DE GRUYTER
OPEN

ICAME Journal, Volume 38, 2014, DOI: 10.2478/icame-2014-0004

Modest XML for Corpora: Not a standard, but a suggestion

Andrew Hardie, Lancaster University

Abstract

*This paper argues for, and presents, a modest approach to XML encoding for use by the majority of contemporary linguists who need to engage in corpus construction. While extensive standards for corpus encoding exist – most notably, the Text Encoding Initiative's Guidelines and the Corpus Encoding Standard based on them – these are rather heavyweight approaches, implicitly intended for major corpus-building projects, which are rather different from the increasingly common efforts in corpus construction undertaken by individual researchers in support of their personal research goals. Therefore, there is a clear benefit to be had from a set of recommendations (*not *a standard) that outlines general best practices in the use of XML in corpora without going into any of the more technical aspects of XML or the full weight of TEI encoding. This paper presents such a set of suggestions, dubbed* Modest XML for Corpora*, and posits that such a set of pointers to a limited level of XML knowledge could work as part of the normal, general training of corpus linguists.*

The Modest XML *recommendations cover the following set of things, which, according to the foregoing argument, are sufficient knowledge about XML for most corpus linguists' day-to-day needs: use of tags; adding attribute-value pairs; recommended use of attributes; nesting of tags; encoding of special characters; XML well-formedness; a collection of* de facto *standard tags and attributes; going beyond the basic* de facto *standard tags; and text headers.*

1 Introduction

In this paper, I wish to lay out explicitly – and argue in favour of – an approach to the use of XML in corpus creation that is in line with much actual practice but which is out of line with what we might call the 'official', or to be more accurate *de jure*, standards that have been established and promulgated. The paper is unconventionally structured, in that the larger, latter part of the paper is devoted to presenting the set of 'modest' suggestions which I argue should form the core of most linguists' education in the use of the XML; the shorter first part of the paper is devoted to laying out the argument in favour of this modest approach.

2 The de jure *standards: Some background*

To explain why I consider it necessary to promulgate a modest approach to the use of XML in corpus linguistics, a brief recapitulation of some history is necessary. Certain fairly limited methods of marking up corpus files were established early on (for instance, COCOA references: see McEnery and Wilson 2001: 34–35). However, a fully systematised and flexible approach to encoding not only structural markup but also text-level and corpus-level metadata as well as analytic annotations did not arise until the *Standard Generalised Markup Language* (SGML) was adopted for this purpose. SGML is well-known as a markup system based on the use of angle brackets to delimit the tags from the actual corpus text. Critically, SGML itself does not actually define any tags. Rather, it consists of a set of general rules, together with a system by which a vocabulary of tags for some particular purpose can be defined. Such a vocabulary is called an *application* of SGML. The single most widely-known application of SGML for corpus encoding is that of the Text Encoding Initiative (TEI). The TEI originated from discussion at a conference in 1987; its first edition was published in 1990 under the sponsorship of three professional organisations in the field nowadays usually known as *digital humanities*: the Association for Computers and the Humanities, the Association for Computational Linguistics, and the Association for Literary and Linguistic Computing (Sperberg-McQueen and Burnard 1990: iii–iv; see also Ide 1996a). It is now curated by a dedicated organisation, the TEI Consortium (or TEI-C for short). The most recent major revision of the TEI Guidelines, the fifth (P5), was published in 2007, but the online edition is continually updated and at time of this writing stood at version 2.5.0 (Burnard and Bauman 2013).

While originally an SGML application, with the advent of XML, the TEI Guidelines were quickly migrated to this newer standard markup language. XML, the *eXtensible Markup Language*, is a somewhat restricted extension of SGML, which omits many of the features that can make SGML complex to process. The development of XML in the late 1990s, under the aegis of the Word Wide Web Consortium (W3C), was driven primarily by the needs of data interchange on the internet (Bray *et al*. 2008: §1.1). However, XML was also quickly established as preferable to SGML for most text-encoding purposes – including corpus-encoding.[1] For example, the use of XML is endorsed by many authors in the edited collection that is, perhaps, the most generally accepted contemporary source on good practice in corpus construction, namely Wynne (2005).

The TEI Guidelines are not solely a standard for linguistic corpora but have a rather wider remit, in keeping with their origin in the digital humanities, as their full title – *Guidelines for electronic text encoding and interchange* – indicates. Typical applications of TEI that are of less interest to corpus linguistics (narrowly defined) include (a) the creation of digital scholarly editions of manuscripts, with detailed manual annotations, preservation of differences between variant exemplars, and so on; (b) the creation of archival-standard digital texts that represent, so far as possible, all features of the original appearance of the document; (c) the recording of full document metadata according to the standards accepted by library science. An SGML application related to TEI that has a more narrow focus on the requirements of corpus encoding emerged in the mid-1990s: the Corpus Encoding Standard (CES: see Ide 1996b). As the introduction to this standard notes:

> The TEI Guidelines are expressly designed to be applicable across a broad range of applications and disciplines and therefore treat not only a vast array of textual phenomena, but are also designed with an eye toward the maximum of generality and flexibility. Most applications will use only those parts of the TEI that are required to meet their needs. The CES is such an application; we have utilized the TEI modular DTD and the TEI customization mechanisms to select those pieces of the TEI that are appropraite [sic] for corpus encoding.
> Ide (1996b: §0.1)

Thus, CES – and its XML-compliant offspring XCES – constitutes only those parts of TEI judged of greatest use for corpora. Today, then, an aspiring corpus builder has two *de jure* international standards to choose from, the TEI guidelines and XCES, both of which provide (a) a formal, machine-parseable specification consisting of a vocabulary of XML elements and their possible attributes, and a grammar of how these elements interrelate (i.e. which elements are allowed to contain which other elements), in the shape of either a Document Type Definition (DTD) or an XML Schema; (b) a set of written guidelines defining and explaining how each of these XML elements are to be used in practice.

3 Limitations of the de jure *standards*

The TEI and CES are both products of their time. In the period from 1987 to 1996, corpus creation was not a task to be undertaken lightly. Collecting any sizeable amount of text in electronic form was still fraught with difficulties. This being the case, it was by and large safe to assume that the 'average' corpus would be widely distributed and re-used. Therefore, dense and careful markup of as many potentially useful features as possible was highly desirable – since the requirements of the corpus user could not be precisely predicted. Moreover, given the scale of corpus building projects, it was also safe to assume that any corpus construction team would have access to personnel with a substantial degree of technical knowhow: computer scientists, or at least programmers. The British National Corpus (BNC; see Aston and Burnard 1998) is perhaps the paradigmatic example of this kind of corpus. Developed by a large consortium including both academic and commercial partners, and with access to the expertise of linguists, computer scientists, and lexicographers, it was constructed with the intention of being made widely available for use in general research on British English. Far more such research than can possibly be mentioned here has, of course, been carried out in the two decades since the BNC was published; and indeed, the BNC has always used TEI encoding, initially in the SGML version but most recently in XML format.

Today, *some* corpus development follows the BNC-style model for development of large, standardised resources, created by large teams or consortia, for a wide general audience. An example would be the ongoing development of the text form of the Early English Books Online (EEBO) collection by the EEBO Text Creation Partnership (EEBO-TCP).[2] In such cases, the use of TEI or a similar standard is wholly appropriate, and projects do indeed make use of TEI or some similar standard; EEBO-TCP uses a system derived from TEI. However, it would be difficult to maintain that these kinds of undertakings constitute the 'average' or typical corpus creation undertaking as they did in the early 1990s. A much larger number of corpora are created by individual researchers for their own research interests, with no expectation that the resource will ever be used by anyone else. This is entirely a consequence of the much greater availability of electronic text and its easy accessibility especially via the medium of the World Wide Web. For this kind of corpus creation enterprise, the assumptions that are true of projects like the BNC do not apply. First, there is no especial need for any markup directed at the speculated requirements of end-users, since there *is* no end-user other than the corpus creator. Second, when a corpus is created by a single linguist, the assumption of a high degree of technical expertise related to computers is by no means always met. While some linguists do have a high level of computer knowhow up to and including programming skill, many do not, and the democratisation of corpus linguistics necessitates that both sorts of researcher should be able to build corpora as required. Today, for instance, in terms of availability of text it is quite feasible for an undergraduate to collect a fairly large corpus as the basis for a coursework assignment; for such a student to implement TEI would be a drastically greater challenge than collecting the text. More generally speaking, expertise in the structure and exploitation of XML document types is a different skillset to expertise in corpus construction and analysis; there is no reason to expect that the two will necessarily intersect in a singular researcher.

In this kind of context, the accepted standards such as TEI and CES start to look somewhat top-heavy or over-engineered, in at least three related senses. First, they are over-weighty in that the standards themselves are extremely complex documents. The printed copy of TEI P1 (Sperberg-McQueen and Burnard 1990) on my bookshelf is 290 pages long; I have not seen a printed copy of the most recent P5 but based on its contents I surmise it must be substantially longer. Likewise, when I once printed out a copy of the CES documentation, it filled a standard ring binder to capacity. The amount of work required for a novice to come to a reasonable understanding of a standard of this scope and complexity is itself very great. Second, the level of markup that these standards dictate is over-weighty. For instance, despite the TEI being defined in such a way that large parts of it are optional and there is a complete architecture for customisation, the minimal TEI file header is a very large, complex block of markup. To insert the minimum required markup for compliance into a corpus is thus, again, a considerable effort. Third, there is a substantial degree of technical overhead in the use of these standards in terms of getting the appropriate DTD working together with an XML parser and/or with whatever other software is to be employed. Perversely, the use of a full encoding standard with a DTD can make it actually harder to use the data, since one slip on the part of the corpus builder can lead to a failure of the document to validate, usually forcing XML-aware software to abort with a hard-to-understand error message.

The opposite extreme to these heavyweight standards is to rely solely on plaintext corpora, with no explicitly encoded internal structure at all beyond possibly a division into multiple files. However, this would hardly be desirable as a universal recommendation. There are indeed research questions for which raw text alone will suffice. But in many cases the possibility of marking up structure or analytic annotation into a file is a valuable one. XML is ideal for these purposes, because of its standard nature and also because so much corpus software is (at least partially) XML-aware. However, for the reasons of weight noted above, the *de jure* standards do not provide a very good way for most linguists to approach such applications.

What then is the solution? Two points may be made. First, a number of practices in XML markup for corpora have become common to the point of being *de facto* standard, so that it is not necessary to use a maximalist standard in full in order to get the benefits of XML in general and of complying with established standard practice. Two simple examples would be the use of <p> and <u> elements to mark out the limits of paragraphs and utterances respectively. These are both part of TEI, but have also seen wider use, to the point where their use may be considered *de facto* standardised in the field of corpus linguistics. They can easily be used outside the context of a heavyweight standard. So an approach to markup which satisfies the needs of the individual researcher does exist. Second, the level of expertise required to implement a relatively limited quantity of XML markup is substantially less than that required to implement a full heavyweight standard. It is not unreasonable to hope that such a level of expertise could be communicated swiftly and painlessly.

4 Existing lightweight standards and documentation for XML in corpora

Much of the argument in the previous section is not original to me and there have been many attempts – often within the context of the heavyweight standards – to present some part or other of the standards in a lightweight way. However, none of the attempts to date strikes me as wholly adequate to the needs of corpus linguists. For example, the reduced *TEI Lite* standard (Burnard and Sperberg-McQueen 2012), designed “to meet 90% of the needs of 90% of the TEI user community”, is *still* on the heavyweight side, being in excess of 60 thousand words in length. Another rightly renowned component of TEI is the *Gentle Introduction to XML* (Burnard and Bauman 2013: §v), which focuses on XML in general as well as an introduction to how XML is used within TEI. The *Gentle Introduction* is incomparably valuable for readers who wish to learn, for instance, how to write a DTD or to define a vocabulary of entity references. But that very quality makes it excessively technical for someone who wants only to apply a modest level of XML in a corpus they have built. Outside the TEI, Burnard (2005) sets out by providing a lightweight description of several aspects of XML, but moves on to present a recommendation on the use of XML metadata that incorporates a fair proportion of the complexity of TEI. In sum, then, much of the introductory level literature on XML encoding for corpora still goes considerably beyond the level appropriate for a corpus linguist with novice-level technical skills.

5 Criteria for a useful minimalist system

I have argued that there is a benefit to be found in providing a written outline of a minimal set of 'normal' – that is, *de facto* standard – practices for using XML in a way that is easy for anyone to understand, and avoiding all the more technical aspects of the heavyweight standards. What qualities should such an outline possess? Desirable features include the following:

- The outline should be short: readable in a single sitting.
- The outline should be written in a non-technical style accessible to beginners.
- Any linguist from the level of a bright undergraduate upwards should be able to understand it, and by reading it progress from knowing nothing about XML to being able to read and/or apply basic tags.
- The outline should not attempt to present a standard to which adherence is expected, but rather, an open-ended set of suggestions as to what is considered good practice.

In technical terms, the following aspects of XML must be covered:

- How tags are encoded (open, close and empty) .
- How attribute-values are encoded.
- How tags are nested within a single document element.
- The *de facto* standard tags (like s, p, u, and so on) and attributes (like id and n).
- Use of entities for the XML special characters.

(This last is a point that anyone essaying to build a corpus needs to be aware of, lest an instance of the less-than sign < in their data be misinterpreted as an opening angle bracket by some piece of XML-aware software.) By contrast, in the interests of maintaining simplicity, the following more advanced aspects of XML should be avoided:

- Special elements for special types of text such as poems or internet forum posts.
- All special XML syntax, i.e. any <!... or <?... tags.
- Anything related to writing, reading or using DTDs.

- XML Schema.
- Namespaces.
- Xpath.
- XSLT stylesheets.
- XML editors and XML indexing software.

6 Conclusion

Sections 2, 3 and 4 of this paper have built an argument for why a basic outline capturing the aspects of XML implementation most widely used in corpus linguistics could be of more use to most creators of corpora than the full weight of the formal standards of TEI and CES; Section 5 has outlined what I believe to be appropriate criteria for such a basic outline. On the basis of those criteria, I have essayed such a document, which is appended after the text of this paper. This document is dubbed *Modest XML for Corpora*; the adjective 'modest' is adopted here to characterise (a) the modesty of the level of corpus markup that would result from following the recommendations; (b) the modesty of scope arising from the avoidance of all advanced aspects of XML, and (c) the modesty of the strength of the recommendations: the presentation is of a set of suggestions on typical practice, not of a standard that is to be adhered to. I have arbitrarily limited the length of the document to 16 pages in manuscript; more could doubtless have been added, but as noted above, there is a virtue in brevity in a document with this purpose. I have also adopted a purposefully informal tone, taking as my inspiration here the TEI *Gentle Introduction to XML*.

The precise presentation of the suggestions could no doubt be improved upon, but it is my hope that in their present form the *Modest XML* recommendations will prove useful as a tool for informing corpus creators without extensive technical training (novice or otherwise) about best practices in deploying XML markup in their data. Some similar formulation could, without too much difficulty, be made part of the normal, general training of corpus linguists. I have employed the guidelines in their present form in my own teaching of corpus linguistics at postgraduate level, with some indications of success, and I would be delighted for other educators to make use of the *Modest XML for Corpora* suggestions in their teaching, either verbatim as presented here, or in any modified form.

Notes

1. In this context it is worth noting that C.M. Sperberg-McQueen was a co-editor of both the early editions of the TEI Guidelines and the early editions of the W3C's XML standard.
2. See http://www.textcreationpartnership.org/tcp-eebo

References

Aston, Guy and Lou Burnard. 1998. *The BNC handbook: Exploring the British National Corpus with SARA*. Edinburgh: Edinburgh University Press.

Bray, Tim, Jean Paoli, C.M. Sperberg-McQueen, Eve Maler and François Yergeau (eds.). 2008. *Extensible Markup Language (XML) 1.0*. Fifth edition. World Wide Web Consortium: available online at http://www.w3.org/TR/REC-xml (last accessed 30th October 2013).

Burnard, Lou. 2005. Metadata for corpus work. In Wynne (2005).

Burnard, Lou and Syd Bauman (eds.). 2013. *TEI P5: Guidelines for electronic text encoding and interchange. Version 2.5.0. Last updated on 26th July 2013*. TEI Consortium: available online at http://www.tei-c.org/Guidelines/P5/ (last accessed 30th October 2013).

Burnard, Lou and C.M. Sperberg-McQueen. 2012. *TEI Lite: Encoding for interchange: An introduction to the TEI. Final revised edition for TEI P5*. Available online at http://www.tei-c.org/Guidelines/Customization/Lite/ (last accessed 30th October 2013).

Ide, Nancy. 1996a. *The ACH-ACL-ALLC Text Encoding Initiative: A brief overview.* Available online at http://www.tei-c.org/Vault/SC/teij17.txt (last accessed 30th October 2013).

Ide, Nancy. 1996b. *Corpus Encoding Standard*. Version 1.5. Expert Advisory Group on Language Engineering Standards (EAGLES): available online at http://www.cs.vassar.edu/CES/ (last accessed 30th October 2013).

McEnery, Tony and Andrew Wilson. 2001. *Corpus linguistics*. Second edition. Edinburgh: Edinburgh University Press.

Sperberg-McQueen, C.M. and Lou Burnard (eds.). 1990. *TEI P1: Guidelines for the encoding and interchange of machine-readable texts*. Chicago, Oxford: Text Encoding Initiative.

Wynne, Martin (ed.). 2005. *Developing linguistic corpora: A guide to good practice*. Oxford: Oxbow Books. Available online at http://www.ahds.ac.uk/creating/guides/linguistic-corpora/ (last accessed 30th October 2013).

Modest XML for Corpora: The suggestions

1 *Introduction*

These recommendations explain a way of adding markup to corpus texts using a system called XML, the *eXtensible Markup Language*. XML is an extremely powerful and very flexible system, widely used for all kinds of documents and databases. Pretty much any kind of information can be added to a document using XML. In corpus linguistics, however, we most often use XML to indicate features of the text other than its actual words. So for most purposes, we only need a *modest* amount of XML. This includes things like:

- Paragraph boundaries
- Sentence boundaries
- Utterance boundaries
- Page breaks
- Things omitted from the original text
- Anonymisation of spoken texts

This document outlines a series of guidelines that you can follow to use XML to indicate these things in a reasonably standard and easy-to-understand way, without getting entangled with all the technical details of XML. If you are creating a corpus, and you want to go beyond just including the words of the texts, then you should consider following the suggestions here.

2 *What XML is*

XML is a system of markup where the information that we add to the text is represented by tags surrounded by **<angle brackets>**. Anything within angle brackets counts as markup; anything outside the angle brackets is part of the actual text. When a whole corpus text is marked up according to the rules of XML, we call it an *XML file*. The XML file format can be read and understood by many different computer programs, as well as being comprehensible to human beings.

A basic example of this would be using the `<s>` tag to indicate sentence boundaries. Imagine you had the following in one of your corpus texts:

```
Johnny visited his grandma. He brought her some
chocolates.
```

Using XML, you can indicate sentence boundaries like this:

```
<s>Johnny visited his grandma.</s> <s>He
brought her some chocolates.</s>
```

If you are familiar with how the World Wide Web works, you might already know about another angle-bracket-based language: HTML, the markup language used to program web pages. The big difference is that in XML, you can decide for yourself what tags you want to use, define what they mean, and decide on the rules for how they are used. So it's not compulsory to use the `<s>` tag for sentences, though it *is* common practice.

3 XML files

An XML file is a type of *plain text* file. A plain text file is a computer file that consists only of letters, numbers and other characters. There is no formatting information, such as text colour or text font, in a plain text file: the kinds of file that do contain this information, such as word processor files or presentation files, are not plain text and cannot be used as plain text.

You can create and edit plain text files using a text editor program such as *Notepad* on Microsoft Windows. It is normally a bad idea to use word processors, like *Microsoft Word* or *LibreOffice Writer*, to edit XML files. There exist lots of specialised XML editor programs as well, but for the modest level of XML that this document is introducing, you don't need to learn how to use one of these: a text editor is enough.

XML files need to follow all the *rules of XML* which we will encounter in this document. An XML file which follows all the rules is called *well-formed*, which means that computer programs can process it correctly. If your XML file is not well-formed, you will encounter an error when a computer program attempts to process it.

One important rule is that XML files should use proper Unicode encoding (for corpus linguistics, we most often use the form of Unicode called UTF-8). Encoding is a separate question to markup, so we won't explain it here; see McEnery and Xiao (2004).

It's traditional to save XML files with the three-letter extension **.xml** (whereas other kinds of plain text file are often saved with the file extension **.txt**).

4 How to enter tags

When you have your corpus text in a plain text editor, you can add tags simply by typing them in. Tags consist of a label between two angle brackets, like this:

```
<p>
```

We usually try to make the label an easy-to-remember abbreviation for what it is marking up. So, for instance, `<p>` tags are normally used to mark up paragraphs, because *p* is an easy-to-remember abbreviation for *paragraph*.

Remember that is important to always type the angle brackets correctly. If you misplace an angle bracket, the XML file will not be well-formed. Another important rule is that the labels of tags are case-sensitive. So `<p>` and `<P>` count as two different kinds of tag in XML. Because of this rule, we normally stick to lowercase letters for our tag labels. Tag labels can't be more than one word, and normally shouldn't contain any characters other than the letters of the alphabet.

One of the other rules of XML is that all 'whitespace' counts as the same. Whitespace includes line breaks, tabs, and spaces: when a computer reads the XML file, these are all treated as being just the same as a space. Another rule is that putting spaces around the tags themselves is normally optional. So, in XML, this...

```
<u>Hello.</u><u>Hi there!</u>
```

... (where *u* is short for *utterance*) counts as exactly the same as this:

```
<u>
      Hello.
</u>
<u>
      Hi there!
</u>
```

The important thing to remember here is that *line breaks don't mean anything in XML*. In normal typing, we are used to being able to use a double line breaks and/or indent to indicate the end point of a paragraph, for instance:

```
This is my first paragraph.
And this is  my second paragraph.
```

But if we are using XML, the line breaks aren't enough to show the paragraphs. We have to explicitly indicate the start and end points with tags:

```
<p>This is my first paragraph.</p>
<p>And this is  my second paragraph.</p>
```

Very often, as in the utterance/paragraph examples above, we use XML tags to represent *regions* in the text. Each paragraph of a text, for instance, is a region that has a beginning point and an end point. When we use XML to mark up regions, the following rules apply:

- the beginning of each region is indicated by a *start-tag* that consists of just the tag label between angle brackets, for instance `<p>`
- the end of each region is indicated by an *end-tag* that has a forward-slash before the tag label, for instance `</p>`
- every start-tag *must* have a corresponding end-tag
- regions can't overlap (that means one region must end before the next region of the same type starts)

However, we don't always want to mark up regions in our texts. Sometimes, we want to mark up something that occurs *at a particular single point* in the text. For example, if we are encoding a written text we might want to mark up its page breaks – but a page break is a point not a region. For these, there is a special style of XML tag:

```
<pb/>
```

(assuming that *pb* is our abbreviation for *page break*). The forward-slash after the tag label indicates that this is a standalone tag marking a point in the text, which has no corresponding end-tag. Here is an example of region-style tags and a point-style tag being used together to mark up a paragraph that starts on one page and finishes on the next:

```
<p>"It's really a dreadful situation," she said
fretfully. "Who on earth <pb/> can predict how
things will turn out?"</p>
```

5 How to add attributes

5.1 What is an attribute?

Quite often, we don't just want to mark the *position* of a tag in the text: we also want to say something about that tag. For instance, we can use the `<pb/>` tag to mark the position of a page break as illustrated above. But we might also want to specify the *number* of the page that comes after the break. For this, we use a system of markup called *attribute-value* pairs. An *attribute* is a label that specifies what type of information is being recorded; it is then followed by the *value*, which contains the actual information. Attribute-value pairs are encoded inside the actual XML tags, as follows:

```
<pb n="23"/>
```

The rules for attribute-value pairs in XML are as follows:

- The attribute-value pair is placed inside the tag, after the tag label, with a space in between
- The label of the attribute comes first; the same rules apply to attribute labels that apply to tag labels – they should just be a single word, should normally contain only letters, and are case-sensitive (in the example above, the attribute label is `n`)
- The attribute label is followed by an equals sign; after the equals sign comes the value of the attribute.
- The value of the attribute is surrounded by quotation marks.
- Other attribute-value pairs can then be added inside the same tag.

The idea is that the attributes are the same on every tag of the same type in the text, but the values can be different each time. For instance:

```
...
<pb n="23"/>
...
<pb n="24"/>
...
<pb n="25"/>
...
```

It's possible to use either single quotation marks or double quotation marks to mark the start and end of a value, but you shouldn't mix the two types of quotation mark – each attribute-value pair must use one or the other. It's recommended to use double quotation marks most of the time.

Something you should remember: if you type a quotation mark in a word processor, it will normally be turned automatically into a 'curly quote' character – that is, either a 66-style quotation mark (begin quotation) or a 99-style quotation mark (end quotation).

```
 “These are curly quotes (sometimes called
‘smart quotes’).”

"These are straight quotes, the 'correct' ones
for attribute-values."
```

'Curly quote' marks are fine for normal typing, but incorrect for XML: you must never use them for attribute-values as this stops the XML from being well-formed. This is one good reason why it is better to create your XML in a text editor than in a word processor!

When a region-style tag has an attribute, the attribute is marked up once only, on the start tag – even though it applies to the whole region. In the following example, the `who` attribute is added to the `<u>` tag to indicate 'who' is speaking – obviously, this bit of information applies to the *whole* of the utterance surrounded by the `<u> ... </u>`:

```
<u who="PETER">Hello, how are you?</u>
```

There are some other common mistakes that people make when adding attributes. For example:

```
<pb="12"/>
```

The mistake here is that the attribute-value is linked directly to the tag. This isn't allowed in XML. Values have to be linked to attribute names in the way we explained above.

A second common mistake is repeating the attribute-values on the close tag. For example:

```
<u who="PETER">Hello, how are you?
</u who="PETER">
```

You should never use encoding like this. If you do, your file is not a valid XML file.

Finally, you will sometimes see the following in older corpus files from before XML was introduced:

```
<u PETER>Hello, how are you?</u>
```

Here, the value (name of the speaker) is added directly within the tag, without linking it to an attribute. Again, this does not follow the modern rules of XML, so you should never use markup like this.

5.2 How to use identifiers and sequence numbers

There are two particular attributes that you can use on lots of different tags: `id` and `n`.

We saw an example of `n` above being used with the `<pb/>` tag. *n* is short for 'number' and is customarily used to indicate a sequence number. It can be used with any tag. For example, if a spoken text has 100 utterances in it, then the `<u>` tags can contain the attribute-values `n="1"` to `n="100"`. This can be useful for keeping track of where examples in a corpus occur; for instance, you could note that a particular example of a word you are interested in occurs 'in utterance number 48 of text AA2' and this would enable you to find the source of the example whenever you need it.

The `id` attribute is also used to keep track of particular tags. The difference is that, whereas `n` is a sequence number, the `id` attribute doesn't necessarily tell us anything about sequence. Instead, it contains an *identifier* – an arbitrary label that is a 'name' for that particular tag.

There are two generally useful rules for identifiers:

- Identifiers have to be unique – no two tags *in the entire corpus* should ever have the same identifier, even when they are different kinds of tag; this is different from sequence numbers, which *can* be repeated in different texts or for different types of tag
- Identifiers should usually only consist of letters and numbers – if they contain other characters, this can sometimes create problems when the corpus is processed by some computer programs

One typical use of identifiers is to give a unique label to each text in a corpus.

```
<text id="AA01">
```

```
    [The contents of this particular text goes
     hewre.]
</text>
```

An identifier is very useful here because we want each text to be clearly labelled. Typically, you would store each corpus text in an XML file with the identifier as its filename. So the text with identifier `AA01` would be stored in a file with the name **AA01.xml**.

5.3 Recommended and unrecommended ways to use attributes (advanced)

It's possible to use attributes in a way that is perfectly well-formed XML, but that actually makes things difficult later on.

Wherever possible, you should use attributes on a tag for information *specific to that instance of the tag*. Try to avoid repeating information unnecessarily.

Utterance tags are a good example of this. Often, we have lots of information about the speaker – sex, age, social class and so on. It's very tempting to add this information to every `<u>` tag, and this is indeed well-formed as XML:

```
<u who="Mary"   sex="F" age="18">Hello, how are
you?</u>
<u who="Timmy"  sex="M" age="42">Quite well,
thank you.</u>
<u who="Mary"   sex="F" age="18">I'm very glad
to hear it!</u>
```

However, notice that this style results in *lots* of repetition. Every time Mary speaks, we repeat the information that she is a female aged 18. This is not ideal (imagine, for instance, finding out we'd made a mistake and Mary was actually 17 when the text was recorded – we would have to go back and change the markup of every single utterance).

The issue here is that things like sex, age and class are information about the *speaker*, not information about the *utterance*. So, a better way to arrange things is to use a reference to an identifier as the value of `who`, and to store all the speaker information somewhere else. For example, `SP001` could be used as the value of `who` for Mary and `SP002` for Timmy:

```
<u who="SP001">Hello, how are you?</u>
<u who="SP002">Quite well, thank you.</u>
<u who="SP001">I'm very glad to hear it!</u>
```

The identifiers then provide an unambiguous link to the speaker information. This speaker information could be stored, for instance, in the header of the spoken text, in this form:

```
<speaker id="SP001" name="Mary Bloggs" age="18"
sex="F" birthplace="Newcastle-Upon-Tyne"
occupation="chef" />
```

Alternatively, a separate database or spreadsheet can be used to store this information in the form of a table. Either way, pieces of information such as the speaker's sex and age only need to be logged once, rather than on every single utterance.

The `who` attribute is not itself an identifier (the values are not unique – different utterances can have the same value of `who`). But its value *refers to* an identifier, which *is* unique across speakers; so we know unambiguously who is speaking here.

Another tempting mistake is to use more than one attribute to mark up the same kind of information. For example, you might want to mark up pragmatic functions on utterances, in a style like this:

```
<u who="SP001" func="greeting">Hello, how are
you?</u>
```

This is fine so far. However, many utterances have more than one function (the example above is a *question* as well as a *greeting*). How can you encode more than one function? You might be tempted to have multiple `func` attributes, like this:

```
<u who="SP001" func="greeting"
func="question">Hello, how are you?</u>
```

... but this is not allowed (each attribute can only occur once). Another tempting approach is to classify it in terms of 'first function, second function...', and mark it up like this:

```
<u who="SP001" func1="greeting"
func2="question">Hello, how are you?</u>
```

This is well-formed XML. However, it is a bad idea for later processing, because it means the information about pragmatic function is spread across two different attributes – a computer analysing the file won't know automatically that `func1` and `func2` are related. It thus becomes very difficult, for instance, to search for something like ‘all the questions’ because the `"question"` value could be in either `func1` or `func2`. So what might be a better approach?

One way is to combine the values together on a single attribute, like this:

```
<u who="SP001" func="greeting;question">Hello,
how are you?</u>
```

Another way might be to mark the functions up using separate tags:

```
<u who="SP001">
   <func name="greeting"/>
   <func name="question"/>
   Hello, how are you?
</u>
```

Because the `<func>` tags are *inside* the `<u>` tag, they count as *belonging* to it, according to the rules of XML.

6 *How to nest XML tags*

You will nearly always want to use more than one kind of XML tag in a text. That means you have to think about how they fit together. There is one main rule for combining XML tags: pairs of tags can never overlap.

For example, let’s imagine we have both paragraph and sentence tags using `<p>` and `<s>`. After a paragraph tag has been started, we can then start a sentence tag. The rule is that this sentence tag must be closed before the paragraph tag can close. In other words, this is correct:

```
<p><s>This is the story of a man called Bill.
</s></p>
```

... but this is wrong:

```
<p><s>This is the story of a man called Bill.
</p></s>
```

In the latter example, which is not well-formed, the region of the `<p>` and the region of the `<s>` overlap. In the former example, which is well-formed, the region of the `<s>` is completely contained within the region of the `<p>`. This *complete containment* is sometimes called *nesting*.

Note that nesting is not an issue for tags like `<pb/>` which indicate a point rather than a region – they can't ever overlap, precisely because they indicate points in the text.

We sometimes use indenting to illustrate the nesting, as follows (remember that whitespace is irrelevant to the computer, so this is just a visual guide for human readers):

```
<p>
     <s>
          This is the story of a man called
          Bill.
     </s>
</p>
```

In this layout, each extra level of nesting adds one more indentation, and it is easy to see how the tags must correspond. Nesting can go on for as many levels as needed, for example:

```
<u who="SP001">
     <s>
          I er s- started walking and er well
          left the house.
     </s>
     <s>
          Then when I saw th- the
          <unclear>
               circus
          </unclear>
          it was in the er playground there.
     </s>
</u>
```

An XML file is only well-formed when all the tags are correctly nested like this (this is sometimes called *perfect nesting*).

There is one additional rule for XML to be well-formed, which is that the entire XML file must be nested within a single pair of start- and end-tags. That is, the very first thing in the file must be a start tag, and the very last thing in the file must be the corresponding end tag. In corpus linguistics, we very often use a `<text>` tag for this purpose, since most of the time one file corresponds to one text.

7 *How to encode special symbols*

As we've seen, XML is based on two special symbols that mark tags off from the actual text: < and >. If these symbols occur in your actual corpus text, we need to encode them in a special way to make it clear that these are part of the text, not part of the markup. The way XML does this is to represent these special symbols using a short code that starts with an ampersand and ends with a semi-colon. Because ampersand is used for this special purpose, we *also* need a code to represent ampersand itself. The codes are:

- Less-than sign (open angle-bracket): < should be represented as `&lt;`
- Greater-than sign (close angle-bracket): > should be represented as `&gt;`
- Ampersand: `&` should be represented as `&amp;`

The technical term for symbols represented using these codes with an ampersand is *entities*.

< is the symbol that commonly causes problems – any corpus that contains scientific writing is likely to have statements of statistical significance like 'p < 0.05' – just one example of this in your corpus text that has not been converted into an entity is enough to stop the file being well-formed.

For example, imagine that your original text has the sentence 'The results were tested for significance using the t-test, & were found to be significant (p < 0.01).' You would need to encode this in XML as:

```
<s>The results were tested for significance
using the t-test, &amp; were found to be
significant (p &lt; 0.01).</s>
```

An easy way to do this is to use your text editor's find-and-replace tool to *replace-all* instances of `&` with `&amp;` , all examples of < with `&lt;` , and all examples of > with `&gt;` . Two warnings! First, you must do these replacements *before you add any XML tags* or the angle brackets around the tags will also be turned into entities. Second, *you must do the ampersand replacement first*, because if you do it after adding `&gt;` or `&lt;` a *replace-all* of `&` will also affect the `&`-characters at the start of those entities.

There are also special codes for the single and double quotation marks. You don't need to use these in normal corpus text, but you do need to use them inside attribute-values – because otherwise, the computer could mistake them for the quotation marks that end the value. The codes for these symbols are:

- Double quotation mark: `"` should be represented as `&quot;`
- Single quotation mark (apostrophe): `'` should be represented as `&apos;`

For example, the following is not well-formed:

```
<text id="NEWS01" title="Prime Minister says,
"No way"">
     In a statement from Downing Street, the
     Prime Minister said that [...]
</text>
```

The `title` attribute of the opening `<text>` tag should be encoded like this:

```
<text id="NEWS01" title="Prime Minister says,
&quot;No way&quot;">
```

8 *How to check that an XML file is well-formed*

Most web browsers (such as Internet Explorer, Firefox, and Chrome) are designed to read XML files. So an easy way to check if your XML file is well-formed is to open it in a web browser. If the browser opens the whole file correctly, the file is well-formed and contains no errors. If the browser reports an error before it gets to the end of the file, there is a mistake in your XML – maybe an end-tag is missing, or perhaps you have mistyped an angle bracket, or left a less-than symbol in the text without changing it into `&lt;` .

Another useful feature of web browsers is that they will usually lay out the XML tags in indented style (see above) automatically for you, regardless of the layout in the actual plain text file.

9 *The de facto standard tags and their attributes*

Certain XML tags are used so widely in corpus linguistics that it is almost always the best idea to use those same tags, rather than invent your own, if you want to encode the things that those tags represent. For example, it's perfectly possible, if you choose, to use the tag `<para> ... </para>` to indicate the start and end of a paragraph. However, the tags `<p> ... </p>` are so widely used for this purpose, that it makes more sense to stick to this practice wherever possible.

This section lists a number of these *de facto* standard tags, together with the attributes usually used, examples, and some comments.

Normally, you would only use a selection of these tags, choosing the types of markup relative to your purposes. If you used *all* the tags listed here, you would be using a very comprehensive style of markup, well on the way to one of the major, heavyweight standards.

The `id` and `n` attributes which we discussed above aren't mentioned again here, because they are used in the same way for every kind of tag.

9.1 Tags normally used in written texts

- <p>

`<p> ... </p>` is used to mark the start and end points of a paragraph.

- **<s>**

`<s> ... </s>` is very frequently used to indicate sentence start and end points. We've already seen examples of this tag in use. Some part-of-speech taggers automatically insert these tags for you.

- **<head>**

Sometimes, we want to distinguish between normal text paragraphs and paragraphs that are graphically distinct in the original text, for example section headings or the headline of a newspaper article. For instance, a newspaper text could be marked up as follows:

```
<head><s>MAN BITES DOG</s></head>

<p><s>A man bit a dog yesterday.</s> <s>When
interviewed, the man said, "No comment."</s>
</p>
```

If you don't need to draw the distinction between normal paragraphs and headings, it's perfectly fine to just use the normal `<p>` tags for headings/headlines.

`<head>` can have a `level` attribute, where the level of the heading is used to indicate the difference between main headings (level 1), sub-headings (level 2), sub-subheadings, and so on:

```
<head level="1">VOLUME ONE</head>
<head level="2">Chapter One</head>
<head level="3">- 1 -</head>
<p>I wish to tell the story of my life. I was
born ...</p>
```

- **<pb/>**

This tag is used to indicate the position of a page break in a written text (either between two paragraphs or mid-way through a paragraph).

- **<q>**

In some corpora, `<q> ... </q>` tags are used to indicate *quoted* text, such as direct reported speech. That is, instead of this:

```
<s>Mr Johnson said, "Hello, everyone!" as he
walked into the room.</s>
```

... we can have this:

```
<s>Mr Johnson said, <q>Hello, everyone!</q> as
he walked into the room.</s>
```

It is usually a lot of work to correctly insert `<q>` tags, so you would only do it if distinguishing automatically between normal text and reported speech is going to be important for your use of the data.

- **<gap/>**

The `<gap/>` tag is used to make a note of something that has been omitted when the text was transcribed into a corpus file – most commonly figures and illustrations, but also sometimes tables and other irrelevant material. It usually has a `desc` attribute containing a short description of what has been left out. For example:

```
<gap desc="Figure 1.1, apparatus diagram" />
<gap desc="Illustration - picture of a clown" />
<gap desc="Table (half a page) listing mortality
figures in 19th century Wales" />
```

- **<reg>**

The <reg> tag is used to indicate regularised spelling. This is often useful with non-standard types of data – for example, historical corpora predating the full standardisation of spelling, or written texts produced by young children or foreign language learners. Regularisation allows the text to be searched using modern spellings. Normally, a <reg> tag would only surround a single word at a time (there are some exceptions to this). It is customary to use an orig attribute to preserve the original spelling. For example, if you had a historical text with the sentence *Three shippes sayled to Hamborough*, you could mark it up like this:

```
<s>Three <reg orig="shippes">ships</reg>
<reg orig="sayled">sailed</reg> to
<reg orig="Hamborough">Hamburg</reg>.</s>
```

9.2 Tags normally used in spoken texts (or playscripts)

- **<u>**

The <u> tag is used for utterance boundaries. We have seen examples of this in use before. The most common attribute is who, which is used to indicate the speaker of the utterance. This has also been discussed above in the general outline of XML.

- **<pause/>**

The <pause/> tag indicates the point in an utterance where a pause occurs. It often has the dur attribute to show how long it is (most often measured in seconds). For example:

```
<u>I wanted to <pause dur="2"/> er wanted to go
home</u>
```

The pause tag here is equivalent to the notation like **(2.0)** sometimes used for transcribing pauses in traditional transcription of spoken data.

- **<voc/>**

The <voc/> tag indicates the occurrence of a non-linguistic vocalisation in a spoken text – such as a cough, a sigh, a laugh, or a sneeze. The type of vocalisation is usually specified using a desc (description) attribute, for instance:

```
<u>so <voc desc="cough"/> three llamas right
<voc desc="laugh"/> walk into a bar <voc
desc="laugh"/> and one says </u>
```

- **<event/>**

An `<event/>` tag is just like a `<voc/>`, except it indicates some kind of sound or break on a recording that is not part of an utterance. Here are some examples:

```
<event desc="prerecorded radio jingle" />
<event desc="noise of passing traffic" />
<event desc="end of side one of tape" />
```

- **<stage>**

In scripts (plays and other written-to-be-spoken texts), the `<stage> ... </stage>` tag can be used to mark off stage directions. This can be useful for later analysis if you want to distinguish words that are actually spoken by characters in a play from the other words in the original text. Here is an example of this style of markup applied to Shakespeare's *The Winter's Tale*:

```
<u who="ANTIGONUS">[...] I am gone for ever.</u>
<stage>
      [Exit, pursued by a bear.]
      [Enter an old SHEPHERD.]
</stage>
<u who="SHEPHERD">I would there were no age
between sixteen and three-and-twenty [...]</u>
```

9.3 Tags that could be used in either kind of text

- **<text>**

It is very common to use `<text> ... </text>` as the single tag that encloses an entire corpus text. (Remember that the rules of XML require every file to be contained within a single start-tag and end-tag pair.) If you do this, the `<text>` tag should usually contain the corpus text's unique identifier code using an `id` attribute.

- **<unclear>**

The `<unclear>` tag is very useful for both speech and writing. In a spoken text, you can use it to mark a region (a word or words) whose transcription you are not certain about – perhaps because it was spoken unclearly, or perhaps because the recording quality was poor:

```
<u>So I went to um to my <unclear>village
</unclear> cos I wanted to um you know</u>
```

In a written text, it is used to mark a word or words which can't be made out in the original text – perhaps it is blurry in a printed text, or written in illegible handwriting:

```
<p>Jenny - what do you think of this? Seems
<unclear>fishy</unclear> to me, best, TH</p>
```

There are two common ways to use `<unclear>`. In the examples above, you makes a 'best guess' about what the words are, and surround the uncertain word or words with `<unclear>` tags. If you have no idea at all, you would use an `<unclear/>` tag to indicate the unclear words as a point rather than region, optionally with a `dur` attribute to state how long the unclear chunk is, for example:

```
<unclear dur="3 seconds"/>
<unclear dur="10 words"/>
<unclear dur="whole line">
```

- **<header> and <body>**

Corpus files often have headers – blobs of data *about* the text (such as: title, publisher, date of recording, etc.) encoded at the start of each file. If you are adding headers to your files, you can use `<header>` and `<body>` tags to distinguish the header information from the actual text. The usual way to do this is to have the overall layout of the corpus file as follows.

```
<text id="TEXT01">
   <header>
        [The header information goes here.]
   </header>
   <body>
```

```
              [The actual content of the corpus text
              goes here.]
         </body>
      </text>
```

Note that a `<header>` is not the same thing as a `<head>`! See above for `<head>`.

An approach to creating a modest XML header is outlined near to the end of this guide.

- **<w>**

The `<w>` tag is used around individual word tokens. Its main use is to carry attributes that indicate different types of corpus annotation. Common attributes of this sort include `pos` for a part-of-speech tag and `lemma` (or sometimes `hw`, short for 'headword') for a lemma. For example:

```
<w pos="ART"  lemma="the">The</w>
<w pos="NOUN" lemma="cat">cat</w>
<w pos="VERB" lemma="sit">sat</w>
<w pos="PREP" lemma="on" >on</w>
<w pos="ART"  lemma="the">the</w>
<w pos="NOUN" lemma="mat">mat</w>
<w pos="PUNC" lemma="."  >.</w>
```

Normally you would not add this kind of annotation manually, but would use a computer program to do it for you (e.g. a part-of-speech tagger). If there is no word-level annotation, there is no need for `<w>` tags.

- **<c>**

This tag is sometimes used (for example, in the British National Corpus) as an equivalent for `<w>` that is placed around tokens that are punctuation marks. This allows punctuation tokens (wrapped in `<c> ... </c>`) to be distinguished from word tokens (wrapped in `<w> ... </w>`). So for instance, in the example above, the last token (full stop) could have been marked up with `<c>` instead of `<w>`. As with `<w>`, however, you probably wouldn't use `<c>` unless you needed to encode word-level annotation.

- **<anon/>**

An `<anon/>` tag can be used as a replacement for any word or phrase that needs to be omitted for reasons of anonymisation. You can use a `type` attribute to indicate why the information has been left out, as in this example:

```
<p>I want you to assassinate Mr <anon
type="name"/> for me. He lives at <anon
type="address"/>. Please hurry, because I
really, really hate him.</p>
```

10 Making up your own tags and attributes: Two examples

The fact that there is a set of *de facto* standard tags does not mean that you cannot also use additional tags when you need to. Here are two example cases.

10.1 Example 1: Recording highlighting in the original text

Often, when we build a corpus, we are not particularly interested in preserving text highlighting like **bold** and *italics*. However, sometimes this is linguistically significant (especially in historical text). If so, you might add in tags to indicate regions highlighted in one or both of these ways.

There are lots of ways of doing this. For example, you might use `<b>...</b>` and `<i>...</i>` for bold and italics respectively – these are borrowed from the HTML language used to encode web pages. Or you might borrow the TEI standard's technique, which uses a `<hi>` tag for this purpose, with a `rend` attribute to specify whether it is bold or italic.

So, given this text:

Do you *seriously think* he'll **forgive** you after everything you ***did***?

... we could mark it up in either HTML-style or TEI-style:

```
Do you <i>seriously think</i> he will
<b>forgive<b> you after everything you
<b><i>did</i></b>?

Do you <hi rend="italic">seriously think</hi>
he will <hi rend="bold">forgive</hi> you after
everything you <hi rend="bold italic">did</hi>?
```

10.2 Example 2: Constituent parsing

Constituent parsing is a common form of grammatical analysis that is basically equivalent to drawing a tree diagram for a sentence. Of course, we can't literally draw a tree in a plain text file, but we can use XML tags to indicate the start and end points of things like NPs, PPs, and VPs, as in the following example:

```
<s><np>The cat</np> <vp>sat <pp>on <np>the
mat</np></pp></vp>.</s>
```

The *perfect nesting* rule of XML (see above) is equivalent to two of the basic rules for drawing tree diagrams: (1) branches can't cross; (2) branches can split, but they cannot merge.

In this case, you would create whatever tags and attributes your scheme of grammatical analysis requires. For example, you might have `<np type="definite"> ... </np>` contrasting with `<np type="indefinite"> ... </np>`. It is good idea to choose labels in such a way as to make it *as obvious as possible* what each tag represents.

11 Modest headers

If you have a file header, it is useful to enclose it within a `<header> ... </header>` pair of tags, to distinguish it clearly from the actual data – see the notes on `<header>` and `<body>` above.

The kinds of information that can be included in a corpus text header vary too much to provide detailed recommendations. However, one point to bear in mind is that many XML-aware concordancers are not sophisticated enough to know that text in a header is not part of the actual text, since their only principle is to ignore everything between angled brackets. For this reason, it might be better to enclose textual information in attributes.

For example, the following style (recommended in many of the heavy-duty standards like TEI) may well result in the inner text getting treated by a concordancer as part of the text for corpus searches, frequency counts, and so on:

```
<header>
   <title>Extract from "The Big Book of Facts"
   </title>
</header>
```

By contrast, the following style does not have any text outside the XML tags and so will not be mishandled by a concordancer that works by ignoring anything between angle brackets:

```
<header>
   <title value="Extract from &quot;The Big Book
   of Facts&quot;"/>
</header>
```

12 Summary of the rules of XML

- Tags are marked off from the actual text using **<angle brackets>**
- Regions in a file should be marked as `<tag> ... </tag>`
- Points in a file should be marked up as `<tag/>`
- The tags must be perfectly nested: regions mustn’t overlap
- There must be no unclosed tags
- There must be a single tag that encloses the contents of the entire file
- Whitespace doesn’t matter
- Attribute-value pairs must be placed inside tags (but not closing tags)
- Attribute-values are always paired: each attribute has one single value
- Values must always be quoted
- Special symbols (greater-than, less-than, ampersand) are represented as entities.