

# Simple XML strategies for advanced content ownership

*Richard Pipe  
Consultant – Information Strategies  
Infogrid Pacific Pte. Ltd*

## Overview

This document argues the case for XHTML and CSS as the preferred digitization XML Schema to deliver the highest lowest production costs and highest future value for any type of content.

It specifically argues against using XML schema's such as DocBook, TEI and proprietary DTD/schema's and proposes a simple method to use XHTML – the backbone of the Internet, as the encoding method.

The discussion outlines the distinction between business plus human-to-content interface requirements, vs. computer-to-content interface requirements. It explains how the two items can be simply addressed with the addition of controlled vocabularies rather than more complex XML patterning.

Our objectives (which were largely met) were to improve productivity, never compromise quality, create the most flexible long-lasting and reusable content available, and lower current and future costs.

This document summarizes the internal dialogues as we designed the production and ECM tools for advanced content management for use across a wide range of organizational activities.

## Introduction

For years we have been on the lookout for a better schema or DTD, or strategy to add significant value to clients' business activities and content. However a Tower of Babble-syndrome operates strongly in the content technology sector.

There are any number of content XML DTD's and Schema's vying for acceptance and critical mass, and the proponents of each can explain why their solution is best. Two primary arguments are put forward.

1. Our/this system separates content, structure and styling – that is really important. (We are going to discuss why it is not.)
2. Our/this system has better structure for future use and automated

processing. (But we don't know what it will cost.)

## **Infogrid Pacific Experience**

The fact is experience matters. Our various staff have tagged books and designed processors to create DocBook in three flavours, TEI, Palgrave (does anyone even know it), DITA, OeB in seven flavours, our own and other organizations' custom XML Schema's, and a lot of experimental work.

We have designed two XML-to-multi-format output processors, applied it to millions of pages and processed tens of thousands of books. This is non-trivial as many of the formats need different content-image sizes, ID systems and packaging requirements. We have always been seeking content XML nirvana, not just slapping tags on text for a quick buck.

More recently we have added to our production operations, author formatted academic documents such as theses, examination papers, and rare manuscripts in a number of Asian languages – and that includes the metadata. If you think tagging an academic book is hard – try doing it in Thai, Chinese and Arabic. Of course 3-D object digitization, and audio and video processing are implicit in the overall production solutions framework. Basically we specialize in creating usable digital surrogates of real world and virtual objects.

## **Choosing the DTD**

Regrettably there is no single correct XML schema that adequately addresses every type of document and content, and all have “options” that largely destroy their general applicability. It is impossible to add content definitions “on the fly”.

Various schema's are very particular about the content they apply to and the same schema cannot be applied to books of all types, newspapers, periodicals, topic based content and many others.

DocBook has come up with DocBook Lite because the complexities of the full version create too many problems. The same applies to TEI, it has a partner schema TEI Lite for the little people who cannot fathom its incredible depths. Add to this DITA, NewsML, and a potful of other schema's and any digitization facility has the right to think seriously about throwing it all in.

Even “standards” such as DocBook, TEI and OeB have significant options for customization even if it is open nesting depths, and ID forms which make any final output proprietary in a relative sense. Because the custom options are available, and technologists like to put their stamp on things, we get variants by client/aggregator. So even settling on OeB is not a one-size fits all answer.

Safari™ uses DocBook (their variant) to match their type of content and their technology framework – the content has no-where else to go. NetLibrary™, Baker and Taylor, Mobipocket™ and others all have the ultimate solution for aggregation and distribution of eContent – but the content is conditioned to their technology and not much else.

Everyone has an idea how they are going to use all of this for the holy grail of XML content – variable content publishing, but it is probably never going to happen on a sufficient and affordable scale with the tower of babble working so well.

A digitization producer has to be able to create all of these (and other) formats more or less simultaneously. Customization of a standard schema, no matter how small inevitably results in compromises in con-

tent and an increase in processing overheads. Fortunately the sophistication of eBook readers has been low, and proliferation of eBook formats has slowed except for Sony's latest proprietary format release.

### **Addressing all ownership issues**

Of course tagging is just part of the problem of eContent ownership. You also have to store it, preserve the value implicit in digitization or front-production, and process all these formats to make it slightly useful. And then you really, really want a variable content strategy that is easy, workable, and affordable.

OK, so there are plenty of Open Source and proprietary processors for DocBook and TEI, but you still have to pay for technologists to evaluate, customize and get them actually working.

Then there is the difference between front-list production where production cost reduction is a productivity gain; and back-list production, where digital reformatting of legacy content is seen primarily as a cost centre without a clear return-on-investment.

The result of all of these strategies is a dumbing down of content in one way or another. The XML schema's apparently do everything, but only at an enormous complexity and cost. And future reuse in dynamic reconfiguration, and variable content frameworks is largely rhetoric. Virtually no-one can show it now without making everything proprietary.

Perhaps the only XML schema that has achieved significance in quality is ODF (Open Document Format). This uses a rich set of standard XML elements, which have styles attached on an element by element basis. Of course the applications never let you see the XML, and it is impossible to use it for variable content publishing except in an ODF application framework.

How does anyone contemplating digitization of legacy content work their way through all these problems? What publishers and content owners need is an XML format that does all of this easily and at one low cost.

### **The special issue of variable content publishing**

We use the term "Variable Publishing" to address the concept of content reuse by any means to create a new work from a combination of old and/or new. Variable publishing is a hot topic, and after five years of talk is emerging as a real possibility. It is necessary to consider the specific requirements of variable content in our content preparation strategies and not leave it to later to sort out.

To make variable publishing really work the user has to be able to:

1. Locate the content
2. Know the source and target content use compatible XML schema's/ DTDs
3. Know the content is available for part-purchase
4. Evaluate and purchase the content, preferably with micro-transactions
5. Keep track of the source, attributions and other usage considerations.
6. Add extracts from other works or their own contribution.
7. Style and package the whole thing.

8. Put it up for sale to a potential user community.
9. In addition to the core content flow the user may have to:
10. Locate appropriate media that can be used and clear rights
11. Ensure that other contributor information is used under license terms
12. Generate multiple output formats
13. Create print resolution output for Print on Demand or other print-run production.
14. Not have to be a copyright lawyer to write a book

## An Historical Perspective

For the first 500 years of print publishing content existed quite happily without deconstruction into content, style and structure. These concepts existed of course, but they were abstract concepts and it was probably the last thing anybody was thinking as they set rows of lead characters into lines then into a page block and hammering the hell out of it until it made an acceptable impression on paper after being coated with ink. The good old days when everyone knew where a structure ended – but nobody had to place a closing tag!

The vocabulary that developed around publishing was essentially describing the relationship of a (non-digital) bit of content to the paper on which it was rendered. A useful and standard descriptive vocabulary emerged expressing the methods, forms and functions of the mechanical representation of content.

So (in English) we have terms describing the media of documents such as leaf, folio, recto, verso, cover, etc. and terms describing the content layout such as title, header, figure, table, note, margin note, footnote, and many more.

This vocabulary expanded as the abilities of typesetting systems got better, documents got more complex and publishing became more competitive. Sectionalizing of parts of a document merge the two term lists. Front matter, chapter breaks, and back matter are the places where the media and content terms match.

Of course not all print works are books and not all content is words. Sectionalizing a document is an artificial author/editor activity. So we get production instructions like “figures must appear at the top of the page immediately following the first reference in the text”.

In some books sectionalization and layout is an complete artifice – it exists purely for visual appeal; in others it is attempting to structure the information into sequences or patterns that can be absorbed; in others it is just the slug nature of the content itself.

We are now going to have a look at a little make-believe XML which shows how existing XML systems want to mark up content. The figure on the next page (following the layout rule) shows a simple fragment of content as it would appear on the printed page. Using XML structure this looks something, like this...

```
<Section>  
  <header> .... </header>  
  <para> ....</para>  
  <para> ....</para>  
</Section>
```

## I AM AN AHEAD

I am a small and interesting paragraph of content and also a first paragraph after a header.

I am the next small but interesting paragraph of content and a little longer than the previous one, but still interesting.

### I Am a Bhead

I am a small and interesting paragraph of content and also a first paragraph after a header..

I am the next small but interesting paragraph of content and a little longer than the previous one, but still interesting

### I Am Another Bhead

I am a small and interesting paragraph of content and also a first paragraph after a header..

I am the next small but interesting paragraph of content and a little longer than the previous one, but still interesting

```

<header> .... </header>
<para> ....</para>
<para> ....</para>
<Section>
  <Section>
    <header> .... </header>
    <para> ....
      <span class="pagebreak" />
    ... </para>
    <para> ....</para>
  </Section>
</section>
<A Section>

```

Interestingly creating the XML nested complexity is expensive, and processing it for re-use is expensive because I have to understand the very last elements (those three **</section>** things) before I can put everything into its right place.

If we did an XML representation of what a 17<sup>th</sup> century typesetter was doing as he typeset this book we get:

```

<Make this the really big font from drawer 2 />
<Make this the Normal Font and make the line look like a first
line/>
<Put the lead in so this makes the Normal Para />
<Make this the title font from drawer 4 />
<Do the normal para thing again />
<Finish the page block here and hammer it all together />
<put the lead in so the paragraph reads nicely from the previous
page />
<Keep working />

```

This may be a bit fanciful, but the important issue is each valid content item stands-alone as a block stacked under others. When our 17<sup>th</sup> century typesetter had to re-order content, he moved blocks of type up and down – variable publishing – at least until it hit the paper!

A modern typesetter uses a similar system to our 17<sup>th</sup> Century typesetter (irrespective of what the technology is doing). Take a stack of text paragraphs and put styles on them. Styles are then visible and structure can be implied (but is still an abstract concept). Content is reordered by the ever familiar cut and paste, or move para up and down.

```
<p style="AHead"> ... </p>
<p style="firstpara"> ... </p>
<p > ... </p>
<p style="BHead"> ... </p>
<p style="firstpara"> ... </p>
<p > ... </p>
<p style="BHead"> ... </p>
<p style="firstpara"> ... </p>
<p style="paracontinue"> ... </p>
<p > ... </p>
```

So what is the point of these illustrations? The point is that document content structure was an abstract concept until the invention of XML. It was never a concrete issue, and didn't impact the production work of either our 17<sup>th</sup> century typesetter, nor our 21<sup>st</sup> century typesetter because there were no nested closing tags.

Content Style genres such as paragraph, header items, title items, lists, tables, captioned blocks, note blocks, notes, footnotes and others are content constants. No matter what the method of generation, it is these that must be generated.

With the miracle of XML (which is actually pretty cool in the right place), structure becomes dominant. So now people say they structure documents as they jam content into often ill-fitting predefined XML containers in fixed "standard" XML Schemas because the concept of defining a new element or modifying the existing one is frightening and has massive legacy, cost and technology impacts. Once you have made a DTD decision – you are effectively paralyzed in that schema because of the number of maintenance points, and the backwards compatibility issues.

Now everybody thinks structure, and style is the poor tag-along cousin. So do we care? We certainly do. XML structure is for computers – styles are for real people. The same way style can be implied from structural relationships, structure can be implied from style information. XML is not the only way to record document structure.

What is the way through all of this? Read on.

## Structure or productivity

Complex XML DTD's mean you have to pay the structure price to get your content organized in a computer environment. It is as much a limitation for eContent as the dimensions of a page for printed content. It's not magic.

Unfortunately (or fortunately if you like paper) the emergence of the "new" content structure has not removed the requirement for the "old" page structure. Hundreds of years of referring to documents by page numbers has made sure of that. Publisher's are particularly sinful in this regard. It's only eGenerated information from the "community" that is redefining the page (for example Wikipedia).

### The Page Tyranny

The first barrier to effective retro-digitization is to overcome the dual-

structure of print/pdf documents. There is a page structure, and a document object structure.

XML does not allow inter-leaved structures. It is a single set of strict hierarchical relationships and we have to use little “hacks” to overcome the limitations of a page structure nested in our content XML. Notice the embedded paginated tags in the next paragraphs (with side-notes) which illustrate some of the approaches that are used.

This is generally not a problem until we need to re-paginate AND to restyle a page for presentation using the content boundaries of the original print page. We call it the “para-continue” problem and this

```
<span class="page-break" id="7" />
```

absorbs more processor authoring time than any other factor. It has to be address at all levels of style nuance.

The use of a span statement is as elegant as it gets. It can be ignored for continuous formats, but when used for page breaking the split paragraphs have to be handled.

Original pagination introduces significant complexities into back list  

```
<span class="page-break" id="7">Page 7</span>
```

production. This of course leads to other brilliant ideas (user irritations) such as page turning interfaces. We all know how cuddling up in bed, turning the pages.... yawn, a pretty boring old mantra!

This is a particularly nasty hackish method. Using HTML comments for processing instructions.

Fortunately hundreds of years ago printers and typesetters decided that there should be a front matter and body structure that was then divided optionally into parts and chap-

```
<!-- Page 7 -->
```

ters. And that is where variable publishing is stuck today because the biggest problem in digitizing for variable content is getting across the page boundary either during tagging or in post processing.

This also shows the nasty end-of-line/end-of-page hyphenation problem that is all too frequent.

Original pagination increases complexity in content re-purposing and at least means the creation of an additional “pseudo format”. For some reasons most publishers want their on-line books to look like their print books. This is pretty strange as hyperlinking overcomes the page location problem, but that is how they think, and that is how it is.

Until publishers decide that the electronic copy is the master reference and the print edition a surrogate, this problem will remain.

The pagination requirement means a continuous XML document has to be processed into pages for presentation. That's effectively another format which makes it hard to sell content structures that go across page boundaries, so we end up selling pages instead, even though the content page boundary doesn't make a lot of sense from a document model perspective.

And then they propose an XML solution for a problem created by XML when there wasn't a problem in the first place!

One technology solution is an XML database – a special database designed to deliver segments of documents based on the XML boundaries. We have tried it, and continue to refine our strategies, but except for purpose authored content, and content that is limited in complexity it still has a way to go and is probably a sub-optimal answer. Putting content into technology, instead of using content over technology is also an approach we don't like as it introduces unpredictable future costs when content survival depends on technology.

## XML Schema's: the digitizing perspective

When digitizing and converting large volumes of content from paper to XML the targeted output schema is not as important as the ease with which the documents can be converted from unstructured electronic

documents (PDF/RTF/PostScript) or paper to an XML that can be used for anything/everything.

In simple terms don't digitize to the output format, digitize for the most productive system and then process to the output format.. If the two are highly similar, that's a benefit.

The real objective of a digitizing operation is to get the maximum XML quality and value from auto assisted processes, the minimum editor keystrokes (that improves quality) and keep the customer happy with a low price while handling all the complexity and flexibility requirements. This is so important we have a hierarchical production system that doesn't even consider productivity as an issue:

Handle all Quality, Complexity, Flexibility issues and you have the highest possible productive solution. Anything else means quality failure or compromise which means reduced content value or rework. In either case the job should not have been done in the first place.

How to get trusted quality faster, with complex outputs is not a common conversation and is very different from the technical viewpoint. We know from our experience that quality in digitization is largely rhetoric – there are few sensible auditable systems.

We learned this after creating multiple technology oriented frameworks rather than user oriented frameworks. You only have to use sophisticated products like the XMLSpy range to understand the technology viewpoint of a human interface. It slows the user down and quality is dependent on the goodwill/skill/mood of the editor. There is no linear product with auditing, there is no quality control or assurance methodologies.

What if it was possible to get both efficient AND effective digitization and conversion production operations and a sophisticated XML output with simpler methods? This is the problem that the Strategic content production frameworks solves.

If content is going to be digitized, it needs to be digitized once, and deliver a wide range of current and future (potential) values. Versaware coined the term in 1999 – “Digitize Once-Use Forever”, which everyone uses now (and it has become a bit tedious, because they are not telling the whole truth). The “use forever” means you can probably reuse it, but it will cost a fortune in supporting technologies.

At the end of the day, no matter how you slice, dice or cut and paste it, content is about the presentation and very possibly, the harvesting of information. The value proposition for XML reformatted content distills down to the following activities:

- Styling. You can't ignore styling. It is what makes a published document distinctive, competitive and readable.
- Grouping. Certain content have group relationships that must be maintained for the content to be reused. Examples include lists, figures, Tables of Content and other items.
- Generation. When reusing content certain content needs to be regenerated depending on the context. This includes Tables of Content, Indexes, page references
- Inserting/Moving. Some things (like figures, tables, footnotes, notes) have to be moved and put in the right place.
- Linking. Content often has relationships/references with itself (internal references such as TOC, index, glossary, see-also's etc.) and other

content (bibliographies, external references, emails, downloads, etc.). It is nice if they can link.

- Everything else is the document flow.

Automated reuse implies all of these have to be doable from the XML.

## Blending the old with the new

So after years of focusing on structure, we finally discovered the secret to advanced, amazing XML was to give everything the right name at the right time. This is how automated expert system tagging works, implying blocks from styling information. The style naming strategy has to be usable by: a human (intuitively); the presentation context; and a processor for general processing and reuse requirements.

The decision was made to use XHTML, and more specifically XHTML-2 (a currently un-ratified standard) as the basic tagging schema.

### Give me the tools

It is necessary to define tools that can correctly interpret and apply (with and without operator assistance) hundreds of structures, sub-structures and their sub-types for content to gain significant future value, and for the investment in conversion to be worthwhile.

Everything in content is a paragraph by default, so the big job is easy. Paragraph types that are defined by strict style similarity are also easy to auto or manually tag. Sections that are defined by vocabulary items such as Table of Contents, Chapter 1 (or One) are also relatively trivial to auto tag once sufficient vocabulary nuances are identified – even though there will always be exceptions.

Then the problems start.

If quality and consistency are serious objectives valuable paragraph and content styling have to be largely carried out by inspection and human assistance for all but the most simple documents. Consistently finding all internal links to a pre-defined quantifiable quality standard is impossible to automate.

When converting text to structure, interpretation is from style, not vice-versa. It is therefore important for productivity, quality and flexibility that the digitizing editors (humans) are given the shortest path to success. This means they observe a document, analyse the styles and map the styles to the electronic text and let the processor do the rest..

There is no requirement to go through a filter of XML knowledge to do this. The entire focus is on document content and style interpretation so more people can be trained to do the work better. No person should ever be made to look at XML tags – it's inhuman and hopefully one day will be illegal.

In the boolean/binary world of technology where content, structure and style are separated, structure is seen as generating style – which it does after a bit of assisting technology does its work and a technology matched style sheet exists.

A very difficult aspect of document interpretation is the use of horizontal space in traditional typesetting and word processing. Author create technical documents often liberally use horizontal spacing in an entirely ad-hoc manner, ostensibly to make the document easier to read or

It's hardest when there is nothing to tag. Horizontal white space brings its own share of problems to content encoding.

Parallel linguistic content is without doubt the hardest content to handle. It often ends up an image.

because the author was bored and just playing with the styling tools in his word processor.

Structurally speaking horizontal space means nothing (except perhaps in tables), but when it does mean something such as poetry, aligned lines of linguistic text, or computer code, a very rich repertoire of semi-structural elements is required to interpret it. This means we start to get attributed tags with styling instructions and the whole structural purity thing immediately breaks down.

The alternative is that horizontal space has to become content (such as in nested computer code where tabs, spacing and character placement are almost a religion), or a style statement, something that instructs a processor to preserve white space – but that is often a compromise. Eliminating correct horizontal white space can turn powerful poetry into insignificant text.

Table tagging is another example. Only the most simple and trivial data table will render adequately without column width statements (style statements).

## Structure, content and style

The big thing with the content model from an XML technology-viewpoint is the separation of structure, content and style (SCS). Few people think of their document from the structural (or at least XML structural) perspective except professional document designers and editors using XML.

When authoring a document most authors work linearly on their content and add styling to denote structural divisions later. There is no other interpretation of document structure other than what you see through styling. So traditionally structure is an abstraction expressed through styling.

Even while editing and rearranging this document I cut and paste many sections around and reapplied heading levels. I said "This content will be better if I move it here". Not once did I think structure, I thought only of expressing the ideas, and making their relative importance stand-out with various styles. If I had had to maintain closing tags every time I would have gone mad (or as some people would state – madder!)

Technology said – we will take this abstract structure and turn it into these nested XML block tags and that will be step forward. So the abstract structure now becomes concrete and styles are implied through structure interpretation. We have been diverted to consider the technology need for structure as the most important issue ever since.

A 300 year old book has as clear a structure as a modern academic book. It is implied from the type styling and page layout. Gutenberg can proudly say - "No XML was used in the creation of this book- no closing tags guaranteed".

Common wisdom in technology circles is that style is optionally applied to structure. This is driven purely by the necessity of the closing element (well formed rule) in XML structured documents so software can understand it.

### Keep it simple...

It is equally as relevant that structure can be interpreted and created

from styles – the way we humans actually read a document - “Oh look a really big heading, that must be the start of something new or important.”.

This is the only interpretation method available to a retro-digitization editor working on legacy printed documents as they analyze a document (book). We cannot get past the fact that text content is written as a series of characters, words (strings of characters) phrases, sentences and paragraphs.

In fact from the content perspective, everything is a paragraph with a style applied (Title, Header, List, Table layout, etc.). We have moved to tagging guides from a publishing world driven by style guides, but the details are not there.

The digitizing editor observes the styles applied to the various structures – and most of these have names (indented bodytext, etc). So the digitizing editor or auto processor takes some content and based on styles (and a bit of contribution from content sometimes) tags like this.

```
<p class="title-chapter-number"> ... some content...</p>
<p class="title-chapter"> ... some content...</p>
<p class="title-chapter-sub"> ... some content...</p>
<p class="title-chapter-author"> ... some content...</p>
<p class="A Head"> ... some content...</p>
<p class="bodytext-first-para"> ... some content...</p>
<p class="bodytext"> ... some content...</p>
<span class="pagebreak" />
<p class="bodytext-continue"> ... some content...</p>
etc.
```

Why do we want to put the human through the extra effort of translating style to structure, when this can be done so easily by technology? It is relatively trivial for a processor to take all the class statements that start with “title... and form them into a block.

This simple XSL...

```
<!-- Grouping Chapter Title Header -->
<xsl:template match="/*[@type='ctitle']">
  <xsl:if test="not(precedingsibling::*[1][@type='ctitle'])">
    <div class="group-title-chapter">
      <xsl:element name="{name(.)}">
        <xsl:copy-of select="@*" />
        <xsl:apply-templates />
      </xsl:element>
      <xsl:call-template name="chaptertitle" />
    </div>
  </xsl:if>
</xsl:template>

<xsl:template name="chaptertitle">
  <xsl:for-each select="following-sibling::*[1][@type='ctitle']">
    <xsl:element name="{name(.)}">
      <xsl:copy-of select="@*" />
      <xsl:apply-templates />
    </xsl:element>
    <xsl:call-template name="chaptertitle" />
  </xsl:for-each>
</xsl:template>
```

Transforms the previous linear style statements into this if we want it for some reason (harvesting or replacing title blocks?)...

```
<div type="ctitle" class="group-title-chapter">
  <h2 type="ctitle" class="title-chapter-number"> ... some content...</p>
  <h2 type="ctitle" class="title-chapter"> ... some content...</p>
  <h3 type="ctitle" class="title-chapter-sub"> ... some content...</p>
  <p type="ctitle" class="title-chapter-author"> ... some content...</p>
</div>
```

Voilà – nested blocks without drama – and no human had to handle the horror of closing tags (maintaining well-formedness).

*(For the technically astute who may be reading this, please note we are using the as yet unratified XHTML 2.0 attribute “type” and there may be typos in the code – the issue in this document is the idea, not the expression of the idea!)*

The creation of complex descriptive schema's such as DocBook, TEI, from this structure is also equally simple because of the simple vocabulary used to describe the constituent paragraphs. And because this is XHTML – it is OeB Ready.

### **It's all in the controlled vocabulary**

The Style XML system assumes a controlled vocabulary. The controlled vocabulary is based on a terms that are understood by every document author, editor and reader. If we need to add a new item we can easily add...

```
<p class="title-chapter-contributor"> ... some content...</p>
```

...and the same system will included it in the grouping (assuming it is in the right place). No DTD change authoring drama.

It is easier, more flexible and cheaper to take a list of items, whose purpose is well described, and turn them into any concrete structure because structural membership of content always (nearly) has a semantic style relationship. When we control the semantic – the structural relationship is assured.

### **The real benefits of simple XHTML**

But the real benefit of this is not just simplicity, consider the following immediate benefits:

1. It is OeB compliant.. The processing requirement is only to create the manifest file.
2. No additional processing to view in a browser
3. If the stylesheet is missing it can still be read in a browser
4. If the stylesheet is present it will be beautiful
5. The stylesheet can be amazingly compact and flexible using multi-selectors – change the entire look and feel with just a few style changes.
6. It conforms to the XHTML DTD so it is well-formed and validates.
7. Any language including BIDI is not a problem
8. BIG BENEFIT It is XSLT:FO / CSS-3 ready – these use a block progression model so this is perfect irrespective of the layout complexity of the target document – in fact it makes XSLT:FO and CSS-3 programming easy!
9. Add ID's to the blocks and you are DRM ready to sell by the paragraph, block or other defined structure with no processing overhead. Just add DRM system and shake!
10. The blocks move freely in and between HTML browsers without any technology overhead.
11. It is the most widely supported XML in the entire universe and always will be for current natural lifetimes (we reckon).

12. There is nothing more interchangeable or easily processed.

13. Everyone understand HTML, so even CEO's can use it.

**The final Twist. Why nested XML structures are bad for content and stacked paras are fantastic!**

Don't ever forget, the human reader will NEVER know how clever your deeply nested XML is – and will never care. Only your processing applications care.

Since I have a stack of content paragraphs in XHTML, all nicely named and identified with unique ID's, why not just load them into a simple database table, then create a hierarchical database table pointing to the content to bring them together by page, or document model or any other method I can think of or need. This is trivial from well tagged, lightly nested content.

This means I can do many things on my document amazingly easily.

1. I can output it as XHTML / OeB for normal Online Reading or for processing to “standard” document XML forms such as DocBook, TEI and any other silly custom XML.
2. I can query it for a page structure (Use the Page Hierarchy map of the paragraph stack)
3. I can query for any Document Object Model structure ( query for all the paras in an A-Head)
4. Because my paragraph stack is separate, I can keep the commerce and DRM on those separate and keep track of who is using what, when, how and under what terms, and how much they were billed.
5. If I want to create a controlled document instance (for example first 5 paras of all chapters as a preview) my query is trivial.
6. If I want to create a new journal from an existing library of academic articles I only have to combine a few titles and ID's in an interface.
7. If I want all the abstracts of 100,000 articles, I don't have to parse any XML, just go and read them out.
8. Content can be moved in and out of the database technology framework easily. XHTML master for long-term preservation, instantiated in the database for variable content strategies.

One simple XML, one simple technology framework, infinite content strategies. But its all about getting the data right in the first instance.

**Viewing Content**

There has to be a rendering agent to view a document. The two dominant formats (excluding paper) are currently Browsers and PDF. There are also a number of proprietary readers in the eBook market, but these can be regarded as custom browsers.

Within the organization there are other formats such as the Microsoft proprietary and Open Document standards based formats.

For document distribution apparently we still need paper, although the temperament is moving to screen reading, especially as content is authored, shaped and optimised as mixed-media and optimized for the screen. And mobile devices are improving in usability making the concept of ubiquitous content everywhere a near reality.

The billions who still don't have computer access, generally also have the disadvantage of illiteracy, and/or little access to existing published material in their natural language.

## XHTML (2.0) Rulz!

We advocate an encoding language with the largest viewer base as the logical environment for encoding documents. That means XHTML. Applications that do not have the resources to process and interpret instructional attributes just ignore them. The source document has the potential to be as structured as required, the user context can be changed easily.

XSLT:FO is the most convenient method to produce PDF's for printed documents from (X/HT)ML to any publishing standard. The same is true for Browsers. The Antenna House FO Renderer is a stunning flexible, multi-language processor that with skill, can render any document to publisher standards.

The simplicity of the elements in XHTML is it's strength. From there you can attributize anything to anything. Every attribute is an alias. Therefore it is relatively simple to maintain the core value of a document, while providing significant interpretation hooks.

## Summary

This document was a long way of saying use XHTML, use XHTML value added attributes and you have the content in shape for anything that we can see on a reasonable technology horizon.

It also says this is delivers the lowest total cost of ownership in development, over time and when things have to be reused.

It says the real structure of any document is a stack of paragraphs in a sequence (with minor exceptions for tables and inserted content), each of which has a describable role, and treat it that way.

Use anything more complicated and think heavy-duty DOM and SAX processing and

- Your content will cost you over and over again
- Your technology costs will go through the roof
- The infinite variety of content will constantly bring you to your knees as you redefine your schemas and customizations.

These are important issues with ultra-large content collections, with a wide variety of content types, and where automation is essential.

Variable content reuse is relatively trivial from a content block progression model. It can go into a standard database block by block if necessary. Nested XML unnecessarily increases the complexity of the model.

Content expressed as a simple sequential stacked block model (as opposed to a nested block model) can be transformed instantly and trivially (relatively) to any distribution schema required and is ideally suited to create the area tree model of XSLT:FO and use in standard databases.

Our distributed production solution and online variable content production and presentation solutions use the simplicity and power of XHTML in combination with advanced style tagging tools to achieve unprecedented quality, productivity and the lowest possible long-term cost of ownership. The tools can be configured and changed on a document by document basis to express all content types, and centrally controlled style tagging structures mean extensions can be added on the fly while tools are in use – even with hundreds of encoding workstations dis-

tributed around the world.

Our content ownership strategies include Reader/Writer to exploit the infinite potential of stacked blocks of content for reuse and collaboration.

## About the Author

Richard Pipe is the founder of Infogrid Pacific Pte Ltd. and has been a technology leader in the large-scale digitization arena since 1998, when the business really got under way.

Before starting Infogrid Pacific he was the CEO of Versaware India, Digital Publishing Solutions India and DX Technologies. At peak these companies employed around 1,300 editors and 200 programmers, testers and process engineers. He was the principal designer of the technology frameworks and strategies for these companies including advanced digital reformatting production systems, Online readers, and a range of supporting products. During his tenure the company carried out digitization of over 15,000,000 pages of high-end academic content, newspapers, periodicals a few documents and reinvented the processes and technology four times for quality and productivity optimization.

After navigating the various companies through the dot-com melt-down, an ever changing panel of share-holders and holding management that didn't really get it, he decided it was time to realize the dream through a new company and started Infogrid Pacific.

Besides working on publisher content and technology, Infogrid Pacific specializes in advanced and difficult ECMS problem domains such as digital preservation of cultural artifacts (production of digital surrogates), Asian language production and educational objects in multi-lingual environments.

When not locked away in the back room analysing content, Richard consults and designs information strategies and complete solutions for government and private organizations, institutions and small scale businesses, and oversees Infogrid Pacific's technology and production strategies.