CS2 Current Technologies note 1

XML and structured data

Peter Buneman

1.1 Introduction

First, some truth in advertising. We are only going to deal with *some* current technologies. There is much important current technology such as security, cryptography, peer-to-peer computing, data mining, information retrieval, etc. that we will not cover, even though it could –and should – be taught under this heading.

What we are going to do is to talk about *data* in various forms and contexts. How do we store it, how do we communicate it, and how do we find or extract it? In doing this we shall cover a mixture of established (relational databases) and emerging (XML) technologies. Although this is only a small part of what could be taught under this heading, it is quite useful. People who know about both databases and XML are in demand. Moreover the topic is interesting. There is much interesting research and development to be done.

It is common to find databases whose size is measured in terabytes. A terabyte is a trillion (10^{12}) bytes and people are now talking about petabyte (10^{15} byte) databases. A petabyte is about what you need to store all the 400,000 or so movies in the Movie Database (`www.imd.com`), a pretty comprehensive catalogue of movies that have been publicly distributed.

Consider a couple of problems involving large databases:

- Google currently indexes about 3 billion pages, which collectively require at least a terabyte of disk space. How does Google find the pages you want in a fraction of a second? Remember that it takes between 1/10 and 1/100 of a second to retrieve something "at random" from secondary strorage, and network delays will make this figure much worse.

- How would you move a few terabytes of data from Edinburgh to Glasgow?

Size is not the only problem. Although some scientists and some medical agencies are now contemplating petabyte databases, these are often quite boring from a "complexity" viewpoint. They are things like enormous collections of X-ray or NMR images. The size is mostly in the images themselves. To a database system they are "BLOBs" (binary large objects), and the system knows nothing about their internal structure. What the system knows about is the "catalogue" information: when and where the pictures were taken, the conditions under which they were taken, perhaps the identity of the patient, etc. This is often called *metadata* – data

about data – and in the cases of these petabyte database is relatively small. Moreover it typically has a very simple structure, so searching the metadata presents no particular problems[1].

More interesting and important than sheer size, is the *structure* of the data. How do we *represent* data and how do we *maintain* and *query* it? Once we have understood these issues, we can then deal with size. As we shall see, until we can deal with structure, it is impossible to deal with size effectively.

Concerning the representation of data. One frequently hears the terms "physical" representation – how do we represent the data as bits in priamary or secondary storage, and "conceptual" representation – how do we *think* about data. To understand the difference, consider real numbers. When you type "$sqrt(x + 3.5)$" into your program you don't usually care about how the floating point numbers x and 3.5 are represented in the computer. In fact, you may not even know how they are represented. All you care about is that the operations you specify should work, and should produce the expected output. You are using the conceptual representation, but internally the "physical representation is being used. The same is true – in a much richer way – in databases. Sometimes you will find the term "logical" annoyingly used for "conceptual" (as if to say that the digital representation is "illogical"!) Unfortunately these terms stick. When we talk to a conventional (relational) database we do so in SQL and use – at least initially – the conceptual representation. Another way of thinking about this is that SQL is the *abstraction* of *interface* in programming language terminology.

We are going to start our understanding of data representation not in the established world of relational databases, but in the world of *text*, structured text, and XML. This is an area in which, unfortunately, little thought has been given to size, but leads to some interesting connections with structure.

1.2 Data formats

Well before database technology or document standards were developed there were "data formats". These are typically text files with some structure that is designed to make them easily readable by both humans and machines. For example, suppose you want to store a large number of X-ray images. You will probably first find a way of compressing the image so that it can be efficiently stored. But you then have to deal with the metadata. This needs to be both human and machine readable for obvious reasons. By machine readable we really mean "easy to parse". We don't want to rely on complicated things like natural-language processing to guess at, say, the name of the patient. We'd like some simple method of idicating the name in the metadata in such a way that programs that look at the metadata can easily find the name.

Two rather different data formats are shown in Figure 1. The left-hand side is a format used by a common calendar program. Note how, for example, the "Owner" is indicated. It is a relatively simple matter to write a program that finds the entry with a given "Owner" field and prints out that entry. The left-hand side is *one* entry in a well-known biological database, Swiss-Prot.

[1]The "engineering" involved in storing a petabyte (10^6 disk drives?) should not be underestimated. Nor should the problem of computing with it. How would you apply a computationally expensive pattern recognition program to each of a million X-rays images? This can only be done by distributed computing, and it is what the Grid is about.

```
Appt [
Start [1140]
Length [150]
Uid
[robin.broadband.coma8c057f9204]
Owner [Robin]
Text [23 [Dinner with Jean
Dupont]]
Remind [5]
Hilite [always]
Dates [3/2/2004]
]

Note [
Length [30]
Uid [acom-
puter.ed.ac.uka8c0020057f9205]
Owner [Robin]
Text [16 [DBS Homework due]]
Remind [10]
Hilite [always]
Todo []
Dates [ 23/1/2004]
]

Appt [
Start [720]
Length [60]
Uid [acom-
puter.ed.ac.uka8c002006aa21fb]
Owner [Robin]
Text [14 [DBS Darwin G10]]
Remind [1]
Hilite [always]
Dates [12/1/2004 26/1/2004
2/2/2004]
]

Note [
Length [30]
Uid [acom-
puter.ed.ac.ukd7811b3a227a132]
Owner [Robin]
Text [11 [Term ends!!]]
Remind [5]
Hilite [always]
Dates [14/3/2003]
]
```

```
ID  143B_HUMAN STANDARD; PRT; 245 AA.
AC  P31946;
DT  01-JUL-1993 (REL. 26, CREATED)
DT  01-FEB-1996 (REL. 33, LAST SEQUENCE UPDATE)
DT  01-OCT-1996 (REL. 34, LAST ANNOTATION UPDATE)
DE  14-3-3 PROTEIN BETA/ALPHA (PROTEIN KINASE C INHIBITOR PROTEIN-1)
DE  (KCIP-1) (PROTEIN 1054).
GN  YWHAB.
OS  HOMO SAPIENS (HUMAN).
OC  EUKARYOTA; METAZOA; CHORDATA; VERTEBRATA; TETRAPODA; MAMMALIA;
OC  EUTHERIA; PRIMATES.
RN  [1]
RP  SEQUENCE FROM N.A.
RC  TISSUE=KERATINOCYTES;
RX  MEDLINE; 93294871.
RA  LEFFERS H., MADSEN P., RASMUSSEN H.H., HONORE B., ANDERSEN A.H.,
RA  WALBUM E., VANDEKERCKHOVE J., CELIS J.E.;
RL  J. MOL. BIOL. 231:982-998(1993).
CC  -!- FUNCTION: ACTIVATES TYROSINE AND TRYPTOPHAN HYDROXYLASES IN THE
CC      PRESENCE OF CA(2+)/CALMODULIN-DEPENDENT PROTEIN KINASE II, AND
CC      STRONGLY ACTIVATES PROTEIN KINASE C. IS PROBABLY A MULTIFUNCTIONAL
CC      REGULATOR OF THE CELL SIGNALING PROCESSES MEDIATED BY BOTH
CC      KINASES.
CC  -!- SUBUNIT: HOMODIMER.
CC  -!- SUBCELLULAR LOCATION: CYTOPLASMIC.
CC  -!- TISSUE SPECIFICITY: 14-3-3 PROTEINS ARE LOCALIZED IN NEURONS, AND
CC      ARE AXONALLY TRANSPORTED TO THE NERVE TERMINALS. THEY MAY BE ALSO
CC      PRESENT, AT LOWER LEVELS, IN VARIOUS OTHER EUKARYOTIC TISSUES.
CC  -!- PTM: ISOFORM ALPHA DIFFERS FROM ISOFORM BETA IN BEING
CC      PHOSPHORYLATED (BY SIMILARITY).
CC  -!- ALTERNATIVE PRODUCTS: TWO FORMS ARE PRODUCED BY ALTERNATIVE
CC      INITIATION (BY SIMILARITY).
CC  -!- SIMILARITY: BELONGS TO THE 14-3-3 FAMILY OF PROTEINS.
DR  EMBL; X57346; G23114; -.
DR  MIM; 601289; -.
DR  PROSITE; PS00796; 1433_1; 1.
DR  PROSITE; PS00797; 1433_2; 1.
KW  BRAIN; NEURONE; PHOSPHORYLATION; ACETYLATION; MULTIGENE FAMILY;
KW  ALTERNATIVE INITIATION.
FT  INIT_MET 0 0 BY SIMILARITY.
FT  INIT_MET 2 2 IN SHORT FORM (BY SIMILARITY).
FT  MOD_RES 1 1 ACETYLATION (BY SIMILARITY).
FT  MOD_RES 2 2 ACETYLATION (IN SHORT FORM)
FT  (BY SIMILARITY).
FT  MOD_RES 185 185 PHOSPHORYLATION (BY SIMILARITY).
SQ  SEQUENCE 245 AA; 27951 MW; CE0EADFE CRC32;
    TMDKSELVQK AKLAEQAERY DDMAAAMKAV TEQGHELSNE ERNLLSVAYK NVVGARRSSW
    RVISSIEQKT ERNEKKQQMG KEYREKIEAE LQDICNDVLE LLDKYLIPNA TQPESKVFYL
    KMKGDYFRYL SEVASGDNKQ TTVSNSQQAY QEAFEISKKE MQPTHPIRLG LALNFSVFYY
    EILNSPEKAC SLAKTAFDEA IAELDTLNEE SYKDSTLIMQ LLRDNLTLWT SENQGDEGDA
    GEGEN
//
```

Figure 1: Sample data formats

There are currently about 100,000 entries in this database, so its size (as a text file) is less than a gigabyte. At the bottom of this entry is the data – the protein sequence itself. Above it is the metadata. This includes for example the citations to the papers that describe the sequence as well as a variety of *annotations* and links to other databases. It is sometimes the metadata, rather than the data, that makes this database so valuable.

As an aside, I should mention that I am using the term "database" in its rather general meaning as a "pile of data". Later, we may use it in a rather more specific way, to mean something that is managed by a database management system, but for the time being formatted text files are certainly databases.

There are literally thousands of data formats in use. Look at the .conf, .cf, .config, ... files on your Unix system. There's a separate data format (or "scripting language") for each application. There are numerous formats for storing scientific data, some –such as the format used in Swiss-Prot in Figure 1 are specific to a certain kind of data. Others, such as ASN.1 are quite general and look a bit like the type system of a good programming language. They have representations for a variety of base types (int, float, text, ...), records (tuples/classes), lists etc. If we could agree on a common data format, it would at least help us by providing basic parsing techniques for all data.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="content-type"
content="text/html; charset=ISO-8859-1">
  <title>Example</title>
</head>
<body>
<h3>This is a heading</h3>
Introduction<br>
<ul>
  <li>Point 1</li>
  <li>Point 2</li>
</ul>
The end<br>
<img src="file:///home/peter/cs2/Logo1.gif" alt=""
style="width:  237px; height:  70px;"><br>
<br>
</body>
</html>
```

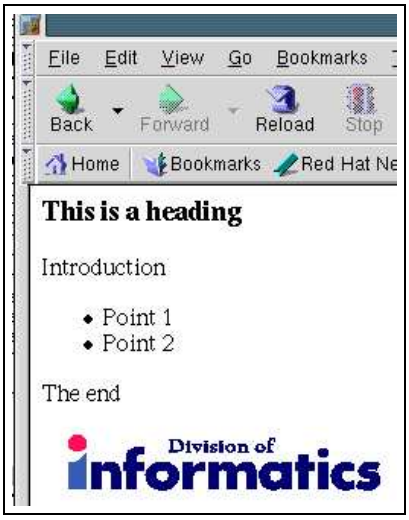


Figure 2: Some HTML and its display

1.3 Markup languages

Another form of structured text with a long history is *markup languages*. One of the best-known of these is HTML, which is used for displaying text in your browser (Figure 2). It's popularity derives from the invention of the Web and its use in browsers. However there are many others in use. This document was typeset with a language called LaTeX. Any WYSIWIG editor has some markup language (often proprietary).

HTML is in fact a restricted form of SGML (Standard Generalised Markup Language). This is a very general language that allows one to specify a number of markup languages. XML is also specified by SGML. In fact we don't need to understand the full generality of SGML to understand HTML and XML. We can start by learning some of the teminology of HTML. The angle-bracketted text items such as `<title>` and `</title>` are called *tags* – opening and closing tags respectively. Between tags we can find both text (called PCDATA for "parsed character data") and other tags. When an opening tag has no matching closing tag it is called a *bachelor* tag.

Within an opening tag we may find *attributes* after the name. Each attribute itself has a name and an associated value. Attributes are typically used, in HTML, to tell us how something is to be displayed for some extra information. (For example `alt = ...` is used to say what text is to be displayed if the browser cannot display the image).

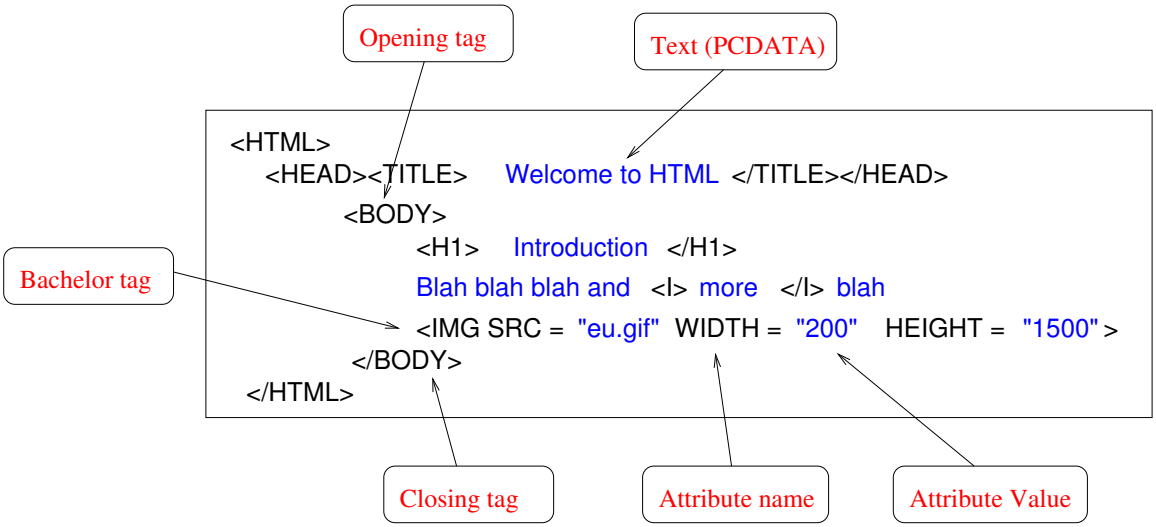


Figure 3: The structure of HTML

1.4 From HTML and data formats to XML

HTML has a *fixed* tag set – one that is designed to describe the format of a document. Now, if we look at the calendar data format in Figure 1 we can think of a number of ways of displaying the calendar, and the structure of the data should support any of these displays. So the first thing we do is to allow ourselves to define the tag set. The second and fundamental observation is that tags and closing tags match up. This is a form of *labelled bracketting*. There are other ways of describing labelled brackets. In the calendar format we would say `Appt [ ... ]` while in XML we would say `<Appt> ... </Appt>`. In XML *bachelor tags are not allowed*[2]. The XML notation for labelled brackets is somewhat more verbose that the calendar notation. It is there because it often difficult to match up opening and closing tags, e.g., `A[B[C[] D[E[] F[G[H[I]]] K[J]]]]`. And `<book> ... hundreds of pages of text ...</book>` is surely better than `book[...]`.

A fundamental restriction on tags is that they must be properly *nested*. You cannot have `<i> italic text <b> bold italic text </i> bold text </b>` even though this makes sense[3].

With this nested tag restriction, XML can be thought of as a tree. Each opening/closing tag pair gives rise to a node – an *element node* and the children of that node are text nodes, attribute nodes or other element nodes.

It should be clear that it is easy to translate the calendar format directly into XML. One might go further and structure the dates. Thus rather than translating

```
Dates [12/1/2004 19/1/2004 26/1/2004 2/2/2004]
```

simply as

```
<Dates> 12/1/2004 19/1/2004 26/1/2004 2/2/2004 </Dates>
```

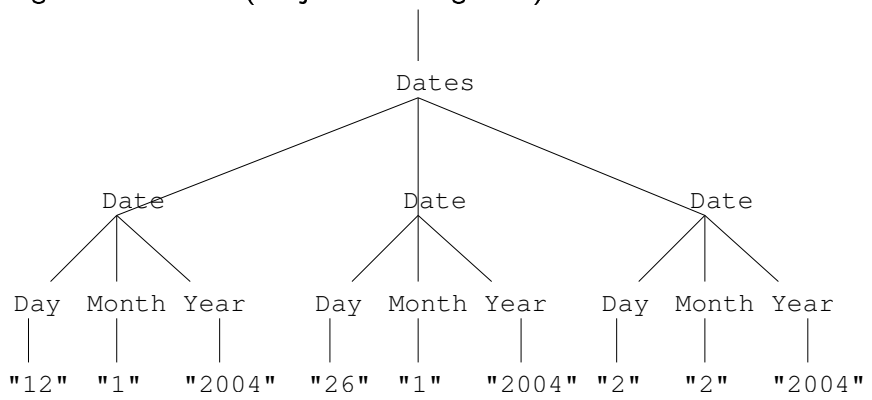
[2]Recent versions of HTML are subsets of XML. Thus we can't have bachelor tags such as `<br>` (for paragraph break. What we do instead is to use `<p/>` as a shorthand for `<br></br>`

[3]HTML parsers are designed not to break. So they will probably accept this but with indeterminate results

one could "structure" the text to get:

```
<Dates>
    <Date> <Day> 12 </Day> <Month> 1 </Month> <Year> 2004 </Year> </Date>
    <Date> <Day> 26 </Day> <Month> 1 </Month> <Year> 2004 </Year> </Date>
    <Date> <Day> 2 </Day> <Month> 21 </Month> <Year> 2004 </Year> </Date>
</Dates>
```

with the following tree structure (for just this fragment):



Some more terminology. The segment of an XML document between an opening and a corresponding closing tag is called an *element*.

```
1. ⟨person⟩
2.   ⟨name⟩ Malcolm Atchison ⟨/name⟩
3.   ⟨tel⟩ 0141 247 1234 ⟨/tel⟩
4.   ⟨tel⟩ 0141 898 4321 ⟨/tel⟩
5.   ⟨email⟩ mp@dcs.gla.ac.sc ⟨/email⟩
6. ⟨/person⟩
```

The text fragments ⟨person⟩ ... ⟨/person⟩ (lines 1-6), ⟨name⟩ ... ⟨/name⟩ (line 2), etc. are elements. The text between two tags is (e.g. lines 2-5) is sometimes called the *contents* of an element.

Note that it is possible to have content which contains a mixture of elements and text:

```
⟨airline⟩
    ⟨name⟩ British Airways ⟨/name⟩
    ⟨motto⟩
        World's ⟨dubious⟩favorite⟨/dubious⟩ airline
    ⟨/motto⟩⟨/airline⟩
```

This is called *mixed content* XML generated from databases and data formats typically does not have mixed content. However it is frequently used in HTML, and remember that HTML is supposed to be a special case of XML.

1.5 Designing an XML format

Let's consider a "database" example:

- Projects have titles, budgets, managers, ...
- Employees have names, employee ids, ages, ...

There are several possibilities:

1. Employees and projects intermixed

```
⟨db⟩
    ⟨project⟩
      ⟨title⟩ Pattern recognition ⟨/title⟩
      ⟨budget⟩ 10000 ⟨/budget⟩
      ⟨manager⟩ Joe ⟨/manager⟩
   ⟨/project⟩
   ⟨employee⟩
      ⟨name⟩ Joe ⟨/name⟩
      ⟨empid⟩ 344556 ⟨/empid⟩
      ⟨age⟩ 34 ⟨/age⟩
   ⟨/employee⟩
   ⟨project⟩ ... ⟨/project⟩
   ⟨project⟩ ... ⟨/project⟩
   ⟨employee⟩ ... ⟨/employee⟩
⟨/db⟩
```

2. Employees and Projects Grouped

```
⟨db⟩
   ⟨projects⟩
      ⟨project⟩
         ⟨title⟩ Pattern recognition ⟨/title⟩
         ⟨budget⟩ 10000 ⟨/budget⟩
         ⟨manager⟩ Joe ⟨/manager⟩
      ⟨/project⟩
      ⟨project⟩ ... ⟨/project⟩
      ⟨project⟩ ... ⟨/project⟩
   ⟨/projects⟩
   ⟨employees⟩
      ⟨employee⟩ ... ⟨/employee⟩
      ⟨employee⟩ ... ⟨/employee⟩
   ⟨/employees⟩
⟨/db⟩
```

3. No tags for employees or projects

```
⟨db⟩
      ⟨title⟩ Pattern recognition ⟨/title⟩
      ⟨budget⟩ 10000 ⟨/budget⟩
      ⟨manager⟩ Joe ⟨/manager⟩
      ⟨name⟩ Joe ⟨/name⟩
      ⟨empid⟩ 344556 ⟨/empid⟩
      ⟨age⟩ 34 ⟨/age⟩
```

```
        ⟨title⟩ ... ⟨/title⟩
        ⟨budget⟩ ... ⟨/budget⟩
        ⟨manager⟩ ... ⟨/manager⟩
        ⟨name⟩ ... ⟨/name⟩
        ...
    ⟨/db⟩
```

Here we have to assume more about the tags and their order.

We might want to express other things in the XML for example that employees work on projects, that the manager of a project must be an employee, or that employee ids are unique. We need to add more to XML in order to state these constraints. In the coming lectures we shall see how we can constrain the structure of an XML document by something aking to the type system of a programming language (though it serves a very different purpose.)

1.6 Attributes

We have already noted that attributes are typically used to describe the content of an element:

```
⟨entry⟩
      ⟨word language = "en"⟩ cheese ⟨/word⟩
      ⟨word language = "fr"⟩ fromage ⟨/word⟩
      ⟨word language = "ro"⟩ branza ⟨/word⟩
      ⟨meaning⟩ A food made ... ⟨/meaning⟩
⟨/entry⟩
```

Another common use for attributes is to express dimension or type:

```
⟨picture⟩
    ⟨height dim= "cm"⟩ 2400 ⟨/height⟩
    ⟨width dim= "in"⟩ 96 ⟨/width⟩
    ⟨data  encoding = "gif" compression = "zip"⟩
        M05-.+C$@02!G96YE<FEC ...
    ⟨/data⟩
⟨/picture⟩
```

It's not always clear when to use attributes:

```
⟨person id = "123 45 6789"⟩
    ⟨name⟩ F. McNeil ⟨/name⟩
    ⟨email⟩ fmcn@barra.org.sc ⟨/email⟩
⟨/person⟩
```

```
⟨person⟩
    ⟨id⟩ 123 45 6789 ⟨/id⟩
    ⟨name⟩ F. McNeil ⟨/name⟩
    ⟨email⟩ fmcn@barra.org.sc ⟨/email⟩
⟨/person⟩
```

There are some important things that distinguish attributes from elements:(a) the value of an attribute is always text (it cannot contain other elements), (b) the order of the attributes is unimportant, and (c) two attributes in a given tag must have different names. A document is said to be *well-formed* if it does not violate (a) or (c), it obeys the "nested tags" rule, and it satisfies some other constraints that need not concern us.

1.7 Style sheets

It is frequently necessary to display an XML document into marked-up text suitable for display. So, for example, we might want to display our Employee/Projects data in tabular form. Thus we would want to turn the XML into HTML, LaTeX, or the internal format of your WYSIWIG editor. In fact one may want to transform HTML into HTML to satisfy different publishing conventions or the formats used for different languages.

XSLT is a language for doing this. It is expressed in XML and consists, roughly speaking of "rules" and "actions". It is built into some browsers. For example, if an `employee` element is encountered, generate table row and then go and apply other rules to the contents of that element to generate the contents of the row. Looking at the three representations of our Employee/Project database, the representation that groups employees and projects separately would seem to yield to this sort of treatment, but it is much more difficult to devise a set of rules that will transform the other two representations into a tabular representation. In fact it is quite hard to specify such transformations in XSLT. We shall not study style-sheet languages in this part of the course. They are easy to pick up. Instead we shall study somewhat more powerful and current technology: an XML query language. This connects nicely with our subsequent studies of SQL and databases.

However it's important to note that, although this query language can express very rich transformations of XML, its output is always XML. Sooner or later, if we want to display something in a format other than HTML we are going to need a style sheet language.

1.8 Some URLs

- XML standard: `http://www.w3.org/TR/REC-xml`

  A caution. Most W3C standards are quite impenetrable. There are a few exceptions to this –some of the XQuery and XML schema documents are readable – but as a rule, looking at the standard is *not the place to start*

- Annotated standard: `http://www.xml.com/axml/axml.html`. Useful if you are consulting the standard, but not the place to start.

- Lots of good stuff at `http://www.oasis-open.org/cover/xml.html`

- Pedestrian tutorials: `http://www.w3schools.com/xml/default.asp` and `http://www.spiderpro.com/bu/buxmlm001.html`

- General articles/standards for XML, XSL, XQuery, etc.: `http://www.w3.org/TR/REC-xml`