interface can retrieve attribute information by namespace name (URI) and local name as well as by qualified name.

**How do I use XML namespaces with DOM level 2?**// Check the local name.  
// getNodeName() is a DOM level 1 method.  
  
if (elementNode.getNodeName().equals("SalesOrder"))  
{  
// Add new database record.  
}  
  
might become the following namespace-aware code:  
  
// Check the XML namespace name (URI).  
// getNamespaceURI() is a DOM level 2 method.  
  
String SALES\_NS = "http://www.foo.com/ito/sales";  
if (elementNode.getNamespaceURI().equals(SALES\_NS))  
{  
  
// Check the local name.  
// getLocalName() is a DOM level 2 method.  
  
if (elementNode.getLocalName().equals("SalesOrder"))  
{  
// Add new database record.  
}  
}  
  
Note that, unlike SAX 2.0, DOM level 2 treats xmlns attributes as normal attributes.

**Can an application process documents that use XML namespaces and documents that don't use XML namespaces?**   
Yes.   
This is a common situation for generic applications, such as editors, browsers, and parsers, that are not wired to understand a particular XML language. Such applications simply treat all element type and attribute names as qualified names. Those names that are not mapped to an XML namespace -- that is, unprefixed element type names in the absence of a default XML namespace and unprefixed attribute names -- are simply processed as one-part names, such as by using a null XML namespace name (URI).   
Note that such applications must decide how to treat documents that do not conform to the XML namespaces recommendation. For example, what should the application do if an element type name contains a colon (thus implying the existence of a prefix), but there are no XML namespace declarations in the document? The application can choose to treat this as an error, or it can treat the document as one that does not use XML namespaces, ignore the "error", and continue processing.

**Can an application be both namespace-aware and namespace-unaware?**   
Yes.   
However, there is generally no reason to do this. The reason is that most applications understand a particular XML language, such as one used to transfer sales orders between companies. If the element type and attribute names in the language belong to an XML namespace, the application must be namespace-aware; if not, the application must be namespace-unaware.   
For a few applications, being both namespace-aware and namespace-unaware makes sense. For example, a parser might choose to redefine validity in terms of universal names and have both namespace-aware and namespace-unaware validation modes. However, such applications are uncommon.

**What does a namespace-aware application do when it encounters an error?**   
The XML namespaces recommendation does not specify what a namespace-aware application does when it encounters a document that does not conform to the recommendation. Therefore, the behavior is application-dependent. For example, the application could stop processing, post an error to a log and continue processing, or ignore the error.  
PART III: NAMES, PREFIXES, AND URIs

**What is a qualified name?**   
A qualified name is a name of the following form. It consists of an optional prefix and colon, followed by the local part, which is sometimes known as a local name.   
prefix:local-part  
--OR--  
local-part  
  
For example, both of the following are qualified names. The first name has a prefix of serv; the second name does not have a prefix. For both names, the local part (local name) is Address.   
serv:Address  
Address  
  
In most circumstances, qualified names are mapped to universal names.

**What characters are allowed in a qualified name?**   
The prefix can contain any character that is allowed in the Name [5] production in XML 1.0 except a colon. The same is true of the local name. Thus, there can be at most one colon in a qualified name -- the colon used to separate the prefix from the local name.

**Where can qualified names appear?**Qualified names can appear anywhere an element type or attribute name can appear: in start and end tags, as the document element type, and in element type and attribute declarations in the DTD. For example:  
  
<!DOCTYPE foo:A [  
<!ELEMENT foo:A (foo:B)>  
<!ATTLIST foo:A  
foo:C CDATA #IMPLIED>  
<!ELEMENT foo:B (#PCDATA)>  
]>  
<foo:A xmlns:foo="http://www.foo.org/" foo:C="bar">  
<foo:B>abcd  
<foo:A>  
  
Qualified names cannot appear as entity names, notation names, or processing instruction targets.

**Can qualified names be used in attribute values?**   
Yes, but they have no special significance. That is, they are not necessarily recognized as such and mapped to universal names. For example, the value of the C attribute in the following is the string "foo:D", not the universal name {http://www.foo.org/}D.  
<foo:A xmlns:foo="http://www.foo.org/">  
<foo:B C="foo:D"/>  
<foo:A>  
  
In spite of this, there is nothing to stop an application from recognizing a qualified name in an attribute value and processing it as such. This is being done in various technologies today. For example, in the following XML Schemas definition, the attribute value xsd:string identifies the type of the foo attribute as the universal name {http://www.w3.org/1999/XMLSchema}string.  
  
<xsd:attribute name="foo" type="xsd:string" />  
  
There are two potential problems with this. First, the application must be able to retrieve the prefix mappings currently in effect. Fortunately, both SAX 2.0 and DOM level 2 support this capability. Second, any general purpose transformation tool, such as one that writes an XML document in canonical form and changes namespace prefixes in the process, will not recognize qualified names in attribute values and therefore not transform them correctly. Although this may be solved in the future by the introduction of the QName (qualified name) data type in XML Schemas, it is a problem today.

**How are qualified names mapped to names in XML namespaces?**   
If a qualified name in the body of a document (as opposed to the DTD) includes a prefix, then that prefix is used to map the local part of the qualified name to a universal name -- that is, a name in an XML namespace. For example, in the following, the prefix foo is used to map the local names A, B, and C to names in the http://www.foo.org/ namespace:   
<?xml version="1.0" ?>  
<foo:A xmlns:foo="http://www.foo.org/" foo:C="bar">  
<foo:B>abcd  
<foo:A>  
  
If a qualified name in the body of a document does not include a prefix and a default XML namespace is in scope then one of two things happens. If the name is used as an element tag, it is mapped to a name in the default XML namespace. If it is used as an attribute name, it is not in any XML namespace. For example, in the following, A and B are in the http://www.foo.org/ namespace and C is not in any XML namespace:  
  
<?xml version="1.0" ?>  
<A xmlns="http://www.foo.org/" C="bar">  
<B>abcd</B>  
<A>  
  
If a qualified name in the body of a document does not include a prefix and no default XML namespace is in scope, then that name is not in any XML namespace. For example, in the following, A, B, and C are not in any XML namespace:   
<?xml version="1.0" ?>  
<A C="bar">  
<B>abcd</B>  
<A>  
  
Qualified names in the DTD are never mapped to names in an XML namespace because they are never in the scope of an XML namespace declaration.

**How are universal names represented?**   
There is no standard way to represent a universal name. However, three representations are common.   
The first representation keeps the XML namespace name (URI) and the local name separate. For example, many DOM level 1 implementations have different methods for returning the XML namespace name (URI) and the local name of an element or attribute node.   
The second representation concatenates the namespace name (URI) and the local name with caret (^). The result is a universally unique name, since carets are not allowed in URIs or local names. This is the method used by John Cowan's Namespace SAX Filter . For example, the universal name that has the URI http://www.google.org/to/servers and the local name Address would be represented as:   
http://www.foo.com/ito/servers^Address   
The third representation places the XML namespace name (URI) in braces and concatenates this with the local name. This notation is suggested only for documentation and I am aware of no code that uses it. For example, the above name would be represented as:   
{http://www.foo.com/ito/servers}Address

**Are universal names universally unique?**   
No, but it is reasonable to assume they are.   
Universal element type and attribute names are not guaranteed to be universally unique -- that is, unique within the space of all XML documents -- because it is possible for two different people, each defining their own XML namespace, to use the same URI and the same element type or attribute name. However, this occurs only if:   
\* One or both people use a URI that is not under their control, such as somebody outside Netscape using the URI http://www.netscape.com/, or  
\* Both people have control over a URI and both use it.  
  
The first case means somebody is cheating when assigning URIs (a process governed by trust) and the second case means that two people within an organization are not paying attention to each other's work. For widely published element type and attribute names, neither case is very likely. Thus, it is reasonable to assume that universal names are universally unique. (Since both cases are possible, applications that present security risks should be careful about assuming that universal names are universally unique.)   
For information about the ability of universal names to uniquely identify element types and attributes (as opposed to the names themselves being unique).

**What is an XML namespace prefix?**An XML namespace prefix is a prefix used to specify that a local element type or attribute name is in a particular XML namespace. For example, in the following, the serv prefix specifies that the Address element type name is in the http://www.foo.com/ito/addresses namespace:   
<serv:Addresses xmlns:serv="http://www.foo.com/ito/addresses">

**What characters are allowed in an XML namespace prefix?**   
The prefix can contain any character that is allowed in the Name [5] production in XML 1.0 except a colon.

**Can I use the same prefix for more than one XML namespace?**   
Yes.

**What happens if there is no prefix on an element type name?**If a default XML namespace declaration is in scope, then the element type name is in the default XML namespace. Otherwise, the element type name is not in any XML namespace.

**What does the URI used as an XML namespace name point to?**   
The URI used as an XML namespace name is simply an identifier. It is not guaranteed to point to anything and, in general, it is a bad idea to assume that it does. This point causes a lot of confusion, so we'll repeat it here:   
URIs USED AS XML NAMESPACE NAMES ARE JUST IDENTIFIERS. THEY ARE NOT GUARANTEED TO POINT TO ANYTHING.   
While this might be confusing when URLs are used as namespace names, it is obvious when other types of URIs are used as namespace names. For example, the following namespace declaration uses an ISBN URN:   
xmlns:xbe="urn:ISBN:0-7897-2504-5"   
and the following namespace declaration uses a UUID URN:   
xmlns:foo="urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6"   
Clearly, neither namespace name points to anything on the Web.   
NOTE: Namespace URIs that are URLs may point to RDDL documents, although this does not appear to be widely implemented. For details, see the next question.   
NOTE: An early version of the W3C's XML Schemas used namespace URIs to point to an XML Schema document containing the definitions of the element types and attributes named in the namespace. However, this proved very controversial and the idea has been withdrawn.

**What is an XML namespace name?**   
An XML namespace name is a URI that uniquely identifies the namespace. URIs are used because they are widely understood and well documented. Because people may only allocate URIs under their control, it is easy to ensure that no two XML namespaces are identified by the same URI.

**Can I resolve the URI used as an XML namespace name?**   
Yes.

**Can I use a relative URI as a namespace name?**   
Yes. However, such usage is deprecated, so you should never do it.

**What is XPointer?**   
XPointer is set of recommendations developed by the W3C. The core recommendations are the XPointer Framework which provides an extensible addressing behavior for fragment identifiers in XML media types.   
XPointer gains its extensibility through the XPointer Framework, which identifies the syntax and processing architecture for XPointer expressions and through an extensible set of XPointer addressing schemes. These schemes, e.g., element() or xpointer(), are actually QNames. The xmlns() scheme makes it possible for an XPointer to declare namespace bindings and thereby use third-party schemes as readily as W3C defined XPointer schemes.

**How do I install the XPointer processor?**   
Download the latest "cweb-xpointer" release from SourceForge. This project uses Apache Maven and Java 1.4+, so you will need to install those as well. Normally you will also want to download one of the XPointer Framework integrations, such as the xpointer+dom4j or the xpointer+jdom package. These "integration packages" provide support for a specific XML Document model.   
The project dependencies are explicitly declared in the Maven POM. This means that Maven can automagically download the required releases of dependent JARs.   
There are several release artifacts. The "uberjar" release provides an executable command line utility (see below) and bundles all dependancies (except for Java itself). If you want to integrate into an existing application, then you should use the cweb-xpointer JAR and also download copies of its dependencies. If you are using a Maven project, then this is all very, very easy.

**What is server-side XPointer?**The XPointer Framework provides an authoritative and extensible interpretation of the semantics of fragment identifiers for XML media types. However, HTTP does NOT transmit the fragment identifier as part of the HTTP request. Therefore XPointer is generally applied by the client, not by the server.   
For example, assuming that http://www.myorg.org/myTripleStore identifies a resource that is willing to negotiate for RDF/XML, then the following is typical of an HTTP request for an RDF/XML representation of that resource and the server's response.  
Request:  
  
GET /myTripleStore HTTP/1.1  
Host: www.myorg.org  
Accept: application/rdf+xml  
  
Response:  
HTTP/1.1 200 Ok  
Content-Type: application/rdf+xml  
  
<rdf:RDF />   
This request asks for the entire triple store, serialized as RDF/XML.   
Server-side XPointer uses the HTTP "Range" header to transmit the XPointer expression to the server. For example, let's assume that the URI of the triple store is the same, but we want to select the subresources identified by the following RDQL query:   
SELECT (?x foaf:mbox ?mbox)  
WHERE (?x foaf:name "John Smith") (?x foaf:mbox ?mbox)  
USING foaf FOR<http://xmlns.com/foaf/0.1/>  
)  
  
In that case the HTTP request, including a copy of the RDQL query wrapped up as an XPointer expression, looks as follows. Note that we have added a range-unit whose value is xpointer to indicate that the value of the Range header should be interpreted by an XPointer processor. Also note the use of the XPointer xmlns() scheme to set bind the namespace URI for the rdql() XPointer scheme. This is necessary since this scheme has not been standardized by the W3C.   
GET /myTripleStore HTTP/1.1  
Host: www.myorg.org  
Accept: application/rdf+xml  
Range: xpointer = xmlns(x:http://www.mindswap.org)x:rdql(  
SELECT (?x foaf:mbox ?mbox)   
WHERE (?x foaf:name "John Smith") (?x foaf:mbox ?mbox)  
USING foaf FOR <http://xmlns.com/foaf/0.1/>  
)  
  
The response looks as follows. The HTTP 206 (Partial Content) status code is used to indicate that the server recognized and processed the Range header and that the response entity includes only the identified logical range of the addressed resource.  
HTTP/1.1 206 Partial Content  
Content-Type: application/rdf+xml  
  
<!-- Only the selected sub-graph is transmitted to the client. --> <rdf:RDF />

**What about non-XML resources?**   
You can use the XPointer Framework with non-XML resources. This is especially effective when your resource is backed by some kind of a DBMS, or when you want to query a data model, such as RDF, and not the XML syntax of a representation of that data model.   
However, please note that the authoratitive interpretation of the fragment identifier is determined by the Internet Media Type. If you want to opt-in for XPointer, then you can always create publish your own Internet Media Type with IANA and specify that it supports the XPointer Framework for some kind of non-XML resource. In this case, you are going to need to declare your own XPointer schemes as well.

**What XPointer schemes are supported in this release?**   
The XPointer integration distributions support shorthand pointers. In addition, they bundle support for at last the following XPointer schemes:  
\* xmlns()  
\* element()  
\* xpath() - This is not a W3C defined XPointer scheme since W3C has not published an XPointer sheme for XPath. The namespace URI for this scheme is http://www.cogweb.org/xml/namespace/xpointer . It provides for addressing XML subresources using a XPath 1.0 expressions.

**How do I configure an XPointer processor?**   
There is no required configuration for the XPointer Framework. The uberjar command line utility provides some configuration options. Applications configure individual XPointer processors when they obtain an instance from an appropriate XPointerProcessor factory method.

**How do integrate XPointer into my application?**   
There are several ways to do this. The easiest is to use the uberjar release, which can be directly executed on any Java enabled platform. This makes it trivial to test and develop XPointer support in your applications, including server-side XPointer. The uberjar release contains a Java class org.CognitiveWeb.xpointer.XPointerDriver that provides a simple but flexible command line utility that exposes an XPointer processor. The XPointer is provided as a command line argument and the XML resource is read from stdin. The results are written on stdout by default as a set of null-terminated XML fragments. See XPointerDriver in the XPointer JavaDoc for more information.   
If you already have a Java application, then it is straight-forward to integrate XPointer support using: org.CognitiveWeb.xpointer.XPointerProcessor You can see an example integration by looking at the XPointerDriver in the source code release.

**How do I implement an application-specific XPointer scheme?**Short answer: Implement org.CognitiveWeb.xpointer.ISchemeProcessor   
The XPointer Framework is extensible. One of the very coolest things about this is that you can develop your own XPointer schemes that expose your application using the data model that makes the most sense for your application clients.   
For example, let's say that you have a CRM application. The important logical addressing units probably deal with concepts such as customers, channels, and products. You can directly expose these data using a logical addressing scheme independent of the actual XML data model. Not only does this let people directly address the relevant concepts using a purpose-built addressing vocabulary, but this means that your addressing scheme can remain valid even if you change or version your XML data model. What a bonus!   
The same approach is being used by the MindSwap laboratory at the University of Maryland to prototype a variety of XPointer schemes for addressing semantic web data.

**How do I support very large resources?**   
You can only do this with server-side XPointer. Further, you need to use (or implement) XPointer schemes that do not depend on a parsed XML document model. Basically, you need to use an XPointer scheme that interfaces with an indexed persistence store (RDBMS, ODBMS, or XML DBMS) which exposes to your ISchemeProcessor the information that it needs to answer subresource addressing requests.   
You will also have to provide shorthand pointer support for your DBMS-based resource. The default shorthand pointer processor assumes that it has access to a parsed XML document, so it can't be used when you have a very large XML resource.

**How do I contribute?**The XPointer implementation is hosted as a SourceForge project. If you want to contribute send an email to one of the project administrators from the project home page.   
The XPointer module uses numerous tests to validate correct behavior of the XPointer processor. One valuable way to contribute is by developing new tests that demonstrate broken behavior. Patches that fix the problems identified by those tests are also valuable, but it is by the tests themselves that we can insure that each release of the XPointer processor will continue to meet the requirements of the various XPointer specifications.

**What's XLink?**   
This specification defines the XML Linking Language (XLink), which allows elements to be inserted into XML documents in order to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to the simple unidirectional hyperlinks of today's HTML, as well as more sophisticated links.   
Definition: An XLink link is an explicit relationship between resources or portions of resources.] [Definition: It is made explicit by an XLink linking element, which is an XLink-conforming XML element that asserts the existence of a link.] There are six XLink elements; only two of them are considered linking elements. The others provide various pieces of information that describe the characteristics of a link. (The term "link" as used in this specification refers only to an XLink link, though nothing prevents non-XLink constructs from serving as links.)

**What are the valid values for xlink:actuate and xlink:show?**   
Don't blame me to put such a simple question here. I saw a famous exam simulator gave wrong answer on this one. Typing them out also help me to remember them. xlink:actuate onRequest, onLoad, other, none xlink:show replace new embed other none

**What is the correct answer of the following question?  
Which of the following is true about XLink and HTML hyperlinks?  
1. XLink can be attached with any element. Hyperlinks in HTML can be attached to only an ANCHOR <A> element.   
2. XLink can refer to a specific location in XML document by name or context with the help of XPointer. HTML ANCHOR<A> does not have capability to point to specific location within an html document.   
3. XLink / XML links can be multidirectional. HTML links are unidirectional.   
4. HTML links are activated when user clicks on them. XLink has option of activating automatically when XML document is processed.**  
  
Only 2 is incorrect, since HTML ANCHOR does have capability to point to specific location within an html document.

**What three essential components of security does the XML Signatures provide?**   
authentication, message integrity, and non-repudiation. In addition to signature information, an XML Signature can also contain information describing the key used to sign the content.

**XLink Processing and Conformance**   
Processing Dependencies: XLink processing depends on [XML], [XML Names], [XML Base], and [IETF RFC 2396]   
Markup Conformance:   
An XML element conforms to XLink if:   
it has a type attribute from the XLink namespace whose value is one of "simple", "extended", "locator", "arc", "resource", "title", or "none", and   
it adheres to the conformance constraints imposed by the chosen XLink element type, as prescribed in this specification.   
This specification imposes no particular constraints on DTDs; conformance applies only to elements and attributes.   
Application Conformance:   
An XLink application is any software module that interprets well-formed XML documents containing XLink elements and attributes, or XML information sets [XIS] containing information items and properties corresponding to XLink elements and attributes. (This document refers to elements and attributes, but all specifications herein apply to their information set equivalents as well.) Such an application is conforming if:   
  
it observes the mandatory conditions for applications ("must") set forth in this specification, and   
for any optional conditions ("should" and "may") it chooses to observe, it observes them in the way prescribed, and   
it performs markup conformance testing according to all the conformance constraints appearing in this specification.

**XLink Markup Design**   
Link markup needs to be recognized reliably by XLink applications in order to be traversed and handled properly. XLink uses the mechanism described in the Namespaces in XML Recommendation [XML Names] to accomplish recognition of the constructs in the XLink vocabulary.