CSS, XSLT/FO, TopLeaf — A New Approach to Rendering XML Documents

Geoff Nolan

Senior Systems Engineer
Turn-Key Systems

A great deal of work has gone into creating such standards as CSS, XSLT and XSL-FO, which allow human readable XML documents to be rendered on-screen, as hard copy or PDF. CSS is easy to work with — using GUIs even non-experts can produce quite complex stylesheets — but lacks key facilities for producing publishable quality pagination. XSLT/FO has virtually unlimited potential for the manipulation and display of source material. However the design of the required formatting objects, not to mention the XSLT transforms which create those objects, is well beyond the capabilities of non-specialists.

The latest version of Turn-Key's TopLeaf rendering system is an attempt to provide the facilities of professional quality typesetting, but using a simple intuitive interface. To achieve this a number of innovative techniques have been developed. This presentation will compare TopLeaf's mapping strategy with classical stylesheets, illustrate some of TopLeaf's unique features, and touch upon the role and relevance of standards in the development of the next generation of rendering tools.

Introduction

XML documents today are typically rendered in one of three ways:

- via CSS stylesheets in browsers and editors;
- with an XSL-FO rendering system, after prior transformation through XSLT;
- using a proprietary system such as FrameMaker, XPP or 3B2.

Since the late 1970s we at Turn-Key Systems have been rendering documents (and in recent years SGML/XML documents) with our own proprietary system, TopLeaf. Accordingly we watched the development of XSLT/FO with great interest. Our own stylesheet language (as is typical) was quite complex, with a steep learning curve, and plenty of Gotchas for the unwary. The prospect of replacing this with standardised XSLT/FO scripts accordingly had great appeal.

However, when XSL-FO hit the streets we realised that rather than solving the problem of specifying a rendering strategy, it had simply moved it. Instead of a proprietary stylesheet language, we now had a standard transformation language (XSLT) targetted at a standard pre-rendered document format (FO). However the user is now faced with two tasks:

- Designing an FO strategy to yield the desired output.
- Developing XSLT stylesheets to map the source document into the required FO format.

Neither task is trivial for even the simplest documents, while for complex requirements only specialists would have the detailed knowledge needed to develop the required stylesheets.

What we were aiming for was a system which would enable an ordinary user to develop quite complex stylesheets, while still allowing the expert the flexibility to meet sophisticated rendering requirements.

At this time we were also the Australian distributor for the SoftQuad (now Corel) XMetaL editor. One of the features of this software is a CSS design dialog, which enables the user to specify the appearance of tagged data with considerable flexibility. Of course CSS, primarily used for screen-based rendering, lacked many of the facilities necessary for publishable quality hard copy or PDF output.

We already had a rendering engine, plus a set of GUI "Wizards" which would generate simple stylesheets which could then be fine-tuned by direct editing. We therefore set out to convert these Wizards into full-fledged Management dialogs, which would completely protect the user from the underlying stylesheets. Along the way we added a number of facilities which added greater functionality while actually simplifying the design process for the user.

This presentation will describe some of these innovations in detail, while comparing aspects of TopLeaf mappings (eg. rule selection, property inheritance) with equivalent mechanisms in CSS and XSLT/FO.

An Overview of the TopLeaf System

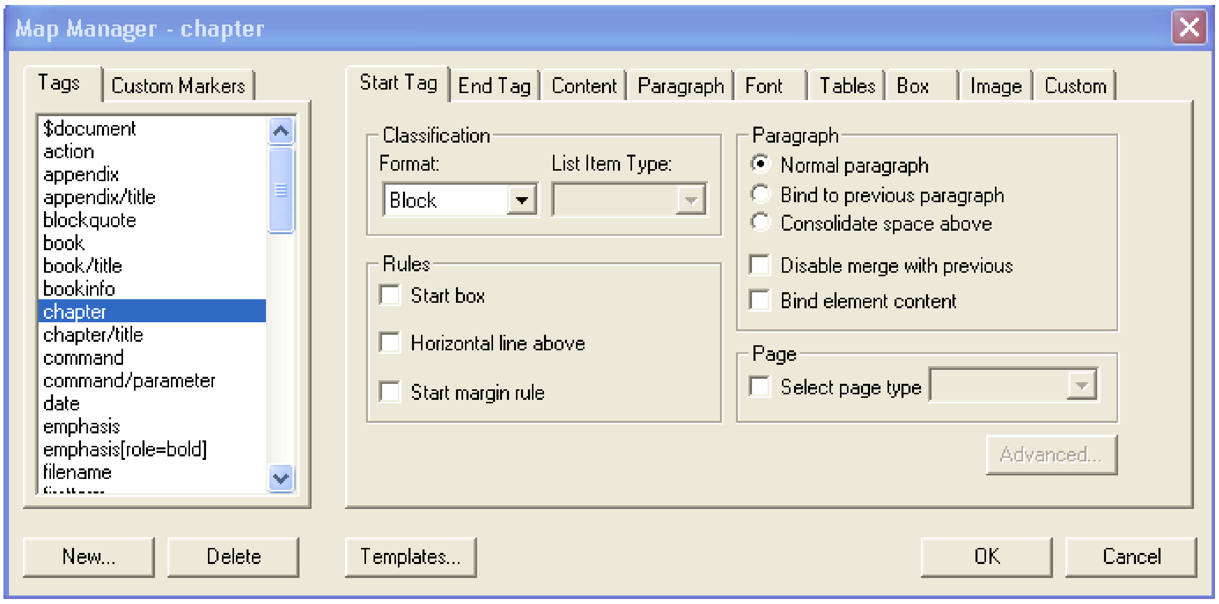
A TopLeaf user specifies document layout via four main dialogs (called Managers) as follows:

- *Layout Manager* — specifies page size, header/footer areas, columns and various job defaults.
- *Map Manager* — defines mappings triggered by tags in the source data.
- *Notes Manager* — specifies layout of footnotes, sidenotes and running heads (eg. continuation headings).
- *HeadFoot Manager* — specifies header and/or footer material set independently of the normal text columns.

The Layout Manager, while interesting in its own right, lies outside the scope of this presentation. The other three managers are broadly similar in function, as notes and header/footers are treated as a special type of tag called a *system marker*. These are handled in just the same way as any ordinary tag in the source document.

Each manager is responsible for maintaining a set of *mappings*. A mapping is set of actions triggered when a given marker (tag) is encountered. Each mapping can include the following classes of actions:

- declare an element to be block, inline, list-item etc.
- select fonts and effects
- specify table or box styles
- specify paragraph properties such as alignment, margins, vertical spacing, hyphenation, widow/orphan control etc.
- control paragraph binding, column changes, page breaks, and display of headers/footers
- display graphics
- store or suppress the content of an element, and optionally assign it to a footnote, running head, table of contents etc.
- generate (for PDF output) a link launch point or destination
- declare explicit pre- or post-content actions.



Format: Block

[✓ Normal paragraph]

As you can see, mappings are analogous to CSS rules and (more distantly) XSLT rules. A simple mapping just takes the name of the corresponding tag (eg. `title`). However mappings can be made more explicit by specifying:

- ancestry — `chapter/title, example//para`
- attribute value — `emph[style="bold"]`
- position within parent — `item!last`

or any combination of the above. As you can see the mapping names are reminiscent of XPath/XQL selectors. Implementing a fuller set of such selectors and using the standard XPath syntax is being considered for the next major release.

Unlike CSS, but like XSLT, only one mapping is applied to a particular element. If several mappings match an element then a set of precedence rules is used to determine which is the most explicit.

If no suitable mapping exists, a default system mapping `$implied` strips the outermost tags and processes the content.

Comparison with CSS and XSLT/FO

Due to its mixed heritage, TopLeaf shares some features in common with both of these other systems.

	CSS-2	TopLeaf	XSLT/FO
Pre-transform	No	Optional	Required
On the fly reorg	No	Moderate	Comprehensive
Headers, Footnotes etc.	No	Yes	Yes
Generated text	Minimal	Yes	Yes
Multiple rule handling	Cascade	Select	Select
Property inheritance	Yes	Yes	Yes

	CSS-2	TopLeaf	XSLT/FO
Real time rendition	Yes	No	No
File Access	Sequential	Sequential	DOM based

At first glance this table suggests that TopLeaf has more in common with XSLT/FO than CSS. In terms of output options and pagination that is true, but the user interface is far more like that of a simple CSS GUI because, generally speaking, no pre-transform is required. The mappings are applied directly to the source document, and any errors or warnings refer back to lines in that document.

Also like CSS TopLeaf accesses documents sequentially, rather than via the DOM. This has the advantages of speed and a much smaller memory footprint. In addition, since TopLeaf generates output directly, the connection between source and output can be more easily designed and implemented from a sequential viewpoint.

While the direct approach has definite advantages, it does require that the output more or less follows the structure of the source document. To offset this limitation, TopLeaf has three additional facilities:

- The ability to capture source data for re-use later in the output rendering.
- User defined Custom Markers (see following section).
- If this is still insufficient, a pre-transform can be applied using XSLT or similar tools.

In practice however pre-transforms are almost never needed since XML documents are either created directly by humans, or constructed by software (eg. database extraction). Human-centric documents are normally rendered more or less in their original form, while even machine generated documents (if destined to be rendered in human readable form) tend to be created with that output format in mind.

TopLeaf Innovations

There are several features which are, to our knowledge, unique to TopLeaf. These allow the user to leverage the simple tag mapping paradigms to control some of the more sophisticated requirements of page based (rather than screen based) rendering.

System Markers

We have already mentioned that TopLeaf treats notes, headers etc. as a special form of tag known as a system marker. This concept also applies to mapping (ie. rule) selection. For example if a section heading is repeated at the top of each page, then we may want it to set differently than in the original text. We may wish it to be all caps, and to ignore **`<emph>`** tags which would have produced font changes in the original heading.

To facilitate this TopLeaf generates such repeated headings in the context of one of the system markers **`$note`** or **`$headfoot`**. Thus we can have a standard **`emph`** mapping which triggers italic font, and a separate **`$headfoot//emph`** mapping which does not.

There is also a special **`$document`** mapping which specifies defaults for the document as a whole, and reimposes those defaults for material set outside the normal text columns.

Block Merging

Like CSS, TopLeaf mappings fall into categories including Block and Inline. While this is a simple and elegant model, there are times when the markup does not lend itself to such interpretations. Consider a table of contents marked up as:

```
<contents>
<heading>Introduction</heading><page>1</page>
<heading>General Principles</heading><page>24</page>
. . .
</contents>
```

While the **`contents`** mapping is clearly a block, what of **`heading`** and **`page`**. If we declare them as blocks, then they will be set as separate paragraphs. If inline, then the whole TOC will be one huge paragraph. The problem is that the markup is deficient in that there is no **`<line>..</line>`** wrapper around each heading/page pair.

The principle in TopLeaf is that any element which *either* begins or ends a paragraph is a block. TopLeaf provides an additional capacity, that any block mapping (in this case **`heading`**) can specify a *Merge with Following* option. This forces the content of the following block to flow on immediately with no paragraph boundary. In this way the correct output format can be achieved without having to restructure the input.

Custom Markers

As previously noted, TopLeaf defines its own internal "tags" as system markers. The associated mappings begin with a **`$`** to distinguish them from regular tag mappings (eg. **`$document, $runhead2, $headfoot`**).

However TopLeaf also allows users to define their own private custom mappings. Like system mappings, these begin with a special distinguishing character **`%`** (borrowed from parameter entities which are likewise user defined).

These mappings are invoked by *custom markers*, which appear as "elements" in the Pre- and Post-content material of any other mapping. Like normal tags they can have attributes, and have start, end, and empty forms. A simple example will illustrate their usefulness.

Say we want a heading to be repeated at the top of each subsequent page. In the mapping for the heading tag we select the option *Assign content to running head level 1*. The heading will now repeat on each new page until switched off or superseded by a new heading.

Now say we want to add — **continued** to each running head. Also simple. Just open up the **`$runhead1`** system mapping and type

```
&mdash; continued
```

However if we want an italic — ***continued*** things become a bit more difficult. We could set the **$runhead1** mapping to select italic font, but this would make the whole heading italic, not just the "continued". What we need is to be able to apply a separate mapping just to part of the string.

This is where custom markers come in. What we need to do is set the **`$runhead1`** Post-content to

```
&mdash; <Ital>continued</Ital>
```

We can then define a new custom mapping **`%Ital`** as a simple inline which selects italic font. Problem solved!

This is only a trivial example. Custom markers can be used to generate new material or effects in any form, and can even call other custom markers as part of their own content. They thus serve to augment the existing document structure to produce almost any desired output. And since they are mapped just like any ordinary tag, the user does not have to learn any new techniques to take full advantage of them.

Standards and Stylesheets

One obvious criticism which could be aimed at TopLeaf is: *Isn't this just another non-standard proprietary system?*

This is a more subtle question than it at first appears. While TopLeaf is of course a proprietary product, it does not actually compete directly with either XSLT or XSL-FO (and certainly not with CSS). You'll notice that I have not mentioned any stylesheet format here. And what actually constitutes a "stylesheet" in TopLeaf is a question to which I have no definitive answer. The user only sees the current GUI settings (which lie outside the realms of current standards) and the Pre- and Post-content material (which are standard well-formed XML fragments).

Indeed it would be quite possible to interpose a TopLeaf-ish GUI on top of a standard XSLT/FO system. And the details of the GUI are open for the world to see, there are no hidden proprietary secrets up our sleeves.

TopLeaf is in fact addressing a different, higher level problem space than current standards. Rather than specifying how to transform an XML document, or how to produce a definitive XML rendering model, it addresses the question:

> *How can we make the full potential of XML document rendering available to the non-specialist user without sacrificing features or quality.*

This sort of human-centric design is what distinguished (say) Lotus 1-2-3 from earlier much more specialised accounting aids. Does this mean that the latest TopLeaf is the forerunner of a new generation of rendering tools? I'd like to think so, but only time will tell.