The XML FAQ

Frequently-Asked Questions about the Extensible Markup Language

Editor: Peter Flynn

Silmaril Consultants, Textual Therapy Division

xmlfaq@silmaril.ie

22 February 2017

Copyright © 1996–2015 Silmaril Consultants. Check for details.

Contents

1 Basics **6**

1.1 What is XML? 6
1.2 What is a markup language? 6
1.3 What is XML for? 7
1.4 What is SGML? 8
1.5 What is HTML? 9
1.6 What are all these standards doing? 10
1.7 Is there an XML version of HTML? 11
1.8 What is HTML5? 11
1.9 Aren't XML, SGML, and HTML all the same thing? 12
1.10 Who is responsible for XML? 12
1.11 Why is XML such an important development? 13
1.12 Why not just carry on extending HTML? 13
1.13 Does XML replace HTML? 14
1.14 Why should I use XML? 14
1.15 Where do I find more information about XML? 16
1.16 Where can I discuss implementation and development of XML? 18
1.17 What is the difference between XML and C or C++ or Java? . . 20

2 Existing users **22**

2.1 What do I have to do to use XML? 22
2.2 What does XML look like (inside)? 23
2.3 Should I use XML instead of HTML? 25
2.4 Someone sent me an XML file. How do I read it? 26
2.5 How do I control the formatting of XML? 26
2.6 Where can I get an XML browser? 29
2.7 How do I execute or run an XML file? 32
2.8 Do I have to switch from SGML or HTML to XML? 33

2.9 Can I use XML for ordinary office applications? 33

3 Authors **35**

3.1 Do I have to know HTML or SGML before I learn XML? 35
3.2 How does XML handle white-space in my documents? 36
3.3 Which parts of an XML document are case-sensitive? 37
3.4 How do I convert my information to XML format? 38
3.5 How can I make my existing HTML files work in XML? 40
3.6 How do I convert XML to other file formats? 44
3.7 If XML is just a subset of SGML, can I use my existing SGML tools? 46
3.8 I'm used to authoring and serving HTML. Can I learn XML easily? 46
3.9 Can XML use non-Latin characters? 47
3.10 What's a DTD and where do I get one? 49
3.11 Does XML let me make up my own tags? 50
3.12 How do I create my own document type? 51
3.13 Can a root element type be explicitly declared in the DTD? . . . 53
3.14 I keep hearing about alternatives to DTDs. What's a Schema? . . 54
3.15 How will XML affect my document links? 56
3.16 Can I encode mathematics using XML? 57
3.17 How does XML handle my metadata? 58
3.18 How do I use graphics in XML? 58
3.19 What is parsing and how do I do it in XML? 61
3.20 How do I include one XML file in another? 64
3.21 When should I use a CDATA Marked Section? 65
3.22 How can I handle embedded HTML in my XML? 66
3.23 What are the special characters in XML? 68
3.24 What other markup systems are there? 70

4 Developers **71**

4.1 Where's the spec? 71
4.2 I'm trying to understand the XML Spec: why does it have such difficult terminology? 71
4.3 What are these terms DTDless, valid, and well-formed? 72
4.4 Which should I use in my DTD/Schema, attributes or elements? 76
4.5 What has changed between SGML and XML? 78
4.6 Can I use JavaScript, ActiveX, etc in XML files? 78
4.7 Can I use Java to create or manage XML files? 79

4.8 How do I get XML into or out of my database? 80
4.9 What's a namespace? 81
4.10 What XML software is available? 82
4.11 Software we're missing 84
4.12 What is my information? DATA or DOCUMENT? 86
4.13 Do I have to change any of my server software to work with XML? 88
4.14 Can I still use server-side inclusions? 89
4.15 Can I (and my authors) still use client-side inclusions? 90
4.16 I have to do an overview of XML for my manager/client/investor/advisor. What should I mention? 90
4.17 Is there a conformance test suite for XML processors? 92
4.18 I've already got SGML DTDs: how do I convert them for use with XML? 93
4.19 How do I include one DTD (or fragment) in another? 94
4.20 How can I include a conditional statement in my XML? 95
4.21 What's the story on XML and EDI? 96

5 Appendices **98**
5.1 References 98
5.2 How far are we going? 99
5.3 Not the XML FAQ 101
5.4 Lost XML software 117

Chapter 1

Basics

General information about XML

1.1 What is XML?

The Extensible Markup Language.

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 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).

1.2 What is a markup language?

A way of describing what's what in a document.

A markup language is a set of words and symbols for describing the **identity**

or **function** of the component parts 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 markup with a stylesheet to transform the document into output for screen, print, audio, video, Braille, or reprocessable data formats.

Some markup languages (especially those used in wordprocessors) only describe **appearances** instead ('this is italics', 'this is bold', 'this has 3mm space below', etc), so these systems can only be used for display, and are not easily re-usable for anything else.

XML is sometimes referred to as 'self-describing' because the names of the markup elements can represent the type of content they hold (eg `title`, `chapter`, `link`, etc).

1.3 What is XML for?

XML is for identification, transmission, and storage.

> 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. Bray, Paoli, Sperberg-McQueen, Maler, and Yergeau, 2004

Despite early attempts, browsers never allowed other SGML, only HTML (although there were plugins). Browsers also allowed (even encouraged) authors to corrupt or break their HTML in order to make it 'easier'. This enabled HTML to become widespread, but 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 on its own for Web pages because browsers still don't provide reliable support for formatting it. Common uses for XML include:

Information identification You can define your own markup, so 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 information across any platforms. Because it is backed by an international standard, it will remain accessible and processable as a data format.

Information structure XML structures can 'nest', so they can be used to store and identify any kind of hierarchical information, especially long, deep, or complex document sets or data sources, which makes it ideal for an information-management back-end to serving the Web. This is one if its most common Web applications, with a transformation system to serve it as HTML.

Publishing The original goal of XML as defined in the quotation at the start of this section. Combining the three previous topics (identity, storage, and 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 (eg Braille, Audio, etc) from a single source document by using the appropriate stylesheets.

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 because of their proprietary or secret data formats. By providing a *lingua franca* for data identity and structure, XML 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 like smartphone apps use XML for data management and transmission, and the web browser or app for display and interaction.

1.4 What is SGML?

Standard Generalized Markup Language, ISO 8879:1986

SGML is the Standard Generalized Markup Language (ISO 8879:1986), the international standard for defining markup to describe 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.html; and Robin Cover's SGML Web pages are at http://www.oasis-open.org/cover/general.html. For a brief summary and list of online and print resources, see http://wiht.link/SGML-intro. For a little light relief, try Joe English's 'Not the SGML FAQ' at http://www.flightlab.com/~joe/sgml/faq-not.txt.

SGML is very large, powerful, and complex. It was in heavy industrial and commercial use for nearly two decades (and still is, in some places), and there is a significant body of expertise and software that grew up with it, most of which is now available for XML.

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.

1.5 What is HTML?

HyperText Markup Language, RFC 1866, the language of Web pages.

HTML is the HyperText Markup Language (originally RFC 1866, now HTML 4.01), which started as a small application of SGML for the Web, originating with Tim Berners-Lee at CERN in 1989–90.

It originally defined a very simple class of report-style documents, with section headings, paragraphs, lists, tables, and illustrations, and a few informational elements, but very few presentational elements Flynn [Flynn], 1995, plus some hypertext and multimedia. See the question on extending HTML.

The current W3C Recommendations are the XML version of HTML, XHTML; and the more mobile- and media-aware variant, HTML5. There is a separate HTML-and-XHTML FAQ maintained by Steven Pemberton at http://www.w3.org/MarkUp/2004/xhtml-faq

1.6 What are all these standards doing?

They all underpin XML, which in turn underpins others

The original standard is Standard Generalized Markup Language, ISO 8879:1986. This is an ISO standard, so it's worldwide and international (see note below). XML itself is not an ISO standard: it doesn't need to be, as it is

1. an application profile of SGML, enabled by the 'WebSGML Adaptation' Technical Corrigendum (2) to ISO 8879;
2. a W3C Recommendation, which means it has been through a formal process roughly equivalent to standardisation

.

On the other hand, HTML does have an ISO reference standard, ISO/IEC 15445:2000(E), but HTML is a concrete markup language, an actual application of SGML and XML, whereas XML and SGML are actually metalanguages used to define other markup languages.

Other XML applications like DocBook and TEI have their own processes for development. DocBook is standardised by the DocBook Technical Committee of OASIS (Organization for the Advancement of Structured Information Standards); the TEI Guidelines are managed by the TEI Technical Council of the Text Encoding Initiative.

The process of developing a standard document interchange specification (which is basically what these all are) is long and complex. Ken Holman has written an essay on how the OASIS technical committee process supports a group of members from an industry or economic sector in creating business exchange document specifications.

> ISO standards like SGML are governed by the International Organization for Standardization in Geneva, Switzerland, and voted into or out of existence by representatives from every country's national standards body.
>
> If you have a query about an international standard, you should contact your national standards body for the name of your country's representative on the relevant ISO committee or working group.

If you have a query about your country's representation in Geneva or about the conduct of your national standards body, you should contact the relevant government department in your country, or speak to your public representative.

The representation of countries at the ISO is not a matter for this FAQ. Please do not submit queries to the editor about how or why your country's ISO representatives have or have not voted on a specific standard.

1.7 Is there an XML version of HTML?

Yes, XHTML from W3C

The W3C Recommendation is 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 were intended to 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).

1.8 What is HTML5?

A new version of XHTML for use in interactive and mobile environments.

To enable XHTML to be more widely used in interactive and handheld environments, the W3C has developed a revision of XHTML called HTML5, which adds more presentational elements, better support for multimedia and mobile devices, and has a less rigid document model.

There is an explanation from Elliotte Rusty Harold, and a HTML5 FAQ from the WhatWG.

HTML5 in conjunction with CSS3 is now regarded as the prevailing standard for web development.

1.9 Aren't XML, SGML, and HTML all the same thing?

No, SGML and XML are metalanguages. HTML, XHTML, and HTML are all applications of them.

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' medical and clinical records to musical notation. SGML is very large and complex, however, and 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 based on SGML, and XML files may still be processed in the same way as any other SGML file (see the question on XML software).

HTML, XHTML, and HTML5 are just those XML applications most frequently used on the Web.

Technical readers may find it more useful to think of XML as being SGML−− rather than HTML++.

(Ed: In respect of this last paragraph, see Question 1.17 on page 20 and Question 2.7 on page 32.)

1.10 Who is responsible for XML?

The W3C

XML is a Recommendation 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.

1.11 Why is XML such an important development?

It overcomes the inflexibility of HTML and the complexity of SGML

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.

1.12 Why not just carry on extending HTML?

HTML is already too overburdened with proprietary add-ons.

HTML was already weighed down 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 organisations to create their own customised 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, mathematics, genealogy, etc).

HTML as originally conceived is now well beyond the limit of its usefulness as a way of describing information, and while XHTML and HTML5 will continue to play an important role for the content they can represent, many new applications require a more robust and flexible infrastructure.

1.13 Does XML replace HTML?

No.

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 on the web, and the current versions of HTML (XHTML and HTML5) are in XML syntax.

XML is designed to make the writing of processing software much easier than with SGML, which is what the original HTML was based on.

1.14 Why should I use XML?

It's a robust, durable, manipulable, and free format for information identification, storage and transfer.

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' your information (well, at least manipulate as if they could understand it).

- XML allows sets of documents which are all the same type to be created and handled 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 or data content of your documents, only the structure of them. If you want styling or formatting, see Question 2.5 on page 26.]

- 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 (persistent) 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 any other format with no loss of information.

- XML lets you separate form (appearance) 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).

- Any of the Design Goals listed in the XML Specification.

Peter Flynn writes:

> WHY NOT JUST USE WORD OR NOTES?
>
> Restricted proprietary data formats are unsuitable for durable public information.
>
> Information on a network which connects many different types of computer has to be usable on all of them. Public information in particular cannot afford to be restricted to one make or model or manufacturer, or to cede control of its data format to private hands. It is also helpful for such information to be in a form that can be reused in many different ways, as this will minimise wasted time and effort. Proprietary data formats, no matter how well documented or publicised, are simply not an option: their control still resides in private hands and they can be changed or withdrawn arbitrarily without notice.
>
> SGML is the international standard for defining this kind of application, and was therefore the natural choice for XML, but those who need an alternative based on different software for other purposes are entirely free to implement similar services using such a system, especially if they are for private use.

1.15 Where do I find more information about XML?

Online and offline resources

Online, there's the XML Specification and the ancillary documentation available from the W3C; Robin Cover's XML Cover Pages with an extensive list of online reference material and links to software; and a summary and condensed FAQ from Tim Bray; and thousands of reference resources available by typing 'xml' into Google or other search engine.

For offline resources, see the lists of books, articles, and software for XML in Robin Cover's XML Cover Pages (articles and books). That site should always be your first port of call for archived resources.

For access to experts, particularly in consultancy and training, see the XML Guild, whose members form a consortium of some of the best independent XML consultants in the world.

The events listed below are the ones I have been told about. Please mail me if you come across others: there are many other XML events around the

world, and most of them are announced on the mailing lists and newsgroups.

Upcoming Events

TEI Conference The 2016 TEI Conference took place at the Austrian Academy of Sciences in Vienna, hosted by the Austrian Centre of Digital Humanities from 26˘30 September 2016.

TEI Conference The 2017 TEI Conference will be held in Victoria, BC (details yet to be advised).

WWW 2016 The 25th International World Wide Web Conference took place at the Université de Québec à Montréal, Canada on April 11˘15, 2016.

XML London XML London (XML and Semantic Web Conference) was held in University College London on June 4–5, 2016.

TUG 2016 The TEX Users Group 2016 conference was held in Toronto, Canada, on 25–27 July 2016.

TUG 2017 The TEX Users Group 2017 conference will be a joint meeting with GUST to celebrate the 25th anniversary of BachoTEX, and will take place in Bachotek, Poland, on April 29˘May 3, 2017. Although primarily related to typesetting, many aspects of LATEX are closely related to the use of XML and the handling of structured documents.

Balisage The Balisage markup conference was in Bethesda MD on August 2–5.

Balisage The Balisage markup conference is the principal technical meeting specifically about markup, including both XML and TEX: it will be in Bethesda MD on August 1˘4, 2017, with a pre-conference Symposium on July 31.

XML Summerschool The annual XML Summer School will be held in St Edmund Hall, Oxford on 18–22 September 2017, with an afternoon Intro to XML on Sunday 17th. This week-long event covers everything from an introduction for the beginner up to XML in publishing, transformation with XSLT2 and XQuery, and the use of Linked Data.

XML Prague XML Prague 2017 will be held on 9–11 February 2017 at the University of Economics, Prague. Videos of earlier meetings are available on their YouTube channel (link on site).

XML Amsterdam XML Amsterdam 2015 was held on November 5 at the Centrum Wiskunde & Informatica (CWI) Science Park, Amsterdam. The next XML Amsterdam will be in 2017 (date to be advised).

Digital Humanities Conference 2017 The Digital Humanities Conference 2017 will be held in Montréal on 8–11 August 2017, hosted by McGill University and the Université de Montréal. The theme of the DH2017

is 'Access'. The use of XML (TEI) markup is a strong feature of DH conferences.
DH2018 will be held in Mexico City, hosted by the El Colegio de México and the Universidad Nacional Autónoma de México (UNAM) in alliance with the Red de Humanidades Digitales (date to be announced).

1.16 Where can I discuss implementation and development of XML?

On mailing lists, Usenet newsgroups, web-based bulletin-boards, and IRC channels

The principal online support media are web forums and mailing lists. Usenet newsgroups for XML exist but are rarely used now, although searching them via Google Groups can turn up a lot of useful information. The IRC network is also used to some extent, and most individual XML projects and programs have their own topic-specific bulletin-boards on their web sites. There is also an unknown number of related question-and-answer forum sites which are findable using search engines.

For off-line support, see Question 1.15 on page 16 for details of conferences and summerschools.

- The most active web forum seems to be StackOverflow: oddly there seems to be no special-purpose StackExchange XML site like there is for other topics (eg LaTeX). The W3Schools XML Forum (wholly unrelated to the W3C) also seems to have recent traffic, as do DevShed and CodingForums. A Google search will turn up many others with lower levels of traffic.

- The main Usenet newsgroup is comp.text.xml, although it is much less used than formerly. Ask your Internet Provider for access to Usenet, or use a Web interface like the searchable archive maintained by Google. If your browser or mailer doesn't provide newsreading facilities, install one that does, or (better) use a standalone newsreader.

  The comp.text.sgml newsgroup is for all practical purposes no longer used. The Microsoft-specific newsgroups are being phased out in

favour of web-based forums hosted by Microsoft themselves.

- The general-purpose mailing list for public discussion is XML-L: to subscribe, visit the Web site and click on the link to join.

- For those developing software components for XML there is the xml-dev mailing list. You can subscribe by sending a 1–line mail message to `xml-dev-request@lists.xml.org` saying just `SUBSCRIBE`. Note that this list is for those people actively involved in developing resources for XML. It is not for general information about XML (use the XML-L list above for that).

- The XSL-List is for for discussing XSL (both XSLT and XSL:FO). For details of how to subscribe, see `http://www.mulberrytech.com/xsl/xsl-list`.

- There is a long list of other discussion groups, mailing lists, and forums on Robin Cover's site at `http://xml.coverpages.org/lists.html`.

Andrew Watt writes:

> There is a mailing list specifically for XSL-FO only, on eGroups.com. You can subscribe by sending a message to `XSL-FO-subscribe@egroups.com`.
>
> Be aware that the Yahoo E-Groups XSL-FO list sends out regular automated spam to non-members falsely claiming that they have asked to join.

Gianni Rubagotti writes:

> A new Italian mailing list about XML is born: to subscribe, send a mail message without a subject line but with text saying `subscribe XML-IT` to `majordomo@ananas.usr.dsi.unimi.it`. Everyone, Italian or not, who wants to debate about XML in our tongue is welcome.
>
> Gianni also runs the Humanities XML List.

J-P Theberge writes:

> A French mailing list about XML has been created. To subscribe, send `subscribe` to `xml-request@trisome.com`.

Murata Makoto writes:

> Please mention this mailing list to your colleagues who use RELAX NG. Go to: http://groups.yahoo.com/group/rng-users/.
>
> MAILING LISTS
>
> When you join a mailing list you will be sent details of how to use it. Please Read The Fine Documentation because it contains important information, particularly about what to do if your company or ISP changes your email address.
>
> Please note that there is a lot of inaccurate and misleading information published in print and on the Web about subscribing to and unsubscribing from mailing lists. Don't guess: Read The Fine Documentation.

1.17 What is the difference between XML and C or C++ or Java?

C and Java are for writing programs; XML is for storing information.

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 Fool!",
    f.put days('31102011','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 xml:id="RQ778">Ringtown Fasteners Ltd</maker>
  <notes>An <tool xml:id="GH25"/>angle-nosed insertion tool</tool> is
    required for the removal and replacement of this part.</notes>
```

```
</part>
```

On its own, an 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. See also the question about how to run or execute XML files.

XSLT2 AND XSL:FO

The water is muddied by the fact that the most popular transformation processing languages (XSLT2 and XSL:FO) are actually written in XML syntax, because they are declarative, not procedural. In these special cases you *can* be said to 'execute' an XML file, by running a processing application like *Saxon*, which compiles the directives specified in the files into Java bytecode to process XML documents. In this sense, you could compare them with other programming languages, but you would be comparing the language facilities, not the XML syntax in which they are written.

William Hammond writes:

(in article `<i7ll1362ib.fsf@hilbert.math.albany.edu>`)

SGML is a category of 'document types', with a configurable shared syntax, most of which (like classic HTML) cannot be compiled to produce executable programs. XML is a subcategory of SGML with syntactic restrictions. For example, with XML the vocabulary of a document type is always case sensitive, while with SGML it may be either case sensitive or case insensitive. So, for example, classic HTML is an SGML document type, and XHTML+MathML is an XML document type.

While some document types correspond to document markup languages, other document types (like a CTAN catalog entry) are just for structured data[...]

I doubt seriously, however, that a computer language like C is in any reasonable sense equivalent to an SGML document type.

Chapter 2

Existing users

(including everyone who uses a browser)

2.1 What do I have to do to use XML?

To read it: use any modern web browser; to create it: use an XML editor.

For the average user of the Web, you don't need anything except a browser which works with XML (see the question about browsers). Remember new XML-related facilities are being invented or implemented all the time (see the W3C web site), so some recent features may not work in all browsers yet.

You can use XML-conformant browsers to look at some of the stable XML material, such as Jon Bosak's Shakespeare plays and the molecular experiments of the Chemical Markup Language (CML). There are some more example sources listed at `http://xml.coverpages.org/xml.html#examples`, and you will find XML (particularly in the guise of XHTML) being introduced in places where it won't break older browsers.

If you want to start preparations for creating your own XML files, see the questions in the Authors' Section and the Developers' Section, particularly the question on Question 4.10 on page 83.

2.2 What does XML look like (inside)?

Pointy brackets like HTML

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 an optional **Prolog**, which can have two (optional) parts:

1. The **XML Declaration**:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   ```

   This specifies that this is an XML document and that it uses the UTF-8 character repertoire (the default; others are available but support is only mandated for UTF-8);

2. A **Document Type Declaration** if you are using a DTD:

   ```
   <!DOCTYPE report SYSTEM "http://sales.acme.corp/dtds/salesrep.dtd">
   ```

   which identifies the type of document (here, 'report') 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>
  <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 DTD, and maybe an **internal subset** (local DTD changes in [square brackets] within the Document Type Declaration like the `ENTITY` declaration below); and an arbitrarily complex nested structure:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE titlepage
  SYSTEM "http://www.foo.bar/dtds/typo.dtd"
[<!ENTITY % active.links "INCLUDE">]>
<titlepage xml: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.foo.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 are usually unformatted because they are for machine consumption, not for human reading, and they 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-XML:ID="BAC352437484">
    <ORDER-UPDATE-DELTA-MODIFICATION-VALUE ORDER-UPDATE-ITEM="56"
```

```
    ORDER-UPDATE-QUANTITY="2000"/>
  </ORDER-UPDATE-DELTA-MODIFICATION-DETAIL>
</ORDER-UPDATE>
```

2.3 Should I use XML instead of HTML?

Yes if you need robustness, accuracy, and persistence.

XML allows authors and providers to design their own document markup instead of being limited by HTML. Document types can be explicitly tailored to an application, so the cumbersome fudging and poodlefaking that has to take place with HTML becomes a thing of the past: your markup can always say what it means. Trivial example:

```
<date YYYY-MM-DD="2005-12-26">last Monday</date>
```

- Information content can be richer and easier to use, because the descriptive and hypertext linking abilities of XML are much greater than those available in HTML.
- XML can provide more and better facilities for browser presentation and performance, using XSLT and CSS stylesheets;
- It removes many of the underlying complexities of SGML-format HTML (which led to them being ignored and broken) in favour of a more flexible model, so writing programs to handle XML is much easier than doing the same for all the old broken HTML.
- Information becomes more accessible and reusable, because the more flexible markup of XML can be used by any XML software instead of being restricted to specific manufacturers as has become the case with HTML.
- XML files can be used outside the Web as well, in existing document-handling environments (eg publishing).

If your information is transient, or completely static *and* unreferenced, or very short and simple, and unlikely to need updating, HTML may be all you need.

2.4 Someone sent me an XML file. How do I read it?

Open it in an XML browser or XML editor.

If the file is well-formed or valid XML, you can just open it with any XML-conformant browser (see Question 2.1 on page 22 and Question 2.6 on page 29). This will display the file in an unformatted view, showing all the markup in a format that lets you fold up or unfold the nested hierarchy (click on the little plus and minus symbols), which will at least let you read something.

If the file contains a link to an XSLT or CSS stylesheet (and the stylesheet was provided or is web-accessible) then the browser should format the file in a readable manner (but beware that in-browser formatting is not robust).

If you want to edit the file, you need an XML editor (see Question 4.10 on page 83). Unless you are very skilled with pointy-bracket markup, do *not* try to edit XML files with non-XML editors.

2.5 How do I control the formatting of XML?

Use CSS or an XSLT2 stylesheet.

In HTML, default styling was built into the browsers because the tagset of HTML was predefined and hardwired into browsers. This is still true for XHTML and HTML5 to some extent. In other 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.

This transformation is usually done by the document owner, on their server, so you just get the HTML anyway, possibly unaware that it was XML originally. But it is also possible to use the (rather limited) built-in XSLT 1.0 transformer in some browsers, and server operators can now also use *Saxon CE*, which is a downloadable in-browser version of XSLT2.

Mike Brown writes:

> XSLT is an XML document processing language that uses source code that happens to be written in XML. An XSLT document declares a set of rules for an XSLT processor to use when interpreting the contents of an XML document. These rules tell the XSLT processor how to generate a new XML-like data structure and how that data should be emitted — as an XML document, as an HTML document, as plain text, or perhaps in some other format.
>
> This transformation can be done either inside the browser, or by the server before the file is sent. Transformation in the browser offloads the processing from the server, but may introduce browser dependencies, leading to some of your readers being excluded. Transformation in the server makes the process browser-independent, but places a heavier processing load on the server.

As with any system where files can be viewed at random by arbitrary users, the author cannot know what resources (such as fonts) are on the user's system, so the same care is needed as with HTML using fonts. To invoke a stylesheet from an XML file for standalone processing in the browser, include one of the stylesheet declarations:

```
<?xml-stylesheet href="foo.xsl" type="text/xsl"?>
<?xml-stylesheet href="foo.css" type="text/css"?>
```

(substituting the URI of your stylesheet, of course). See `http://www.w3.org/TR/xml-stylesheet/` for the full details. The Cascading Stylesheet Specification (CSS) provides a simple syntax for assigning styles to elements, and has been implemented in most browsers.

Dave Pawson maintains a comprehensive XSL FAQ at `http://www.dpawson.co.uk/xsl/`, and his book Pawson [Pawson], 2002 [the Fox book] is available from O'Reilly. XSL uses XML syntax (an XSL

stylesheet is just an XML file) and has widespread support from several major browser vendors (see the questions on browsers and other software). XSL comes in two flavours:

- XSL itself, which is a pure formatting language, outputting a Formatted Objects (FO) file, which needs a text formatter like FOP, XEP, or others to create printable (PDF) output (but see Question 2.5). Currently I am not aware of any Web browsers which support direct XSL rendering to PDF;
- XSLT (T for Transformation), which is a language to specify transformations of XML into HTML either inside the browser or at the server before transmission. It can also specify transformations from one vocabulary of XML to another, and from XML to plaintext (which can be any format, including RTF and LaTeX).

All current versions of Microsoft Internet Explorer, Firefox, Chrome, Mozilla, Safari, and Opera handle XSLT 1.0 inside the browser. Beware obsolete browsers like MSIE5.5 which needs some post-installation surgery to remove the long-obsolete WD-xsl and replace it with the current XSL-Transform processor.

WYSIWYG for XSL

There have been attempts to produce pseudo-WYSIWYG editors for creating XSL[T] stylesheets, but they have mostly been restricted to simple mapping between input elements and output elements (eg a DocBook `para` to a HTML `p`). Anything beyond this seems likely to fail because of the infinite complexity of what people want to do with their information. If you have access to the ACM database, see the paper by Pietriga, Vion-Dury, and Quint on VXT, from the ACM DocEng'01 (Atlanta) Proceedings.

Generating HTML on the server

There is a growing use of server-side processors like Cocoon and others, which let you create, store, and manage your information in XML but serve it auto-converted to HTML or some other format, thus allowing the output to be used by any browser. XSLT is also widely used to transform XML into non-SGML formats for input to other systems (for example to transform XML into LaTeX for typesetting).

Alternatives to XSL:FO

Instead of generating PDF via an FO processor, it is possible to use XSLT2 to transform XML to LaTeX for typesetting PDF (as is done for the print versions of this FAQ, from DocBook to LaTeX). This has the advantage of being able to make use of LaTeX's extensive library of prewritten formatting modules ('packages'), which avoids much of the wheel-reinventing currently required with XSL:FO.

Alternatively, David Carlisle's *xmltex* reads XML directly, offering another practical if experimental solution to typesetting XML. One use of a TeX system that can typeset XML files is as a backend processor for XSL:FO, serialised as XML. Sebastian Rahtz's PassiveTeX uses *xmltex* to achieve this end.

The TeX FAQ is at `http://www.tex.ac.uk/faq`. Silmaril maintains the online version of Peter Flynn's book on LaTeX, Formatting Information, which has some examples of XSLT2 conversion Flynn, 2014.

SGML systems used a similar stylesheet mechanism: some of the common ones were the FOSI (Formatted Output Specification Instance), which was standard in defence and industrial engineering applications, especially when using the Arbortext editor (Adept, then Epic, probably something else next week); the DynaText/DynaWeb stylesheet used in SGML publishing to the web; and the Synex stylesheet used in browsers based on the Synex engine (eg Panorama, whose styling interface was partly adopted in XMetaL), the expertise of whose designers persists in the DocZilla browser.

2.6 Where can I get an XML browser?

All modern browsers support XML

Current state of existing browser support for XML (1 August 2014):

- Current versions of Microsoft Internet Explorer, Firefox, Safari, Chrome, Mozilla, and Opera all appear to support XML with CSS and/or XSLT 1.0 stylesheets. The editor would welcome additional information and corrections.

- Don't use Netscape (any version), Internet Explorer 6 or earlier, or any early versions of Mozilla if you want XML support: they either don't have it or were hopelessly broken. Upgrade to a modern browser as soon as possible.

The remainder of this list is of historical interest only.

- Microsoft Internet Explorer 5.0 and 5.5 handled XML, processing it by default using a built-in stylesheet written in a Microsoft-specific, obsolete predecessor of XSLT called XSL (not to be confused with the real XSLT). The output of the stylesheet is DHTML, which, when rendered in the browser, shows a coloured, syntax-highlighted version of the XML document, with collapsible views. If the XML document references a stylesheet, that stylesheet will be used instead, within the limitations of MSIE's incomplete implementation of CSS. MSIE 5.0 and 5.5 can also use stylesheets in another obsolete syntax called WD-xsl, which should be avoided. These versions can be upgraded to support real XSLT: see the MSXML FAQ.

  MSIE 6.0 and later use real XSLT 1.0, but can use both the obsolete syntaxes as well.

- Mozilla Firefox 0.9 up, Netscape 6 and 7 (there is no Netscape 5), and Galeon all have full XML support with XSLT and CSS. In general, Firefox is more robust than MSIE, and provides better standards adherence.

  I have a user report that Netscape 4.6 and 4.8 supports XML, but no independent verification.

- The authors of the former MultiDoc Pro SGML browser, CITEC (whose engine was also used in Panorama and other browsers), joined forces with Mozilla to produce a multi-everything browser called DocZilla, which read HTML, XML, and SGML, with XSLT and CSS stylesheets. This ran under Windows and Linux and was at release 1.0 at the time it became unavailable. This was by far the most ambitious browser project, and was backed by very solid markup-handling expertise.

I have less information on the XML capabilities of the Mac OS X browser Safari, which is based on the KHTML engine used in Konqueror. Konqueror itself does not appear to support XML or XSLT (at least in KDE under Fedora Core, for example), but Safari 1.3.2 (v312.6) under OS 10.3 did provide partial support for XML, but does not honour an external DTD modified by an internal subset (thanks to John Haynie for testing this).

Mike Brown writes:

> The concept of 'browsing' is primarily the result of HTML having the semantics that it does. In an HTML document there are sections of text called anchors that are 'hyperlinked' to other documents that might be at remote locations on a network or filesystem. HTML documents provide cues to a web browser regarding how the document should be displayed and what kind of behaviours are expected of the browser when the user interacts with it. The HTML specification provides many suggestions and requirements for the browser, and provides specific meanings for many different examples of markup, such as the fact that an `<img>` element refers to an image that should be retrieved by the browser and rendered inline with the adjacent text.
>
> Unlike HTML, XML does not have such inherent semantics at all. There is no prescribed method for rendering XML documents. Therefore, what it means to 'browse' XML is open to interpretation. For example, an XML document describing the characteristics of a machine part does not carry any information about how that information should be presented to a user. An application is free to use the data to produce an image of the part, generate a formatted text listing of the information, display the XML document's markup with a pretty color scheme, or restructure the data into a format for storage in a database, transmission over a network, or input to another program.
>
> However, despite the fact that XML documents are purely descriptive data files, it is possible to 'browse' them in a sense, by rendering them with stylesheets. A stylesheet is a separate document that provides hints and algorithms for rendering or transforming the data in the XML document. HTML users may be familiar with Cascading Style Sheets (CSS). The CSS stylesheet language is general and powerful enough to be applied to XML documents, although it is oriented toward visual rendering of the document and does not allow for complex processing of the document's data. By associating an XML document with a CSS stylesheet, it may be possible to load an XML document in a CSS-aware web browser, and the browser may be able to provide some kind of rendering of it, even if the browser does not otherwise know how to read and process XML documents. However, not all web browsers will load an XML document correctly, and they are not required to recognise the XML markup that associates the document with a stylesheet, so one cannot assume that XML documents can be opened with just any web browser.
>
> A more complex and powerful stylesheet language is XSLT, the

Transformations part of the Extensible Stylesheet Language, which can be used to transform XML to other formats, including HTML, other forms of XML, and plain text. If the output of this transformation is HTML, it can be viewed in a web browser as any other HTML document would.

The degree of support for XML and stylesheets in web browsers varies greatly. Although loading and rendering XML in the browser is possible in some cases, it is not universally supported. Therefore, much XML content on the web is translated to HTML on the servers. It is this generated HTML that is delivered to the browsers. Most of Microsoft's web site, for example, exists as XML that is converted to HTML on the fly. The web browser never knows the difference.

See also the notes on software for authors and XML for developers, and the more detailed list on the XML pages in the SGML Web site at `http://xml.coverpages.org/`.

2.7 How do I execute or run an XML file?

Not a meaningful question. XML is a data format, not a programming language.

You can't and you don't. XML itself is not a programming language, so normal XML documents 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 XML browser.

XSLT2 AND XSL:FO

The water is muddied by the fact that the most popular transformation processing languages (XSLT2 and XSL:FO) are actually written in XML syntax, because they are declarative, not procedural. In these special cases you *can* be said to 'execute' an XML file, by running a processing application like *Saxon*, which compiles the directives specified in the files into Java bytecode to process XML documents. In this sense, you could compare them with other programming languages, but you would be

comparing the language facilities, not the XML syntax in which they are written.

2.8 Do I have to switch from SGML or HTML to XML?

Not if you don't want to

No, existing HTML applications software will continue to work with existing files. There is now very little SGML left, but what there is will continue to work with its existing software. But as with any enhanced facility, if you want to view or download and use XML files, you will need to use XML-aware software. Much more has been developed for XML than there ever was for SGML, so almost all current effort is concentrated on XML (and XHTML and HTML5).

2.9 Can I use XML for ordinary office applications?

Yes, use MS-Office, Libre Office, Open Office, WordPerfect, or others.

Yes, most office 'productivity' suites already do this, and save their documents along with stylesheets, images, etc in a Zip file:

- Libre Office, OpenOffice, and NeoOffice (Mac) have been saving their files as XML by default for many years (`.odt`, `.ods`, and `.odp` file types are all Zip files). The packages are essentially variant implementations of OpenOffice, and all comprise a wordprocessor, spreadsheet, presentation software, and a vector drawing program, and they share related Schemas. The Office Document Format (ODF) was the first official International Standard (ISO/IEC 26300) for office documents. All of them can read and write Microsoft Office files.

- Corel's WordPerfect suite has shipped with a fully-fledged XML editor for many years (which also does full SGML as well). It can save the formatted output as a Microsoft Word `.doc` file, but it uses its own stylesheet technology to format documents, not XSLT2 or CSS. It can

also save its own (WordPerfect) document format to an XML representation.

- The AbiWord wordprocessor (all platforms) can open and edit Word and OpenOffice documents, but it can also save them in DocBook XML or even LaTeX format (although it does not provide native XML editing) which makes it an excellent converter.

- Microsoft Office 2003 provided a 'Save As…XML' to all parts of the suite except Powerpoint, using WordML to represent the visual appearance of the document, although it will preserve style names if they are in use.

  Office 2007, 2010, and later all save natively as XML documents (`.docx`, `.xlsx`, and `.pptx` file types, which are Zip files). They use Office Open XML (OOXML, similar but unrelated to WordML) which is Microsoft's equivalent to ODF. It is a parallel ISO standard.

  Word 2003 shipped with a real XML editor as well, supporting other W3C Schemas as well as its own (but not DTDs), and this also provided a method for binding element types to Word's named styles (like Microsoft's earlier product SGML Author for Word did).

- Avoid Microsoft's 'Works' package, as it is incompatible both with XML and all other Office software.

- I have no information on Lotus office products.

There is more detail under 'XML File Formats for Office Documents' in the XML Cover Pages which briefly describes and points to further information on: GNOME Office, KOffice, Microsoft XDocs, OASIS TC for Open Office XML File Format, 1DOK.org Project, and OpenOffice.org XML File Format.

Chapter 3

Authors

(including writers of HTML and Web page owners)

3.1 Do I have to know HTML or SGML before I learn XML?

No, but it's useful.

You don't need any foreknowledge, but it's useful because a lot of XML concepts, terminology, and practice derive from two decades' experience of SGML (and the systems which came before it).

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 SGML or XML.

For most practical purposes you don't need any knowledge of SGML or HTML: only if you intend delving into the internals of document type design.

3.2 How does XML handle white-space in my documents?

Parsers keep it all. It's up to the application to decide what to do with it.

All white-space, including linebreaks (Mac CR, Win CR/LF, Unix LF), 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). The parser identifies the context in which the white-space was found (element content, character data content, or mixed content), if this information is available, 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.

This is one of the few really radical changes from SGML, where all white-space in element content was discarded by the parser before it got anywhere near the application. See Question 3.2 on the facing page below for why.

There are two different types of white-space:

- **insignificant white-space** (discardable white-space) which occurs between structural elements in element content. This is space which occurs where only other elements are allowed, where text never occurs. It is usually inserted automatically by an editor or manually by an author to help with the visual clarity of the markup, and often has nothing to do with spacing you see when the document is processed or formatted. In XML, this space *will* get passed to the application (in SGML it got suppressed, which is why you can put all that extra space in old-style HTML documents and not worry about it);
- **significant white-space** which occurs inside elements which can contain only text (character data content, like a HTML `title`) or text and markup mixed together (eg paragraphs). In XML, this space will still get passed to the application exactly as under SGML.

In both cases, it is the application's responsibility to handle the space correctly (XSLT2, for example, provides a `strip-space` instruction to specify

how to handle it). The parser must therefore inform the application that white-space has occurred in element content, if it can detect it, so that it can be discarded. (Users of SGML will recognise 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.

Peter Flynn writes:

> WHY?
>
> In SGML, a DTD is compulsory, always. A parser therefore always knows in advance whether white-space has occurred in element content (and can therefore be discarded) or in mixed content or character data (where it must be preserved). XML allows processing without a DTD or Schema, where it may be impossible to tell whether space should be discarded or not, so the general rule was imposed that *all* white-space must be reported to the application.

3.3 Which parts of an XML document are case-sensitive?

All of it, both markup and text.

All of an XML document is case-sensitive. This is significantly different from HTML and most other SGML applications, where the default was to ignore case. It was done to allow markup in non-Latin-alphabet languages, and to obviate problems with case-folding in writing systems which are inherently 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 `xlink:href="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.

3.4 How do I convert my information to XML format?

Write or use a converter in a language that understands XML

If the source file format has some kind of consistent and recognisable structure, even simple line-breaks or spacing, it's usually possible to write pattern-matching routines in many languages to isolate the information falling into such patterns and output it with tags around it.

XSLT2 has a pattern-matching syntax for doing exactly this kind of 'up-conversion', and other processors such as *Omnimark* offer similar

features. Such conversions may also create a temporary 'half-way' format to which a second conversion is applied to create the final XML format.

If the source files are in a known format (CSV, for example), there may be existing routines available for download or purchase which can create some XML format. A second XML-to-XML conversion can then be used to create the final format required.

Database management systems may have built-in 'export-to-XML' routines which can create a similar 'half-way' format for subsequent conversion (see Question 4.8 on page 80 for an example).

If the information is completely unformatted, or so badly or inconsistently formatted that automated conversion is impossible, it will have to be edited by hand into XML format. Wordprocessor documents are the classic example of this. There are companies which specialise in this kind of work, particularly around the Pacific Rim, who have long experience in dealing in all kinds of weird and wonderful formats, and will send fully-fledged XML back to you.

Two formats frequently requested as sources are better-supported:

LaTeX Well-formed LaTeX documents (those that do not use homebrew macros, especially those using plain TeX or obsolete commands) can be converted using the TeX4ht package. At the time of writing (2015) this is unsupported since the untimely death of its author, but is fully functional.

TeX4ht can convert to HTML and ODF (*OpenOffice* format) in various ways, so the resulting file can easily be opened in *OpenOffice* and saved as a `.docx` file. There are command-line options for the `oowriter` program (or `lowriter` if you are using *Libre Office*) which allow for scripted bulk conversion.

Other facilities are available in some editors and online services (such as the blogs and forums which support LaTeX formatting in web pages). These may also be used for conversion.

Microsoft *Word* *Word* (`.docx`) files are Zip files containing XML documents along with the associated images and stylesheets. By default, *Word* documents consist only of paragraphs (w:p elements)And tables, the only other block-level element in normal use.

. All the metadata about document structure is provided as font and spacing information, which can only reliably be interpreted by a human, making meaningful conversion exceptionally difficult.

However, if named styles (from the built-in style menu or created by the author) are used *consistently*, it is possible to write an XSLT2 script to match them and output more usable XML markup.

Some editors (eg *XMLMind*, *AbiWord*) and other systems now provide conversion from *Word*, both to a purely visual (HTML) format, mimicking the appearance of the original, and to a 'semantic' vocabulary such as DocBook or DITA, with no formatting.

The XSLT2 route also applies to *OpenOffice*/*LibreOffice*, which also stores XML in a Zip file. The markup is different, but can be converted along the same lines.

3.5 How can I make my existing HTML files work in XML?

Either make them XHTML/HTML5, or use a different document type.

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 or HTML5.

You may need to convert existing HTML files because XML does not permit end-tag minimisation (missing `</p>`, etc), unquoted attribute values, and a number of other SGML shortcuts which are commonly used in HTML. Many HTML authoring tools already produce almost (but not quite) well-formed XML by making sure that end-tags are used, attribute values are quoted, etc — however, many do not.

You may be able to convert HTML to XHTML using the Dave Raggett's HTML Tidy program (a HTML5 version also exists), which can clean up some of the horrible mess of pseudo-markup left behind by incompetent HTML editors, and even separate out some of the formatting to a stylesheet, but there is usually still some hand-editing to do.

Most modern website design programs, including DreamWeaver, still don't produce anything like well-formed HTML, largely because they are intended for making pages look pretty, rather than getting the markup right. Using a website design program and its HTML pages as the sole repository of your information can be a dangerous and expensive mistake. If you're working the other way round, however, getting the information design right in XML first, and then exporting it to a page design produced using a website design program, it's probably less important that the HTML is a mess, because browsers are very forgiving.

Converting valid HTML to XHTML

If your HTML files are valid (full formal validation with an SGML parser against one of the published DTDs, not just a simple syntax check), then try validating them as XHTML with an XML parser. If you have been creating clean HTML without embedded formatting then this process should throw up only mismatches in upper/lowercase element and attribute names, and EMPTY elements like `img`, plus any non-standard element type names if you use them. Simple hand-editing or a short script should be enough to fix these changes.

If your HTML validly uses end-tag omission and unquoted attribute values, this can be fixed automatically by a normalisation program like *sgmlnorm* (from the *OpenSP* package, which is part of *OpenJade*), or by the ***sgml-normalize*** function in an editor like *Emacs/psgml* (don't be put off by the names, they both do XML).

If you have a lot of valid HTML files, you could write a script to do this in a programming language which understands SGML markup (such as *Omnimark*, *SGMLC*, or one of the popular scripting languages (eg *Perl*, *Python*, *Tcl*, etc), using their SGML/XML libraries); or you could even use editor macros if you know what you're doing.

If your HTML is invalid or badly-formed, try the *HTML Tidy* program mentioned above. If that doesn't fix them, I'm afraid you'll need to write something special using the procedure below, or do it all by hand-editing, or copy-and-paste from a browser.

Converting to a new document type

If you want to move your files out of HTML into some other DTD entirely, there are many native XML industrial DTDs, and modular XML versions of popular DTDs like *TEI* (literary, historical, and linguistic documents) and *DocBook* (computer documentation) or *DITA* (technical documentation) to

choose from. There were several sites for the exchange of new XML DTDs, but writing new ones is now rare.

You can of course just make up your own markup: so long as it makes sense and you create a well-formed file, you should be able to write a CSS or XSLT stylesheet and have your document displayed in a browser.

Converting invalid HTML to well formed XHTML

If your files are invalid HTML (95they can be converted to well-formed DTDless files as follows:

1. replace any DOCTYPE Declaration with the XML Declaration `<?xml version="1.0" encoding="UTF-8"?>` (or using the appropriate character encoding).
2. If there was no DOCTYPE Declaration, just prepend the XML Declaration. Either way, the XML Declaration, if used, *must* be line 1 of the document.
3. Change any EMPTY elements (eg every `BASE`, `ISINDEX`, `LINK`, `META`, `NEXTID` and `RANGE` in the header, and every `AREA`, `ATOPARA`, `AUDIOSCOPE`, `BASEFONT`, `BR`, `CHOOSE`, `COL`, `FRAME`, `HR`, `IMG`, `KEYGEN`, `LEFT`, `LIMITTEXT`, `OF`, `OVER`, `PARAM`, `RIGHT`, `SPACER`, `SPOT`, `TAB`, and `WBR` in the body of the document) so that they end with `/>` instead, for example `<img src="mypic.gif" alt="Picture"/>`;
4. Make all element type names and attribute names lowercase;
5. Ensure there are correctly-matched explicit end-tags for all non-EMPTY elements; eg every `<para>` must have a `</para>`, etc;
6. Escape all `<` and `&` non-markup (ie literal text) characters as `<` and `&` respectively (there shouldn't have been any isolated `<` characters to start with, anyway!);
7. Ensure all attribute values are in matched quotes (values with embedded single quotes must be in double quotes, and *vice versa* — if you need both, use the `"` character entity reference);
8. Ensure all script URIs which have `&` as a field separator are changed to use `&` or a semicolon instead.
9. Ensure all scripts (eg Javascript) which have `<` or `&` characters (mathematical less-than tests, and Boolean AND conditionals) are *either* given as CDATA Marked Sections, or (if browser processors accept them) changed to use `<` and `&` or a semicolon respectively.

Be aware that some obsolete HTML browsers may not accept XML-style EMPTY elements with the trailing slash, so the above changes may not be

backwards-compatible. An alternative is to add a dummy end-tag to all EMPTY elements, so `<img src="foo.gif"/>` becomes `<img src="foo.gif"></img>`. This is valid XML but you must be able to guarantee no-one will ever put any text content inside such elements. Adding a space before the closing slash in EMPTY elements (eg `<img src="foo.gif" />`) may also fool older browsers into accepting XHTML as HTML.

If you have to answer Yes to any of the questions in the Question 3.5 below, you can save yourself a lot of grief by fixing those problems first before doing anything else. You will likely then be getting very close to having well-formed files.

Markup which is syntactically correct but semantically meaningless or void should be edited out before conversion. Examples are bogus spacing devices such as repeated empty paragraphs or linebreaks, empty tables, invisible spacing GIFs etc. XML uses stylesheets, and CSS3 means you won't need any of these.

Unfortunately there is rather a lot of work to do if your files are invalid: this is why many Webmasters now insist that only valid or well-formed files are used (and why you should instruct your designers to do the same), in order to avoid unnecessary manual maintenance and conversion costs later.

Checklist for invalid HTML

If your HTML files fall into this category (HTML created by most WYSIWYG editors is usually invalid) then they will almost certainly have to be converted manually, although if the deformities are regular and carefully constructed, the files may actually be almost well-formed, and you could write a program or script to do as described above. The oddities you may need to check for include:

- Do the files contain markup syntax errors? For example, are there any missing angle-brackets, backslashes instead of forward slashes on end-tags, or elements which nest incorrectly (eg `<B>starting <I>inside one element</B> but ending outside</I> it`)?
- Are there elements with missing end-tags that cannot be inferred by (eg) *sgmlnorm*?
- Are there any URIs (eg in `hrefs` or `srcs`) which use Microsoft Windows-style backslashes instead of normal forward slashes?

- Do the files contain markup which conflicts with HTML DTDs, such as headings or lists inside paragraphs, list items outside list environments, header elements like `base` preceding the first `html`, etc? (another sloppy editor trick)
- Do the files use imaginary elements which are not in any known HTML DTD? (large amounts of these are used in proprietary markup systems masquerading as HTML). Although this is easy to transform to a DTDless well-formed file (because you don't have to define elements in advance) most proprietary or browser-specific extensions have never been formally defined, so it is often impossible to work out meaningfully where the element types can be used.
- Are there any invalid (non-XML) characters in your files? Look especially for native Apple Mac Roman-8 characters left by careless designers; any of the illegal Windows characters (the 32 characters at decimal codes 128–159 inclusive) inserted by Microsoft editors; and any of the ASCII **control characters** 0–31 (except those permitted like TAB, CR, and LF). These must be converted to the correct characters in UTF-8 (or whatever you are using).
- Do your files contain invalid (old Mosaic/Netscape-style) comments? Comments must look

  ```
  <!-- like this -->
  ```

  with double-dashes each end and no other double (especially not multiple) dashes in between.

3.6 How do I convert XML to other file formats?

Write a conversion in a language that understands XML

While it is possible to write conversion routines by inventing your own XML parser, it is not recommended except as an exercise for students of computing science. All major languages have XML libraries that do all the heavy lifting of parsing (and validating, if needed).

You do need to know what's in the XML document before you start: there is no magic wand that will automatically deduce what things mean and where they are located in the file. If you have been handed some XML files out of the blue, you will need to go and find the creator or some documentation about them. The first 2–3 lines of the file may hold a clue as to what type of

XML they are. You will almost certainly need a copy of the DTD or Schema to which the files have been created.

The options for programming are:

- Use a language designed for the task. XSLT2 has all the facilities for handling XML built in from the start, and standalone processors are available for all platforms. Many XML editors have a copy of XSLT (2, hopefully) built in, so they offer an integrated development environment for editing and conversion. XSLT2 conversion can also run inside server packages like Apache Cocoon.

- Use an XML processing or pipelining package. These are (usually) commercial products which provide extensive document management, document database, and document conversion and editing functions, often as part of a much larger enterprise information solution, using XSLT2 or their own in-house systems. Two popular ones are MarkLogic and OmniMark.

- Use a conventional compilable language. Java or C (or one of its many variants) would be common; Pascal, FORTRAN, or COBOL are rare these days, but XML libraries do exist for them). BASIC, anyone?

- Use a scripting language. Perl, Python, Tcl, VBscript, or even Powershell are all popular, and XML libraries exist for them; the Python ones have an excellent reputation.

- Combine XML utilities with standard shells. Here is an early example of an XML-to-CSV routine which uses *onsgmls* to expose the ESIS, and *awk* to reformat it. Similar processes can be developed using the LTXML2 toolkit.

- There are downloadable (sometimes free) programs claiming to be 'easy' XML converters. The editor would like to hear recommendations or warnings ☺

The process of converting XML to other formats is sometimes referred to as 'down-converting', as it may involve the unavoidable loss of information (usually metadata) when the target format simply doesn't have a way to represent it.

3.7 If XML is just a subset of SGML, can I use my existing SGML tools?

Yes, if they are up to date

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. (see the pointers in Question 4.10 on page 82).

Unless there are very special reasons, you should probably plan to move your SGML to XML anyway.

3.8 I'm used to authoring and serving HTML. Can I learn XML easily?

Yes

Very easily, but even after nearly 20 years there is still a need for more tutorials, simpler tools, and more open examples of XML documents. 'Well-formed' XML documents may look similar to HTML except for some small but very important points of syntax.

The big practical difference is that XML has to stick to the rules. HTML browsers let you serve them even fatally broken or ridiculously corrupt HTML because they don't do a formal parse but just elide all the broken bits instead. With XML your files have to be completely correct or they simply won't work at all. One outstanding problem is that some browsers claiming XML conformance are also broken, and some browsers' support for XSLT

processing and CSS styling is still dubious at the best. Try yours on the list of real hotel web sites.

3.9 Can XML use non-Latin characters?

Yes, this is the default

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/` and an example of the range of alphabets and symbols at `http://www.cogsci.ed.ac.uk/~richard/unicode-sample-3-2.html`.

> While XML software may allow you to enter any Unicode character into a document, your readers can only see the characters if their computer has a suitable font! Not all typefaces and font files have the entire Unicode repertoire (ones that do are *huge*).

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 (accents). 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).

Peter Flynn writes:

> The encoding specification can refer to any character set your software supports, but the XML Specification only *requires* that applications support UTF-8 and UTF-16. Some of the common encodings supported by software include:

US ASCII Characters TAB, LF, CR, space, and the printable characters 33 to 126 (decimal) only (all other control characters are forbidden by XML).

ISO 8859 1 (Western European Latin-1) As ASCII plus codes 128 to 255 (decimal). Covers most (but not all) western European accented letters.

ISO 8859 2 TO 15 These other planes of ISO-8859 cover the remaining and different sets of Latin-based alphabetic and other symbols.

'CODEPAGES' AND OTHER OBSOLESCENT SETS Some software may also support various obsolete 'codepages', such as IBM-850, Microsoft Windows-1252, Apple Macintosh Roman-8, DEC Multinational and other non-standard character encodings, but these are generally non-portable and should be avoided where possible.

One common practice in western Europe is to use ISO-8859-1 so that the majority of common accented letters can be used as single bytes, and to use character entity references or numeric entities for all other characters. This has the advantage that such files can be opened in almost any single-byte editor. The drawback is that numeric entities are not mnemonic, and character entities have to be declared in DTD or internal subset, but if they are rare, this may not be a serious problem.

Bertilo Wennergren writes:

UTF-16 is an encoding that represents each Unicode character of the first plane (the first 64K characters) of Unicode with a 16-bit unit — in practice with two bytes for each character. Thus it is backwards compatible with neither ASCII nor Latin-1. UTF-16 can also access an additional 1 million characters by a mechanism known as surrogate pairs (two 16-bit units for each character).

'. . . the mechanisms for signalling which of the two are in use, and for bringing other encodings into play, are [. . .] in the discussion of character encodings.' The XML Specification explains how to specify in your XML file which coded character set you are using.

'Regardless of the specific encoding used, any character in the ISO 10646 character set may be referred to by the decimal or hexadecimal equivalent of its bit string': so no matter which character set you personally use, you can still refer to specific individual characters from elsewhere in the encoded repertoire by using `&#dddd;` (decimal character code) or `&#xHHHH;` (hexadecimal character code, in uppercase). The terminology can get confusing, as can the numbers: see the ISO 10646

Concept Dictionary. Rick Jelliffe has XML-ised the ISO character entity sets. Mike Brown's encoding information at http://skew.org/xml/tutorial/ is a very useful explanation of the need for correct encoding. There is an excellent online database of glyphs and characters in many encodings from the Estonian Language Institute server at http://www.eki.ee/letter/.

3.10 What's a DTD and where do I get one?

A specification of document structure. You can write one or download them.

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 Schema does the same thing in XML Document Syntax, so it can be read as an XML document itself; and Schemas allow more extensive data-typing.

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, what they can contain, and how each relates to the other, so that applications which need to know this in advance (processors, browsers, editors, search engines, navigators, and databases) can set themselves up correctly. The example above lets you create lists like this:

```
<List>
  <Item>Chocolate</Item>
  <Item>Music</Item>
  <Item>Surfing</Item>
</List>
```

As explained in Question 3.2 on page 36, the indentation in the example is just for legibility while editing: it is not required by XML. It could just as easily be written like this:

```
<List><Item>Chocolate</Item><Item>Music</Item><Item>Surfing</Item></List>
```

A DTD therefore 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 well-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') like images; 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 Cover Pages for pointers). Many of them can be downloaded and used freely, but some are restricted to certain industries, or are proprietary; but you can also 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.

Some XML editors use a binary compiled format of DTD produced by their own management routines to allow a single person in an organisation to be in charge of modifications, and to distribute only an unmodifiable (binary compiled) version to users.

The alternatives to a DTD are various forms of Schema. These provide more extensive validation features than DTDs, including character data content validation.

3.11 Does XML let me make up my own tags?

Yes but they're not called tags. They're element types.

XML 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.

The same applies if you are thinking in terms of 'fields' (see Question 4.8 on page 80). Wrong paradigm, wrong language.

Bob DuCharme writes:

> Don't confuse the term 'tag' with the term 'element'. They are not interchangeable. An element usually contains two different kinds of tag: a start-tag and an end-tag, with text or more markup between them.
>
> XML lets you decide which elements you want in your document and then indicate your element boundaries using the appropriate start- and end-tags for those elements. Each `<!ELEMENT...` declaration defines a type of element that may be used in a document conforming to that DTD. We call this type of element an 'element type'. Just as the HTML DTD includes the `H1` and `P` element types, your document can have `color` or `price` element types, or anything else you want.
>
> Normal (non-empty) elements are made up of a start-tag, the element's content, and an end-tag. `<color>red</color>` is a complete instance of the `color` element. `<color>` is only the start-tag of the element, showing where it begins; it is not the element itself.
>
> Empty elements are a special case that may be represented either as a pair of start- and end-tags with nothing between them (eg `<price retail="123"></price>`) or as a single empty element start-tag that has a closing slash to tell the parser 'don't go looking for an end-tag to match this' (eg `<price retail="123"/>`).

3.12 How do I create my own document type?

Analyse the class of documents, and write a DTD or Schema

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 encode: reports, invoices, letters, configuration files, credit-card verification requests, novels, plays, theses, or whatever. Once you have the structure correct, you write code to express this formally, using DTD or Schema syntax.

If you want to create a DTD, you need to learn the XML Declaration Syntax (very simple: declaration keywords begin with `<!` rather than just the open angle bracket, and the way the declarations are formed also differs slightly). Here's an example of a DTD for a shopping list, based on the fragment used earlier:

```
<!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 Maler and el Andaloussi (&, M.,

1995): this was written for SGML but perhaps 95than full SGML — see the list of restrictions which shows what has been cut out.

> 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.

3.13 Can a root element type be explicitly declared in the DTD?

No, use the Document Type Declaration.

This is done in the document's Document Type Declaration, not in the DTD.

Bob DuCharme writes:

> In a Document Type Declaration like this:
>
> ```
> <!DOCTYPE chapter SYSTEM "docbookx.dtd">
> ```
>
> the whole point of the `chapter` part is to identify which of the element types declared in the specified DTD should be used as the root element. I believe the highest level element in DocBook is `set`, but I find it hard to imagine someone creating a document to represent a set of books. We are free to use `set`, `book`, `chapter`, `article`, or even `para` as the document element for a valid DocBook document.
>
> [One job some parsers do is determine which element type[s] in a DTD are not contained in the content model of any other element type: these are by deduction the prime candidates for being default root elements. (PF)]
>
> This is A Good Thing, because it adds flexibility to how the DTD is used. It's the reason that XML (and SGML) have lent themselves so well to electronic publishing systems in which different elements were mixed and matched to create different documents all conforming to the same DTD.
>
> I've seen schema proposals that let you specify which of a schema's element types could be a document's root element, but after a quick look at section 3.3 of Part 1 of the W3C Schema Recommendation and the

RELAX NG schema for RELAX, I don't believe that either of these let you do this. I could be wrong.

3.14 I keep hearing about alternatives to DTDs. What's a Schema?

Like a DTD for validating content as well as structure.

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://schema.org/docs/faq.html` 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, where DTDs continue to be used.

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:

```
<?xml version="1.0"?>
<invoice xml: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.

> Authors and publishers should note that the English plural of Schema is Schemas: the use of the singular to do duty for the plural is a foible dear to the semi-literate; the use of the old (Greek) plural schemata is unnecessary didacticism.
>
> Writers should also note that the plural of DTD is DTDs: there is no apostrophe — see Truss (Truss, 2003).

Bob DuCharme writes:

> Many XML developers were dissatisfied with the syntax of the markup declarations described in the XML spec for two reasons. First, they felt that if XML documents were so good at describing structured information, then the description of a document type's own structure (its schema) should be in an XML document instead of written with its own special syntax. In addition to being more consistent, this would make it easier to edit and manipulate the schema with regular document manipulation tools. Secondly, they felt that traditional DTD notation didn't allow document type designers the power to impose enough constraints on the data — for example, the ability to say that a certain element type must always have a positive integer value, that it may not be empty, or that it must be one of a list of possible choices. This eases the development of software using that data because the developer has less error-checking code to write.

Peter Flynn writes:

> A DTD is only for specifying the element structure of an XML file, with a very limited amount of control over attribute values. It gives the names of the elements, attributes, and entities that can be used, and how they fit together. DTDs are designed for use with traditional text documents, not rectangular or tabular data, so the concept of data types is not as relevant: text is just text. If you need to specify numeric ranges or to define limitations or checks on the character data (text) content, a DTD is the wrong tool.

3.15 How will XML affect my document links?

XML Links are much more powerful, but not yet implemented in browsers

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 (`quandaentry`);
2. the second child object is the answer (the `answer` 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.

3.16 Can I encode mathematics using XML?

Yes, using MathML.

Mathematics is enabled 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 normal XML documents.

3.17 How does XML handle my metadata?

Any way you want.

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.

Id you are generating HTML5 from your XML, be aware that HTML5 does not use Dublin Core as HTML did (see `http://www.w3.org/TR/html5/document-metadata.html`) but it has a registration mechanism for extensions to the predefined set of metadata names.

3.18 How do I use graphics in XML?

Reference them as for HTML or use XLink. Or embed SVG.

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 3.15 on page 56), 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, Scalable Vector Graphics (an XML specification for vector graphics) 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 becoming supported in browsers, and can be exported from standard graphics (drawing) programs like *GIMP*.

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).

Embedded binary graphics

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 Question 3.18 on page 61).

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. External entities are available only when using a DTD, not a Schema.

Bob DuCharme writes:

> All the data in an XML document entity must be parsable XML. You can define an external entity as either a parsed entity (parsable XML) or an unparsed entity (anything else). Unparsed entities can be used for picture files, sound files, movie files, or whatever you like. They can only be *referenced* from within a document as the value of an attribute (much like a bitmap picture on an HTML Web page is the value of the `img` element's `src` attribute) and not part of the actual document. In an XML document, this attribute must be declared to be of type `ENTITY`, and the entity's declaration must specify a declared `NOTATION`, because if the entity isn't XML, the XML processor needs to know what it is. For example, in the following document, the `colliepic` entity is declared to have a JPEG notation, and it's used as the value of the empty dog element's `picfile` attribute.
>
> ```
> <?xml version="1.0"?>
> <!DOCTYPE dog [
> <!NOTATION JPEG SYSTEM "Joint Photographic Experts Group">
> <!ENTITY colliepic SYSTEM "lassie.jpg" NDATA JPEG>
> <!ELEMENT dog EMPTY>
> <!ATTLIST dog picfile ENTITY #REQUIRED>
> ]>
> <dog picfile="colliepic"/>
> ```
>
> The Entity method is particularly useful when you have many images, or many repeated uses of the same images, because you only declare them once, at the top of the document, making image management much easier.
>
> The XLink and XPointer linking specifications describe other ways to point to a non-XML file such as a graphic. These offer more sophisticated control over the external entity's position, handling, and appearance within the XML document.

Peter Murray-Rust writes:

> GIFs and JPEGs cater for bitmaps (pixel representations of images: all made up of coloured dots). Vector graphics (scalable, made up of drawing specifications) are addressed in the W3C's graphics activity as Scalable Vector Graphics (see http://www.w3.org/Graphics/SVG). With the specification now complete, it is possible to transmit the graphical representation as vectors directly within the XML file. For many graphics objects this will mean greatly decreased download time and scaling without loss of detail.

Max Dunn writes:

> SVG has really taken off recently, and is quite an XML success story [. . .] there are [many] conformant implementations. We recently started an SVG FAQ at http://www.svgfaq.com/.
>
> XSLT can be used to generate SVG from XML; details are at http://www.svgfaq.com/xslt.asp (be careful to use XSLT, not Microsoft's obsolete WD-xsl). Documents can also interact with SVG images (see http://www.xml.com/pub/a/2000/03/22/style/index.html).

3.19 What is parsing and how do I do it in XML?

Parsing is splitting up information into its component parts

Parsing is the act of splitting up information into its component parts (schools used to teach this in language classes until the teaching profession caught the anti-grammar virus).

'Mary feeds Spot' parses as

1. Subject = Mary, proper noun, nominative case
2. Verb = feeds, transitive, third person singular, active voice, 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; *iCal* and Google *Calendar* contain a parser which reads an `.ical` appointment attachment in your email, and works out what information is in it. Give them 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 the DTD or 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 xml:id="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 `xml:id` (predefined type 'ID') 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). This ends up as a kind of family-tree structure in the application's memory (tree structures are a common way for programs

to store related data).

As well as built-in parsers, there are also stand-alone parser-validators (see Question 3.19), 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.

Bill Rayer writes:

> For standalone parsing/validation use software like James Clark's onsgmls or Richard Tobin's rxp. Both work under Linux and Windows/DOS. The difference is in the format of the error listing (if any), and that some versions of *onsgmls* do not retrieve DTDs or other files over the network, whereas *rxp* does.
>
> Make sure your XML file correctly references its DTD in a Document Type Declaration, and that the DTD file[s] are locally accessible (*rxp* will retrieve them if you have an Internet connection; *onsgmls* may not, so it may need a local copy).
>
> Download and install the software. Make sure it is installed to a location where your operating system can find it. If you don't know what any of this means, you will need some help from someone who knows how to download and install software on your type of operating system.
>
> For *onsgmls*, copy `pubtext/xml.soc` and `pubtext/xml.dcl` to your working directory.
>
> To validate `myfile.xml`, open a shell (command or terminal) window (Linux) or an MS-DOS (command) window (Microsoft Windows). In these examples we'll assume your XML file is called `myfile.xml` and it's in a folder called `myfolder`. Use the real names of your folder and file when you type the commands.
>
> **For *onsgmls*:** `$ onsgmls -wxml -wundefined -cxml.soc -s myfile.xml`
>
> There are many other options for *onsgmls* which are described on the Web page. The ones given here are required because it's based on an SGML parser and these options switch it to XML mode and suppress the normal output, leaving just the errors (if any).
>
> In Microsoft Windows you may have to prefix the `onsgmls` command with the full path to wherever it was installed, eg `C:\Program Files\OpenSP\bin\onsgmls`.

For *rxp*: `$ rxp myfile.xml`

rxp also has some options which are described on its Web page.
In Microsoft Windows you may have to prefix the `rxp` command with the full path to wherever it was installed, eg `C:\Program Files\ltxml2\bin\rxp`.

3.20 How do I include one XML file in another?

Use a general entity, same as for SGML, or use XInclude.

One method is to use Document Entities, which work exactly the same as for SGML, but require a DTD or internal subset. First you declare the entity you want to include, and then you refer to it by name as an Entity Reference:

```
<?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 4.19 on page 94) 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).

Schemas do not support entities, so the alternative is to use XInclude. This is a W3C specification for including one XML document (or fragment) inside another.

```
<?xml version="1.0"?>
...
<article xmlns="http://docbook.org/ns/docbook"
      xmlns:xi="http://www.w3.org/2001/XInclude">
   <info>
     <xi:include href="metadata.xml" parse="xml"
         xpointer="title"/>
   </info>
   <sect1>
      ...
   </sect1>
</article>
```

Your processing software must be able to handle XInclude for this to work. The XPointer syntax can direct the parser to a specific location within the document, unlike entities, where the entire document is included.

3.21 When should I use a CDATA Marked Section?

CDATA is only for text containing markup-like characters.

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 equivalents.*

If you try, for example, to use:

```
some text with <![CDATA[<em>markup</em>]]&gt; in it.
```

in the expectation that the embedded markup would remain untouched, it won't: it will just output

```
some text with &lt;em>markup&lt;/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 bogus, corrupt, or 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.

Read Question 3.22 as well, which is very closely related.

3.22 How can I handle embedded HTML in my XML?

Provide for it in the output, use a deep copy, or try disable-output-escaping.

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 or HTML 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 few 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 reproduces what was there untouched, eg if you have to preserve `<b>some text</b>` as-is, supply a template to do it:

  ```
  <xsl:template match="h:b">
    <b>
      <xsl:apply-templates/>
    </b>
  </xsl:template/>
  ```

  (If you are handling elements from several different DTDs or Schemas, you will probably need Namespaces to keep them distinct, hence the `h:` prefix.)

- Use XSLT's 'deep copy' instruction, which outputs nested well-formed markup *verbatim*, eg

  ```
  <xsl:template match="h:b">
    <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>]]&gt;</xsl:text>
  ```

  This falls into the 'dirty tricks' department, and is usually deprecated. Some processors do not support it.

- 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.

Read Question 3.21 on page 65 as well, which is very closely related.

3.23 What are the special characters in XML?

Just five: `<` *(<),* `&` *(&),* `>` *(>),* `"` *("), and* `'` *(').*

For normal text (*not* markup), there are no special characters except < and &: just make sure your XML Declaration 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.

Apart from the invisible ASCII control characters (the ones you can't type), all other characters are just normal text. Currency signs (€, £, $, f, Rs, ฿, and others), all the punctuation (except < and &), and all other letters, signs, and symbols in any language or writing system are just text (assuming you have the correct character encoding).

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 set of names which you can declare in your DTD (eg `<!ENTITY euro "€">`) which then lets you use the name `€` 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 the five character entities listed at the top of this question are assumed to be predeclared, and you can use them without declaring them separately (indeed, most software prevents you redeclaring them):

< The less-than character (<) starts **element markup** (the first character of a start-tag or an end-tag).

& The ampersand character (&) starts **entity markup** (the first character of a character entity reference).

> The greater-than character (>) ends a start-tag or an end-tag.

" 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.

&APOS; 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* character entities you need to use, so it would be good practice also to declare any of the five above that you plan on using. 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.

> There are circumstances where you can use special characters as themselves, such as in CDATA Sections. Most control characters are prohibited in XML: see the Specification for exact details.

There are also no reserved words as such in the user namespace of XML: you can call an element `element` and an attribute `attribute` and so on as in the following (perverse) example:

```
<?xml version="1.0"?>
<!DOCTYPE DOCTYPE SYSTEM "SYSTEM" [
<!ELEMENT DOCTYPE (ELEMENT+)>
<!ATTLIST ELEMENT ATTLIST ENTITY #IMPLIED>
<!NOTATION DOCTYPE SYSTEM "ENTITY">
<!ENTITY NOTATION SYSTEM "ENTITY" NDATA DOCTYPE>
]>
<DOCTYPE>
  <ELEMENT ATTLIST="NOTATION">foo</ELEMENT>
</DOCTYPE>
```

where the file `SYSTEM` contains the declaration: `<!ELEMENT ELEMENT (#PCDATA)>` and the file `ENTITY` does not even exist ☺

There are **keywords** like `DOCTYPE` and `IMPLIED` which are reserved Names, but they are prefixed by a flag character (the Markup Declaration Open character or the Reserved Name Indicator) so that they cannot be confused with user-specified Names.

3.24 What other markup systems are there?

There are several other popular systems available

MarkDown This is both a plaintext formatting syntax, and a Perl script that converts it to XHTML. The objective was that the souce file should not look like markup (so it could be read just as a plaintext document), but that it should be sufficiently robust to withstand conversion.

Details are at `http://daringfireball.net/projects/markdown/`.

Wiki/blog formats There are several of these: the best-known is probably the MediaWiki format used by Wikipedia, which provides fairly extensive plaintext facilities for creating the entries used in the site, which are then converted to HTML for display.

Similar conceptual formats are used by some other wiki systems such as PmWiki and Google Code; and there are hybrids which also use HTML like Fossil.

LaTeX The LaTeX document-preparation and typesetting system Lamport [Lamport], 1994 uses macros built on TeX syntax to implement a partially-structured document model. The syntax uses the backslash to flag markup elements and formatting instructions, and curly braces to delimit arguments and restrict the scope of typographic actions.

The software is available on all platforms, and creates PDF or PostScript output. Conversion into LaTeX from XML-based systems can be done with XSLT, but conversion out of LaTeX syntax is problematic because it has features not easily representable in other systems.

SGML SGML and some of its progenitors remain available but are rarely used.

There is an extensive list of other markup languages in the Wikipedia page.

Chapter 4

Developers

Programmers and implementers

4.1 Where's the spec?

Right here

Right here: Bray et al. (2004) (see `http://www.w3.org/TR/REC-xml`). Includes the EBNF, and all the normative material. There are also versions in Japanese; Spanish; Korean; a Java-ised annotated version, and an annotated version in book form DuCharme [DuCharme], 1999.

Eve Maler maintains the DTD used for the spec itself; the DTD is also to encode several other W3C specifications, such as XLink, XPointer, DOM, XML Schema, etc. There is documentation available for the DTD. Note that the XML spec needs to use a special one-off version of the DTD, since the real original DTD used for it has long since been lost.

4.2 I'm trying to understand the XML Spec: why does it have such difficult terminology?

It has to be formal to be accurate.

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 document 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.

Those new to the terminology may find it useful to read something like the Sperberg-McQueen and Burnard, 2002 or DuCharme, 1999.

4.3 What are these terms DTDless, valid, and well-formed?

Well-formed means just syntactically correct; valid means it conforms to a DTD or Schema.

XML lets you use a Schema or Document Type Definition (DTD) to describe the markup (elements and other constructs) available in any specific type of document. However, the design and construction of Schemas and DTDs can be complex and non-trivial, so XML also lets you work without one. DTDless

operation means you can invent markup without having to define it formally, provided you stick to the well-formedness rules of XML syntax.

To make this work, a DTDless file is assumed to define its own markup purely by the existence and location of elements where you create them. When an XML application encounters a DTDless file, it builds its internal model of the document structure while it reads it, because it has no Schema or DTD to tell it what to expect. There must therefore be no surprises or ambiguous syntax. To achieve this, the document must be 'well-formed' (must follow the rules).

To understand why this concept is needed, look at standard HTML as an example:

- The `<img>` element is declared (in the [SGML] DTDs for HTML) as EMPTY, so it doesn't have an end-tag (there is no such thing as `</img>`);
- Many other HTML elements (such as `<para>`) allow you to omit the end-tag for brevity.
- If an XML processor reads an HTML file without knowing this (because it isn't using a DTD), and it encounters an `<img>` or a `<para>` (or any other start-tag), it would have no way to know whether or not to expect an end-tag. This makes it impossible to know if the rest of the file is correct or not, because it has now no evidence of whether it is inside an element or if it has finished with it.

Well-formed documents therefore *require* start-tags and end-tags on every normal element, and any EMPTY elements must be made unambiguous, either by using normal start-tags and end-tags, or by appending a slash to the name of the start-tag before the closing > as a signal that there will be no separate end-tag.

All XML documents, both DTDless and valid, must be well-formed. They must start with an XML Declaration if necessary (for example, identifying the character encoding or using the Standalone Document Declaration):

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<foo>
  <bar>...<blort/>...</bar>
</foo>
```

David Brownell writes:

> XML that's just well-formed doesn't need to use a Standalone Document Declaration at all. Such declarations are there to permit certain speedups when processing documents while ignoring external parameter entities — basically, you can't rely on external declarations in standalone documents. The types that are relevant are entities and attributes. Standalone documents must not require any kind of attribute value normalisation or defaulting, otherwise they are invalid.

It's also possible to use a Document Type Declaration with DTDless files, even though there is no Document Type to refer to:

Richard Lander writes:

> If you need character entities [other than the five built-in ones] in a DTDless file, you can declare them in an internal subset without referencing anything other than the root element type:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE example [
<!ENTITY mdash "&mdash;">
]>
<example>Hindsight&mdash;a wonderful thing.</example>
```

So…here are the rules:

Well formed XML

- All tags must be balanced: that is, every element which may contain character data or sub-elements must have both the start-tag and the end-tag present (omission is not allowed except for EMPTY elements, see below);
- All attribute values must be in quotes. The single-quote character (the apostrophe) may be used if the value contains a double-quote character, and vice versa. If you need isolated quotes as data as well, you can use `'` or `"`. Do not under any circumstances use the automated typographic ('curly') inverted commas substituted by some wordprocessors for quoting attribute values.
- Any EMPTY elements (eg those with no end-tag like HTML's `<img>`, `<hr>`, and `<br>` and others) must *either* end with `/>` *or* they must look like non-EMPTY elements by having a real end-tag (but no content).

Example: `<br>` would become either `<br/>` or `<br></br>` (with nothing in between).

- There must not be any isolated markup-start characters (`<` or `&`) in your text data. They must be given as `<` and `&` respectively, and the sequence `]]>` may only occur as the end of a CDATA marked section: if you are using it for any other purpose it must be given as `]]>`.
- Elements must nest inside each other properly (no overlapping markup, same as for HTML);
- DTDless well-formed documents may use attributes on any element, but the attributes are all assumed to be of type CDATA. You cannot use ID/IDREF attribute types for parser-checked cross-referencing in DTDless documents.
- XML files with no DTD are considered to have `<`, `>`, `'`, `"`, and `&` predefined and thus available for use. With a DTD, all character entities used must be declared, including these five.

Valid XML

Valid XML files are well-formed files which have a Document Type Definition (DTD) or Schema and which conform to it. They must already be well-formed, so all the rules above apply.

A valid file begins with a Document Type Declaration specifying a DTD, or code specifying a W3C Schema. It may have an optional XML Declaration prepended.

```
<?xml version="1.0"?>
<!DOCTYPE advert SYSTEM "http://www.foo.org/ad.dtd">
<advert>
  <headline>...<pic/>...</headline>
  <text>...</text>
</advert>
```

The XML Specification predefines an SGML Declaration for XML which is fixed for all instances and is therefore hard-coded into all XML software and never specified separately (except when using an SGML/XML switchable validator like *onsgmls*: see below).

Peter Flynn writes:

> The SGML Declaration for XML has been removed from the text of the Specification but is available as a separate document). As this appears to suffer occasionally from bitrot or neglect, there is a copy here (WebSGML TC) and here (Extended Naming Rules TC), and a version for *onsgmls* here.

The specified DTD must be accessible to the XML processor using the URI supplied in the SYSTEM Identifier, either by being available locally (ie the user already has a copy on disk), or by being retrievable via the network. Note that DTD specifications *must* be URIs (local, relative, or absolute). Proprietary-specific filesystem references (eg `C:\dtds\my.dtd` are not URIs and cannot be used: use the `file:///C|/dtds/my.dtd` format instead.

It is possible (many people would say preferable) to supply a Formal Public Identifier with the PUBLIC keyword, and use an XML Catalog to dereference it, but the Specification mandates a SYSTEM Identifier so this must still be supplied after the PUBLIC identifier: no further keyword is needed. A PUBLIC identifier constitutes a claim to ownership only of the identifier, not to the DTD itself (although ion many cases that is implied).

```
<!DOCTYPE advert PUBLIC
   "+//Silmaril//DTD Foo Corp Advertisements//EN"
   "http://www.foo.org/ad.dtd">
<advert>...</advert>
```

The test for validity is that a validating parser finds no errors in the file: it must conform absolutely to the definitions and declarations in the DTD.

XML (W3C) Schemas are not usually linked directly from within an XML document instance in the way that DTDs are: the relevant Schema (XSD file) for a document instance is normally specified to the parser separately, either by file system reference, or using a Target Namespace.

4.4 Which should I use in my DTD/Schema, attributes or elements?

See http://xml.coverpages.org/elementsAndAttrs.html

There is no single answer to this: a lot depends on what you are designing the document type for.

Traditional editorial practice for normal text documents is to put the real text (what would be printed) as character data content, and keep the metadata (information about the text) in attributes, from where they can more easily be isolated for analysis or special treatment like display in the margin or in a mouseover:

```
<l n="184">
  <spara>Portia</spara>
  <text>The quality of mercy is not strain'd,</text>
  ...
</l>
```

But from the systems point of view, there is nothing wrong with storing the data the other way round, especially where the volume of text data on each occasion is relatively small:

```
<line speaker="Portia" text="The quality of mercy is not strain'd,">184</line>
```

A lot will depend on what you want to do with the information and which bits of it are easiest accessed by each method. A rule of thumb for conventional text documents is that if the markup were all stripped away, the bare text should still be correct, readable, and usable, even if unformatted and inconvenient. For database output, however, or other machine-generated documents like e-commerce transactions, human reading may not be meaningful, so it is perfectly possible to have documents where all the data is in attributes, and the document contains no character data in content models at all. See http://xml.coverpages.org/elementsAndAttrs.html for more information.

Mike Kay writes:

> From a user: '[...] do most of you out there use element-based or attribute-based xml? why? '
>
> Beginners always ask this question. Those with a little experience express their opinions passionately. Experts tell you there is no right answer. (http://lists.xml.org/archives/xml-dev/200006/msg00293.html)

4.5 What has changed between SGML and XML?

Stricter syntax and no options.

The main syntactic change is that EMPTY elements in DTDless documents *must* use the Null End-Tag trick (eg `<img src="pic"/>`) because without a DTD or Schema there is no way for the parser to know not to expect an end-tag. If an element type is declared as EMPTY in the DTD/Schema then it can use *either* the NET *or* the full end-tag syntax (eg `<img src="pic"></img>`).

Other syntactic changes are that *all* attribute values must be quoted; there is no minimisation of attributes or elements; and everything is case-sensitive. One important addition is that multiple ATTLIST declarations are allowed, so an internal subset can add to the attributes already declared for an element type.

The principal changes in Document Type Definitions (DTDs) are in what you can specify. To simplify it and make it easier to write processing software, a large number of SGML markup declaration options have been suppressed (see the list of omitted features). The biggest change in vocabulary management is the introduction of W3C Schemas, which allow a level of content-type validation not available in DTDs, and are themselves expressed in XML Document Syntax.

The main addition here is namespaces, which enable Schemas and documents to distinguish element-type and attribute-type source (ownership, origin, or application). This lets you have element types with the same name but different meanings in the same document, eg `DocBook:table` and `TEI:table`. An extra Name Start Character (the colon) was added in XML Names to allow this. Despite its classification, a colon may only appear in mid-name, *not* at the start or the end, and the prefix `xml:` is Reserved.

4.6 Can I use JavaScript, ActiveX, etc in XML files?

Not in the XML file itself, but via a stylesheet.

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, and script files for Javascript etc, 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 nor 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.

> If you need to embed scripts in a web page that you are generating from XML, you need to make sure that the two markup characters < and & are either escaped as `<` and `&` respectively, or that each script's content is enclosed in a CDATA Section so that it doesn't get seen as markup.

4.7 Can I use Java to create or manage XML files?

Sure.

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. There is a suite of Java tutorials (with source code and explanation) available at `http://developerlife.com/tutorials/`.

Please do not mail the FAQ editor with questions about your Java programming bugs. Ask one of the Java newsgroups instead.

4.8 How do I get XML into or out of my database?

Ask your database manufacturer

Almost all database management systems now 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.

Alternatively, most database systems now provide an XML dump format that lets you export a table as-is, for example by surrounding the field values with tags called after the fieldnames. For example, the `-X` option to the `mysql` command will do this, eg

```
$ echo 'select * from news;' | mysql -X -u username -p password dbname

<?xml version="1.0"?>
<resultset statement="select * from news"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
        <field name="id">1</field>
        <field name="stamp">0</field>
        <field name="title"></field>
        <field name="date">0000-00-00</field>
        <field name="time">test</field>
        <field name="description">News Engine test item 1</field>
  </row>
</resultset>
$
```

In less trivial, but still simple, cases, you could export by writing a report routine that formats the output as an XML document by adding the relevant tags as literals before and after each data value; and you could import by

writing an XSLT or similar transformation that formatted the XML data as a load file in your database's preferred format. For example, with the following data:

```
<news>
  <entry xml:id="N1" stamp="0" date="0000-00-00" time="test">
    <title></title>
    <description>News Engine test item 1</description>
  </entry>
</news>
```

you could turn it into a MySQL statement with *lxprintf*:

```
$ lxprintf -e entry \
  'INSERT INTO `news` VALUES (%s,%s,"%s","%s","%s","%s");\n' \
  'substring(@xml:id,2)' @stamp title @date @time description \
        mynews.xml

INSERT INTO `news` VALUES (1,0,"","0000-00-00","test","News Engine test item 1");
$
```

> Users from a database or computer science background should be aware that XML is not a database management system: it is a text markup system. While there are many similarities, some of the concepts of one are simply non-existent in the other: XML does not possess some database-like features in the same way that databases do not possess markup-like ones. It is a common error to believe that XML is a DBMS like Oracle or Access and therefore possesses the same facilities. It doesn't.

Database users should read the article Salminen and Tompa [&], 2001 [thanks to Bart Lateur for identifying this.] Ronald Bourret also maintains a good resource on XML and Databases discussing native XML databases at http://www.rpbourret.com/xml/XMLAndDatabases.htm.

There is some information about the XQuery (XQL) Language in the note on Searching.

4.9 What's a namespace?

A named DTD/Schema or fragment identified by a URI (URL).

Randall Fowle writes:

> A namespace is a collection of element and attribute names identified by a Uniform Resource Identifier reference. The reference may appear in the root element as a value of the `xmlns` attribute. For example, the namespace reference for an XML document with a root element <x> might appear like this:
>
> ```
> <x xmlns="http://www.company.com/company-schema">
> ```
>
> More than one namespace may appear in a single XML document, to allow a name to be used more than once. Each reference can declare a prefix to be used by each name, so the previous example might appear as
>
> ```
> <x xmlns:spc="http://www.company.com/company-schema">
> ```
>
> which would nominate the namespace for the 'spc' prefix:
>
> ```
> <spc:name>Mr. Big</spc:name>
> ```

James Anderson writes:

> In general, note that the binding may also be effected by a default value for an attribute in the DTD.
>
> The reference does not need to be a physical file; it is simply a way to distinguish between namespaces. The reference should tell a person looking at the XML document where to find definitions of the element and attribute names using that particular namespace. Ronald Bourret maintains the Namespace FAQ at http://www.rpbourret.com/xml/NamespacesFAQ.htm.

4.10 What XML software is available?

Thousands of programs: too many to list here.

Hundreds, possibly thousands, of programs. Details are no longer listed in this FAQ as they are now too many and are changing too rapidly to be kept up to date: see the XML Web pages at http://xml.coverpages.org/ and watch for announcements on the mailing lists and newsgroups.

For a detailed guide to some examples of XML programs and the concepts behind them, see the editor's book Flynn, 1998. There are several implementations of the key XML processing and query languages (XSLT2, XQuery, and XSL:FO), the most popular of which is probably Saxon.

SGML software such as *Jade* and *SP* (now *OpenSP*) remain the source of several utilities which also handle XML, especially the *onsgmls* validating parser and *sgmlnorm* (see Question 3.5 on page 41).

Details of some XML software products are held on the XML Cover Pages. For browsers see the question on XML Browsers and the details of the xml-dev mailing list for software developers. Bert Bos keeps a list of some XML developments in Bison, Flex, Perl, and Python. The long-established conversion and application development engines like Omnimark, and SGMLC all have XML capability and they all provide APIs.

XML Editors

Choosing an editor is one of the hardest tasks, because everyone has different requirements and levels of knowledge, and what appears to be incredibly simple to one user may seem dauntingly difficult to another. All XML editors guide the user in the construction or maintenance of XML documents — that's their purpose in life.

The simplest ones just keep track of matching pointy brackets, start-tags and end-tags, and balanced quotes, leading to a well-formed file. More powerful editors can read a DTD or Schema and provide menu choices for element manipulation and attribute editing, and prevent the creation of invalid documents. The most powerful ones can also be used for DTD or Schema development, and for XML processing with XSLT or XSL:FO.

Some are text-mode editors — they show all the markup and the text with nothing hidden, often using colour to distinguish markup characters and indentation to show the structure. Most editors have a synchronous typographic mode as well, using a stylesheet to format the display, so you appear to be editing a fully typeset view of the document (often called WYSIWYG, although it's actually not). Text-mode editors worry some users because the pointy brackets are visible (they think it's programming); synchronous typographic editors worry other people because the pointy brackets are *not* visible, which makes it hard to see where stuff begins and ends.

The more sophisticated editors are programmable, so the nature and

effect of the markup and the user's actions can be limited or enhanced by scripts in JavaScript, VBscript, Python, Tcl, Lisp, etc; or XSLT.

Do *not* be tempted to use a non-XML editor like *Notepad*, *vi*, or *textedit* for XML documents: it will only end in tears, anger, and recriminations. Get properly-equipped. (Microsoft's separate *XML Notepad* product *is* usable for editing small instances, but not for DTD or Schema development.)

Wikipedia has a comparison of editors at `http://en.wikipedia.org/wiki/Comparison_of_XML_editors` but it doesn't give any qualitative information. There is an old (2004) comparative paper on choosing an XML editor from Thijs van den Broek which may help with methodology although the products are not current.

Several of the editors are available free of charge or free for demo/examination, but as far as is known, the only entirely open-source one is *Emacs* with *nxml-mode* (for W3C and RNG schemas) or *psgml-mode* (for DTDs).

There is a page of useful links for XML users in Dutch at http://xml.beginthier.nl/.

Information for developers of Chinese XML systems can be found at the Chinese XML Now! website of Academia Sinica: http://www.ascc.net/xml/ This site includes a FAQ and test files.

4.11 Software we're missing

Useful programs that have disappeared from sight

There are probably dozens of useful and productive programs that have vanished. Sometimes the company simply went bust and disappeared; sometimes it was a private or open-source venture that the author had to abandon; and sometimes (worst of all) the company was taken over by another company which didn't really know about XML, and was then bought out by another which had no clue at all, and the products became abandonware.

This is a list of those we knew and loved, but in some cases we do still have the installation disks, and the software may still execute given a suitable (outdated) platform.

Near & Far (Microstar) A graphical SGML DTD-design program for

Windows 95, which just about made it into the XML era (it could export in XML Declaration Syntax, but not import). Notable principally for its ease of use and its attractively-designed interactive diagramming of the document structure (still used by a few *aficionados* for documentation).

DynaText/DynaTag/DynaWeb (Electronic Book Technologies) A suite of Windows 95/NT software comprising an SGML database and web server for serving HTML converted from Word documents.

The key component was DynaTag, which let you import a Word document, and identify patterns in the (non-XML) markup which mapped to elements in an ad-hoc output SGML document. The user-selectable mappings let you encapsulate list items into a real list, and generally fight your way through the mess of Word `.doc` files to produce tagged output. This could then be post-processed into conformance with a known DTD. With sufficient examples of similarly-constructed Word files, it could be let loose on a whole collection of them for bulk conversion.

Microsoft SGML Author for Word Despite its name, not an editor but a converter. It used mappings from Word styles to SGML markup and really could do circular lossless conversion. Aimed at authoring communities who either couldn't or wouldn't learn to author or edit in SGML, it allowed authoring in Word and editing in SGML, and to then have the document converted back to Word for the authors to re-edit. Probably a check-box item for a contract, it was never supported by Microsoft, and even their own helpdesk didn't believe it was genuine.

Panorama (SoftQuad) Plugin for Netscape to allow SGML downloads with a referenced stylesheet (which worked flawlessly once you got the hang of the slightly odd referencing). It also implemented HyTime bidirectional links, and you could publish your linksets.

Also available as Panorama Publisher, essentially an expanded version in standalone form for large-scale document publishing. Its stylesheet editing pane remains one of the best examples of interactive style manipulation.

Author/Editor (SoftQuad) One of the first genuinely synchronous typographic editors for SGML (and XML, in its later incarnation as

XMetaL). In widespread use for many years along with its simplified sibling, *HoTMetaL*.

4.12 What is my information? DATA or DOCUMENT?

It depends on what you're using it for.

Some important distinctions exist between the major classes of XML applications and the way in which they are used.

Two classes of applications are usually referred to as 'document' and 'data' applications, and this is reflected in the software, which is usually (but not always) aimed at one class or the other.

Document-style applications These are like traditional publishers' work: text and images in a structured environment, with fonts and formatting. In most cases this includes Web pages as well as material destined for PDF or print like books and magazines. The hallmark of document applications is that they make heavy use of Mixed Content (eg subelements in text).

Data-style applications These are found mostly in e-commerce, web services, and process or application control, with XML being used as a container for information being stored or passed between systems, usually unformatted and unseen by humans. Their hallmark is the absence of Mixed Content, and the prevalence of numeric or categorical data.

There is a third major area, Web Development, whose requirements are often hybrid, and span the features of both document and data applications because they contain partly static descriptive text and partly dynamic data.

While in theory it would be possible to use data-class software to write a novel, or document-class software to create invoices, it would probably be severely suboptimal. Because of the nature of the information used by the two classes, data-class applications tend to use Schemas, and document-class applications tend to use DTDs, but there is a considerable degree of overlap.

The way in which XML gets used in these two classes is also divided in two: XML can be used manually or under program control.

Manual usage This means editing and maintaining the files with an editor, from the keyboard, seeing the information on the screen as you do so. This is suitable for individual documents, especially in the publishing field, for web pages, and for developers working on single instances such as sample files or web site templates. Manual processing also implies running production programs like formatters, converters, and database queries on a one-by-one basis, using the keyboard and mouse in the normal way. Much of the software for manual usage can be run from the command line, which makes it easy to use for one-off applications and in hidden applications like Web scripts.

Programmable usage This means writing programs which call on software services from APIs, libraries, or the network to handle XML files from inside the program. XML files in data applications are almost never edited by hand. This is the normal method of operating for e-commerce applications, web automation, web services, and other process or application controls. There are libraries and APIs for many languages, including Java, C, and C++ as well as the usual scripting languages like Python, Perl, Tcl, Ruby, etc.

In addition to these axes, there are currently two different ways of processing XML, memory-mapped or event-triggered, usually referred to by the names of their original instantiations, the Document Object Model (DOM) and the Simple API for XML (SAX) respectively. Both use a model of document engineering based on a tree-like structure of hierarchical document markup known as a Grove (a collection of trees, effectively an in-memory map of the result of parsing the document markup). In this model, every 'node' (item of information) from the outermost element down through every element and attribute to each piece of unmarked text can be identified. For applications using Schemas, a Post-Schema-Validation Infoset (PSVI, equivalent to a grove) is defined, which specifies what information a parser should make available to the application.

Joe Fawcett writes:

(in article `<eFIrHKtCGHA.2920@tk2msftngp13.phx.gbl>`)

Briefly 'node' is a generic term for any of the many types of XML building blocks, including **element**: `<myElement/>`; **attribute**: `<myElement myAttribute="myValue"/>`; and **text node**: `<my Text Node>`

There are also comments [**Comment Declarations**], **Processing Instructions** and the invisible **Document Node** representing the **root** of the XML document, as well as others.

Grossly oversimplified, a **DOM-based application** reads an entire XML document into memory and then provides programmable access to every node in every tree in the grove; whereas a **SAX-based application** reads the XML document, and events are triggered by the occurrence of nodes as they happen, for which rules or actions have been pre-programmed. (In reality it's more complex than that, and both methods share a lot of concepts in common.)

Both models provide an abstract API for constructing, accessing, and manipulating XML documents. A binding of the abstract API to a particular programming language provides a concrete API. Vendors provide concrete APIs which let you use one or other method to query and manipulate XML documents. Both types of parser have been implemented in many languages and under many operating systems and interfaces. There are FAQs for both DOM and SAX.

4.13 Do I have to change any of my server software to work with XML?

Make sure your server sends XML files as `text/xml`

If you are just serving static files. 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. Popular server software like Apache HTTPD knows this already.

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.

4.14 Can I still use server-side inclusions?

Yes, just make sure the output conforms to XML

This works 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-replacer scripting languages 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.

4.15 Can I (and my authors) still use client-side inclusions?

Yes, just make sure the output conforms to XML

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.

4.16 I have to do an overview of XML for my manager/client/investor/advisor. What should I mention?

Non-proprietary multi-purpose flexible markup

Tad McClellan writes:

- 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 numerical or categorical 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 [or one program with 20 outputs (Ed)] 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.

4.17 Is there a conformance test suite for XML processors?

Yes, there was: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xslt

James Clark has a collection of test cases for testing XML parsers at http://www.jclark.com/xml/ which includes a conformance test against 'canonical XML'.

Carmelo Montanez writes:

> NIST has developed a number of XSLT/XPath tests, which will be part of the official OASIS XSLT/XPath suite (not yet released). The expected output may be slightly different from one implementation to another. The OASIS XSLT technical committee has a solution for that problem, however our tests do not yet implement such solution. Please forward any comments to carmelo@nist.gov.

Jon Noring writes:

> For those who are interested, I took the current and complete Unicode 3.0 'cast' of characters and their hex codes, and created a simple XML document of it to test XML browsers for Unicode conformity. It is not finished yet — I need to add comments and to fix the display of rtl characters (ie Hebrew, Arabic). It is found at: http://www.windspun.com/unicode-test/unicode.xml. It is quite large, almost 900K in size, so be prepared. IE5 renders many of the characters in this XML document — and for the ones it does render it appears to do so correctly. I look forward to when Opera will do likewise. I haven't tested the current version of Mozilla/Netscape for Unicode conformity.

4.18 I've already got SGML DTDs: how do I convert them for use with XML?

Edit by hand or use software like Near+Far Designer.

There are numerous projects to convert common or popular SGML DTDs to XML format (for example, both the TEI DTD (Lite and full versions) and the DocBook DTD are available in both SGML and XML, in Schema and DTD formats).

Seán McGrath writes:

> TO CONVERT SGML DTDS TO XML:
>
> 1. No equivalent of the SGML Declaration. So keywords, character set etc are essentially fixed;
> 2. Tag minimisation is not allowed, so `<!ELEMENT x - O (A,B)>` becomes `<!ELEMENT X (A,B)>` and `<!ELEMENT x - O EMPTY>` becomes `<!ELEMENT X EMPTY>`;
> 3. `#PCDATA` must only occur at the extreme left (ie first) in an OR model, eg `<!ELEMENT x - - (A|B|#PCDATA|C)>` (in SGML) becomes `<!ELEMENT x (#PCDATA|A|B|C)*>`, and `<!ELEMENT x (A,#PCDATA)>` is illegal;
> 4. No CDATA, RCDATA elements [declared content];
> 5. Some SGML attribute types are not allowed in XML eg NUTOKEN;

6. Some SGML attribute defaults are not allowed in XML eg CONREF and CURRENT;
7. Comments cannot be inline to declarations like

```
<!ELEMENT x - - (A,B) -- an SGML comment in a declaration -->
```

8. A whole bunch of SGML optional features are not present in XML: all forms of tag minimisation (OMITTAG, DATATAG, SHORTREF, etc); Link Process Definitions; Multiple DTDs per document; and many more: see http://www.w3.org/TR/NOTE-sgml-xml-971215 for the list of bits of SGML that were removed for XML;
9. And [nearly] last but not least, no CONCUR!
10. There are some important differences between the internal and external subset portion of a DTD in XML: Marked Sections can only occur in the external subset; and Parameter Entities must be used to replace entire declarations in the internal subset portion of a DTD, eg the following is invalid XML:

```
<!DOCTYPE x [
<!ENTITY % modelx "(A|B)*">
<!ELEMENT x %modelx;>
]>
<x></x>
```

For more information, see McGrath (McGrath, 1998).

4.19 How do I include one DTD (or fragment) in another?

Use a parameter entity, same as for SGML

This works exactly the same as for SGML. First you declare the entity you want to include, and then you reference it by name as a parameter entity:

```
<!ENTITY % mylists SYSTEM "dtds/listfrag.ent">
...
%mylists;
```

Such declarations traditionally go all together towards the top of the main DTD file, where they can be managed and maintained, but this is not essential

so long as they are declared before they are used. You use Parameter Entity Syntax for this (the percent sign) because the file is to be included at DTD compile time, not when the document instance itself is parsed.

Note that a URI is compulsory in XML as the System Identifier for all external file references: standard rules for dereferencing URIs apply (assume the same method, server, and directory as the containing document). A Formal Public Identifier can also be used, following the same rules as elsewhere.

4.20 How can I include a conditional statement in my XML?

You can't, as such: XML isn't a programming language.

You can't as such: XML isn't a programming language, so you can't say things like

```
<foo if{DB}="A">bar</foo>
```

But you can have conditional criteria in a Schema, DTD, or a processor, and some DTDs provide attributes for conditional processing.

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 Boolean 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 have used 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.

In processing languages such as XSLT, there are constructs for conditional processing, both for simple IFs and for exclusive case-by-case choices:

```
<xsl:if test="@foo='bar'">
  <xsl:text>Hello, world!</xsl:text>
</xsl:if>

<xsl:choose>
  <xsl:when test="$type=1">
    <xsl:apply-templates select="//*[@class='special']"/>
  </xsl:when>
  <xsl:when test="$type=2">
    <xsl:apply-templates select="/foo/bar"/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:apply-templates/>
  </xsl:otherwise>
</xsl:choose>
```

DocBook and many other DTDs and Schemas provide attributes on some elements for the specification of **effectivities**, saying which parts of the document apply in which circumstances. Processing software can then isolate these and process them accordingly.

4.21 What's the story on XML and EDI?

Getting there: still needs more work and agreement.

Electronic Data Interchange has been used in e-commerce for many years to exchange documents between commercial partners to a transaction. It requires special proprietary software and is prohibitively expensive to implement for small and medium-sized enterprises. There are moves to enable EDI documents to travel inside XML, as well as proposals to replace the existing EDI formats with XML ones. There are guideline documents at `http://www.eccnet.com/xmledi/guidelines-styled.xml` and `http://www.geocities.com/WallStreet/Floor/5815/guide.htm`.

Probably the biggest effect on EDI is the rise of standardisation attempts for XML business documents and transactions. The standard jointly sponsored by OASIS and United Nations/CEFACT is ebXML (Electronic Business XML) which provides Schemas for the common commercial transaction document types. Normal office documents (letters, reports, spreadsheets, etc) are already being done using the materials under the charge of the OASIS Open Office XML Formats TC, detailed above. Other standards such as OAGI and RosettaNet are undergoing interoperability testing with ebXML.

In addition to full standards, there are many sets of shims, interoperability tools, and component libraries such XML Common Business Library (xCBL).

Chapter 5

Appendices

5.1 References

There is a much larger XML and SGML bibliography at http://xml.coverpages.org/biblio.html.

This list covers only documents directly referenced in this FAQ.

References

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., & Yergeau, F. (Eds.). (2004, February). *Extensible Markup Language (XML) 1.0*. W3C. Boston. Retrieved from http://www.w3.org/TR/REC-xml/

DuCharme, B. (1999). *XML: The Annotated Specification*. Upper Saddle River, NJ: Prentice Hall PTR. Retrieved from http://www.snee.com/bob/xmlann

Flynn, P. (1995, December). Making more use of markup. In *Proc. SGML'95, Boston, MA (7 December 1995)* (p. 158–167). Retrieved from http://imbolc.ucc.ie/~pflynn/articles/moreuse.html

Flynn, P. (1998). *Understanding SGML and XML Tools*. Boston, MA: Kluwer. Retrieved from http://www.amazon.com/exec/obidos/tg/detail/-/0792381696/qxml:id=1128202814/sr=1-1/ref=sr_1_1/102-0476289-3244914?v=glance&s=books

Flynn, P. (2014, July). *Formatting Information: An introduction to typesetting with LaTeX* (7th ed.). Cork, Ireland: Silmaril Consultants. Retrieved from http://latex.silmaril.ie/formattinginformation/

Lamport, L. (1994). *LaTeX: A DOCUMENT PREPARATION SYSTEM* (2nd ed.). Reading, MA: Addison-Wesley.

Maler, E. & el Andaloussi, J. (1995). *Developing SGML DTDs: From Text to Model to Markup*. Upper Saddle River, NJ: Prentice Hall PTR. Retrieved from http://www.amazon.com/exec/obidos/tg/detail/-/0133098818/qxml:id=1104447963/sr=8-1/ref=sr_8_xs_ap_i1_xgl14/002-9386245-9385639?v=glance&s=books&n=507846

McGrath, S. (1998). *XML by Example: Building E-Commerce Applications*. Upper Saddle River, NJ: Prentice Hall PTR. Retrieved from http://www.amazon.com/exec/obidos/tg/detail/-/0139601627/qxml:id=1104449400/sr=8-1/ref=sr_8_xs_ap_i1_xgl14/002-9386245-9385639?v=glance&s=books&n=507846

Pawson, D. (2002). *XSL-FO: Making XML Look Good in Print*. Sebastopol, CA: O'Reilly. Retrieved from http://www.oreilly.com/catalog/xslfo/

Salminen, A. & Tompa, F. (2001, November). Requirements for XML Document Database Systems. In *Proc. ACM Symposium on Document Engineering, Atlanta, GA (2001-11-00)*. Retrieved from http://db.uwaterloo.ca/~fwtompa/.papers/xmldb-desiderata.pdf

Gentle Introduction to XML. (2002). In M. Sperberg-McQueen & L. Burnard (Eds.), *TEI P4: Guidelines for Electronic Text Encoding and Interchange*. Oxford, Providence, Charlottesville, Bergen: Text Encoding Initiative Consortium. Retrieved from http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SG.html

Truss, L. (2003). *Eats, Shoots & Leaves: The Zero-Tolerance Approach to Punctuation*. London: Profile Books. Retrieved from http://www.amazon.com/exec/obidos/tg/detail/-/1592400876/qxml:id=1104449308/sr=8-1/ref=pd_csp_1/002-9386245-9385639?v=glance&s=books&n=507846

5.2 How far are we going?

To infinity and beyond!

Running a search facility on this FAQ has produced some interesting results from the notifications of both matches and non-matches. Sex has dropped to 10th place.

- The most frequent request (5individual characters, either as character entity names or as numeric values, or one of the markup characters (`<` or `&`).
- In recent months the second largest category has stabilised as the word `dtd` (3
- Third comes CDATA at 2given the abuse so widespread).
- Fourth equal at 1of which is dealt with in detail here as they have their own FAQs.

The entertaining bits are deep in the tail, like the user from Broomfield, CO, who typed in 'How can I analyze a telephone to understand it better?' (taking it to pieces is probably a start); the one from the Phillipines who wanted to know how to 'describe the five fundamental interactions between X-rays or Gamma rays with matter' (try DS9); the one from Culver City, CA, who asked 'how are echinodermata organisms different from lower invertebrates?' (like I care?); and the one from Lexington, KY, who asked 'How do I add two text fields?' (got me there, d00d, how do you multiply a lettuce and a cucumber?).

```
Date: Fri, 09 Jul 1999 14:26:17 -0500 (EST)
From: The Internet Oracle <oracle@cs.indiana.edu>
Subject: The Oracle replies!
To: <address-removed>
X-Planation: X-Face can be viewed with ftp.cs.indiana.edu:/pub/faces.

The Internet Oracle has pondered your question
deeply. Your question was:

> Oh Oracle most wise, all-seeing and all-knowing,
> in thy wisdom grant me a response to my request:
>
> Is XML really going to cut the mustard?

And in response, thus spake the Oracle:
Well, since XML is a subset of SGML, and SGML
```

```
has a <cut mustard> tag, I'd have to say yes.

You owe the Oracle a B1FF parser.
```

For the SGML-curious among our readers, that's:

```
<!element cut - o empty>
<!attlist cut mustard (mustard) #required>
<!-- :-) -->
```

5.3 Not the XML FAQ

Infrequently Asked Questions

This is a list of topics that people have asked about or searched for in relation to the XML FAQ, which are not necessarily directly connected to XML and its technology, nor *frequently* asked questions. It also includes some fall-back definitions for the benefit of users who have come to XML by different routes and may not have been exposed to a document publishing background.

Readers may also want to look at Joe English's 'Not the SGML FAQ' at http://www.flightlab.com/~joe/sgml/faq-not.txt.

AJAX Asynchronous HTTP, Javascript, and XML. A technique for improving the interactivity of web pages whereby in-browser scripting detects user activity and pre-fetches the required data asynchronously from an XML-backed data-store, instead of waiting until the user clicks on a link and requesting it synchronously from the server.

ATTRIBUTES These are items of **metadata** or **metainformation** (information about information) which can be added to the start-tag of an element. Usually attributes are a way of refining the meaning, function, or some other quality of an element. They take the form of a name and a quoted value joined by an equals sign, eg

```
<part xml:id="B22" catnum="51N1573R" level="App">Left-handed
      Screwdriver</part>
```

Attribute names must follow the XML rules for Names (see the spec). If your application does not use a DTD or Schema, the attribute values are treated as plain text (CDATA) and cannot have any special meaning to XML (with the exception of `xml:id` and `xml:lang`, see below). In a DTD or Schema, attributes can be assigned datatypes, the most common being (using DTD terminology for simplicity):

ID or IDREF ID attribute values must be XML Names (no spaces; must begin with a letter) and they must be unique in a document. An IDREF attribute value can occur any number of times, but it must be the value of an ID attribute in the same document. ID and IDREF are most frequently used for cross-referencing within documents.

Note that an ID attribute can have any name: it doesn't have to be *called* 'ID', although it frequently is. Conversely — as a matter of best practice — you should never use the name 'ID' ('id') for an attribute which is not of type ID, simply because it's confusing. If your application has unique identity values that the community calls IDs, and which are *not* XML Names, either name the attribute something different (eg 'Product-ID') or document *heavily* that the value is not an XML ID.

There is a W3C Recommendation that document type designers should use the *attribute name* `xml:id`, and this can be interpreted by parsers as being a unique ID without the need for the document to use a DTD or Schema.

CDATA Just text.

Token List The attribute must have one of a restricted number of values (specified in parentheses in the declaration, separated by vertical bars), eg

```
<!ATTLIST part level (App|Jny|Mst) #REQUIRED>
<!ATTLIST Q.27 resp  (Yes|No) "Yes">
```

In the first example there is no default, and a value is compulsory. In the second, 'Yes' is the default value (if the attribute is omitted, the parser will take the default value from the declaration).

ENTITY The attribute value must be a declared Entity.

NMTOKEN An XML Name Token is like an ID value (no spaces) but it *can* begin with a non-letter (eg a digit or punctuation).

Special attributes In addition to `xml:id` (mentioned above), there are two others allowed by the XML Specification:

xml:space to signal an intention that in that element, white space should be preserved by applications;

xml:lang to specify the language used in the contents and attribute values of any element.

See sections 2.10 and 2.12 of the Spec for more detail.

In Schemas a much greater range of datatypes is available than in DTDs, and complex validation criteria can be attached to each.

Attributes in a DTD can be declared as `#REQUIRED` (compulsory), `#IMPLIED` (optional), or `#FIXED` (predefined and invariable).

There is not intended to be any limit on the length of an attribute value, but you should check that your processing software can handle unusual data volumes if you intend to use very large lengths.

BPEL The Business Process Execution Language is an XML-based specification of the steps required for a cooperative business process to take place between consenting servers.

Byte Order Mark A two-byte signature (`0xFEFF`, defined in Unicode and ISO 10646) which must be prepended to the XML document when using the the UCS-2 encoding, in order to allow processors to differentiate between the UCS-2 and UTF-8 encodings.

Colour XML is designed for identifying information about the structure and content of text documents, rather than their appearance. Although it is perfectly possible to identify and store information about appearances, this information is usually kept in a CSS or XSL stylesheet. If you need to record information about the formatting or appearance of an existing document, there are features in the TEI Schema/DTD for doing so.

Data export A common requirement in the flat data model used in many

e-commerce systems is to export XML data to the CSV (Comma-Separated Values) data format used as input to spreadsheets. There is a simple example of a short script to do this here. More complex and sophisticated routines could easily be written using XSLT or other XML processing software. Users should note that while conversion to CSV is adequate for simple data formats, it is an inappropriate format for normal XML text documents which use Mixed Content models.

Data import Many XML projects require the import of existing documents in non-XML formats. The import of existing HTML documents is explained in Question 3.5 on page 40, and if you can convert your documents to XHTML; this is probably the simplest method. OpenOffice saves Open Document Format (ODF) files, which are the international standard for office XML documents. Word files can be saved as WordML (2003) or Office Open XML (2007: Microsoft's alternative to ODF). In both cases an XSLT transformation can be written to create a suitable XML import format. For complex documents in other formats, however, specialist conversion software is needed. Some XML editors are beginning to offer inbuilt conversion of other formats, and there are many standalone conversion systems available (some at high cost) for formats which are otherwise not easily machine-accessible via markup, like PDF, PostScript, LaTeX, Quark XPress, and most proprietary document formats. The critical point is that almost all non-XML (non-SGML) document are formatted to make them human-readable and pretty, not to make them machine-readable. It is therefore often the case that the information required to make the document meaningful in XML simply doesn't exist in these formats. The only alternative for this class of documents is to have them rekeyed or scanned into XML by one of the many companies in the Indian subcontinent or the Pacific Rim.

Disadvantages XML markup has a few disadvantages:

- It can be verbose unless element and attribute names are chosen with care. In large documents the markup overhead need not be large, but in short messages it can be significantly more than the actual data, especially when the element or attribute names are concocted by machine.

- Overlapping markup is not permitted (an element cannot start inside one element and end inside another): element markup must nest hierarchically.
- Most applications require the document to be loaded to memory in its entirety before it can be parsed and processed. This can become a problem for truly huge documents (larger than the addressable memory of a computer system). Arguably, XML is the perhaps wrong tool to use for files this size, but there are streaming systems which will enable them to be processed.
- Some of the software is truly mediocre.

Editing To edit (open) an XML file you should use an XML editor. It is possible to open an XML file using any standard plaintext editor or even a wordprocessor, but be aware that they may try to reformat the file incorrectly because they don't understand XML.

Entities An **entity** is a unit of storage in XML. It can be as small as a character or as large as a whole document. Four types of entity are **declarable**:

General entities which can be like string-replacement macros:

```
<!ENTITY IBM "International Business Machines">
```

These can be used for shorthand data entry or to guarantee uniform spelling like `&IBM;` and they get replaced when the file is parsed.

They can also represent external files:

```
<!ENTITY chap5 SYSTEM "chapter5.xml">
```

which can be used as a file-inclusion mechanism at the point where you insert `&chap5;`. External general file entities must not contain the XML Declaration or any Document Type Declaration.

Document entities These are like external general file entities except that they specify the type of data they contain, using a declared Notation, so that the parser and application can decide how to handle them (eg include them or hand them to another program specific to their type of medium):

```
<!ELEMENT link (#PCDATA)> <!ATTLIST link to ENTITY #REQUIRED>
...
<!NOTATION PDF PUBLIC
  "-//Adobe//NOTATION Portable Document Format//EN//PDF"
  "http://partners.adobe.com/public/developer/pdf/index_reference.html">
<!ENTITY pricelist SYSTEM "/sales/pricelist.pdf" NDATA PDF>
...
<para>Please refer to our <link to="pricelist">current price list</link>.</para>
```

This provides an extremely robust method of defining an external entity once and allowing it to be referenced multiple times (if the external filename changes, you only have to update the entity declaration).

Character entities like `á` to represent characters that users without the required keyboard features may want to enter like 'á';

Parameter Entities are like General Entities but can only be referenced within a DTD. They are used for control of content models, inclusion or exclusion of declarations, and modification of modular constructs:

```
<!ENTITY % local.qandaset.mix "|bibliodiv">
```

(to use an example from the DTD for this FAQ) where the mix of element types in the content model for `qandaset` is specified by the entities `qandaset.mix` (defined by DocBook) *and* by `local.qandaset.mix` (definable by the user [me]) so that the DTD can be tweaked without having to be edited.

General entity names, including XML document entities and character entities, always start with an ampersand (`&`) and end with a semicolon (`;`), and can be used anywhere in your document. Parameter entities can only be used in a DTD: they start with a percent sign (`%`) and end with a semicolon.

Enumeration To count the number of occurrences of a node in an XML document, you can use the ***count*** function in XSL[T], eg

```
<xsl:value-of select="count(//chapter)"/>
```

To apply a counter to a repetitive element type, use the ***xsl:number*** element, eg

```
<xsl:number select="appendix" level="any" format="A"/>
```

For more on XSLT, see Question 2.5 on page 26.

Environment variables XML is a markup language, not a programming language, so it has no concept of environment variables. However, if you are using a DTD, and accessing your XML files under program control (eg in a script rather than by hand) it is possible to modify the value of declared attributes or entities (eg with a stream-editor like sed) before the file is opened, and thereby to pass values from the external environment into the document. A similar approach would be possible with Schemas.

Escaping Escaping means temporarily switching the way a program works to do something different with the data. In SGML, it was conventional to use only ASCII characters in your documents because keyboards, screens, and fonts for other characters were often unavailable. To escape from the limitations of this format for non-ASCII characters like accents and symbols a set of mnemonic names was available, prefixed by an ampersand (`&`) to turn the escapement on, and followed by a semicolon (;) to turn the it off, so an á was given as `á`.

XML allows you to use Unicode, so any character or symbol in any language can be entered as itself. If you are using UTF-8 encoding in your documents, there is no need to use escaping except for the two markup symbols (`<` and `&`). However, not everyone has a Unicode editor, and complete Unicode fonts are very large, so it is conventional in alphabetic languages to pick an encoding which allows you to use the majority of the characters you need, and to use escaping for the occasional other characters.

Floating-point You cannot declare character data content or attribute values as floating-point (or many other data types) using DTDs. To do that you need to use a Schema.

GTT The Gnome Time Tracker is a component of the Gnome interface used extensively on Linux systems. Part of its internal data is configured in XML.

Games I am not aware of any computer games written using XML yet, although XML is used in some of the internal control and configuration files used by games.

Idempotency A term used in the HTTP specification to describe the side-effect-free nature of repeated requests for a resource.

Javascript ECMAscript (to give it its real name) has nothing to do with the Java language. It's designed to run inside browser windows, navigating or acting on the markup of a page to create dynamic content, validate forms, or instantiate objects in ways that are not possible with static HTML. It is also designed so that it cannot write to the user's local filesystem, for obvious security reasons, so it cannot easily be used to create XML files locally, although there are some back-doors in Microsoft software which allow modified pages to be saved to disk.

Line breaks XML files can be created using any of the three standard newline representations: CR (Mac), LF (Unix), or CR/LF (Windows). Use of anything else may lead to undefined behaviour (so old DOS editors that use LF/CR may create unusable files). XML processors normalise all line-ends to LF.

Line-breaking in your output is governed by your rendering engine (eg a browser, a typesetter, etc). Your DTD or Schema may define special elements or entities to be used on rare occasions when a forced linebreak is required, but this is not normally something done in XML (exception: reconstruction of historical documents using the TEI).

Loops To process some XML repetitively, you need to use a processing language which allows looping or the cyclical handling of a defined set of nodes. For example in XSLT, to output all chapter titles to make a table of contents (ie out of natural document position), you could say:

```
<xsl:for-each select="//chapter">
  <li>
    <xsl:value-of select="title"/>
```

```
  </li>
</xsl:for-each>
```

Multimedia The Synchronized Multimedia Integration Language (SMIL) provides an XML vocabulary for simple authoring of interactive audiovisual presentations. SMIL is typically used for 'rich media'/multimedia presentations which integrate streaming audio and video with images, text or any other media type.

Patents, Copyright, and Intellectual Property I'm not a lawyer, and this is not legal advice. If you're worried, see a psychiatrist first ☺

Since the USA (and, increasingly, elsewhere) stopped sanity-checking patent applications, pretty much anyone can patent anything in these countries, regardless of whether or not it already exists. If you are sufficiently intellectually bankrupt, you can then start sending invoices to companies and even individuals demanding payment of license fees for continued use.

XML was drafted during 1995 and first published in 1996, so anyone claiming they invented pointy-bracket self-defining hierarchically-nested structured markup after that is probably a few elements short of a Schema. XML is based on SGML, which is an international standard codified as ISO 8879:1986, and it was preceded by numerous other closely-related markup systems, so anyone claiming they invented it after that date is equally wide of the markup.

Lots of subsequent derivative technologies which owe their existence to the SGML and XML groundwork quite possibly *are* valid patents, in the same way that fire was not originally patented but matches and lighters were.

Patents were originally designed for new physical inventions. Their use for methodologies and algorithms extended the concept into the realm of ideas, which many people regard as deeply suspect. The patenting of natural phenomena like genes (which are pre-existing parts of Nature like politicians and pond scum), is meaningless and intellectually void, although legally enforceable in the USA and elsewhere.

Copyright subsists automatically in anything you create, but in some countries (notably the USA and France) you cannot enforce this unless

you register your interest. Copyright persists for a number of years after your death (EU: 75, different elsewhere) in order to let your descendants benefit from sales of your work.

Copyright is for the physical form of intellectual expression like books, newspapers, works of art, web sites, or computer programs. It exists to prevent others stealing your work and selling it. You can quote snippets of other people's work without permission, such as a line of a poem, or a bar of music, or a sentence from a novel, provided you say whose it is and where to find it: otherwise you need to ask permission beforehand. Copyright already provides more than adequate protection for computer programs, making the use of patents for them unnecessary overkill.

Intellectual Property identifies you as the owner of the thoughts and ideas which may find their physical manifestation in patentable inventions or copyrightable publications. Even if you sell off your patents, and for long after your copyrights have expired, you can still be seen as the person who dreamed up the idea, and some countries (eg the UK) allow you formally to assert your right to be so identified, regardless of what happens to the book or the gizzmo.

You should *always* acknowledge the intellectual property of others, especially when you use it in furtherance of your own aims. Pretending that someone else's smart ideas are your own is probably a worse offence than trying to patent fire, water, the wheel, or XML.

Pipelining Technique for reducing complex sequential and parallel processing requirements to a set of components which can be completed under program control. The term is taken from the Unix facility for redirecting the output of one command into the input of another (called a 'pipe'), in effect creating a chain or pipeline through which data passes on its way from source to result.

The W3C has a Note pending submission on an *XML Pipeline Definition Language* which could be used to define a pipeline in a portable, vendor-independent manner.

RSS The Really Simple Syndication format was designed to allow news sites to process updates by machine, and it evolved into a semi-standard

format for blogs and other frequently-changing sites to notify the world of changes. Unfortunately it was never properly defined, and has multiple incompatible and undocumented versions. It was about to be superseded by a vastly better language called Atom, but Microsoft have recently announced their support for RSS, so it looks like we may be stuck with a lemon for years to come.

'Newsreaders' (RSS readers) are available for all platforms, both standalone and as browser plugins. Do not confuse these with programs of the same description designed to provide access to the Usenet News service, which is a different thing entirely (and which you will need to read at comp.text.xml).

Rendering Using XSLT or XSL:FO transformation (or other similar conversion systems), information marked up in XML can be rendered to almost any target: HTML, PDF, audio, Braille, and almost any plain-text format (eg LaTeX). How it appears (or sounds) is the result of using stylesheets or other transformation logic activated by the markup.

SML The Spacecraft Markup Language is an application of XML.

The Standard ML programming language is not.

Did you mean SGML?

SOAP A W3C standard for the 'definition of the XML-based information which can be used for exchanging structured and typed information between peers in a decentralised, distributed environment'. Most commonly used in Web Services for message-passing.

Originally the Simple Object Access Protocol, the acronym is now undefined, or expressed as the Service-Oriented Access Protocol. Guru99 has a good tutorial on SOAP.

Searching You can search individual XML files on a sequential, stand-alone, unindexed command-line basis using programs such as *lxgrep* or *lxprintf*, parts of the LTXML2 toolkit. Many editors include a search facility as well

XSLT allows a limited search facility simply by using functions like `contains`, `starts-with`, and `ends-with`. XSLT2 adds Regular Expressions. XQuery is a fully-fledged search language for XML.

The *Saxon* XSLT processor comes with an implementation of XQuery (see also the XQL FAQ), which can accept queries either from the command line or from a file. Saxon can also use a control file to specify groups of XML files to be searched together.

For indexed searching (for speed) you need an XQuery search tool that implements an indexing engine which reads and understands markup. These are usually implemented as part of a 'native' XML database system such as *eXist* (and many others), which run either stand-alone or in parallel with an XML server like *Cocoon*.

Traditional relational databases (MySQL, Oracle, etc) tend to store XML as undistinguished strings or BLOBs, using bolt-on XML backends to handle the markup on import and export. 'Native' XML databases have the XML handling built-in, and can be configured for granularity, to store at a specific element level, making markup-sensitive searching much more effective.

Serving XML See Question 4.13 on page 88

Sorting To sort a repetitive set of XML elements in XSL[T], use the ***xsl:sort*** element, eg

```
<xsl:for-each select="//acronym">
  <xsl:sort select="@abbrev"/>
  <xsl:value-of select="@abbrev"/>
  <xsl:text>: </xsl:text>
  <xsl:apply-templates/>
</xsl:for-each>
```

Special characters XML has only two special markup characters in normal documents:

- The open angle bracket or less-than sign (`<`) which begins a start-tag or end-tag like `<report>` or `</table>`;
- The ampersand character (`&`) which starts an **entity reference** like `á` for á or `§` for §.

Contrary to popular opinion, the closing angle bracket or greater-than (`>`) and the semicolon (`;`) are not special characters in normal text: they

only acquire their temporary special meaning once one of the two markup characters has been encountered.

In DTDs, the percent sign (%) has a special meaning in **entity declarations**: it defines the entity as a **parameter entity**, meaning that it can only be used inside the DTD, not in a document text, and only for data substitution (a kind of simple macro).

The exclamation mark (!) acquires a special meaning immediately after a less-than sign: when followed by one of the declaration keywords in a DTD it signals the start of Declaration; when followed by two dashes it signals the start of a comment (ended by another two dashes and a greater-than sign.

TMX TMX is a standard method to describe translation memory data that is being exchanged among tools and/or translation vendors for human-language translation (part of the OSCAR project from LISA).

Tables You can define tables any way you wish in XML (see Question 3.11 on page 50) but there are a few existing table models which have become so widely-used (and supported by software) that it would need a very compelling reason to invent something new. There are more details in Flynn, 1998 §2.3.7.

HTML HTML tables were invented by Mosaic (now Netscape) and first appeared in the HTML2 DTD. In all versions of HTML and XHTML they define a very simple but practical model, with very few refinements, suitable for web use and for rudimentary printing. Their chief advantage is that in a browser the cell heights and widths (and thus the column widths) expand or contract automatically to accommodate the amount of text contained in them. Most other table models assume the widths of the columns and the height of the cells will be specified in advance (which you can do in HTML but this is rarely used).

CALS Computer-Aided Logistics and Support (and several other acronyms over the years) was (is) part of the US military project to ensure a consistent markup for all documentation, originally in SGML, now in XML. As part of this activity the CALS table model has become the most widely-used in technical documentation,

especially for Interactive Electronic Technical Manuals (IETMs), with extensive support in all the major editors, and it is the default table model in the DocBook DTD and Schema. The CALS definitions are very powerful but quite complex, and can handle virtually all requirements for spanning, ruling, and aligning.

SASOUT This model has been used extensively in the social sciences and elsewhere for defining tables based on the semantics of the data, rather than the appearance. At one time they were an alternative in DocBook (enabled by a simple parameter entity switch).

TEI The TEI model is designed to allow the encoder to represent existing tables being transcribed from historical, literary, or archive material, rather than for the generation of new data. The markup is at the same level of simplicity as the HTML model, but it is designed to allow the inclusion of the much denser markup and metadata needed in research texts.

LaTeX The LaTeX model is not of direct concern to the XML user except insofar as LaTeX is a common target for transformations from XML using XSLT in order to create PDFs. Like CALS, LaTeX tables can handle almost any formatting, but the default alignments assume that each column format is defined beforehand, and that each cell will occupy one line of data: an additional package (array) is needed to handle multi-line cells in the way that other models do.

In XML, it is not necessary to use tables to mark up lists as is often done in wordprocessors, because the processing facilities of languages like XSLT allow you to transform the document to use non-tabular methods (like HTML's `divs`). Table markup should therefore be confined to 'real' tables (data arranged in rows and columns) and not abused simply because you want something displayed on a level with something else: it is better to pick markup which is designed to do the job properly rather than to distort existing facilities.

Wordprocessor users are usually unaware that many structures that they currently use wordprocessor tables for are in fact segmented lists, which wordprocessors are incapable of handling correctly. One of the major reasons for doing it properly is that the data can then be

reprocessed to make sense when read in the natural order.

Text document formatting functions Because XML is a metalanguage to let you define and name your *own* information structures, it has no built-in knowledge of anything to start with. It therefore has no inherent understanding of any document specifics like bulleted lists, sections, footnotes, or any of the common online features like drop-down menus, forms (inputs, check boxes, radio buttons, and text areas), scripts, mouseovers, or other bells and whistles — these are things which *you* have to use XML to define, in a DTD or Schema for your specific application. Contrary to the impression given by some manufacturers these things are *not* built into XML itself. You first choose or design a document type (Schema or DTD) to represent your information accurately, then you can generate effects like the above by using CSS styling, or writing an XSL[T] transformation of your XML to HTML, Word, LaTeX, PDF, or whatever other format is capable of instantiating them.

There *are* additional native-XML proposals and recommendations at the W3C for XML Forms handling, XML Linking, XML Security, and a lot of other features, but these are architectural enabling mechanisms, not drop-in replacements for HTML.

UML The Unified Modeling Language has nothing to do with XML, although there are many points of contact, and some software is available to express some UML structures in XML for the purposes of inter-process messaging.

URI parsing errors See 8 on page 42.

Variables XML doesn't have variables or parameters, nor does it have fields or records. These are all terms from programming and database technology, and do not have exact equivalents in XML.

XML identifies your information with **elements** and **attributes**.

WAP The Wireless Application Protocol (WAP) is now handled by the Open Mobile Alliance.

Well-formed See Question 4.3 on page 74.

White-space See Question 3.2 on page 36.

XLL The XML Linking Language comprises the XLink specification and the XPointer specification. For details, see the XML Linking Working Group at the W3C.

XLS Microsoft proprietary spreadsheet file format written by their *Excel* spreadsheet program. XLS files are not XML files, but modern versions of *Excel* save their data as `.xlst` files in Microsoft's Office Open XML format (OOXML).

Do not confuse XLS with XSL (see Question 2.5 on page 26).

XML This is the XML FAQ. Everything in it is about XML. For introductory explanations, see Section 1 on page 6.

XML and security, privacy, and identity standards Eve

XML Protocol There is a Working Group for Web Services at the W3C, and part of their remit is to work on an XML Protocol. See `http://www.w3.org/2000/xp/Group/` for details.

XMLHTTP Feature implemented in MSXML and elsewhere to allow the retrieval of web pages, binary data, or scripted responses under program control (like using *curl*, *wget* or *dog* in a shell script). Used asynchronously in AJaX applications to pre-fetch data, saving time to make it appear that an application is operating locally.

XUL The XML User Interface Language, designed for specifying the user interface in the Mozilla browser.

asp.net ASP (Active Server Pages) is a Microsoft language for serving dynamic web pages, similar in concept to JSP, PHP, and others. In itself, ASP has nothing inherently to do with XML, although like any server-side system, it can be used for serving XML just as well as an other type of file.

.NET itself is an application platform and methodology for web services development on Microsoft servers. Most web services are predicated on XML as the 'common carrier' of inter-business messaging, so .NET has a significant XML component.

Marc Hadley writes:

> There are many alternatives to ASP, most of which use a similar page based approach. Java based alternatives include Java Server Pages (JSP), Java Server Faces (JSF) and Cocoon (which includes eXtensible Server Pages — XSP). Popular scripting language alternatives include Zope (Python) and Rails (Ruby) [all of which have extensive XML support. ¯ Ed.]

5.4 Lost XML software

Some of the best software that has disappeared

The most common cause of lost good software seems to be that the company making it got taken over through no fault of their own, by a corporate shark who didn't know what they were buying, or who simply didn't care. In these cases it wasn't the product that was at fault — often it was popular and selling well; it just fell foul of corporate stupidity.

***Near&Far* (MicroStar)** A standalone visual (graphical) SGML DTD design tool, originally for Microsoft *Windows 95*. N&F made it very easy to prototype a new document type, although later stages of development were usually hand-tuned. It was also an excellent tool for displaying the structure of a newly-encountered DTD. When XML arrived, they kept the internal SGML model but provided a 'save-as' in XML syntax.

Many current design tools have similar embedded functionality (eg *XML Spy*), but there is no equivalent standalone tool of the same quality. A development to use *RelaxNG* to generate different syntaxes would be a major advance.

MicroStar was bought by OpenText Corp and the product was dropped on the floor just at the point when it would have been most useful. If you have a copy (one was embedded in the WordPerfect SGML/XML editor), it still executes under XP, and in Codeweavers' *Wine* under Linux.

***DynaWeb* (EBT)** A family of products: *DynaBase*, the underlying SGML database; *DynaWeb*, a Windows server with a graphically-managed stylesheet system for serving XML or SGML converted to HTML, and

an excellent markup search facility; and *DynaTag*, a GUI system for converting *Word* and *Frame* documents to SGML or XML, based on the original *RainbowMaker* commandline converter.

EBT was bought up by Inso Corp, and the product was ignored for several years. However, a page on Indo's server now claims to provide details, but it is not known if the product is still available. It appears that they inherited some users, so for a while they still had a *DynaWeb* training page.

The good news is that Red Bridge Software now occupies the old EBT factory (under the Red Bridge in Providence, RI), selling a content management system that includes *DynaTag* and some other elements of the original range.

***Panorama* (SoftQuad)** An SGML browser from SoftQuad with an SGML-syntax stylesheet which worked both standalone and as a Netscape plugin, based on Synex *Viewport*. This let users open direct links to SGML documents: *Panorama* would download both instance and DTD via an entity resolver, perform a tokenised parse, and apply the specified stylesheet.

Its unique features included switching between multiple stylesheets, a search result density indicator, and the ability to implement double-ended HyTime links, which let anyone publish their own set of links, even multi-ended links, and even between documents that they didn't own. The browser plugin was free, and the full version included the stylesheet editor.

SoftQuad faltered after Yuri Rubinsky passed away, and was taken over by Corel (*WordPerfect*), where the product was ignored.

SoftQuad's *Author/Editor* SGML editor product transmuted into *XMeTaL*, which is still available from JustSystems.

If you have more information about useful products that have disappeared, please email the editor.

Search results