Maintaining Related EDDs

Maintaining Related EDDs

by Tassos Anastasiou and Lynne A. Price, Ph.D.

Tassos Anastasiou is an independent consultant specializing in Frame Developer's Kit (FDK) applications. He is a former employee of Frame Technology Corporation.

Lynne Price is president of Text Structure Consulting, Inc., a consulting company that specializes in FrameMaker+SGML application development and training. Prior to founding Text Structure Consulting in 1996, Lynne was a software engineer at Adobe, which she joined through Frame Technology as one of the original developers on the project that eventually became FrameMaker+SGML. Lynne has been active in the SGML community since 1985. She participates in both US and international SGML standards committees. Lynne's interest in structured documentation began in graduate school. She completed a Ph.D. in Computer Sciences at the University of Wisconsin-Madison in 1978. Her dissertation was titled *Representing Text Structure for Automatic Processing*.

Problem statement

Hewlett-Packard uses FrameMaker+SGML to prepare user documentation and training materials. The FrameMaker+SGML application is based on the DocBook 3.0 DTD. Printed user documentation can be prepared in any of 5 different page layouts: there is an indented style that uses side heads and a full-page, non-indented style, both available in each of two page sizes as well as an additional style for reference material typical of unix man pages. Material displayed online and training material provide additional variations of the basic book design. While printed and online documentation all use the same element structure, training material uses a few elements and attributes that are not permitted in user documentation.

The EDD for this application is close to 200 pages long. It is not practical to maintain a separate EDD for each variation. To reduce the amount of editing and bookkeeping needed to maintain various templates, and to ensure that a change that applies to all styles need be specified only once, Hewlett-Packard maintains a single EDD. This paper describes, with examples, how the necessary variability is supported in this single file. The methods used are applicable to other collections of related EDDs.

In brief, three techniques are used:

- User variables define repeated strings that may change in different styles. The variable definitions are maintained in simple style-dependent variable definition files and imported into the EDD as necessary as one step in preparing a style-specific version of the EDD. For example, one file defines variables used in all templates in the larger page sizes, and another defines variables used for smaller page sizes. Similarly, one defines variables used in indented styles and another in non-indented styles. For the large, indented style, variable definitions are imported from the file with definitions for large pages and from the file with definitions for indented styles. Rather than requiring the EDD developer to import all the variable definitions used for each style, Hewlett-Packard performs this processing with an FDK plug-in.
- Since variables cannot contain structured material, conditional text is used for structured segments of the EDD that are needed in some styles but not others. The correspondence of condition tags to styles is not one-to-one; as with the variable definition files, there are condition tags for groups of related styles. For example, there are condition tags for training, for all documentation styles, for indented styles, and so on.
- Text insets are used for fragments of the EDD that appear multiple times. for example, since the same formatting is used for cautions and notes, the specifications are made in a text inset that is included in the definitions of both the caution and the note element. Use of text insets in an EDD requires some planning so that FrameMaker+SGML will be able to process the element definitions the EDD contains. Lynne Price described an approach using SGML text insets in “Using FrameMaker+SGML to Build FrameMaker+SGML Applications” at the 1999 FrameUsers Conference. Hewlett-Packard has chosen instead to use FrameMaker text insets. This alternative avoids the need of maintaining parallel SGML And FrameMaker+SGML versions of each reusable fragment. However, it does

require minor changes to the metatemplate (the template used for EDDs) and an FDK plug-in to preprocess the EDD before its element definitions can be imported into a document. This paper includes complete descriptions of the metatemplate changes and the functionality of this new FDK plug-in.

These techniques address two goals: they allow style-specific information to be switched in and out and they support reusable fragments within the EDD. Reusable fragments provide a single definition that can be edited once to change multiple occurrences of a fragment throughout the EDD and also eliminate the possibility of typing errors that occur in rekeying. Variables support both the use of style-specific information and provide reusable fragments, conditional text is style-specific but does not by itself provide reusability, and text insets are reusable but not style-specific (unless they contain conditional text or variables).

Variables

The master EDD uses variables for property values such as dimensions and font family names that are specified as simple strings and for variable parts of general rules. For example, the element type called `Chapter` in user documentation is changed to `Topic` for training while `Part` is changed to `Module`. An example of how this variability appears in the master EDD is shown in Figure 1, which shows the user-documentation form of a general rule. Strikethrough text indicates variables:

General rule: FrontCover?, Title?, BookInfo?, ToC?, LoT*, (Glossary | Preface)*, (((~~Chapter~~)+, Reference*) | ~~Part~~+ | Reference+), (Appendix)*, (Glossary)*, (Index)*, LoT*, ToC?, BackCover?

Figure 1 Using variables in the master EDD

Within the EDD, a character format called `variable` turns on the strikethrough character property. All variables are defined to use this format. There is a variable called `Chapter`, for instance, that for user documentation is defined to be:

```
<variable>Chapter
```

but for training is defined to be:

```
<variable>Topic
```

Since FrameMaker+SGML ignores formatting when it imports element definitions from an EDD, neither use of variables nor the character format change affects the way the EDD is processed. However, EDD developers find the visual clue that a variable is present helpful when they edit material such as the general rule shown above.

Variable definitions are maintained in separate files with the extension `.var`. The body page of each `.var` file lists the variables defined in its variable catalog. For easy reference, the entries are listed alphabetically. For example, Figure 2 shows entries in `training.var`:

This file defines variables used to specify the element tags for training. In particular:

- AdmonitionIndent is ~~1.125"~~.
- AutonumFormat is ~~AutoNumSmall~~.
- Chapter is ~~Topic~~.
- ChapterTitleTab is ~~36~~.
- FigureTitleIndent is ~~90~~.
- ListMarkIndent is ~~0.25"~~.
- Part is ~~Module~~.
- PartIntro is ~~Introduction~~.

Figure 2 A variable definition file

Of course, the text in a variable definition document is entirely comments for the benefit of the developer. The document's functionality is contained entirely in its variable catalog. Preparing a particular variation of the EDD may involve importing variable definitions from several files. The small, indented style of user documentation, for instance, needs variables from `small.var`, `indented.var`, and `documentation.var`.

Conditional text

Since variables cannot contain elements, they cannot handle all the variations that may be needed in related EDDs. Conditional text provides additional flexibility. The master EDD uses different condition tags for material that is specific to a single EDD variation or to a group of related variations. There is a condition for all small templates, for instance, and one for training material. As shown in Figure 3, conditional text can be used to indicate that part of a string pertains to only some conditions. Here, for instance, the general rule indicates that `PartIntro` is permitted in user documentation but not training material. (While the examples shown in this paper do not use color, the actual EDD uses color in condition indicators to help the developer visualize the variants relevant to any part of the EDD.)

General rule: (DocInfo?, Title), PartIntro?, (RefEntry)+

Figure 3 A conditional string

Conditional text is also used for elements that may appear in some variant EDDs but not others. Figure 4 shows an attribute definition list in which the first attribute type is defined for all variations of the EDD, but the last two are conditional.

Attribute list

1. **Name:** Id **Unique ID**
   Optional
2. **Name:** PageBreak **Choice**
   Optional
   Choices: PageBreak, NoPageBreak
   Default: PageBreak
3. **Name:** ViewTopicTitle **Choice**
   Optional
   Choices: Display, Hide
   Default: Display

Figure 4 Complex conditional structures

Conditional element hierarchies can contain variables. Figure 5 shows specification of paragraph indent properties. In fullpage styles (indicated by overlined text), the first indent is set to 0, while in indented styles (indicated by grayed text) the left indent is moved by a variable amount.

Indents

First indent: 0 pt
Move left indent by: ~~66 pt~~

Figure 5 Combining conditional text with variables

When editing structured conditional documents such as the master EDD, developers have learned to be careful when selecting elements for assignment of condition tags. If a condition tag is assigned to a paragraph element but not the following end-of-paragraph character, hiding the condition can leave a stray end-of-paragraph that may cause the result to be an invalid document. Also, editing with all conditions showing prevents many editing mistakes.

Text insets

Variables provide two advantages in the master EDD. In addition to customizing the EDD through changing variable definitions, variables are reusable—changing a definition affects all occurrences of the variable in the EDD. Conditional text provides customizability without reusability. A third technique is needed for reusability of element structures. Text insets are one possibility. The master EDD uses text insets to its own reference pages for reusable elements or groups of elements.

Some changes to the meta-EDD (the EDD for EDDs) are necessary to support this ability. First, each text inset must be a well-formed hierarchy with a single highest-level element. So that text insets can be used for sequences of parallel elements (more than one but not all the rules within an element's text format rules, say, or

several adjacent attributes definitions but not an entire attribute definition list), a new element called `TextInset` was defined. `TextInset` is a container defined in the meta-EDD with a general rule of `<ANY>`. It is valid at the highest-level and thus in a valid master EDD can be used as the highest-level element of text insets. Figure 6 shows the structure of such a text inset. Here, the intent is that the `PropertiesFont` and `PropertiesNumbering` can be used as often as needed as all or part of the content of `ParagraphProperties` elements in the EDD.

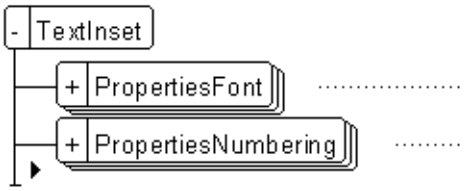

Figure 6 The structure of a text inset

However, a text inset to this `TextInset` element inserts the `TextInset` element as well as its children. Therefore, the metatemplate was modified to allow `TextInset` elements to occur anywhere by defining `TextInset` to be an inclusion on the EDD's own highest-level element, `ElementCatalog`. Permitting `TextInset` anywhere causes two additional problems. The content of a `TextInset` may be intended for required content of its parent. For example, a `TextInset` that consists of a `GeneralRule` will be imported at a point where a `GeneralRule` is required and will substitute for that required `GeneralRule`. Thus, the metatemplate must allow the `TextInset` to substitute for the `GeneralRule`. In addition to allowing `TextInset` elements anywhere, the text-inset metatemplate makes all elements optional, even those that are required in the original metatemplate.

The other problem with use of the `TextInset` element is that the Import Element Definitions command cannot process an EDD that contains `TextInset` elements. Thus, before any of the variant EDDs are imported into a document, the `TextInset` elements must be unwrapped.

Incidentally, the format rule for the `TextInset` elements uses a distinct character format so that the developer can recognize them as he reads the EDD.

An additional change to the meta-EDD was made for the convenience of the developer. While the original FrameMaker+SGML metatemplate allows a `Comments` element to appear at the beginning or end of each element definition, it does not allow other `Comments` elements. Complicated element definitions can extend for many pages and involve numerous format rules and subrules. It is very helpful for the reader of an EDD to see comments explaining such components of a single element definition. Since this project required its own metatemplate, an additional change was to allow `Comments` throughout an element definition. Just as the `TextInset` element must be unwrapped before the element definitions in the EDD can be imported into a template, so the additional comments elements must be deleted.

Processing the conditional EDD

The different EDD variants could be extracted manually from the master EDD by importing appropriate variable definition files, showing and hiding relevant conditions, unwrapping all `TextInset` elements, and deleting all `Comments` elements through a global Find/Change command. Each variant could then be manually imported into the associated templates. However, such manual processing is not only tedious, but potentially error-prone unless the user keeps careful notes on which of the various steps have been performed. It is much easier to use an FDK client to perform all of these steps. The client developed for this project takes as input the master EDD and a table that specifies how to process it. Such a table is shown below

Style	Variable Definitions	Condition Tags	Templates
`BigFull`	`Big.var` `Full.var`	`Full`	`BgFlChap.fm` `BgFlGloss.fm`
`BigIndented`	`Big.var` `Indented.var`	`Indented`	`BgInChap.fm` `BgInGloss.fm`

`SmallFull`	`Small.var`
	`Full.var`
		`Full`	`SmFlChap.fm`
			`SmFlGloss.fm`
			`ReleaseNote.fm`

`SmallIndented`	`Small.var` `Indented.var`	`Indented`	`SmInChap.fm` `SmInGloss.fm`
`Training`	`Training.var` `Big.var`	`Full` `Training`	`TrngMan.fm`

The FDK client unwraps the `TextInset` elements in the master EDD and deletes all `Comments` elements (those that would be valid under the original metatemplate as well as those that would not). For each row of the table, it performs the following steps:

- Imports variable definitions from the files listed in the second column.
- Shows conditions listed in the third column and hides all other conditions.
- Imports element definitions from the original metatemplate and validates the document to ensure that use of the `TextInset` element did not obscure invalid structures.
- Saves the result under the name specified in the first column, appending the `.edd` extension. (Thus, the variant EDDs from this example would be called `BigFull.edd`. `BigIndented.edd`, and so on.) While saving these intermediate files is not necessary, their presence enables the developer to debug and test each variation separately
- Imports the saved EDD into all templates listed in the last column.

Conclusions

The work described in this paper is an example of the kind of tool that can be developed by treating an EDD as a structured FrameMaker+SGML document and using some of FrameMaker's features to process it. The effort is close to complete. The FDK clients have been written and a master EDD created. While the EDD has not been fully tested, indications are that this is an effective tool. Each developer who edits the EDD must fully understand all three of the techniques described above. This training enables maintenance of a much more consistent and manageable set of variations than would be possible with separate EDDs.