XML 2002, Baltimore

Baltimore, MD USA
December 8-13, 2002

Using XSL Formatting Objects for Production-Quality Document Printing

Kimber W. Eliot
ISOGEN International, LLC
eliot@isogen.com

Abstract

The XSL Formatting Objects specification has been a published recommendation for over a year. During that time a number of commercial XSL-FO implementations have become available that make it possible to use XSL-FO for true production-quality creation of printed documents. While there are functionality limitations in XSL 1.0 that limit the types of page layouts that can be created, the page layouts that are supported are sufficient for most technical documentation, such as user manuals, maintenance manuals, and so on.

This paper evaluates the XSL 1.0 specification and the currently-available implementations against the print requirements typical of a variety of document types, including technical documents, magazines, and newspapers. We then report our experience in using XSL-FO with commercial tools to produce hardware user manuals for a line of consumer computer peripherals. We discuss the XSLT and XSL issues, as well as XSL-FO extensions that may be required to satisfy typical print production requirements. Finally, we provide a set of recommendations, based on the current state of the XSL specification and the current state of tools, as to when the use of XSL-FO is appropriate and which XSL-FO implementations are best suited to which tasks or disallowed by certain sets of requirements.

Using XSL Formatting Objects for Production-Quality Document Printing

§ Introduction

The purpose of this paper is to answer the question “can XSL Formatting Objects (XSL-FO) be used to produce production-quality printed and online documents?” The short answer is “yes, within some reasonable limits.” That's a good and useful answer, but just saying “yes you can” fails to convey just how important that “yes” is. The fact of the suitability of XSL-FO[XSL-FO] for production-quality print production is the realization of a dream more 20 years in the making, a dream that a great many people have shared and that a great many dedicated people have worked very hard to realize.

In an attempt to convey what this “yes” really means, I start with a short history lesson, then follow with a discussion of XSL-FO as a technology, its general suitability to various publishing tasks, and the current state of XSL-FO implementation. This paper is not a tutorial on how to create XSL-FO style sheets.

XSL-FO, The Dream and the Reality or How I Came to Write This Paper

The precursor to SGML, GML, was an application of IBM's Data Composition Facility (DCF), a full-function page composition system that IBM had developed to do document and bill publishing. GML came out of the box with high-quality page composition features (although somewhat limited stylistic choices unless you wanted to write a lot of DCF code to implement your own presentation styles). For its internal use (and later as a product), IBM developed BookMaster, a GML application that directly supported all of the print (and online) publication requirements for IBM technical manuals, both hardware and software. BookMaster was also a DCF application. It provided more stylistic choices and an easier way to define new styles, but it was still essentially a giant DCF macro package. When we started defining IBM's SGML replacement for BookMaster in the very early 1990's, we quickly realized that doing the same high-quality page composition from SGML instead of from GML was going to be a challenge at best. This was because SGML, being completely generic and implementation independent, did not have a built-in page composition system.

The whole point of SGML was that you could do many different things with the same documents. But that also meant that to do any one thing you would have to buy or build it, in addition to whatever you did to create and manage the SGML itself. At IBM we had the advantage that we could still use BookMaster and DCF to compose from SGML simply by writing a relatively simple SGML-to-BookMaster transform. DCF was a mainframe-only product (DCF itself is largely implemented in IBM 370 assembler, if I understand correctly). This was at a time when mainframes as work support systems were being replaced by PCs. It was clear that using BookMaster to compose SGML documents was not a viable long-term solution. It was about this time that I left IBM to become a consultant for a small company. IBM went on to use Xyvision XPP and Omnimark to build a

very sophisticated SGML-based publishing system for its technical documents, but at great cost (this was 1994-1995 and IBM's requirements were quite large, as you might imagine). There were simply no other viable technical solutions that would satisfy IBM's technical documentation requirements.

The original developers of the SGML standard had realized that publishing from SGML would present this sort of challenge and recognized that what was needed was a standardized, implementation-independent way to compose SGML documents for print and online use. Soon after the publication of the SGML standard in 1986 they started development of the DSSSL standard (Document Style Semantics and Specification Language)[DSSSL]. Also at about this time, the U.S. Department of Defence started developing their own standard for SGML style sheets, the Format Output Specification[FOSI]. Many SGML practitioners like myself started beating their heads against the practical problems of publishing from SGML. There were many problems with the tools used to address these needs, from poor parsers to editors that put odd things in files to composition engines that were expensive to buy, difficult to configure (being essentially typesetting macro languages like DCF), etc. Tools came and went. The DSSSL standard was finally published in 1996 and we had great hope for it, but alas, it came to naught[1]. The only free implementation, James Clark's Jade tool, was good but not feature complete and lacked a few layout features needed for full production use. The only commercial DSSSL implementation was built in Japan by Next Solution and not actively marketed in North America or Europe. At the same time we started to realize that, for all its brilliance and elegance, DSSSL was never going to be viable simply because its syntax, while completely appropriate for the task at hand, made DSSSL style sheets essentially unmaintainable. They were unmaintainable for the simple reason that few people would take the time to learn the syntax because it was very different from traditional procedural languages (DSSSL is Scheme-based).

At the same time, XML popped into existence and our whole way of thinking about how to do processing and how to design processors for generalized markup changed and evolved in important and valuable ways. We realized that simplicity and directness trump elegance, that learnability and maintainability must be given great weight. And we learned that people seem to get the idea of XSLT[XSLT] in a way that they never got DSSSL. So, many of the very same people who developed DSSSL (and who had also been key figures in defining XML), along with many other new players, started defining XSL and XSLT, with the express goal of making it possible to compose XML documents into high-quality print and online pages. They took everything they had learned in the previous 15 or 20 years of painful experience and used it to craft the XSLT and XSL-FO standards. XSLT took the spotlight because its value as a way to generate HTML quickly and easily or to generate new XML forms from existing XML, coupled with its ease of learning, rapid adoption, and wide deployment by a number of very solid implementations, all free, made XSLT an immensely important tool in every XML practitioner's toolkit.

But the dream of composing high-quality pages from XML was still just a dream. Even in 2001, the choices were still limited to the venerable SGML-based composition systems that had been around forever: XPP, 3B2, Datalogics

Composer, Epic (nee Adept) Publisher, FrameMaker+SGML, etc. All good systems, but all showing their age. There were a few experimental FO implementations developed with the XSL specification, such as PassiveTeX [PSVTEX], but they were not yet sufficiently complete to enable general production use.

Then, in October of 2001, the XSL FO 1.0 recommendation was finally published and with it came one free FO implementation, the Apache project's FOP, and two commercial implementations, RenderX' XEP[XEP] and Antenna House's XSL Formatter[AHXF]. At the same time, Arbortext indicated it was developing a full-featured FO implementation for its Epic product[EPIC] and Next Solution, the company that had implemented DSSSL, announced development of an FO implementation as well.

Suddenly the dream seemed very close, at long last, to being a reality. But could it be?

At this same time, late fall of 2001, one of ISOGEN's customers realized that the SGML composition tool they were using to publish user and service manuals for computer peripherals would not be able to handle a number of very important national languages, in particular, Chinese, Japanese, Korean, Thai, Hebrew, and Arabic. This company is a world-wide market leader in its space, so internationalized documents are key to its business. What to do?

I looked at the options and decided that, given what I knew about XML, XSLT, and XSL-FO and what I had seen of the available FO implementations, that it would be worth the risk to try an FO-based solution. The risk was that we wouldn't be able to make it work, either because the tools would fail or the standard wouldn't provide the features we needed. The potential reward was being able to produce documents in all the languages we needed at a significantly lower dollar cost than using the only other solution I could identify (one of the venerable, old-school SGML publishing systems) and significantly lower implementation and maintenance cost because of the inherent goodness of XSLT.

There were some tense moments along the way, and a few late nights, but I can report that we were successful in developing an FO-based solution that does everything we need it to, at a lower implementation cost, lower software cost, and lower maintenance cost, than the system we replaced. At the time of writing, October 2002, all technical issues relating to the formatting requirements and the functionality of the FO implementation used have been resolved (and all but a couple small issues had been resolved by May of 2002) and the system is ready for production use. From this experience I am confident not only that XSL FO can be safely used to do high-quality print production of XML documents but that it is without question a superior solution to anything I have used in the past, as long as you can satisfy your page layout requirements.

I am also delighted that the dream of practical, affordable page composition from XML is finally being realized. There are still a few kinks to work out, but it is definitely a 95% solution, not an 80% solution.

How XSL-FO Works

XSL Formatting Objects is based on a two-stage process for getting from input XML document to composed pages. The XSL-FO recommendation defines a

document type, Formatting Objects, for documents that represent the components of composed, but unpaginated, documents. An FO document instance is then interpreted by FO implementation to create a paginated rendition of the formatting objects by applying the layout and formatting semantics defined in the XSL-FO recommendation. The typical output of an FO engine is a Postscript or PDF file or, when printing directly to a physical printer, printed paper pages. The formatting objects themselves represent the usual typesetting and page layout constructs that have been well understood for decades: page sequences, text flows, blocks, inlines, and so on. These constructs carry a large number of possible properties or characteristics that describe the details of the presentation: geometry, font, color, etc. Many of the formatting characteristics used in XSL-FO are taken directly from the CSS (Cascading Style Sheet) specification.

One important and distinguishing aspect of the FO design is its support for internationalized documents. FO is designed explicitly to not be biased in favor of any particular writing order, writing direction, page orientation or other culture-specific aspect of text presentation. Thus FO has been designed from the start to support, for example, right-to-left writing systems like Hebrew and Arabic and top-to-bottom writing directions like Traditional Chinese, as well as Western writing systems. It has also been designed to accommodate complex glyph layout requirements, such as those of Thai. Of course, this does not mean that all FO implementations can support all writing directions, scripts, or glyph construction rules. But it does mean that authors can be explicit about the features they need page composition systems to support.

You can create FO documents in any number of ways, but the typical approach uses XSLT transforms to generate FO documents from XML documents in exactly the same way XSLT is used to generate HTML documents. It's the same basic task and uses the same basic implementation techniques. The only difference is the output document type. In fact, it's often possible to re-use schema-specific XSLT business logic between HTML transforms and FO transforms.

A Note About XSL-FO Terminology

XSL Formatting Objects has been carefully designed to be culturally neutral. Thus it avoids use of terms like “top” “bottom”, “left”, and “right”, preferring the terms “before”, “after”, “start”, and “end”. These terms are always in reference to the orientation and direction that has been set for blocks and inline text. For Western left-to-right, top-to-bottom layouts, before is top and start is left.

All references to left, right, top, and bottom in the FO specification are for compatibility with CSS but are formally mapped to the equivalent culture-neutral specifications.

A Note About FO Implementations Discussed in This Paper

One purpose of this paper is to evaluate the different FO implementations for suitability to specific requirement sets. All discussion of product features is with respect to the generally released versions of these products available as of 15 October 2002. Where a beta version is publicly available I have mentioned it if relevant, with the appropriate qualifications.

Note that all of the commercial FO implementations are evolving rapidly and that features missing in October 2002 may well be implemented or announced by the time you read this. As a long-time integrator and user of SGML and XML tools I have been struck by the high speed with which the FO implementations have been developed and by the responsiveness of all the vendors to tool issues and feature requests. After a decade of largely incremental improvement to a fairly static suite of SGML and XML tools, it is exciting to see these new FO implementations being implemented so aggressively and with such high quality of implementation, service, and support.

The FO implementations available at the time of writing are:

- Epic Publisher V4.3 from Arbortext[EPIC].
- FOP, part of the open-source Apache project[FOP].
- PassiveTeX, from Sebastian Rahtz[PSVTEX].
- IBM XSL Formatting Objects Composer[XFC].
- XEP from RenderX[XEP].
- XSL Formatter from Antenna House[AHXF].

These products are discussed in more detail in 4 [below]. PassiveTeX and XFC are not included in this discussion simply because they are not yet sufficiently feature complete to be candidates for general production use. But that will almost certainly change with time.

§ Strengths and Limitations of XSL-FO

XSL Formatting Objects has a number of strengths that derive from being XML-based and from the use of XSLT to generate FO instances. It also has strengths that derive from the FO design itself, which reflects hundreds of person years of experience with doing page composition from generalized markup, and in particular, the direct lessons learned from DSSSL, FOSI, and other approaches to standardized page composition.

XSL Formatting Objects has unavoidable limitations from two principal causes: missing layout features and the limitations inherent in the two-step XML-to-pages processing model.

Inherent Limitations of XSL-FO

The XSL FO 1.0 recommendation naturally fails to define a number of features that one might need. Correctly, I think, the FO working group decided to publish something, even though it was incomplete with respect to all the known requirements, rather than wait until the specification was 100% complete. For example, the Working Group realized that it had not provided a full solution for index generation but also realized that a complete solution would require a lot of careful thought and design.

The two-step FO processing model means that there is no standardized way to get feedback from the pagination step to the FO generation step. This makes it difficult to implement presentation features that are dependent on knowing exactly what page something will occur on. Thus, it is impossible using standard features of FO to make choices in the XSLT transform based on whether or not a particular formatting object will appear on a particular page. However, it would be possible for specific implementations to provide this feedback, but that

mechanism would not be a standard, at least in the short term (there's no reason an FO feedback mechanism couldn't be standardized but there is currently no effort to do so as far as I know).

Performance Issues with XSL-FO

The abstract, generalized nature of XSL-FO naturally imposes some performance penalty simply because it requires more computing power to implement a generalized solution than it does to implement a task-specific solution. In addition, the two-step XSLT process imposes another performance penalty, especially if the transformation step results in a file on disk that is then read by the XSL-FO implementation, rather than doing the transformation into a shared memory area that is then used directly by the FO processor.

For most applications these performance limitations are not an issue. A typical FO process instance is comparable to, for example, using Epic Publisher with a FOSI-based style sheet to compose the same document. However, for some high-volume publishing applications the current FO implementations may not be fast enough, although normal hardware solutions can be applied.

There is no inherent reason why an XSL-FO implementation cannot be as fast as the fastest non-FO markup-based composition system, but the focus of FO implementors to date has been on feature completeness more than on raw performance (although all the products have demonstrated significant performance improvements from version to version over the last year). I expect that as the FO implementations become more mature and the use of FO becomes more widespread that the focus of implementors will turn more toward performance.

As a rule of thumb, if Epic Publisher or Framemaker+SGML will meet your performance requirements then any of the FO implementations will also meet your performance requirements. If these products would not meet your performance requirements then the FO implementations probably will not meet them either. And of course, as for all performance issues, Moore's law continues to hold, meaning that many short-term performance limitations may have an easy hardware solution in the near term.

Another significant value of XSL-FO is its ease of implementation and maintenance over time, as discussed in the following sections. For many use cases the value of these characteristics of FO far outweighs any short-term performance issues, especially when performance issues can be addressed by simply upgrading computing hardware. That is, a near-term investment in extra hardware will be more than recouped by both a near-term savings in initial development cost and a medium- and long-term savings from reduced maintenance and incremental development costs.

Strengths Deriving from XSLT

The use of XSLT to generate FO instances from XML documents realizes a number of advantages that stem from strengths of XSLT itself. XSLT is relatively easy to learn, it is widely supported and widely used. There are a large number of skilled XSLT practitioners, which significantly reduces the long-term maintenance cost of XSLT transforms. XSLT has good modularity features, allowing the use of traditional programming techniques for creating re-usable code modules that can

be applied to a number of different document types or presentation styles in a way that few other print production systems can provide. XSLT has good conditional processing features, making it cost effective to have a single style sheet that can account for variations in either the input documents or the output FO instances, again reducing implementation and maintenance costs and improving reusability.

The ability to write XSLT extensions in Java also makes it easier, or sometimes simply possible, to solve specific implementation problems that would be difficult or impossible to do with XSLT alone, including integration of XSLT transforms with underlying information systems, database publishing, and so on.

Strengths Deriving from the FO Design

The design of the Formatting Object document type itself generally makes it easier to create sophisticated page layouts and formatting effects compared with similar SGML- or XML-based composition systems. The FO design reflects decades of experience with doing high-quality page composition, including lessons learned from DSSSL, CSS, FOSI, and various commercial products. The various FO constructions and characteristics will be largely familiar to anyone with page layout experience, whether from a desktop publishing background or a markup-based publishing background. The design is very consistent and largely intuitive. A great deal of careful thought has gone into the FO design and it shows. All of these qualities translate into a formatting system that is relatively easy to learn and easy to apply to specific layout problems.

ISOGEN's experience over the last year or so is that creating an XSLT- and FO-based style sheet requires about one half the effort of creating the equivalent style sheet in a proprietary system. In addition, the incremental cost of adding new document types or new layouts to an existing family of document types or layouts goes down over time as you refine your XSLT code to be more modular, making it easier to add new functionality or new input or output choices. No other SGML- or XML-based composition system has this characteristic.

Strengths Deriving from the XSLT and FO Community

The XSLT and FO communities are quite large and very supportive, as well as a number of good books on applying XSLT and FO. There are a number of online forums and websites where answers to questions will be answered very quickly. There are three, almost four[2], commercial implementations of the FO standard, all useful and appropriate for production use. All of this works to greatly reduce the risk of using FO as a solution for print publication. The competition in the marketplace serves to both encourage vendors to constantly improve their support for FO features, add features to support requirements not met by the FO specification, improve performance, and generally increase the value of their products. This is a level of competition that has not been seen before in the markup-based composition marketplace.

The existence of a number of FO implementations, all of which are appropriate for production use within the bounds of their individual limits, helps to protect FO users from abuse by product vendors. More than almost any other XML technology domain, FO implementations must compete almost entirely on value and not on proprietary lock-in through extensions. This is in part because

the features defined by FO are sufficiently complete that they don't provide much room for proprietary extension and in part because XSLT makes it easy to create style sheets that will work with any implementation (because it is easy to create conditional processing based on the target FO implementation).

§ Comparison of XSL FO To Typical Technical Documentation Requirements

There are few things that FO cannot do. Therefore, it is easier to list those requirements that FO does not meet, rather than those it does. FO provides facilities for all the basic page layout requirements: headers, footers, margin areas, multiple columns, top and side floats, page flows with complex pagination rules, page number references, dynamic, page-sensitive content in headers and footers, tables, graphics, footnotes, and so on. It provides facilities for specifying writing direction (left-to-right or right-to-left), block placement direction (top-to-bottom, bottom-to-top, left-to-right, right-to-left), mixed left-to-right and right-to-left text, and fine control of glyph placement, including different baseline orientations. It enables the easy creation of rules and boxes. It allows the use of a variety of color models (RGB, CMYK, etc.). It provides for absolute positioning of blocks on a page.

The layout requirements that FO as specified does not currently satisfy are:

- Multiple, independent, multi-page flowed areas within a single page body (e. g., two independent articles on the same page that continue to different pages). There are no known proprietary extensions for satisfying this requirement.

  While there's no reason in theory that FO could not be refined to meet the requirements of, for example, magazine design, it is unlikely that it will be any time soon simply because the business case for standards-based authoring and production in magazines is much less compelling than it is for technical documentation (although ISOGEN has at least one customer that would very much like to use XML for design-heavy magazines if it were practical).

  For example, one could imagine an interactive, XML- and FO-based design tool that provided all the design functionality of a Quark or Adobe InDesign, but implementing such a tool would be a challenge and the market, at least in the short term, would be fairly small.
- Revision marks (“rev bars”) that reflect the block-progression-direction extent of text sequences (i.e., the vertical extent of a changed phrase within a paragraph). All of the vendors are working on or have implemented extensions for producing revision marks in some form.
- Before floats within a single column. Before floats are supported, but the float is always to the top of the body area. At least one product (Epic) provides an extension for column floats.
- Dynamic, page-sensitive information in figure captions, table captions, and table headers (although this can be faked using floats or tricks with page headers or footers in some cases). Epic Publisher provides an extension for before floats that can be used to satisfy this requirement.

- Composition of back-of-the-book indexes such that lists of page number citations are reduced to unique page numbers, as well as the automatic creation of page ranges (although it's not clear that page ranges can be reliably created automatically--they can always be created using markup in the original document that identifies the start and end of the range to be indexed). All the commercial FO products provide extensions that satisfy this requirement to one degree or another.
- Writing systems that use alternating start-to-end and end-to-start inline progression directions (some forms of ancient Hungarian, for example). There are no extensions for this requirement (this is a pretty esoteric requirement that would not normally be needed by technical documentation, although it might be needed for scholarly publication).
- Text that flows around arbitrary curved areas (but text flowing around rectangular areas is possible using side floats). There are no extensions that satisfy this requirement.
- Page-location sensitive inclusion or exclusion of content. For example, there is no direct way to condition the text of a cross reference based on whether or not the target of the reference occurs on the same page as the reference itself. There are no extensions that satisfy this requirement.
- Any other presentation tuning semantics that require feedback from the pagination step to the initial FO generation step. But note that this feedback could be provided in implementation-specific ways, enabling a mutli-pass process in which second and subsequent FO generation instances would have access to information about what page a given input node occurred on in the previous pagination instance. None of the current products provide this feature.
- Rotation of flowed text at angles other than multiples of 90 degrees. The FO specification only provides for rotation in increments of 90 degrees (that is, parallel to or perpendicular to the block progression direction). There are no extensions for other rotation options. However, embedded SVG can be used with FOP and XSL Formatter to create a wide range of text effects (XEP removed embedded SVG support in version 3 but RenderX has indicated that SVG support will be restored in a later release).
- Creation of PDF bookmarks, links, and annotations. While PDF is not, strictly speaking, a standard, it is a defacto standard and is almost universally used for online delivery of print-quality documents. PDF includes a number of useful features for online display, including navigation bookmarks and hyperlinks. The FO specification does not provide any direct facilities from which these PDF artifacts could be derived. However, all the FO implementations provide extensions for creating bookmarks, links, and other PDF-specific, online artifacts.

All of these requirements have been submitted to the XSL editors via the XSL editors mailing list.

§ Current FO Implementations

All of the FO implementations clearly document their support for the various FO features. You can use this documentation to determine if a particular product will

support the specific requirements of your documents and processing system. This section does not re-state the feature support details provided by each implementation. Rather, this section talks in general terms about the key features and characteristics that distinguish the various FO implementations.

Of the four full-featured FO implementations currently available, XSL Formatter implements the most FO features, although XEP version 3 almost matches it. FOP, as an open-source, volunteer product is the least feature complete and is not generally suitable for production use at this time. Epic is slightly limited by the fact that some FO constructs, mostly to do with page geometry and page master sequences, have no direct mapping into Epic's underlying formatting engine, which was originally engineered to support the FOSI style language. However, Arbortext has for the most part provided extensions that work around these limitations. Epic has been used for years to do high-quality production of technical documentation, so its support for features that are actually needed by most technical documents is quite good (in other words, the features of FO that it doesn't support are features that you probably don't need anyway if you are producing typical technical manuals).

There are at least two other freeware or open-source FO implementations, but they are not sufficiently feature complete to currently be considered as candidates for production-quality document production of typical technical manuals. Sebastian Rahtz' PassiveTex[PSVTEX] is a TeX-based FO implementation. From the PassiveTeX home page: “provides a rapid development environment for experimenting with XSL FO, using a reliable pre-existing formatter.” IBM's Alphaworks is also developing and FO implementation, IBM XSL Formatting Objects Composer[XFC]. The current version lacks a number of key features that also makes it inappropriate for production use for technical manuals. In particular it does not yet implement footnotes or marker references. Marker references are pretty much a hard requirement for doing technical manuals (enabling dynamically-generated running heads and feet in a page-specific way).

Overview of FO Implementations

Epic from Arbortext is a Windows- and Unix-based FO implementation built around the FOSI-based Epic Publisher composition engine. Epic can be used as a standalone composition engine or in its integration with the Epic Editor SGML and XML editor. The version at the time of writing is 4.3. Arbortext has announced the development of version 5, slated for mid-2003 release, and promises a more complete FO implementation at that time. It can be used either as a command-line tool or interactively through the Epic Editor user interface.

The FOP FO implementation is implemented in pure Java as Apache-licensed open source. It lacks a number of important FO features but is actively being developed. It can be used either as a command-line tool or integrated with other Java tools using its Java API. FOP is also integrated with the eXcelon Stylus Studio XSLT development environment.

The XEP product is implemented in pure Java and can be used with any JVM. It exposes a Java API. It can be used as a command-line tool or integrated with other Java tools using its Java API. RenderX also licenses a version of XEP integrated with the XML Spy editor. ISOGEN has created simple integrations of

XEP with Epic Editor and SoftQuad XMetal, available from the ISOGEN Web site.

The XSL Formatter product is currently Windows-only, although a Java version is being developed. XSL Formatter exposes both COM and Java APIs. It can be used as a command line tool, as an interactive tool using its graphical user interface, or integrated with other tools using its COM or Java APIs. ISOGEN has created simple integrations of XSL Formatter with both Epic Editor and SoftQuad XMetal, available from the ISOGEN Web site.

Support for Non-Western Languages

XML and XSL-FO make it easy as it can be to author and produce documents in almost any national language, including the traditionally "hard" languages[3] such as Thai, Arabic, Hebrew, and the various ideographic languages. However, just because you can create an XML document in Thai or Hebrew it doesn't mean that your FO implementation will be able to render it properly.

A quick note on the jargon of internationalization: in this context the term "language" means a national language, like English or Chinese. The term "locale" refers to a subgroup of users of a particular language. For example, U.S. English and U.K. English, where "U.S." and "U.K." are distinct locales. Language and locale are used to construct ISO language codes, where the first two characters are the language and the second two are the locale. For example, "zh" is the language code for Chinese, "zh-CN" is the language and locale code for Simplified Chinese. The term "script" refers to a system of characters or glyphs outside the context of any language. Thus the Roman script is the system of characters used for most Western languages. The term "glyph" refers to the presentation form used for a given character. A "character" is an abstract thing that may be mapped to many different glyphs. For example, the character "a", Unicode value 0x0061, represents the idea of the letter "a". For a given presentation the character "a" may be mapped to any number of graphic representations of the letter "a", the glyphs. A set of glyphs makes up a font. For most Western languages and scripts there is usually a one-to-one mapping of characters to glyphs in a particular font (because most Western scripts do not use different glyph forms for the same character). For many non-Western scripts, such as Arabic, there may be a one-to-many mapping from characters to glyphs.

Non-Western languages present a number of formidable challenges for print presentation:

- Proper rendering of glyphs. Of modern languages, Thai is probably one of the most challenging. Thai glyphs consist of a number of different components that are combined together in subtly different ways depending on adjacent glyphs and other variables. This makes composing Thai text quite a challenge. Likewise, languages that use Arabic scripts, such as Arabic and Farsi, also have a number of rules for glyph substitution based on where a particular character occurs within a word (initial, medial, or terminal). Rendering of these scripts requires implementation of the appropriate glyph transformation, substitution, or construction algorithms, which may vary based on language or locale.

- Rendering bi-directional text, that is, text with a left-to-right writing order mixed with right-to-left text, such as English words within a Hebrew sentence. The Unicode standard defines an algorithm for determining the writing order of sequences of characters but it is difficult to implement at best. From a composition and layout standpoint, handling bi-directional text can be a challenge as well.
- Correct font selection. For ideographic languages such as Chinese and Japanese there are different locale-specific fonts that must be used. Chinese and Japanese use many of the same ideographic characters (and thus the same Unicode characters) but use slightly different glyphs for these characters. Thus a composition system must be configured to select the appropriate locale-specific font for ideographs, for example.
- Thai word breaking. The Thai language has no well-defined notion of “word,” meaning that there are no natural word breaks as there are in most languages. Thus a composition system must either implement a Thai word breaking algorithm (as Microsoft Word does, for example) or the input to the XSLT or FO processor must have word breaks added. No FO implementation currently provides a built-in Thai word breaking algorithm. However, ISOGEN has adapted a Thai word breaking algorithm provided by IBM as parf of its ICU4J package into a SAX filter that can be integrated with any SAX parser, such as Saxon or Xerces. This filter inserts zero-width space characters into PCDATA content as it passes through the parser so that the data passed to the XSLT processor already has appropriate word boundaries. The algorithm is not perfect but is good enough for most applications[THAI].

Of all the FO implementations, XSL Formatter[AHXF] has the best support for non-Western languages, including build-in locale-specific font configurations, complete Thai glyph construction, and full support for bi-directional text. RenderX is close, supporting bi-directional text and a limited implementation of the Unicode bidirectional algorithm, but requires more effort to configure locale-specific fonts.

Support for Graphics and Mathematics

All the FO implementations support the common bitmap formats GIF, JPEG, and TIFF (although XSL Formatter requires a separate, nominally-priced license for GIF rendering. XEP and Epic support scaling of bit-mapped graphics, although the quality of the scaled result may be poor in some instances (RenderX recommends scaling graphics to the appropriate size before including them in the FO instance to avoid any problems with dynamic scaling, either by the FO renderer or by the presentation device (i.e., PDF)).

All the FO implementations support EPS graphics. XSL Formatter and XEP only support interpreted EPS graphics when using their Postscript output options (as opposed to their direct-to-PDF options). For direct PDF generation, they both use the EPS preview image, if present. Note that both XSL Formatter and XEP include PDFMark in the Postscript they generate, meaning that you can create “online” PDFs from both tools using a Postscript-to-Distiller process instead of the direct-to-PDF process. Epic does not have a separate direct-to-PDF option, instead requiring the use of Distiller to create PDFs. Thus interpreted EPS graphics are always supported by Epic.

Epic supports CGM graphics, as does XSL Formatter when using the free ISOView CGM viewer plug-in on Windows.

FOP and XSL Formatter support the use of embedded SVG graphics. XEP had partial SVG support in version 2 but removed it in version 3, although RenderX has announced the intent to restore SVG support in the near future.

XSL Formatter supports embedded MathML through the use of a Windows MathML rendering plug-in. Epic supports the use of TeX for mathematics (the underlying Epic composition engine is TeX based).

XSL Formatter supports the Windows WMF (Windows Metafile) format.

Support for Non-RGB Color Models

All of the FO implementations implement RGB color exclusively. The generation of Postscript or PDF that uses CMYK or another color model requires post-processing. There are a number of RGB-to-CMYK post processors available for both Postscript and PDF, including a PDF plug-in.

§ Future Enhancements of the XSL-FO Specification

It appears that the XSL FO specification is not being actively developed at the moment. The XSL working group is focusing all of its energy on XSLT 2.0. This suggests that enhancements to XSL-FO in the short term will be either proprietary extensions or community-developed extensions in the model of the EXSLT extensions[EXSLT]. It is probably be prudent to gain a bit more implementation and use experience with XSLT 1.0 before attempting the enhancement of XSL-FO to address any outstanding requirements. It is also the case that several of the remaining requirements present serious design and implementation challenges that will not be solved quickly.

§ Conclusions

Our experience to date with using XSL-FO for production of high-quality printed documents, primarily technical manuals, has been tremendously positive. At almost every step of the process of implementing FO-based publishing solutions the task has been easier than we expected. We have been pleasantly surprised by the how much easier it is to create and maintain FO style sheets using XSLT as the transformation technology than any other publishing technology we have used in the past. We have been impressed by the quality of the FO implementations and the responsiveness of vendors to problem reports and feature requests.

It seems clear that by mid-2003 there should be at least three, if not four, fully-featured, stable, and mature FO implementations from which to choose, with a high degree of interoperability among them. As the open-source implementations improve, that will put increasing pressure on the commercial vendors to increase the value of their products, most likely by providing better performance and a greater range of support for graphics formats and specialized requirements not met by the FO specification directly.

Thus, the answer to the question of “can FO be used for production print production?” is a largely unqualified “yes.” As long your layout and formatting

requirements can be satisfied by FO or FO plus the available extensions, there seems to be little reason not to use an FO-based solution.

Notes

1. However, a small but dedicated band of DSSSL enthusiasts have kept the flame alive by forming the Open Jade project to continue development of the Jade DSSSL engine.
2. Next Solution's FO implementation, announced at XML Europe 2002 and still, presumably, under development at the time of writing.
3. Hard only from a Western perspective, where we have been spoiled by having a very simple script and writing system. If the first digital computers had been developed in China or Thailand or India then working with ideographic languages or languages with complex glyph construction rules would not be hard at all because computers would have had to handle it from the start. We are only now, with the advent of widespread support for standards like Unicode and XML coupled with fully internationalized operating systems, able to correct the decades of culturally insensitive computer systems developed by Western engineers.

Bibliography

[AHXF] Antenna House XSL Formatter product. A Windows-based FO implementation. Version current at time of writing is 2.3. See http://www.antennahouse.com for more information. Free evaluation version available

[DSSSL] ISO/IEC 10179:1996, Document Style Semantics and Specification Language (DSSSL). See http://www.jclark.com/dsssl for more information.

[EPIC] Epic page composition system (an optional feature of the Epic SGML/XML editor). See http://www.arbortext.com for more information. Available on Windows and Unix platforms (but not Linux).

[EXSLT] A set of community-defined extensions to XSLT 1.0. See http://www.exslt.org.

[FOP] Apache Project's FO implementation. Open source, volunteer-developed FO implementation. See http://www.apache.org for more information. Implemented in Java.

[FOSI] Formatting Output Specification Instance, defined in U.S. Department of Defense standard MIL-PRF-28001. See http://navycals.dt.navy.mil/28001/28001c.pdf.

[PSVTEX] PassiveTex. A TeX-based FO implementation developed by Sebastian Rahtz. See http://www.tei-c.org.uk/Software/passivetex/.

[THAI] Thai word breaker SAX filter, written in Java. Available from the ISOGEN Web site, http://www.isogen.com/downloads. Open source under LGPL license. Based on IBM ICU4J Thai word breaker.

[XEP] RenderX XEP product. A Java-based FO implementation. Version at time of writing is 3.02. See http://www.renderx.com for more information. Free evaluation version available.

[XFC] IBM XSL Formatting Objects Composer. Developed by IBM alphaWorks. See http://www.alphaworks.ibm.com/tech/xfc.

[XSL-FO] XSL 1.0 Recommendation ("Formatting Objects"), published by the W3C October 2001. See www.w3.org/TR/xsl.

[XSLT] XSL Transformations (XSLT) 1.0 Recommendation, published by the W3C November 1999. See www.w3.org/TR/xslt.

XML 2002

Baltimore, MD, USA, December 8-13, 2002

This paper produced from XML source via XSL, Saxon and Antenna House's XSL Formatter product.

