Formatting with FrameMaker + SGML's EDD

Daniel K. Schneider, TECFA, University of Geneva
http://tecfa.unige.ch/guides/xml/frame-sgml/
Version 0.5 - may 2001
Version 0.3 first useful draft. Version 0.4: added pointer to sdocbook. Version 0.5 said something about attributes
Thanx to Jan den Hartog (objective i), Kevin Lossner (Easy Software AG), Marcus Carr (Allette Systems) , Ceryle Gaehl (Australian Medicines Handbook) for helpful comments.

Summary

This is a very short introduction to formatting with Framemaker + SGML's EDD. I use ***version 6*** for Solaris and Windows. You also must read the developer's manual (in particular chapters 7, parts of 8, 9) which you can find as a 550 page PDF file in some directory in your distribution. In addition, is strongly suggested that you download all the files from http://tecfa.unige.ch/guides/xml/frame-sgml/ including the ones in the Stepbystep and sdocbook directories and that you look at the source code.

The best example I have at the date of this writing (check yourself) is an incomplete EDD for Norman Walsh's Simplified Docbook XML DTD: http://tecfa.unige.ch/guides/xml/frame-sgml/sdocbook/

This is a first rough draft (including the English). I don't know when I will have time to improve it.

Introduction

EDD formatting is not meant for the faint hearted. In typical SGML ideology a "End User" is supposed to write text and not to define EDD rules. This class distinction certainly doesn't apply to the place I work. I think that documentation is too difficult reading (badly organized in some areas). Well the system is very complex and its not easy for technical writers. Anyhow, I think that EDD is a fine piece of work and that once you master it you can very efficiently maintain and modify layouts. The only trouble is bad XML support (you suffer before you understand how to import and export, no XSL, old XLink and more). Personally, I only know the basics and just don't have time to do more ...

Prerequisites

This document assumes that you already have an EDD with general structure rules defined. The target audience are people who import an XML DTD, a procedure I outlined in the companion document quick-fm-xml-guide.pdf. I also assume that you know how to use standard Framemaker, in particular the Paragraph Designer (although strictly speaking you don't need to know much about standard Frame, it is just general concepts for formatting plus a few FrameMaker specific things like paragraph numbering).

Major Steps

Step 1:Try to grasp the logic of EDD formatting

FrameMaker+SGML is very different from FrameMaker

FrameMaker (standard) is based on paragraph definitions. While you can easily map your old paragraph definitions to SGML elements in order to quickly import legacy text, it's not the recommended way to do it. EDD's do all (and more) that SGML or XML DTDs can do, what XSL/FO does, what paragraph definitions in FrameMaker do. It's structure + style.

(1)The Foundation: a simple Body TAG

All EDD rules are based on the predefined Body TAG (which is a regular FrameMaker paragraph definition that you can modify btw). By default all EDD rules inherit definitions from this Tag. As you can see later, I suggest defining a very small number of base paragraph tags and build upon them.

(2)Define changes

Instead of let's say fixing a header as 18pt, you would make +6pt in relation to some default header.

(3)Think context

EDD allows you to define rules for different contexts. That basically means that you can work with a much smaller tag set than in typical Framemaker. E.g. you can define rules for first and last elements in a list, for elements (e.g. titles) that have exactly or approximately such and such a parent, etc.

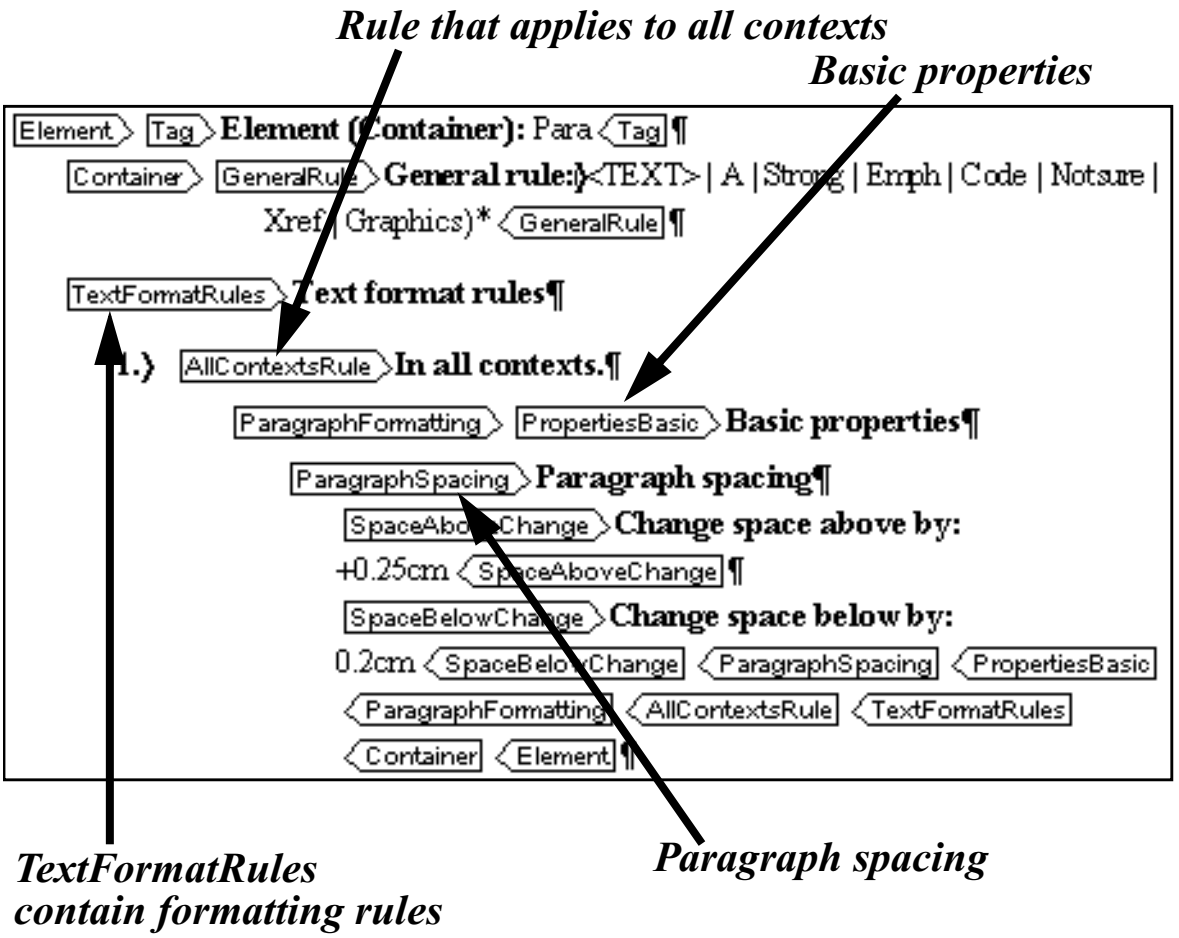
(4)Be aware of special cases

There are many special cases you need to handle, some easy and some harder (at least for me at present). E.g. you have

- Inlined elements (i.e. elements that do no start a new FM paragraph, like a "strong" or "code" tag
- Cross-references
- Tables
- Generated things
- Graphics (e.g. anchored frames)

Step 2:Understand the anatomy of a simple EDD rule

Let's have a look an example. Note that you can you always should make visible the EDD element tags when working on an EDD file (Menu: View->Element Boundaries (as



tags) Alternativly you can work with the Structure View Window.

The organization of a simple EDD definition for a typical element is the following:

```
Element
  Tag
  Container (many others exist, e.g. graphics or crossref)
     General Rule  (corresponds to an ELEMENT definition)
       Text Format Rules
         Context Rule
           .. Formatting nested down to many levels ...
```

The organization of the Formatting Rules corresponds roughly to the organization of FrameMaker's Paragraph Designer (Menu: Format->Paragraphs->Paragraph Designer). You can look at this tool, however do not use it (except for the few base paragraphs from which you will inherit). Also, remember that you always should use "change" something.

Step 3:Identify the elements that need formatting

Not all of your elements need formatting, i.e. the ones that just hold child elements, usually don't need any (unless you want to play with spacing, boxes, background etc., but that's more advanced stuff not covered here.). For starters you roughly may have:

- Paragraphs
- Lists and items

- Titles
- Listings (i.e. code)
- Inlined Text markup
- Things related to tables

For each of these you need to learn some more and some different EDD formatting. For each group you can also try to figure out what formatting attributes they have in common (or by default)

Since as we explained before EDD rules are based by default on a paragraph tag called "Body" you should think about these questions:

- Do I want to inherit from one single tag only? What other basic elements do I need? But remember: You do NOT need many as you will see later
- What is the base font, left indentation, and in between paragraph spacing?

Well, this step wasn't really important, but a tiny little bit of planning will not hurt. We are ready to start now!

Step 4:Create EDD and template files

This is needed, if you haven't done so already of course. The EDD file defines the structure and the formatting for a whole class of documents. A template file is simply an empty FrameMaker file that you use as a starting point for a new document (It simply acts as template because it sits in some special directory on your machine). Basically, there are 2 situations:

- You imported a DTD into Framemaker. In this case, you certainly will have a EDD file. To start working on content (or a Template file), all you need to do is to open a blank page and import the EDD Element Definitions. See the separate document "quick-fm-xml-guide.pdf" for this topic !
- You start from scratch. In this case you first must first define an EDD file (Menu: File->Developer Tools->New EDD. Then you must define the logical structure of a document, a procedure which I will not explain here (because this document is for folks who know XML and tend to start at least from some DTD, even if they work with an incompatible EDD later on)

Step 5:Create a few basic standard Frame paragraphs

(1)Do this in separate file!

Create a new FrameMaker file to hold paragraph definitions. I'd call it something like <DTD/EDD name>-paras.fm

If you have a template file, import the styles from there if you want (so you can import them back later on!).

Anyhow, do NOT make these definitions in your text or template files. Here is why: Many elements will use the same paragraph tag, but they all look different. So when you change a definition in the Template or text file you will never really know what the base configuration of your paragraphs was. You will have to painfully reset the paragraph definition to what you forgot it was. The good News is that when you import paragraphs

from your styles files and ensuite the EDD definitions you can override all local definitions in your Template file or elsewhere (means that recovery is easy if you stick to my plan).

(2)Define a few basic standard FrameMaker formats

For starters I suggest these (or less!):

- Body
- Title
- Code

Make them look nice. For all of these, do not think in terms of standard FrameMaker or Word paragraph definitions. Think of something from which element formatting rules will inherit! That basically means use standard indentations, use the smallest font your smallest title will use and so on ... If you don't understand this idea, read the next steps and come back.

Warning: Do not rely on default Paragraph Names if you work in a multi-language organization. Maybe the default paragraph for Titles is "Titre" or whatever. This is another reason why you should start by defining at least one "base paragraph". (Later on, to be safe you have to tell EACH element in your DTD to rely on one of YOUR paragraph definitions. I did not do this in my little Stepbystep EDD (but should so).

Step 6:Define rules for standard paragraphs

For testing purposes, I suggest that you start editing some text in your template file (you can always empty it later). Then as soon as you make a modification in your EDD file go and import (Menu: File->Import->Element Definitions)

The basic tasks are (logically) the same as in standard FrameMaker. You have to specify font size, spacing above and below, numbering for starters. BUT, use ***changes*** to define fonts and spacing. This will make things more easily to maintain. Base things, like standard font-family, standard size, language, etc. should ***not*** be defined here but in "base paragraphs" as explained in step 5 [p. 4].

Here is an example I use for the overall title of a document (not considered as a "Title" heh):

```
Element (Container): Doctitle
 General rule: <TEXT>
 Text format rules
    Element paragraph format: Title
    In all contexts.
       Default font properties
       Change size by: +8pt
       Basic properties
       Paragraph spacing
         Change space above by: 1cm
         Change space below by: 1cm
```

Note that this definition inherits properties from a paragraph called Title (read everything again if you don't understand this). The rest simply adds pt's and spacing. To get something like this (provided that you have a Doctitle element defined) you must:

- Find the Element in your EDD file (use "find" if your EDD is large)
- Insert your Mouse after GeneralRule or AttributeList if you have one. (remember to turn on View->Element Boundaries as Tags) and see what the Elements menu offers (about 10 elements)
- Insert "TextFormatRules"
- Insert "ElementPgfFormatTag" if you wish. This will allow to define the base paragraph (other than Body) from which you want to inherit
- Insert "AllContextsRule" for simple unique elements (the same element does not appear at different levels/places needing some special formatting)
- And so on .... (sorry this gets boring)

Note: If you find that this thing is totally mysterious, just look some other EDD file, e.g. the one used for this document. You may even export it to xml, have Emacs indent and color it and study it as an XML structure :)

Step 7:Simple text range elements

Inlined elements are called "Text Range Elements" in F+S. Probably "inline" is not the correct word to use. I am talking about elements that are used for special formatting inside a same paragraph (in typographical terms). E.g something like <body> This is <emph>super</emph> cool </body>

The important bit is to insert "***TextRangeFormatting***" in the beginning of a Context rule. You must realize that this rule is of a different sort, albeit still a Container. Look at the source of this document and the Stepbystep-edd.fm file if it's not clear. Basically you want to change a simple thing, like font family, angle or weight.

```
Element (Container): Emph
General rule:<TEXT>
  Text format rules
     In all contexts.
       Text range.
         Font properties
           Angle: Italic
           Weight: Bold
```

Step 8:Simple title rules

If you look at the DTD used for this document (Stepbystep.dtd) or the EDD if you prefer, you notice that the Element Title appears in several contexts. Each Step, each Substep, each Goal, etc. can have a title. So this requires different formatting.

Look at the source of the EDD for one suggestion. I am getting lazy here :). Basically you need to fix spacing, indentation, font size and numbering and repeat this several times.

You also should label each context rule, because you need this to produce fancy headers (on top of printed pages) and PDF bookmarks (!). E.g. if you have something like:

```
If context is: Step
Context label: StepTitle
```

you can then configure Running Headers of the Master page to display as header the content of highest level Title element it can find on the page:

```
<$elemtext[Title(StepTitle),Title(OtherTitle),Title(DefaultTitle)]>
```

Finally, put the AllContextsRule (In all contexts) in front of ContextRules else it will override all Context Rules *(I am not sure about this anymore)*

Step 9:Special Elements

You can handle Xrefs, Tables and Anchored Frames somewhat. Don't have time to document this just now (3/01). See the sdocbook-edd for an example.

Step 10:Printing Attributes and attributes for printing

You can print attributes within prefix and suffix rules (maybe in other places too). If you just need to print a optional attribute value (impled) but don't need any decorations like parenthesis just do the following:

```
In all contexts.
Prefix: <$attribute[attr_name]>
```

If you want to surround attributes with something, e.g. *** *** to indicate that some element has special status, you must test if the attribute has a value. I don't know how to do this properly, but testing if the attribute value is equals or bigger than the first value in a list of values works (ugly). In the example below we test if the current element (e.g. Title) has an attribute Status with value bigger than rough-draft" in the list of choices. If true we print:

```
Prefix rules
If context is: [Status>="rough-draft"]
Prefix:  +++ <$attribute[Status]> +++
```

FYI here is the DTD definition of this Status attribute (a "choice" type in the EDD)

```
<!ATTLIST Title Status (rough-draft | draft | bad | ok | good) #IMPLIED
>
```

You can also use attributes to conditionnally print out text. A typical example are typed lists (bullets vs. numbers) like in the following case. Note that we do not test attribute values of the current element but of the parent element called List

```
Text format rules
1.If context is:List [Type ="Bullet"]
  Numbering properties
      Autonumber format:•
      Character format:bulletsymbol
  Else,if context is:List [Type ="Numbered"]
      Numbering properties
         Autonumber format:<n+>
```

More information about testing attributes (including occurencies in more complicated contexts are on page 36ff, 123ff, 149ff of the lovely developper's manual.

Step 11:Exporting to XML

(1)Export

```
File-Save as ..
```

Select XML as format

It should work, i.e. you will have a file with the XML contents. The XML file has a style sheet declaration that points also to the generated CSS file (not XSL/FO!)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml:stylesheet href="Stepbystep.css" type="text/css"?>
<Stepbystep><Doctitle>XML with FrameMaker + SGML Quick Guide</Doctitle>
....
```

(2)Repair the CSS declaration in the XML file

Just a note of warning about the CSS: You text will look rather ugly with IE explorer and Mozilla. The generated CSS is minimalistic (even if you have nice EDD formatting rules). You can CSS with Webworks (a go-along product) but I don't have it installed (yet).

In addition if you try to display it with Mozilla (M18 in my case) you have to edit the declaration. It has wrong syntax, but probably works with IE explorer (don't know about recent versions). Yet another lost hour because of F+S strangeness. (see http://www.w3.org/TR/xml-stylesheet/). Anyhow, replace:

```
<?xml:stylesheet href="Stepbystep.css" type="text/css"?>
by
<?xml-stylesheet href="Stepbystep.css" type="text/css"?>
```

Step 12:Do some server-side stuff with it

This is not part of this document. If you want to make your XML document available to the masses, you'd have to translate it server-side to HTML. Try Cocoon and learn XSLT.

Open questions

- I have no clue if and how attribute values can be displayed. This is a pain because I got several DTDs where information sits in attributes.
- Configuration of the HTML and PDF translators (rather easy) should be a bit explained here.
- Tables and other fancy stuff is missing.