Pushing FrameMaker: a case study

Steve Rickaby describes the construction of a complex book design and writing environment in structured FrameMaker.

As this is the last article in this long series about Adobe FrameMaker, I thought it would be interesting to look at a real-life case study. I have titled the article *Pushing FrameMaker* because it took the application (and the author!) to several limits, as I hope to show. It also provides a good example of what can be achieved with FrameMaker and, as the pilot once put it after a nasty landing at Gatwick in a thunderstorm, 'courage, skill and determination'. Plus a little help from your friends.

The brief

Thomson Education, as it was then, commissioned an attractive design for a series of university-level textbooks on Java from Design Deluxe of Bath (www.designdeluxe.com). Although I had little contact with the designers, whose work was completed before I became involved with the project, I assume that they would have worked in Quark or InDesign and would have had no knowledge of FrameMaker. Their deliverables were a set of page designs, as PDF files, and a full written typographical specification, which turned out to be immensely useful.

Meanwhile, the author team in Sweden had decided to work in structured FrameMaker as a route to selectively reusing the material. To this end they had created their own version of DocBook and had drafted parts of books using that as their Element Definition Document (EDD). The authors, supported by the publisher, wanted to be able to continue writing in an environment that gave them the closest possible simulacrum of the final book design.

A final requirement was to deliver the writing environment to the publishers in a documented and reusable form. (I use the term 'writing environment' here, as this was the authors' perspective, but strictly speaking this was a FrameMaker *structured application*, with a few extra bells and whistles.)

This article will make more sense if you look at the final design on Amazon using its 'look inside' feature at tinyurl.com/4745l9e, as I will be referring to aspects of the book's design throughout. If the TinyURL link doesn't work, the full Amazon URL for the book is given at the end of this article. Failing both, Google *Java Actually: A Comprehensive Primer in Java Programming*.

Technical challenges

The first and most obvious challenge was to implement a complex design in structured FrameMaker: specifically, to decide whether to specialise an existing EDD such as DocBook, or to create a new EDD 'from the ground up'. In the light of little relevant knowledge at the time, but some good advice, I decided on the latter route. This turned out to be the right choice, but for some quite complex reasons. I discussed this sort of decision in *First steps in structure: EDDs* in the Summer 2007 issue of *Communicator*.

The second challenge was to create a 'point and click' writing environment for the authors. This was to enable them to work with the complex elements that made up the design without having to involve themselves in the intricacies of structured FrameMaker.

Finally, there was a collection of tricky presentational features the designers had included that weren't directly implemented in FrameMaker, such as partially reversed-out headings (the section numbers, figure, program and table titles), graphical rules that do more than just go above or below a paragraph (see Figure 6 later in this article), complex table designs (see 'Best Practice', Figure 7), and the fore-edge tabs, which move down the page edge as the chapters and appendices increment. I described how these tabs were done in *Mastering master pages II* in the Autumn 2006 issue of *Communicator*, so I won't detail them again here: the key was to link the TabSpace font, in which the width of each 'character' is proportional to its ASCII value, to the chapter counter.

I will describe how these three challenges were met in the order I've listed them, although in reality they were tackled in the reverse order; we had first to ensure that all the design features were achievable, in case any had to be dropped and the design revised. In practice, very little fine-tuning of the design proved necessary.

Constructing the EDD

There was one further challenge I've not yet mentioned: this project was my introduction to structured FrameMaker. To some extent this naivety was an advantage, but in other areas, such as constructing list elements, I needed expert help. My first step, therefore, was to buy a training package on EDD design. As I'd been using unstructured FrameMaker for over a decade, structured FrameMaker, although challenging, came as something of a pleasant surprise.

Once I'd opted to create an EDD from scratch, the second decision was how to control formatting. In structured FrameMaker it is possible to include formatting instructions in the EDD, to refer out of the EDD to a document or template's paragraph and character tags, or to use a combination of both. To make the structured

application as easy to understand as possible, I decided to control only structure using the EDD, and to place all formatting in the FrameMaker templates, invoking the required paragraph and character tags from the element definitions. The resulting EDD contained about 115 elements, although most of these were 'under the hood', grouped into the 21 high-level structural elements exposed to the authors (see Figure 1).

The corresponding templates used some 120 paragraph and 26 character tags, all applied under the control of the element definitions in the EDD. The mapping of function to paragraph and character tags was more extensive than might normally be used for a FrameMaker template. This was partly to do with the relative complexity of the design, and partly to achieve a 'clean' mapping of elements to paragraph tags in the EDD with minimum reuse of function and/or style overrides. There were also a number of paragraph and character tags that existed solely to implement parts of the design functionality. I prefixed their names with a tilde '~', both to indicate that they should not be selected by authors and to move them to the bottom of the relevant palettes.

The ability to use container elements to group simpler elements proved to be vital when programming around some of the challenges posed by the book design: see *Reversed headings* and *Complex graphical rules* later in this article.

Point and click writing environment

This proved the easiest part of the project. Some years previously I had discovered the very useful range of FrameMaker plug-ins produced by SiliconPrairie Software (www.siliconprairiesoftware.com). One that I'd never made much use of was Auto-Text: this enables you to define a set of named text or graphical items, which then appear on their own menu and can be inserted into the current document by selecting them. The items are drawn from a (FrameMaker) configuration document that is read when the application starts up. To my surprise and delight, it turned out that Auto-Text could be used just as easily with structured elements as for unstructured content. The resulting configuration document is not, of course, a valid structured document, as it consists only of a sequence of unrelated elements, but that doesn't matter. Nor is the Auto-Text menu context-sensitive, but this turned out not to matter in practice either: the key feature was the ability to insert groups of elements that formed complex structures into a working structured document with a single mouse click (Figure 1).

Detailed design challenges

This section describes some of the detailed presentational challenges and how they were overcome.

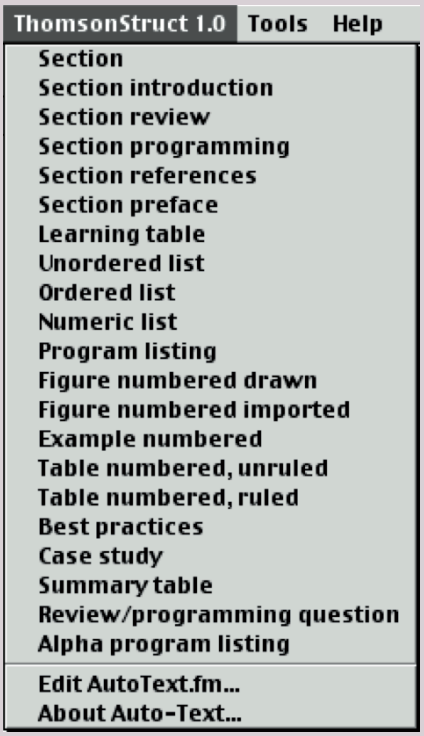

Figure 1. The Auto-Text menu, presenting high-level elements for insertion

Reversed headings

The recent release of Version 10 of FrameMaker now offers coloured backgrounds to text, but this wasn't available to us at the time. A key feature of the design was the use of *partial* white-on-colour headings: see the book's 'look inside' view on Amazon and Figures 4 and 5. Finding a way of creating these caused a great deal of head-scratching, and some ingenious suggestions from FrameMaker gurus such as Bernard Aschwanden of Bright Path Solutions and Hayden 'Document' Jones. The obvious method of using unruled one-row tables ('obvious', as FrameMaker allows table cells to be shaded) posed some nasty table-of-contents issues, and tables are cumbersome in structured FrameMaker. I tried placing the autonumbers in an inline anchored frame within the heading, but this gave problems with cross-references: both methods just required too much manual adjustment to get them to work acceptably.

It was Hayden Jones' idea of placing a graphic from a reference page 'behind' the text using negative leading (vertical spacing) that finally solved the problem. This method has the advantage of combining the autonumber and heading into one: the autonumber (only) is in white, with an opaque graphical 'backdrop'. The only disadvantage of this approach is that the size of the backdrop graphic is fixed and does not scale with the autonumber (and the page has to be manually refreshed if the heading is edited, an old FrameMaker bug). Overall, this seemed the best compromise. I 'solved' the problem of the width of the reference-page graphic not growing

Element (Container): TitleHead
General rule: <EMPTY>

Text format rules

1. **Count ancestors named:** Section
   If level is: 3
   Use paragraph format: ~NullCFlag
   Else, if level is: 2
   Use paragraph format: ~NullBFlag
   Else, if level is: 1
   Use paragraph format: ~NumberFlag
   Else
   1.1. **If context is:** Title < Chapter
   Use paragraph format: ~TitleOffset
2. **If context is:** Title < (SectionProgramming | SectionReview)
   Use paragraph format: ~NumberFlagEoC
3. **If context is:** Title < (SectionReferences | SectionPreface)
   Use paragraph format: ~NullBFlag

Figure 2. The element definition for the TitleHead element

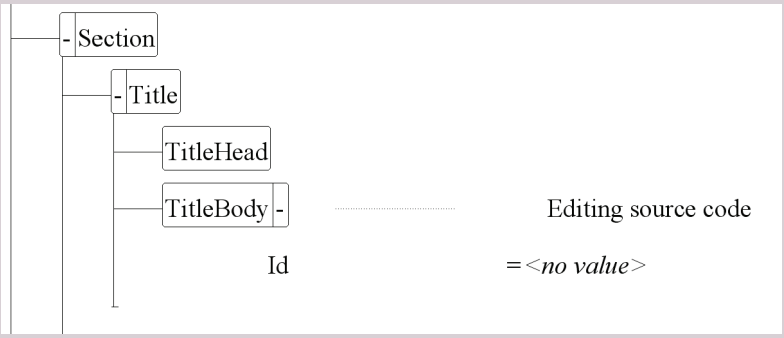

Figure 3. The container element Title, grouping TileHead and TitleBody to form the reversed-out heading

1.6 The components of a program

Figure 4. The resulting heading with reversed autonumber

FIGURE 1.1 Main activities in writing programs

Figure 5. The same technique used for a figure number

INTRODUCTION

This chapter illustrates some important programming concepts by way of an example. We will look at how to write, build and run programs. Later chapters will provide a more thorough explanation of the concepts introduced here.

Figure 6. The 'Introduction' section heading

BEST PRACTICE

Choose a good editor and spend a few hours learning its features. In the long run, this effort will pay off handsomely in terms of productivity.

Figure 7. The 'best practice' call-out, a FrameMaker table

with the width of the autonumber by using a separate paragraph tag with a wider graphic for numbers greater than 10 — one of the few manual overrides needed. I could then 'bolt' the reference-page graphic together with the title paragraph using a container element, as shown in Figure 3 for a **Title** element. This element is a container that includes an empty paragraph to hold the graphical backdrop, the **TitleHead** element, plus a paragraph to hold the title itself, the **TitleBody** element.

Figure 2 shows the element definition for **TitleHead**. The important thing here is the invocation of the paragraph tag **~NumberFlag** (highlighted) for the level 1 headings. This 2-point tag, positioned *before* the title paragraph, uses the paragraph 'frame below' feature to insert the background graphic from the reference page, and negative below-paragraph spacing to position it 'behind' the white text. This works in conjunction with the corresponding paragraph tag for the heading itself – applied by the **TitleBody** element – which uses the same negative vertical spacing above the title paragraph. Together these produce the effect shown in Figure 4. Exactly the same technique produced the other reversed headings, such as those for figures (Figure 5).

Complex graphical rules

The book's design called for numerous ruled objects; in all but one case it was possible to implement these using FrameMaker tables, for example the 'Best practice' table shown in Figure 7; these occurred frequently throughout the book. The EDD's definition of this table is too lengthy to include here, but its element structure is shown in Figure 8. The 'BEST PRACTICE' text is inserted using the paragraph tag's autonumber field, the tag itself being inserted using a context rule.

The original design also called for the many examples of program code – using text insets – to be enclosed in ruled frames. This was the only aspect of the design that proved to be impossible to implement cleanly in FrameMaker, the problem being code examples that flowed between pages. In the end we compromised by including rules above and below the code examples using container elements, removing the need for another table element.

The 'all but one case' of ruled objects referred to above was the (unique in this design) presentation of the title of the chapter introductions, as shown in Figure 6. Again, a lot of experimentation was involved in getting this to work: the eventual solution was essentially the same as that used for the reversed text in headings described in the preceding section. However, in this case the 2-point paragraph containing the graphic had to be positioned *after* the title paragraph to get it to display correctly.

Complex table designs

I've touched on one table above in the context of graphical line rules. Custom ruling and shading in tables allow a lot of design flexibility in creating complex objects in FrameMaker. Our book's original design included other call-out objects, such as the chapter summary table shown in Figure 9, as well as a 'case study' table, but these were not used in the final books as the authors didn't need them. They proved to be relatively easy to implement as FrameMaker tables.

The blue table 'border' shown in Figure 9 is actually part of a tinted page background. Each chapter concluded with review material and exercises that were distinguished from the remainder of the material using tinted pages. I implemented these using custom master pages, applied automatically by the StructMasterPageMaps table. This was triggered by the review and exercise section header paragraph tags, although the corresponding elements would have worked just as well.

Documenting the design

The documentation for the writing environment, its associated EDD and FrameMaker template (the components of the structured application) provided detailed instructions for using the application and listed the purpose of each element, paragraph tag and character tag. I grouped it into four major sections: using the templates, a description of the 'presentation layer' (tags), a description of the 'structure layer' (elements), and detailed notes on special features such as the automatic fore-edge tabs. It ran to 44 pages. This also provided a very useful *aide memoire* for me, enabling me to pick up the details quickly after breaks in work. I also provided wrapping tables for the books' generated files (table of contents, list of figures, list of programs, index) so that they could be structured if required.

The only deliverable not yet mentioned was an unstructured version of the application, provided so that the design could be used in books that didn't use structured FrameMaker. This used the FrameMaker template alone, making extensive use of the Paragraph Designer's 'next para tag' feature to ensure that the composite paragraph sequences that were grouped using container elements in the structured application would be inserted in the correct sequence in the unstructured version.

In conclusion

I felt that this would be an interesting case history to end this series, as there were few, if any, of FrameMaker's features that it didn't use, so the title *Pushing FrameMaker* is I hope justified. The benefit for me was that it provided a 'baptism by fire' to structured FrameMaker.

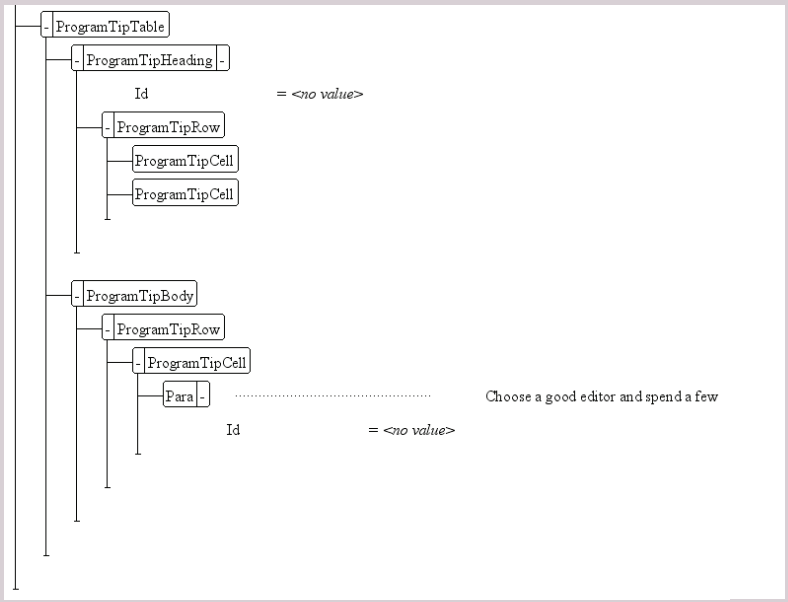

Figure 8. The elements that compose 'best practice' table

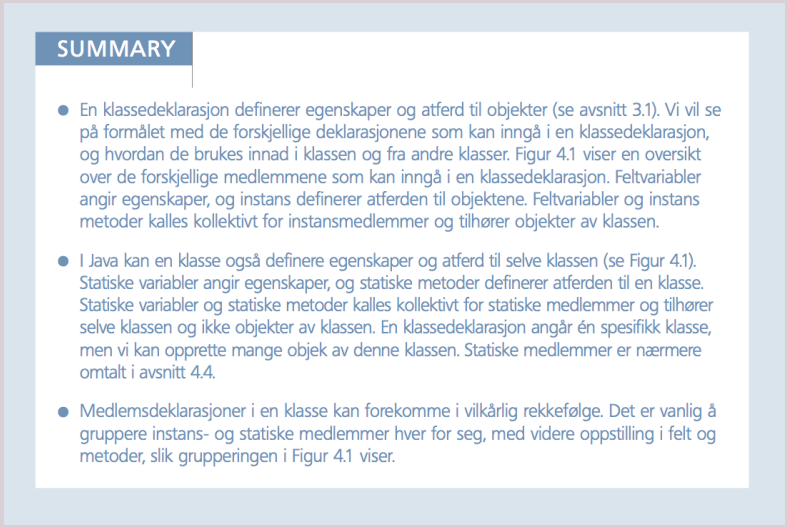

Figure 9. The chapter summary table

And finally…

I took over this series, so capably started by Jane Dards, in the Spring 2006 issue of *Communicator* and have been involved with it since then. I would like to thank our many other enthusiastic contributors: Terry Smith, Russ Ward, Tammy van Boening, Lynne A. Price and Andy Lewis, and of course our ever-patient editors, Marian Newell and Katherine Judge: it's been a pleasure working with you all. C

Steve Rickaby BSc MISTC has been a freelance technical author and editor for over 20 years, and has used FrameMaker for most of that time.
E: srickaby@wordmongers.com
W: www.wordmongers.com

Reference

To see the final book from the design discussed in this case study, go to tinyurl.com/4745l9e. If that does not work, the full URL is www.amazon.com/Java-Actually-Comprehensive-Primer-Programming/dp/1844809331/ref=sr_1_8?ie=UTF8&s=books&qid=1295539503&sr=1-8.