Adopting XML — A Business Perspective

Geoff Nolan

Senior Systems Engineer

Turn-Key Systems
Sydney NSW, Australia
gjn@turnkey.com.au
www.turnkey.com.au

Turn-Key Systems is one of Australia's oldest software companies, established in 1971. It has long specialized in professional publishing software and content based markup. In 1997 the company decided to convert its operations to XML. The company has particular expertise in data conversion and DTD design, it is the Australian distributor for XMetaL, and has developed TopLeaf (an XML looseleaf rendering system) and X-Ice (an open source MS Word to XML converter).

Geoff Nolan is an honours graduate in computer science who joined Turn-Key in 1984. He has worked on the automated rendering of Yellow Pages directories, integrated legal publishing systems, and point-in-time legislation delivery. He became an early XML convert in 1997, and specializes in markup design, data conversion, and XML storage and maintenance strategies. He has co-authored several patents for XML enabling technologies and presented XML technical papers in Australia, the US and Europe.

Introduction

Making the decision to adopt XML, or having it thrust upon you, is only the first step in a major transformation in how your organization relates to its data. You should be aware at the start that there will be costs and difficulties associated with the switchover. However, if you are properly prepared, the costs can be minimized and the benefits substantial.

This tutorial addresses the issues you are likely to confront in converting to, and working with, XML data. We will begin with the conversion process itself, the choice of which data lends itself to conversion, and the equally important selection and preparation of staff to facilitate the change.

Next we will cover the tools of the trade — software which will help you convert and maintain your data. Finally we discuss the uses to which your new data will be put, and how to structure your data to support its existing and potential functions.

Making the Switch

For some organizations the decision to adopt XML is a virtual no-brainer. For others the decision will come only after the most careful evaluation of potential benefits and pitfalls. Only one thing is certain: the switchover from traditional data maintenance to structured documents will be the most painful part of the exercise. However a properly planned and executed implementation will involve considerably less pain and less cost, while helping to ensure that desired outcomes will be fully realized in the shortest possible time.

Migrating New Material

While some XML enthusiasts will say "Convert Everything!", this is manifestly bad advice for most organizations. Our business, for example, still uses Microsoft Office products very frequently. In fact we have even reverted to using proprietary formats for some files which were previously in XML!

The selection of which data to migrate to an XML production mode and which to leave alone is often the difference between a successful and profitable implementation, and a costly failure.

Put simply, you should consider migrating documents which:

- have more than one potential end-use
- can be used in the creation of other documents
- are subject to search and retrieval beyond simple word searches
- have a potentially long life-span

Let's consider the above items one by one. Because XML tends to emphasize content over format, it lends itself very well to documents which may need to appear in different forms. This tutorial is itself coded in XML for that very reason. There are at least three forms in which this document will appear:

- A US Letter sized PDF.
- An A4 sized PDF for distribution outside the USA.
- A set of HTML pages for the Turn-Key website.

Now you might say that Microsoft Word (WordPerfect etc.) is perfectly capable of producing output in these three formats. This is certainly true for A4 and Letter documents, and partially true for HTML. And indeed many businesses use Word in exactly this fashion.

However, the XML solution is both easier to adapt to a production environment and slightly more efficient. When a large number of documents are involved, this represents a substantial saving in processing costs. In addition, the efficiency and effectiveness of HTML translation is far higher from XML than from any proprietary format (see Tools of the Trade below).

The advantages of XML are much more obvious when it comes to re-using all or part of a document. Such re-use may range from automatically generating a subject index for your data, through to lifting whole sections and pasting them into new documents. Experience has shown us time and again that, irrespective of special styles and templates that may be used, it is virtually impossible to restrict the "creativity" of authors using Word. And even if you succeed, all you have done is produce an inefficient and non-standard XML equivalent!

Of course once you have an XML document, you immediately have access to a wide and growing range of applications which can access and process your data. Already there are document management systems which support sophisticated search and retrieval functions, and converters such as XSLT which can reshape your data to meet a range of requirements.

And finally a much overlooked point. XML is a format which not only allows transmission of data between computer systems, but also transmission of data through time. For example, an SGML document from the mid 1980s would still be relatively easy to process today. The same could not be said for a WordStar document of the same era. Proprietary formats come and go, and even Word document formats have changed substantially over the years. But the information content of a XML document using basic tagging and a standard ASCII or Unicode encoding will retain its value, even if the associated processing applications are no longer in use.

However, even with its advantages, XML should not be the automatic choice for every type of document. Proprietary systems generally have advantages in three areas:

- cost of preparation
- flexibility
- familiarity

There is often a cost overhead in the preparation of a structured document, particularly if that document is small and free format. Examples are memos, flyers, brochures and other throw away items. If style is important but the content will never be used again, it is almost always more efficient to use whatever application is most expedient.

Even some larger documents may be best left with current production methods. XML is most effective for classes of documents with a well defined common structure. Some quite substantial documents bear little structural similarity to other documents within the organization, or contain numerous internal inconsistencies. In such cases the cost of analyzing and formally specifying the structure (e.g. in a DTD) may well exceed the benefits gained, especially if the authors/editors find the structured environment too restricting.

Finally, your staff may be perfectly happy with existing practices and software, and strenuously resist any attempt to force them into a new mold. The cost of such resistance can be substantial, and may well outweigh any potential benefits from the change. In this case the organization needs to carefully assess whether a migration to XML will actually be of any long term benefit.

Note that this section refers only to the migration of document production to XML. We now turn to the more difficult question of what, if anything, to do about existing documents.

Archived Data

When thinking about archived data, the problem is less straightforward. Unless the data is very tightly structured, there will be a substantial conversion cost. So the fact that new material for a particular document class is in XML is no guarantee that upgrading the legacy material will be a viable proposition. In general you should aim to convert as little existing data as possible, consistent with your business and production requirements.

You should of course convert any data which forms an integrated whole with on-going work. For example, if you are publishing new volumes or supplements to an existing work then the total production cost of the work will generally be lower if it is all converted to a common format. The same applies to material which is likely to be revised or otherwise accessed for some time to come. Remember that conversion to XML is the cheapest and most effective means of storing data over long periods of time.

Teamwork

The decision to adopt XML must be actively supported by all affected personnel, and those personnel must themselves be supported by the business as a whole. All too often we see management take the decision to adopt, and then leave implementation to individual departments. This invariably leads to a patchy and inefficient setup, especially if no staff or budgeting allowances are provided for, and existing cost and production deadlines are expected to be met.

Management must always be aware that adopting XML is a long term corporate strategy, and one which will require a substantial investment of time, money and resources. Any plan requiring cost recovery within 12 months is doomed at the outset, as the initial cost of a good implementation is almost all up front. The benefits, though substantial, will not become apparent until the conversions are complete and the new practices and procedures in place. Unlike, say, investment in new computer systems which depreciate to almost nothing over a few years, the benefits of XML adoption steadily increase as time goes by.

Most successful implementations have a *champion*. This is a senior manager or technical officer who has a commitment to oversee and coordinate the adoption process. A successful champion must be able to:

- prepare a plan and a schedule for XML adoption, including a clear set of goals and sign-off conditions
- ensure that individual departments play their part in the implementation
- extract from senior management relaxation of budgetary and time constraints which are incompatible with the change
- organize additional personnel, training, documentation and other facilities required by affected departments
- promote an atmosphere of acceptance within the organization, and address pockets of doubt or apprehension

It must be stressed that XML adoption is a different order of change from, say, upgrading to a new version of Windows. It is likely that all levels of staff will be affected to some degree in terms of workflow and procedures. Anyone interacting with the data will have to change not only their software (e.g. XMetaL instead of MS Word) but also the way they think about their documents.

Working with XML is no harder than creating documents using traditional tools. After all, the aim of the exercise is generally to increase the efficiency of production of the documents in question. The major difference, and one which causes far more trouble than simply learning a new authoring tool, is the perceived lack of flexibility (or even creativity) in the production of structured documents. MS Word, like most traditional document authoring tools, allows almost unlimited flexibility in the styles and formats that can be applied. Even the strictest style guides and templates still leave room for the canny operator to "work around" problems which may crop up from time to time. But it is this very flexibility which compromises the true informational content of the data, and which makes automatic Word to XML conversion such a high risk and/or uncertain affair.

XML tools on the other hand enforce absolute adherence to the prescribed document structure. This is done not by ringing bells or posting nasty error messages, but rather by simply not providing access to any action which would break the rules. Thus, if an element cannot be inserted at a particular point in a document, then the drop down insertion menu simply does not contain that element. If text is not allowed, then pressing the alphabetical keys has no effect. And so it goes — the available tools are constantly modified so that only actions valid at the current location are possible.

Authors and editors new to this environment find it cumbersome and restrictive. It is only later that they realise that they no longer have to reformat (say) an extract from a monthly discussion paper in order to include it in a managerial summary. As the impact of these benefits starts to sink in (and provided that there *are* benefits) we often see a major turnaround in attitude. Not having to worry about format comes to be seen as a liberating experience which entirely eliminates many useless and tedious tasks. But both the staff and the organization will benefit if the advantages are explained in advance, and the staff properly trained both in the new procedures and in the problems they are likely to encounter during the adoption process.

Workflow

Before you convert your data you need to consider how it will be managed under an XML regime. The purists say that all document authoring and editing must take place in an XML environment, and in an ideal world we would agree. However, there are many circumstances in which a different workflow model must be used. The most common is where an organization has little or no control over the raw data it must use.

Author, Author!

Whether to author documents in XML or convert them from some other format is a question that almost every XML adopter must face sooner or later. When the documents arise from an external source (such as an independent author or government department) the question answers itself. And if a company already has a smoothly running document production system, it may wish to leave well enough alone, and restrict XML processing to downstream editors.

Nevertheless there are substantial benefits to a full XML environment. Perhaps the greatest is doing away with the need for document conversion.

The phrase "We can automatically convert *our* Word documents to XML" is often heard, but we have yet to see a single instance of it working in practice. By applying stricter and stricter style rules, or tighter and tighter templates, it is possible to approach the goal of fully automatic conversion. But by the time you get close enough to make this a working proposition, you have reduced Word to an unwieldy and feature-poor replica of a proper XML authoring tool you could have used from the start!

Authoring in XML is not just a matter of using the right software, it's also a state of mind. A traditional mindset regards the strictures of using XML as an unnecessary complication and an obstacle to getting the job done. An XML author regards those same strictures as an essential part of getting the job done *right*. A document which proofs beautifully, but which causes major headaches when reprocessed, is of little net benefit to an organization. In fact the effort of cleaning up the mess, and then having to reconcile the changes with versions already released, usually far exceeds what would have been required to do the job properly in the first place.

So, while there are many valid reasons not to author in XML, remember that there *will* be a cost involved.

Conversion Strategies

The simple rule here is: If you have to convert at all, then only convert once.

Systems which purport to convert data back and forth have a superficial appeal — "Have all the benefits of XML in-house while still allowing your authors and editors to use their preferred document processors!". The fact remains however that the informational content of XML is generally higher than a version of the same document in any other format. Thus each conversion from XML to (for example) DOC involves a net loss of information. It is possible to preserve some of this information in the form of styles, comments, and similar workarounds. But the non-XML application cannot in general ensure that this information is preserved during an editing session.

So as a general rule, once your data has been converted to XML it should stay in that form.

But this naturally raises the question of what to do when your authors/editors have to correct or expand the original document. Though it seems counter-intuitive, our experience has been that the most successful approach is to send a non-XML author a printed copy of the document so that corrections can be marked by hand. If substantial new material is required, then the author should create mini-documents to be inserted into the main work. In either case, the corrections to the XML master are made in-house by XML editors.

While this process seems inefficient, the alternative is generally to convert the XML back to its original format, then reconvert the amended document back to XML. This automatically implies that the working master of a document is sometimes in the repository, and sometimes out with an author in an uncontrolled format. This in turn can lead to subtle errors which manifest in spectacular (and costly!) fashion further down the track.

How do we actually achieve the conversion? You can write your own script, use a pre-existing tool (of which there are many), or use the other application's "Save as XML" option (and then try to make some sense of the result). If you are converting MS Word files, then Turn-Key provides an excellent solution which is outlined in the following section.

Always remember though, the best conversion is no conversion at all. So if it's at all feasible, you can save yourself a great deal of time and cost by training your authors to use XML from the start!

X-Ice

The X-Ice (XML Interactive Conversion Environment) program is available from the Turn-Key website, and it's free! The one catch is that (for the current version at least) you need a running XMetaL to build your output file — an evaluation version will do. X-Ice provides a bridge between the MS Word API and the XMetal API, extracting a paragraph at a time and sending XMetaL the data it needs to build the XML equivalent.

You can specify your own DTD, and your own conversion rules. The conversion is interactive and pauses when the system can't find a rule to apply, or the result would produce invalid XML. You then have four options:

- modify the Word document to match the rules;
- modify the XML document to ensure a valid result;
- modify the rules to handle the new situation;
- tell the system to insert the bad data as a comment and continue.

The rules themselves are entered via a GUI which handles the basics. More complex operations (such as tagging based on textual content) are handled by adding snippets of Perl code to the basic rule. The system is simple to operate (unless you want to do something really arcane) and has been battle tested on some large and complex conversions. Best of all, as the rules are modified the system requires less and less human intervention.

How Many XML's

It's possible to get the impression that you must explicitly mark up your document to handle any forseeable application. This is not entirely correct — you should mark up to *ultimately support* any forseeable application. The document that you author and edit is a master document. In other words it is the one from which all other variants can be derived. The tagging of a master document should directly support only two functions:

- it must contain sufficient tagging to *reliably imply* any forseeable variant;
- subject to the above it must be amenable to understanding and modification by human editors.

For example, this tutorial is marked according to the XML 2001 DTD. It contains very few tags in common with HTML, and yet we can be entirely confident that a valid and meaningful HTML variant can be produced. The HTML will be entirely different from the original in a number of ways: the markup will be entirely different; it will be spread over several files; it will include a hyperlinked table of contents.

The important point is that we don't have to worry about any of this when keying the original document, since we first ascertained that the markup in question did reliably imply all the features we wanted in our HTML.

The same applies if we want to map our document into (say) a relational database table. Take a phone number which needs to be mapped into the relational fields *country_code*, *area_code*, *local_number*. We could mark the number:

```
<phone-number><country-code>1</country-code>
<area-code>412</area-code>
<local-number>555-4821</local-number></phone-number>
```

or we could use the simple:

```
<phone-number>+1-412-555-4821</phone-number>
```

Which is correct? There is no simple answer. The first form looks very explicit, but how would it handle phone numbers in small countries with no area codes? The second is straightforward, but could a missed key by an operator render the whole number unreadable? For that matter what would a missed key do to the first version?

The format of your master document must fit in with your overall workflow, editorial guidelines, and other software. Provided that this is achieved, the precise level and style of markup is really quite secondary.

You may find that once you've accumulated a body of XML documents, new uses for the data will become apparent. An example might be auto-generation of tables and indexes. Provided that you have marked the content that is of interest, you will be able to introduce these new applications with little or no change to the markup of existing documents.

Tools of the Trade

The question of which tools to use with your XML documents is one of the most difficult to address. Ever since its inception, XML has spawned a wide and ever increasing set of associated standards, new methodologies, and supporting software. And since new and upgraded tools are constantly appearing, it is impossible to make firm recommendations in a static document such as this tutorial. So we will mention only a few of the most basic tools of the trade, and give some advice on what you should be looking for in a useful application.

Apart from the home sites of the individual products, there are a number of web pages which list available XML applications. One of the best is *xmlsoftware.com* which presents lists of tools by category. Another good place to begin is *google.com* — just type in the name of the application you're looking for, and you'll get back a list of sites, most relevant first.

XML Editors

For most organizations the first item on the shopping list will be the basic author/editing system. We have had experience with three systems: ArborText Epic (formerly Adept), SoftQuad's XMetaL, and Altova's XML Spy. These are arranged in descending order of cost (and time since initial release). All three will do a good job on the desktop, and the choice will probably boil down to details of price, programming interface, and handling of associated standards such as the DOM, Schemas, XSLT, and stylesheets (CSS and FO).

We would not normally recommend editing tools based on Word, WordPerfect etc, as both performance and standards support are necessarily not as good as tools specifically designed for XML. However, if your main concern is having a unified tool on every desktop, then these solutions might be worth considering.

Repository/Database

Storage of your XML documents is a more complex matter. There are several competing technologies available at the moment, and many companies (including Turn-Key!) are working on experimental applications which attempt to address the shortcomings of the existing offerings. The currently available strategies fall into four general groups:

- *Simple file hierarchies* — the cheapest and simplest option is to set up a directory hierarchy and access your documents as individual files. Judicious choice of structure and naming conventions, use of entity inclusions etc. can make this solution surprisingly effective. The major drawback is functionality. Versioning has to be done manually, and search and retrieval facilities are crude.
- *Document repositories* — many document repository systems (such as Documentum) offer some XML specific support. These systems offer the traditional services of versioning, tracking, workflow etc, but support for XML standards such as the DOM may be incomplete.
- *RDBMS add-ons* — major players such as Microsoft and Oracle are beginning to add XML functionality to their relational database products. While the concept is new, the underlying database engines are stable mature designs with performance and scalability to burn. These systems work well, but they are expensive and often support XML standards in unorthodox fashion. The so-called object-relational mapping techniques used to map the hierarchical XML content into flat table records have some major limitations, but these should not be apparent to the casual user.
- *"Native" XML Object Stores* — applications such as Excelon and Tamino have eschewed traditional relational models in favor of an XML object technology which supports the XML object model (DOM) and XML Query syntax directly. They tend to be easier to use and more fully featured than other systems. Their principle weakness is in sheer brute strength handling of large quantities of data. This is an emerging technology, and each release sees a substantial increase in performance, but if lightning response times are required then the RDBMS systems still have the edge.

There are a few other innovative repository tools out there, but examining them is beyond the scope of this document. If you are (say) looking for super fast query response or want to handle huge documents efficiently, then you should hit the search engines and newsgroups. New solutions come on line every week or two.

Rendering Systems

Once you have your data neatly marked up you will normally want to render it in some visible form. This section deals with tools which produce quality typeset renditions for hard copy, PDF etc. Conversion to HTML for web page creation will be covered in the following section.

There are a number of systems based on traditional composition engines. These include FrameMaker, XyVision, Ventura and InterLeaf. Turn-Key's own TopLeaf (on which this document was set) also falls into this category. As yet most of these systems use their own internal control files to perform the translation from XML to print, though there is an increasing tendency to offer support for XSLT and/or CSS.

Selection will be based on a number of factors such as price, features, expertise within the organization, special features (e.g. TopLeaf is particularly strong in looseleaf production). One thing to avoid are systems which convert the XML file into an internal markup which is then subject to "final correction". If this feature is offered, don't use it! Corrections applied to anything other than the master XML file run directly counter to best practice. At best they are recurrent waste work, at worst they can lead to serious errors and anomalies.

A new class of rendering engines is worthy of mention. RenderX and FOP are specifically designed to render XSL:FO (flow objects) as printable pages. RenderX is a commercial product, while FOP is an open source initiative of the Apache project. However, these systems come with a caveat. XSL is a very new standard, having been finalized in October 2001. Support for flow objects in both systems is incomplete, though they should work well for "normal" print jobs. More significantly, you need an XSL:FO file to drive them. Flow objects are not intended to be edited directly, but are the result of a transform of your original XML data. As yet we are not aware of any XSL:FO creation system which offers features comparable to a traditional composition engine. But it's only a matter of time.

HTML and Beyond

XML data has the potential to be mapped into any number of forms. An HTML equivalent for display on the Web is a very common requirement, though other formats (e.g. RDBMS record files and even differently marked XML) are both possible and often useful.

The mapping between these forms is normally achieved in one of two ways. Firstly applications such as OmniMark, and high level languages such as Perl can be used. While these are programming solutions, the process can be relatively painless. Perl for example has a number of freely available XML modules which handle many of the basic functions. A complete DOM interface is available if needed.

However, XML provides its own means for reformatting XML documents. XSLT provides a set of actions which can be expressed in the form of an XSLT stylesheet. Such a stylesheet is itself an XML document, and systems have been designed to accept a stylesheet and source document, and output the result of the transformation (which may or may not itself be an XML document). In particular such a system can be used to convert a source document to a set of flow objects which can in turn be rendered as print (see previous section). In current usage however, the most common target is HTML (or XHTML, the XML compliant equivalent).

Whatever the desired end product, an XSLT transform requires two utilities:

- an XSLT *editor* to help create the stylesheet; and
- an XSLT *engine* to perform the transformation.

The number of tools of each type is staggering. They range from full featured GUIs to program libraries for Java, Perl etc. The *xmlsoftware.com* website is a good starting point for selecting a tool to meet your needs.

Markup Design

The mistake people tend to make in this area is to concentrate on DTD and/or Schema development. The starting point should always be the marked up data. Start by getting the markup into the form you need, then develop the DTD/Schema to match. This section contains a few tips which address the most frequently encountered issues in markup design.

How Much Markup

The simple answer is: Mark up as much as you need and no more.

But what does this mean in practice? Essentially markup is an investment in your data which involves an initial outlay, but can yield substantial benefits. Like any other investment there is no point outlaying in areas which yield no returns. Every tag and attribute in your data represents time and money in its creation and continued maintenance. A useful tag will pay its way many times over, but over-tagging is worse than pointless.

Exactly what constitutes useful markup varies according to how the data is to be used. Thus a sentence in a grammar text might be marked:

```
<para><verb mood="imperative">Experience</verb>
<article>the</article> <noun>luxury</noun>
<preposition type="possessive">of</preposition>
<compound-adjective><adjective>red</adjective>
<noun>leather</noun></compound-adjective> <noun>upholstery</noun><punc>!</punc></para>
```

whereas the same sentence in a car company sales brochure appears as:

```
<para>Experience the luxury of
<accessory part-code="UH07" color-name="Carmine Sunrise">red
leather upholstery</accessory>!</para>
```

So you should mark up everything that is, or may be, useful for your own purposes

Separate Structure from Content

You will find life far easier if you use one set of elements to convey the structure of your document and another set to encapsulate the content. To check your markup, divide the tags into three classes as follows:

- Any element which introduces element content (i.e. can never contain text directly) is a *structural* element. This includes empty (singleton) tags.
- Any non-structural element which can appear directly within a structural element is a *container*.
- The remaining elements are *delimiters*.

Structural elements provide a skeleton for the document, while containers introduce the content. Delimiters are typically used to distinguish some fragment of the content. For example, in this tutorial document, structural elements include: `gcapaper`, `author`, `section`, `randlist`; containers include: `title`, `para`, `code.block`; and delimiters: `b`, `i`, `keyword`, `acronym`, `code`.

You will have successfully separated structure and content if *there are no nested containers*. For example a `para` can directly contain any number of delimiters (to indicate keywords or italicized phrases within the text) or sub-structures (such as lists). It would however not be permissible to place another `para` or even a `code.block` directly inside a `para`. Another way to express this rule is: a container may never appear in any position where text might appear (i.e. in mixed context).

Note that it is perfectly OK to *indirectly* nest containers, provided that at least one structural tag separates them in the hierarchy. Thus, while `para/para` and even `para/keyword/para` break our separation rule, `para/randlist/li/para` does not.

There are certain circumstances in which this rule may be weakened. The classical example is the table cell (entry, td, etc). If we think of the cell as a structural element, then we should never include text directly within it. On the other hand, if it is a container, then we could never have a cell consisting of several paras, since that implies nested containers. In fact a cell is definitely a structural element, but one which carries an implied container if and only if it contains text directly. Thus:

- `<cell><para>Hello world!</para></cell>` is acceptable, since this is simply a container within a structural element;
- `<cell>Hello world!</cell>` is also OK, since a cell which contains text directly implies an (anonymous) container; but

- `<cell>Hello <para>world!</para></cell>` breaks our separation rule, as here the para is contained directly within the implied cell container.

In other words, a cell may contain either text or sub-containers, but not both!

While the separation rule sometimes forces us to introduce a little additional tagging, it should always be followed wherever data may be re-used, as the ability to clearly differentiate between structure and content allows you to keep better control of your document, and simplifies both markup design and a number of downstream processing tasks.

Attributes vs Elements

One question we are frequently asked is whether to include an item of data as an attribute, or within its own sub-element. Consider a book chapter, which has a main title plus a short title which is used in running heads. Do we mark this as:

```
<chapter><title>Innovative Procedures and their Effect
on Management Decisions</title>
<short-title>Innovative Procedures</short-title>
```

or

```
<chapter short-title="Innovative Procedures">
<title>Innovative Procedures and their Effect on Management
Decisions</title>
```

We have a rule to decide issues such as these. In short, data may only be expressed as an attribute if:

- it has a fixed format; and
- it does not form part of the substantive text of a document.

Fixed format data includes:

- selections from a known set of options (e.g. alignment options, US states)
- text whose format is known in advance (e.g. zip codes, URIs)
- numerical data (e.g. "-31", "12pt", "33.3%")

In particular, fixed format data must *never* contain XML tagging, and will not normally require any use of character entities.

So going back to our short title example, we can now apply the rules. Clearly the data is not fixed format. It should also probably be regarded as substantive text. Thus on both counts data of this type must be expressed as a sub-element, and so the first coding is the correct one.

Why is it so important to make this distinction? There is no reason in principle why the short title could not be something like "Implications of *Roe v Wade*", and the coding:

```
<chapter short-title="Implications of <i>Roe v Wade</i>">
```

is not legal XML, since attributes may not contain internal tagging. And while it costs very little to use the correct markup in the first place, it is a difficult and expensive undertaking to address such problems once a large body of data has been accumulated.

A Capital Idea!

It's remarkable how often even XML professionals forget the simple maxim: *Never* key text in all upper case.

Irrespective of how text is displayed, it should always be keyed in mixed case. Headings may take initial capitals where appropriate. But the only material which should be keyed in all caps is text that can never appear any other way (e.g. "NYPD", "IBM", "Them vs Us in the US").

The simple fact is that translating mixed case into all upper is a trivial exercise, while the reverse is not the case. For example, is "VANDERMEER" actually "Vandermeer", "vanDermeer", "VanderMeer" or some other variant. By keying in all caps we have actually lost some of the informational content of our data. Sooner or later we may want to re-use this data, say in generating a table of contents where case will be significant.

Another trap for the unwary are small caps. These little guys look great when used with discretion, but all too often we see markup such as:

```
<song>O<s>LD</s> M<s>AC</s>D<s>ONALD</s></song>
```

when all we really needed was:

```
<song>Old MacDonald</song>
```

Yet another example of confusing style with content. I'll leave it as an exercise for the reader to work out which variant is more often mis-typed by the editors!

Conclusion

Naturally in a document of this size we can only cover a fraction of the issues you may encounter. Nevertheless we do tend to see the same problems cropping up again and again. So study our advice on migration, workflow and markup, adapting it to your own needs where necessary. It should help you avoid the worst pitfalls of the XML adoption process, while ensuring that your data will continue to serve you well both now and in the future.