

XML Capabilities of Popular Publishing Applications

Produced by Really Strategies, Inc.

618 S. Broad Street

Lansdale, PA 19446

May 2006

www.reallysi.com

Contents

Introduction: The XML Tipping Point?	1
Data Storage: From Shredding to Native XML Support	2
RDBMS XML Support	2
Native XML Repositories	3
Conclusions	3
Composition: How Adobe Put its Best Foot Forward	4
InDesign and XML.....	4
Improving the base	6
Conclusions	6
Content Management: Documentum and XML	7
Documentum XML Basics	7
XML Capabilities Beyond Storage.....	8
Documentum and XML Editors	9
Support for other XML-Related Standards.....	10
Conclusions	10
Microsoft Word: Finally XML?	11
Get XML Out.....	11
Put XML In.....	12
Play by Your Rules: Attaching a Schema.....	13
Conclusions	15
Final Thoughts	16
Additional Resources	17
About Really Strategies	17

Introduction: The XML Tipping Point?

XML is a foundation to enable the dream of next generation publishing models: Making it possible and affordable to create content once and use it many times in print and digital formats, to harness content in new ways to hook it into electronic products, to harvest data hidden inside it, and to extract portions of content from across large content collections to create new, targeted products.

To achieve the dream, software that touches content needs to be XML-aware, and those XML-aware software products need to excel in two key ways. First, products need to make foundational capabilities like editing, transformation, and search easier, more flexible, and faster to deploy. Second, software for working with XML content should actively promote the transition to new publishing processes that simultaneously result in both high quality content and products. Achieving the second is dependent on addressing the first.

Unfortunately, ten years after the introduction of XML, even usability, flexibility, and rapid rollout are mostly a dream. In the last several years, however, there has been important progress in the right direction. This white paper describes XML-related developments in key types of software used by publishers, including:

- **Data storage**, including the advancement of XML capabilities in relational databases and the emergence of powerful native XML repositories.
- **Composition systems**, focusing on Adobe's Creative Suite, which has started to make great strides in supporting XML compared with its competition.
- **Content management systems**, notably Documentum, probably the most widely used content management system by publishers.
- **Microsoft Word**, the most widely used authoring tool, and one that has for years challenged publishers trying to fit it into an XML workflow.

This white paper does not attempt to cover the breadth of available XML-enabled software, but focuses on widely-used applications that have added greater XML capabilities that move us closer toward having the foundation described above.

More than ever, support for XML is rapidly expanding and evolving. The landscape is constantly changing, and we look forward to the day when we can survey products that make it possible not only to work with XML, but to fully exploit it to fulfill the dream of next generation publishing.

Data Storage: From Shredding to Native XML Support

All publishers need somewhere to put their content. For publishers using XML, the most typical storage method has typically been a relational database. If more than simple file storage is required, publishers map XML structure to traditional relational database models and “shred” or fragment documents into fields as they are loaded. (For example, the sections in a chapter might be stored as fragments so that they can be retrieved as separate items.) In this scenario, metadata often becomes fielded entities in the database and XML content blocks are stored as CLOBs (Character Large Objects) or BLOBs (Binary Large Objects).

Shredding requires significant upfront design work to map XML content into relational fields. Elements and attributes that must be independently accessible must be identified early. Later changes – such as new elements to chunk or XML schema or DTD modifications – require changes to the underlying data structure, and potentially the need to export and reload the content into the database. It can also have extremely negative performance implications for content loading, search, retrieval, and export if there are more than a few levels of nested XML chunks.

These drawbacks matter little for data (highly fielded information with relatively static structure) that happens to have been encoded as XML. But experts concede that shredding is less than ideal for complex XML content – documents – that might change over time; in other words, it is problematic for almost all publishing environments. Instead, publishers need inherent XML-awareness and XML functionality with improved document retrieval performance, querying, and document management. The development of XQuery, a W3C specification for querying XML, is evidence of the limitation of relational methods (specifically SQL) when working with XML for documents.

XML support in data storage is at a revolutionary stage. In the past year or two, we have seen the emergence of “native” XML repositories (some existed prior to that, but only received much attention in the past few years) as well as ever increasing XML support, or at least claims of it, from the big RDBMS vendors: Microsoft, Oracle, and IBM.

RDBMS XML Support

Recognizing a growing usage of XML, RDBMS vendors have worked to increase XML support. For the most part, the systems still require shredding XML into a relational format, but vendors have added XML field types and support for XQuery to help simplify XML application development.

At the time of this writing (April 2006), Oracle and Microsoft still require storing XML content in a relational model, and as a result suffer from the issues mentioned earlier in this section. Oracle recently purchased Sleepycat Software, producers of the open source Berkely DB XML, but it is unclear whether Oracle will integrate Sleepycat’s XML capabilities into its core offerings.

IBM’s Viper, code name for the next version of IBM DB2 Universal Database 9.0, promises to bring a convergence of relational database technology with native XML storage. If the promises are true, this could be a very exciting next step in the evolution in data storage. DB2 pairs a native XML repository along with the traditional relational database, and the query engine is “bilingual”: It understands both XQuery and SQL and allows endless combinations and nestings of the two – SQL inside XQuery inside SQL inside XQuery, and so on. It is neither a native XML repository nor RDBMS, but a combination of the two.

IBM claims that DB2 allows for foreign key relationships between XML nodes and relational fields. Indexes can be created for XPath expressions (no predicates) to improve performance. Viper also has some appealing schema management capabilities, including the ability to modify schemas without reloading content (something required by the relational-only approach). Unfortunately, it appears that Viper will not allow the combination of full text and XQuery queries found in truly native XML repositories. IBM has not set an official release date, but they have made beta copies available and it is believed it will be released in mid to late 2006.

When tracking the growth of XML-awareness in relational databases, it's helpful to remember that the vendors are not targeting publishers in particular. They are focused on use within their largest markets, especially enterprise-sized businesses, whose needs don't always align with those of publishers.

Native XML Repositories

Native XML repositories have no relational database. They simply store XML as XML. It's really not much more complicated than that. XML of varying formats can easily be loaded into the repository without predefined mapping, because there is no mapping. The XML *is* the data model, as well as the fundamental unit of storage (not database tables, columns, and rows).

Generally, native XML repositories offer exceptionally strong support for XML-related standards, such as XQuery and XSLT. Through these tools, the nature of the storage format, and indexing of the content, these repositories provide powerful and quick access to and manipulation of the XML.

In the end, simpler solutions usually win out over complex ones, and native XML storage is definitely simpler – and more powerful – than shredding. For these reasons, publishers have increasingly invested in native XML storage over the past 12 to 24 months as the repositories matured and as business demanded better and faster XML content access. Two of the more well known repositories are the commercial MarkLogic Server and the open source eXist.

The publishing industry is a primary market for Mark Logic. The company understands publisher requirements to support capabilities like dynamic content selection and assembly, automatic pagination, back-of-book indexes, and PDF generation. Target customers include scientific, technical, and medical (STM) publishers; news and trade publishing organizations; abstractors and indexers; aggregators; market researcher organizations; vertical search portals; and education providers. MarkLogic supports XQuery but has also added its own extensions to provide even greater XML querying, full-text search, and transformation capabilities. eXist is also offers strong XQuery with extensions, but, as is to be expected from open source software, lacks the performance and scalability of MarkLogic Server for massive content sets.

Conclusions

Navigating the native XML storage landscape is a challenging journey because of rapid change. There are many questions to ask when considering an approach today: What is my current platform and environment? Even if another system is better, is it worth the investment to switch? Will my current system “catch up?” Do I need both an XML repository *and* a relational database? How tightly do they need to be integrated, if at all?

Publishers must be careful to clearly identify the business drivers important to their organization. After defining these goals and prioritizing requirements, the project leadership must consult with its technical team to determine what approach makes sense within that publishing environment.

Composition: How Adobe Put its Best Foot Forward

Batch composition systems like XPP, 3B2, or TeX have included XML capabilities for years, and continue to improve. However, most composition is performed with desktop publishing tools – specifically Quark XPress and Adobe InDesign. The lack of XML support in those products has been a significant barrier to XML adoption.

For years Quark XPress was the leading desktop program used by publishers, supporting most designers' print needs. What it didn't – and doesn't – do well is support interoperability or standards, especially XML. It includes some built-in tools, and other vendors have added more support, but, in the end, XPress just isn't developer-friendly. Having achieved a near monopoly in page composition applications, Quark lost its customer focus and didn't innovate in areas outside design.

Enter Adobe InDesign.

Adobe saw the chance to take the lead in the page composition market and followed through remarkably well with InDesign. By the time InDesign Creative Suite was released, Adobe made enough designer-friendly and time-saving innovations that InDesign became the better application and, with a dedicated customer focus, InDesign began taking away large swaths of the Quark XPress market share.

Adobe also saw the importance of meeting the needs of the technical crowd responsible to integrate desktop publishing into the larger toolset. Ask any vendor who has added support for both InDesign and XPress to their production management system; they'll say InDesign is a lot easier to work with.

As part of this big-picture focus, Adobe added XML support within InDesign. At first it wasn't much to write about. The primary problem was that the user interface was too clunky to use in the publishing process, and the XML support had some basic deficiencies. That was okay during the first few releases as very few publishers were able to figure out how to use XML in their desktop publishing processes anyway, and the demand for XML as an intermediary for online versions of print was not nearly as high as it is today. With some work and some compromises, it was useable and passed as being better than what Quark XPress offered out of the box.

Today, having captured a substantial part of the market, Adobe realizes that the time has come when XML is a must-have. The demand for online content is increasing significantly. Being focused on the customer, Adobe has clearly decided to take XML seriously. While still not perfect, Adobe's CS2 increases support of XML in some important ways.

InDesign and XML

Editing and Tagging

InDesign CS2 allows users to work with tags – XML elements – to mark up content.

XML tags can be applied manually within InDesign, although it is cumbersome. Other XML constructs like attributes can also be added, although InDesign does not help the user by reading the DTD to identify available attributes or attribute values. It is also possible to use scripting and programming to do more complex automated tagging work.

InDesign includes a useful structure panel to show and navigate the XML tagging. Color and markers are used to indicate tagging in the document text itself. XML tags are visible in the Story Editor.

Tags can be created through InDesign, or by loading a DTD. Tag sets can also be exported and imported.

InDesign allows the user to validate against the DTD, although this is not always practical in a production setting. InDesign does not enforce DTD conformance during editing.

Style-to-tag mapping

A production specialist can map character and paragraph styles to tags and vice versa. After a tag set has been created in a styled InDesign document, the style-to-tag map can be automatically applied to tag the entire file. Style mapping is an easy approach for tagging, but requires that style names not contain spaces or other characters that are illegal for XML elements.

Import and export

InDesign CS2 can import XML files into a document for manual formatting and layout or, if the XML's structure corresponds to one that exists in the document, into a template or a model for autoflow or to replace existing content.

Along with a configurable Word and Excel import feature, content can be converted to XML from Word or RTF documents. This is a traditional method for mapping and exporting XML from page composition files and continues to be valuable.

InDesign allows all or part of a file's XML structure to be exported, with several useful options. Alternatively, XML can be exported from InCopy files or from related tools such as the K4 workflow system and the latest Adobe Acrobat Professional.

Tables can be automatically tagged on export, a very helpful feature. Special character handling is a particular plus in InDesign. Exported XML contains numeric character references ("entities") using the UTF-8 encoding.

InCopy

InCopy is a simplified version of InDesign for writers and editors, and has been shown to be highly intuitive for their needs. It's used to edit text in the context of a page layout – so the user can, for example, edit to fit – but without allowing the user to change the page design. Under the covers, an InCopy file is an XML file, tagged to a proprietary InDesign schema and used for several purposes by InDesign: saving backwards from CS2 to CS, for Snippets (which are InDesign page parts that can be shared), and to exchange text with InDesign.

Several editorial workflow systems from developers like DTI, SoftCare, WoodWing, BaseView/Harris and others incorporate InCopy. Because all the text in their systems exists in InCopy, it can potentially be extracted at any time in the publishing process as XML (assuming one is prepared to process Adobe's schema, which is format-oriented).

Scripting and extensibility

Just about everything in InDesign and InCopy is scriptable through AppleScript on the Mac, VBScript on Windows, and JavaScript on either. Every part of the XML tagging interface may be addressed through a script, which allows a great deal of customization. If a user can describe something she'd like scripted, it's probably possible. For C++ developers, InDesign includes an API.

Improving the base

While InDesign is capable of being an integrated part of XML-based solutions, there is still room to grow. It would be nice to see improvements such as:

- Inserting XML elements or processing instructions that indicate page break locations and the page number
- Exporting InDesign notes as XML processing instructions or comments
- Creating drop-down lists for assigning attribute values, and generally making it easier to efficiently assign attribute values
- When exporting, including items from master pages in the content for all the pages in which they are presented
- More easily associating a graphic or inset that is outside the main text flow with a particular location in the text
- Including XMP metadata for images and documents in the XML export (see more on XMP later in this white paper)
- Incorporating contextual and attribute rules into tag mapping
- Allowing the user to specify some XML tagging that should be ignored (not removed) when automatic tagging is performed
- A method for including empty elements (elements that don't contain any content)

Conclusions

Seeing that Adobe is embracing XML capabilities with its system, we anticipate seeing many future improvements. However, it should be noted that additional transformation work will need to be applied to the exported XML to effectively load it into a content management system or to deliver it in its final form. One example is that InDesign still produces a non-standard table structure. While all of the elements are there to recreate a table, the XML will require another transformation to make it standardized or otherwise useful.

Publishers should not expect to use InDesign as an XML editor. InDesign is still primarily a page design tool; page design is a different process than creating structured content, and in some ways the two activities are incompatible. Furthermore, if a publisher needs to create complex, deeply-tagged XML that conforms 100% to a DTD or schema, it should consider ways to do so through either post-processing or significant changes to layouts and composition processes, or even through use of another layout program like FrameMaker. Bottom line: What InDesign provides today is an important step forward towards adding value to content earlier in the content creation process, but the real issue is defining new workflow processes that leverage these capabilities.

Content Management: Documentum and XML

In July 2003 Documentum issued a white paper with the title “Native XML Management with Documentum,” which identified their solution as “the industry’s only enterprise-grade content management solution with native XML capabilities across the entire platform.” And this is true, but only as true as saying that Documentum has native MS Word, PDF and, let’s say, Visio capabilities across the entire platform.

The reason EMC can make this statement is that the Documentum architecture is very flexible with respect to different types of content. It manages all files by storing them on the file system in their original format and stores only metadata in its database. Therefore Documentum can easily support any format present or future. So Documentum is capable of storing native XML – but how extensive are its XML capabilities beyond simple file storage? How accessible is the valuable XML markup? And how extensible is the completed solution?

Documentum XML Basics

Documentum allows users to load XML files into the Documentum repository through either the Documentum Desktop or Documentum Webtop interfaces. (Note that Documentum has had problems with XML capabilities in the Documentum Desktop on the Macintosh platform. It is unclear whether those issues have been resolved.) When a user loads an XML document, he can select a set of configurable rules that will determine how the XML is processed and stored. These rules are expressed in what Documentum calls an “XML Application”.

One of the key rules contained in the XML Application is which XML elements become “chunks” in the repository. Think of chunks as the *record-level* or *manageable* objects. Standard content management operations can be performed on these chunks – including check-out, check-in, versioning, attaching a chunk to a workflow or, perhaps most importantly, referencing a chunk from another document (such as for content re-use). One **cannot** perform these operations on XML elements that have not been defined as chunks. This approach should seem familiar: Documentum has used the typical relational shredding scheme described in storage section above, and it leads to the expected performance and maintenance issues. Changes can require a good deal of work.

Documentum’s XML Application allows the user to map values in the XML elements and attributes to properties (metadata) of the chunk. Users can search for content based on these properties using the Documentum Desktop or Webtop interfaces. Developers can easily access these properties when developing customizations. The flip side is that accessing the values of elements and attributes that are not mapped to properties is very difficult. Also, Documentum’s properties cannot express anything like the rich hierarchical structures in XML; subelements and attributes are essentially lost. (Think, for example, about how many publishers structure an author’s name – first, last, etc.)

Other rules expressed in the XML Application allow application developers to specify things such as where a chunk gets stored in the repository, which access permissions are associated with the chunk, and how links in the XML are processed.

Documentum provides a very nice user interface, the Documentum Application Builder, which, among other things, guides developers through setting up an XML Application to define the rules for processing XML. It’s possible to create any number of XML Applications for applying different sets of rules to different content.

When Documentum breaks an XML file into the predefined chunks, it creates “virtual documents” that represent the hierarchical relationships expressed in the XML. The concept of virtual documents in Documentum predates its XML support. Virtual documents can be used to create a collection of objects for any type of content. And Documentum’s Virtual Document Manager allows one to manage those virtual documents by, for example, adding and removing objects or binding the virtual document to specific versions of the associated objects. When virtual documents are employed as part of Documentum’s XML solution, they become much more complex because it is not uncommon to have a virtual document referencing other virtual documents down to several layers. For example, a book might be a virtual document which contains chapters, the chapters contain sections, and the sections contain topics, all of which are also virtual documents. This is one area where Documentum’s XML solution seems to be stretched to its limits. Deeply chunked content with multiple layers of hierarchy in virtual XML documents leads to significant performance issues and exposes some bugs in Documentum’s support for virtual XML documents.

XML Capabilities Beyond Storage

Documentum provides a broad set of content management capabilities as core components and also offers extended capabilities in the form of content services and platform extensions. The core content management capabilities include functions like check-in and check-out, versioning, access control, search, workflow management, and rendition management. Extended capabilities include content transformation, auto-categorization, content distribution, rich media management, and collaborative services. The details of all of these capabilities are beyond the scope of this paper; however, following is a brief overview of how well the core capabilities support XML.

General CMS Capabilities

As previously stated, it is important to understand that Documentum’s content management operations can only be applied to the XML elements that have been loaded into the repository as chunks (based on the rules in the XML Application). So, if an XML file consists of a <book> element which contains several <chapter> elements which each contain several <section> elements, and the XML Application was configured to “chunk” the <book> and <chapter> elements, a user would not be able to check-out, check-in, and version <section> elements. Of course, the publisher could configure the XML Application to chunk the <section> elements, but there is a performance tradeoff to chunking the content deeply, and the publisher might not be ready to make chunking level decisions at the initial design and deployment of the system. This is because chunking needs are often not obvious until after the system has been deployed – that’s when new workflows are truly tested and when new ways to use content are more fully exploited.

Check out, check in, versioning, and access control are basic functions and there isn’t a need for these actions to “understand” much about the XML markup within the chunk being processed. Although we could make a case for being able to perform actions based on values in the underlying XML (such as applying different versioning rules or triggering other actions upon check-in), such capabilities are handled effectively by mapping the XML values of interest to properties of the Documentum chunk.

Search

Documentum has good search capabilities from both the user interface and developer perspective, but there are some significant limitations with respect to searching XML. The Desktop and Webtop user

interfaces provide simple search and advanced search capabilities. The simple search is a one box, one step search (ala Google) which can be configured by an administrator to include searching on full text and specified properties. The advanced search allows users to select specific object properties to include in the search, perform full-text searches, or combine the two.

For developers Documentum provides a SQL-based search language called DQL. It has all of the power of SQL but is tailored to Documentum. What Documentum does not have is the capability to search full text with an understanding of the underlying XML markup (as in XQuery). For example, a user cannot, through either the user interface or developer tools, perform a search to find all <section> elements that are children of a <chapter> element where the topic attribute of the <chapter> includes “anatomy” and the usage attribute of the <section> is “unrestricted” (unless, of course, all of these things are mapped to object properties). This is perhaps the most significant limitation of managing XML in Documentum. The semantic value of the XML becomes inaccessible because, at the file level, Documentum does not distinguish XML content from any other format.

Workflow

Documentum’s workflow capabilities are very powerful. They allow developers to define workflow templates as a series of manual or automated activities, assign content objects (including XML) to an instance of a workflow template, route workflow activities to various users, loop back to previous steps, and associate workflow steps with content life cycle states. Documentum includes a graphical interface (the Workflow Manager) for defining these templates and the Desktop and Webtop interfaces each include an InBox for users to manage their assigned activities. As with other Documentum capabilities, the workflow component has no access to the XML elements or attributes. If a publisher wants a workflow, or an automated activity within a workflow, to behave differently based on values in the underlying XML, those values must be mapped to object properties. For example, if an XML document should be routed to different users for review and approval depending on the values in a “topic” attribute, the attribute must first be mapped to an object property. This is fairly easy to do but requires more up-front planning to implement and more effort to modify later than if the workflow component could directly query the underlying XML.

Rendition Management

Documentum’s rendition management functionality allows users to manage a content object and various renditions of that content (e.g., HTML or PDF) as a single component. A user could, for example, manage as one component a chapter of a book in XML along with its HTML and PDF renderings. Various other operations and customization can then deliver the appropriate format to a given user or application. Rendition management is not a component that would obviously benefit from the ability to interact with the XML structures in the file. These renditions themselves can be generated outside of Documentum or by using Documentum’s Content Transformation Services.

Documentum and XML Editors

There are integrations for Documentum with the two leading XML editing tools, Arbortext Epic and XMetaL. The XMetaL integration adds Documentum functions to the editor menus and also provides an explorer-like user interface for accessing content in the repository, allowing users to perform basic content management operations such as log in, check in, check out, copy or link (via simple drag-and-drop), all within the XML editor user interface. Users can search or browse the repository to find content.

They can then choose to view content (read only without locking it in the repository) or edit content (locked for exclusive write access). They can check the content back into the repository and indicate how it should be versioned, and can also set object properties either directly on the Documentum object or in the XML (for mapped properties). The Epic integration is very similar.

The integration enforces whatever access restrictions are defined for the user within Documentum. One area that is not well integrated is the workflow component. Users cannot directly assign a workflow to a content object. They must first locate the workflow template, start a workflow instance based on the template, and then assign content to that workflow instance rather than simply selecting a content object and workflow template. Users are also unable to access their workflow assignments through the XML editor interface but must use the Documentum InBox. Still, these are very good integrations and are fairly easy to extend to meet custom requirements.

Support for other XML-Related Standards

- Documentum supports SGML DTDs, XML DTDs, and W3C XML Schema. Documentum validates the content against the DTD or schema when the content is loaded or checked-in.
- Documentum supports XSLT stylesheets. Stylesheets can be applied by users or automatically via workflows or other applications.
- Documentum claims to support XLink but there is not much more information available as to the specific capabilities, and we have not encountered an implementation using XLink.
- Documentum does not support XQuery for querying XML. Do not mistake Documentum's XDQL as being akin to XQuery. With XDQL a developer issues a DQL query and the result is delivered as XML. DQL doesn't allow one to query XML in the repository.

Conclusions

Although Documentum does indeed provide XML capabilities across its entire platform, the degree of XML support varies from component to component. And all components are hampered by the fact that the Documentum content server does not support intelligent XML searching. The server treats all content the same at the file level. Therefore, a publisher must determine which elements and attributes are important early in an implementation and then map those to chunks and properties with its XML Application (similar to the "shredding" of XML content into relational databases mentioned previously in this paper). This approach is not very flexible. If it is determined later that additional elements or attributes are of importance, it will be necessary to update the XML Applications and reload content to affect those changes.

That's the state of things today. It will be interesting to see if EMC identifies support for richer XML capabilities as an important market need and how they respond.

Microsoft Word: Finally XML?

In most circumstances, publishers avoid saddling authors and even editors with content structuring tasks, and particularly with XML tagging. Yet publishers do need to apply structural and semantic markup in content to maximize its value. Most publishers cannot expect content experts, who may be freelancers or other external resources, to use something as sophisticated, technical, and costly as an XML editor like XMetaL or Epic. Often they don't want to anyway – publishers pay authors to author, not to tag content.

So most authoring and editing is done in the tool everyone has, Microsoft Word. Publishers have implemented different processes, or even tricks, to transform Word files to a structured format like XML. Word itself has not been particularly helpful in this regard.

Since the release of Microsoft Word 2003 Professional Edition there's been a great deal of interest among publishers about its XML capabilities. Can publishers sneak some tagging into Word in a way that is acceptable (even invisible) to non-technical users? Could Word export usable XML? Could it perhaps even replace full-blown XML editors? Below is an overview of the key capabilities of this version of Word's XML features, followed by a conclusion and thoughts on Microsoft's promises for the next version of Office, scheduled for late 2006.

Get XML Out

Word allows the user to “Save As...” XML, but the XML is bloated and full of superfluous information that Word needs in order to reopen the XML document with every Word feature intact, as if it were saved as a .doc file. It is similar to saving Word as HTML and viewing the source. It's certainly not a human-friendly format. In fact, there's so much markup it's quite hard to find the content. Following is a small sample of “WordML”:

```
<w:docPr>
  <w:view w:val="print"/>
  <w:zoom w:percent="100"/>
  <w:doNotEmbedSystemFonts/>
  <w:proofState w:spelling="clean" w:grammar="clean"/>
  <w:attachedTemplate w:val=""/>
  <w:defaultTabStop w:val="720"/>
  <w:punctuationKerning/>
  <w:characterSpacingControl w:val="DontCompress"/>
  <w:optimizeForBrowser/>
  <w:validateAgainstSchema/>
  <w:saveInvalidXML w:val="off"/>
  <w:ignoreMixedContent w:val="off"/>
  <w:alwaysShowPlaceholderText w:val="off"/>
  <w:compat>
    <w:breakWrappedTables/>
    <w:snapToGridInCell/>
    <w:wrapTextWithPunct/>
    <w:useAsianBreakRules/>
    <w:dontGrowAutofit/>
  </w:compat>
```

```

</w:docPr>
<w:body>
  <wx:sect>
    <w:p>
      <w:r>
        <w:t>In most circumstances, publishers avoid saddling

```

However, with a little knowledge of the WordML schema, a developer can write an XSLT to deliver the essential structure and content of the document, for example, capturing just

```

<w:body>
  <wx:sect>
    <w:p>
      As
</body>
<section>
  <paragraph>

```

Thankfully, Word lets us apply such a transformation as part of the Save As function (Fig. 1):



Figure 1

When the “Apply transform” box is checked the “Transform” button may be used to pick an XSLT file written to perform the transformation. In Figure 1 the XSLT file is named “simple.xml.” Now the “Save” button will deliver the transformation results rather than WordML. The output will be a brand new XML, HTML, or plain-text file (based on the XSLT’s output method), which may or may not look anything like the original document. It all depends on what the XSLT is designed to accomplish.

Put XML In

Again, with some knowledge of Word’s XML schema, it’s possible to write XSLT to add the minimum markup that will enable Word to open an XML file as WordML. But there is no Word dialog that allows the user to select the XSLT. Instead, the XSLT must be referenced *inside* the XML file using a processing instruction like this:

```
<?xml-stylesheet type="text/xsl" href="simple2wordml.xsl"?>
```

where the href attribute points to the XSLT file.

Play by Your Rules: Attaching a Schema

So far this probably seems like a minor simplification of an old problem: transforming data to and from Word. Whereas formerly it may have been solved by VBA (Visual Basic for Applications) scripting, or via costly third-party conversion software, it can now be solved via XSLT. But is it really necessary to play by Word's rules? Can publishers instead make Word play by *their* rules, and manage the markup dictated by the publisher's schema instead of Microsoft's? The answer is a qualified "yes."

Opening an arbitrary XML file in Word has surprising results (Fig. 2):

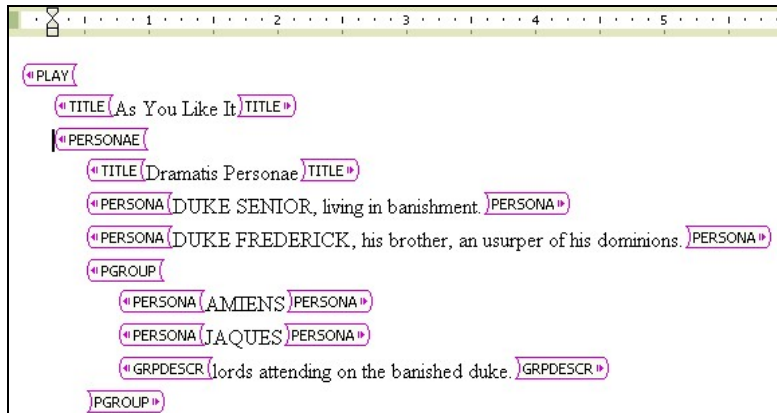


Figure 2

Tags! Structure! But don't get *too* excited. Unlike Epic or XMetaL, where the user is constrained to follow a DTD or schema, it's far too easy to go "off the path" editing the XML directly in Word. Figure 3 illustrates that it's possible to insert text outside of the <title> element.

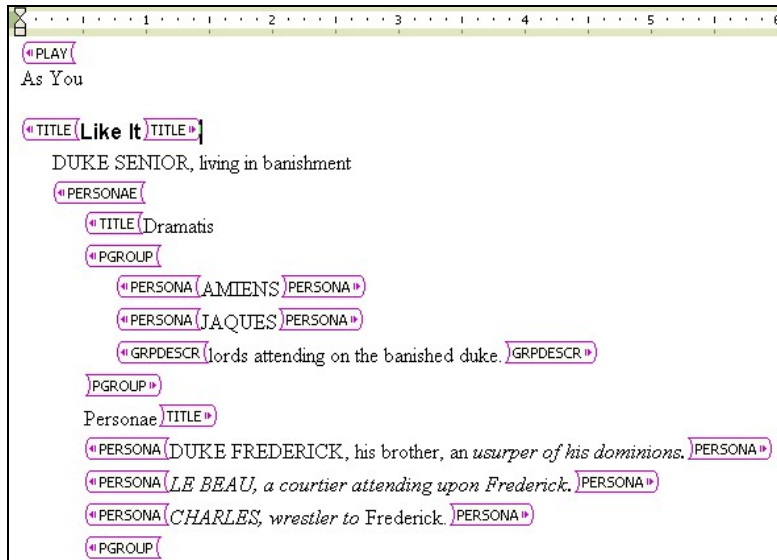


Figure 3

In fact, text can go anywhere, whole chunks of markup can be moved into invalid locations, and unrestricted access to Word's formatting capabilities is allowed to add styles across any markup boundary. It

is similar to the use of styles within Word. Sure they are helpful, and if a user understands how to use them, they mimic markup, but they are very easy to ignore or misapply.

Only one more step is required before this can be called XML editing.

From the tools menu the Templates and Add-Ins panel can be invoked, which includes a tab dedicated to XML options. Here is how XML namespaces – the *user's* XML namespaces – can be associated with a W3C XML Schema (not a DTD) to use for validation. Now when an XML document is opened with Word, and Word encounters a namespace that's been registered here, it “attaches” the corresponding schema to the document. In practical terms that means it provides *some* of the smart editing and validation similar to XMetaL or Epic. The gateway to these extra features is the Task Pane (press CTRL-F1 in Word to see the Task Pane). The Task Pane performs multiple tasks, from style management, to template selection, to online research, to XML Structure (Fig. 4):

In the Shakespeare example, the cursor is inside the ACT tag, so the corresponding tree view and a list of elements that can go inside ACT are displayed. But don't be fooled: these are *all* the possible child elements of ACT, regardless of any ordering constraints. If an out-of order element is inserted, some rather cryptic feedback is revealed (Fig. 5):

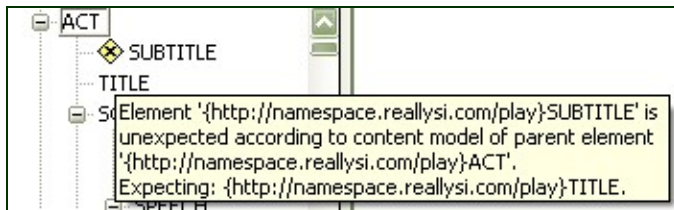



Figure 5

The user must decipher this – not hard in this example, but if there are multiple  flags in the structure view and the schema is unfamiliar, it can be a nightmare getting a document to validate.

Once finished editing a document, care must be taken to save it via the “Save As” dialog, selecting the misnamed “Save data only” checkbox (Fig. 6):

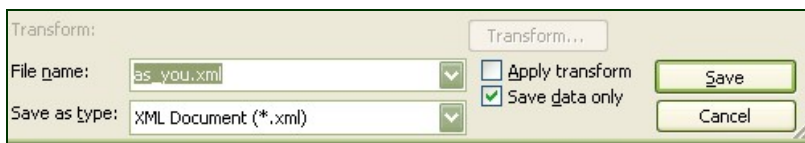


Figure 6

In Word, “data” is all content and the markup, but without WordML. If the checkbox is not selected, the result is an ugly amalgam of WordML interleaved with other tagging.

Attribute values have not been discussed because there's not much to say: attributes are only revealed by right clicking on a tag to get a context menu. Next “Attributes” must be selected to get a dialog showing all the assigned or available attributes and their values. This is a major drawback if the user needs

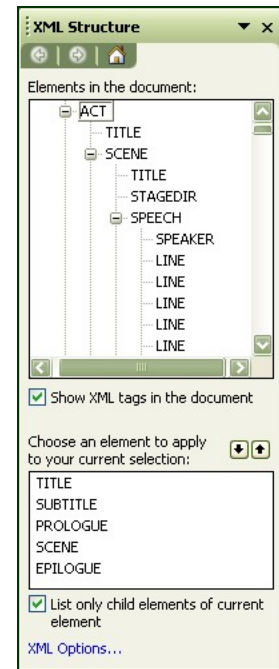


Figure 4

quick access to attributes; for example if the schema uses a single <emphasis> tag with a style attribute to get bold, italic, underline, and so on.

Conclusions

The ability of Word 2003 to open and save XML in the form of WordML offers a new starting point for writing import and export programs: XSLT. But despite some initially enticing interface components in Word's XML view of a document, its XML capabilities fall far short of the kind of structured editing offered by dedicated XML-only editors. We can only recommend using Word for native-XML editing by extremely XML-savvy users, who are also highly knowledgeable about the markup constraints and peculiarities of their documents. However, it is quite possible to make a decent transition from Word authored documents into an XML process. It is also possible to create a round-trip process for XML files that need review by a non-XML savvy author or editor: set up their computer to work with the schema, send the user XML to edit in Word, clean up the file in Word after it's returned, save the XML.

Additionally, Microsoft expects to release a new generation of Office, due out at the end of 2006. This is the first time since Office 97 that they're changing to a new file format, called "the Microsoft Office Open Format." And the change is radical. They are unraveling Office documents by storing their constituent objects (XML content in custom schemas, content in the Office schemas, images, charts, and so on) in independent files, each of which can be accessed, read, written to, and otherwise processed on its own. Relationship files describe how the objects fit together into a document. The objects, relationship files, and a file describing the file types are gathered together into a zip file with an appropriate Office extension (e.g., "docx"). Each Office application opens its own zip file format directly – most users will never even know that it's a zipped file.

Microsoft has been clear that their purpose is not to create an alternative to native XML editors. Instead, their goal is to create an environment in which business users can more easily search data in back end systems and include that data in documents in ways that allow the data to be automatically kept up to date. For example, if a sales manager authors a document about her company's projected revenue, she can use an InfoPath form, with XML under the covers, that is embedded directly in the document she's working on to report on the current numbers from all her salespeople, embed the result in her document, and refresh that number at will as the end of the quarter approaches. This naturally assumes a programmer did some work to set it up for her, presumably in a template.

The data could also be delivered in the form of an XML document that is included in the document's zip package but not displayed directly – it just becomes a data source living behind the scenes and traveling along with the document even when the system from which it was extracted is not accessible. From Microsoft's perspective, the goal is productivity gain through gluing data to documents.

We never know what to expect from Microsoft, but this does sound promising.

Final Thoughts

Publishers no longer question the validity of XML, but ask instead how best to incorporate XML into their workflows and which tools they should be using. At last, XML support is becoming stronger and more intricately woven into many publishing applications, and there is a reasonable answer to these questions. The software landscape is constantly changing for the better. XML-based standards such as XQuery, XSLT, and XPath can enable new publishing opportunities as they are integrated into publishing tools. Major vendors are indeed adopting XML and building robust support – although not usually with publishers in mind.

For those publishers who have not yet made the leap to XML, these developments mean it is time to make the change. Those who already invested in XML should consider toolset improvements. For all publishers, it is also time to begin thinking about what comes next: How could XML-aware software not only enable the creation of XML content, but change the way publishers do business to make the next generation publishing dream a reality?

Here's one example that we're excited about: Adobe's XMP.

XMP (eXtensible Metadata Platform) is an Adobe XMP specification for capturing metadata as XML “islands” inside non-XML files. This means a photograph or video or desktop publishing file can contain structured metadata in an accessible format. Some standards groups have already created metadata sets appropriate to various content types and for various purposes. This is not only technically interesting; it promotes new publishing workflows that could have a dramatic impact how rich media and other file types are created and managed. Software products like InDesign and Photoshop make it easy for publishers to add XMP metadata forms that can be easily transported to an external user's computer. Vendors like Pound Hill Software are created tools to make this easier, and hopefully to do things like enable some level of workflow control on end users' computers (for example, to require that a metadata field be filled out before a photo is submitted to a publisher). Our hope is that the important parts of a publisher's digital asset management system will essentially be “published” to vendors with no system integration required. Digital asset management vendors are adding capabilities to mine the XML metadata and do things like automatically route the files (including rejecting them) based on their metadata, as well as to make the metadata searchable and updatable. This is an example of XML-based technologies changing the way the members of the publishing ecosystem work together, and of potentially enhancing collaboration in an exciting new way.

There is a lot of work to be done before XMP is fully supported by software, but the vision first promoted by Adobe is an exciting step towards a next generation publishing models.

Additional Resources

Adobe CS2: <http://www.adobe.com/products/creativesuite/main.html>

Brian Jones, Microsoft Engineer, Blog: http://blogs.msdn.com/brian_jones/

David Kellogg, Mark Logic CEO Blog: <http://marklogic.blogspot.com/>

eXist: <http://exist.sourceforge.net/>

IBM DB2: <http://www-306.ibm.com/software/data/db2/udb/viper/>

MarkLogic Server: http://www.marklogic.com/products/ml_server.html

Oracle 10g: <http://www.oracle.com/database/index.html>

Really Strategies Blog: <http://blog.reallysi.com>

SQL Server 2005: <http://www.microsoft.com/sql/default.mspix>

XMP Open: <http://www.xmp-open.org/>

About Really Strategies

Really Strategies, Inc. is a privately held company that was founded in 2000 to provide world-class content solutions and services to publishers, media companies, and other content-centric companies. From content creation to delivery, Really Strategies helps bring strategy, content, and technology together to analyze, architect, and implement appropriate tools and technologies. Our solutions encompass XML editorial tools, XML repositories, content management systems, and editorial and production systems. Our services include workflow reengineering; technology evaluation; DTD and Schema development; business, functional, and technical requirements development; and electronic product development strategy. As a recent recipient of the Deloitte Technology Fast 500 Award, Deloitte and Touche Rising Star Award (Delaware Valley), the Philadelphia 100® Award, and one of the "Best Places to Work" in Philadelphia Award by the Philadelphia Business Journal, Really Strategies is committed to building the premier content solutions and services firm.

Contact Information

info@reallysi.com

www.reallysi.com

(215) 631-3107

618 Broad Street, Lansdale, PA 19446

Barry Bealer, President/CEO

bbealer@reallysi.com

Lisa Bos, Executive Vice President, Chief Architect

lbos@reallysi.com