

First steps in structure: an overview

Many experienced FrameMaker users may have never used structured documents. **Steve Rickaby** attempts a beginner's guide.

This article and its successors are intended to provide an easy introduction to structured working in FrameMaker for readers who have not used structured documents before.

FrameMaker users who have upgraded beyond Version 6 will have noticed that it now supports both unstructured and structured working. A version of FrameMaker that supported structured documentation has in fact been available since FrameBuilder, which was an alternative to FrameMaker 3.0 in the early 1990s. With its acquisition of FrameMaker in 1995, Adobe introduced a separate FrameMaker+SGML product. FrameMaker 7.0 saw the fusion of 'plain' FrameMaker and FrameMaker+SGML into one application.

Here we will introduce the idea of structured working and describe some of its advantages, while later articles will delve into the mechanics of setting up a simple structure definition.

document structure. SGML and HTML are well-known examples of languages for describing document structure (although SGML is, strictly speaking, a *metalanguage*).

FrameMaker users are familiar with the use of paragraph and character tags to define and control the appearance of text objects. There is a one-to-one correspondence between a paragraph or a character sequence and the tag applied to it. For example, a chapter document may have a chapter title, sections with headings to which a **Heading** tag is applied, body paragraphs to which a **Body** tag is applied, list paragraphs to which a **Bulleted** tag has been applied, and so on.

However, beyond its physical placement, there is no concept of the heading to which a specific body paragraph 'belongs'. Equally, there is nothing to stop an author following one **Heading** with another: the only indication of the document's structure is the physical placement of its component objects.

Enter structured documents: in a structured document, a separate syntax describes the document's structure independently of its presentation. For the simple example above, a sequence of heading and body paragraphs might be *wrapped* in a **Section**, and sections in a **Chapter**.

FrameMaker uses the term *element* to describe the objects that represent structure. It's helpful to think of elements as a separate level of document organisation from the language of tags, table styles, markers and so on that you already know (although these still work as before). This separate level of organisation is hierarchical: sections contain a title and body paragraphs, and are themselves contained in chapters — or whatever document structure suits your purpose.

Figure 1 illustrates this: in the structured version, the section title, body and list elements 'know' that they are part of a section. We'll see how this is achieved later.

When thinking about elements, it's helpful to remember that an element may exist only to contain other elements, as with **Section** in Figure 1. Others, such as **Body** elements, clearly must contain text. There is no default relationship between elements and paragraph tags.

The extra level of organisation provided by elements can be used by FrameMaker to apply *rules*, to:

- Control the document's structure by restricting the elements allowed in a specific *context*.
- Apply formatting to text objects automatically based on their context, relieving authors from this task and allowing them to concentrate on writing.

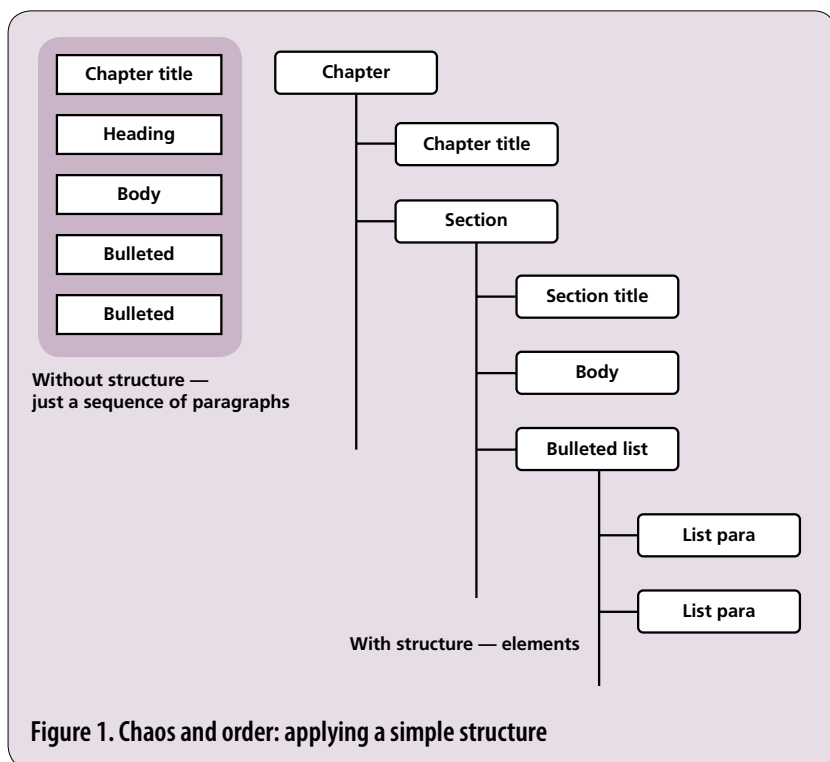


Figure 1. Chaos and order: applying a simple structure

What is a structured document?

Every document possesses structure: it is obvious that a book, for example, contains chapters made up of sections, paragraphs, lists and so on. A structured document is one that adds some form of notation to provide a description of its organisation that is separate from the document's presentation. A suitable notation in this context is one that can be used to define a concrete language that describes

Why should I use structured documents?

The introduction of a structural description of documents brings many advantages:

- A much greater level of automation becomes possible, such as the context-dependent application of formatting mentioned above.
- A document's structure can be *validated*, that is, checked against the structure definition and any errors and omissions flagged.
- Structure enables design devices that would be tedious and error-prone to apply in unstructured FrameMaker to be wrapped in elements and used much more easily.
- The ability to interact with documents at a structural level makes edits to the structure easier and less error-prone, as well as making objects such as markers much easier to select.
- Formatting rules within the structure definition enable document content to be reformatted in response to structural changes, for example changing one element into another with a single command and having all child elements reformat automatically.
- Meaning can be introduced into document structures. For example, the documentation of a software programming interface might include name, interface definition, parameter definition list, usage and error messages for each procedure call. Such elements can be given descriptive names in the structure definition, and completeness can be checked and enforced.
- Locally applied formatting can be removed by reapplying the structure definition to a document.
- Document contents can be repurposed much more easily.
- Extra information about parts of a document can be introduced through the use of *attributes*, data fields that 'belong' to elements but which do not appear in the document itself.
- Inter-working with document tagging formats such as SGML and XML becomes possible.

As with everything, there is a downside. Setting up a structured writing environment carries a reasonably high overhead, which includes getting to grips with structure definitions, analysing your documents, defining and testing the required structures, and becoming familiar with the parts of FrameMaker's user interface concerned with structure. Acquiring deep proficiency with these, as with unstructured FrameMaker, make take years. However, the message behind these articles is that making a start is not as difficult as you might think — and not as difficult as it's sometimes made out to be.

By now, readers who are familiar with Word's Outline view may be wondering what all the fuss is about. However, Outline view only understands Word's predefined heading styles and body text. In structured FrameMaker, by contrast, every object within a document is controlled by the structure definition, and that definition is completely under the documenter's control.

How structure is defined

Until now we've purposely referred only to the 'structure definition' for a document, without describing how it is achieved. FrameMaker uses an *element definition document* (EDD) to hold the structure definition for a document or family of documents. An EDD contains the element definitions, and is itself a structured FrameMaker document. The language required to define elements and their rules in an EDD is built into FrameMaker itself.

Just as you can import the unstructured aspects of a document — paragraph and character tags, page layouts, table definition, variables and so on — from one FrameMaker document to another, so you can import element definitions between an EDD and a FrameMaker document, and between FrameMaker documents. Importing element definitions into a structured FrameMaker document reapplies the structural definitions to the document, and you can opt to remove local formatting overrides, enforcing the formatting defined by the EDD. However, importing element definitions into an *unstructured* FrameMaker document sadly does not magically apply structure — we'll see how that can be achieved in a future article.

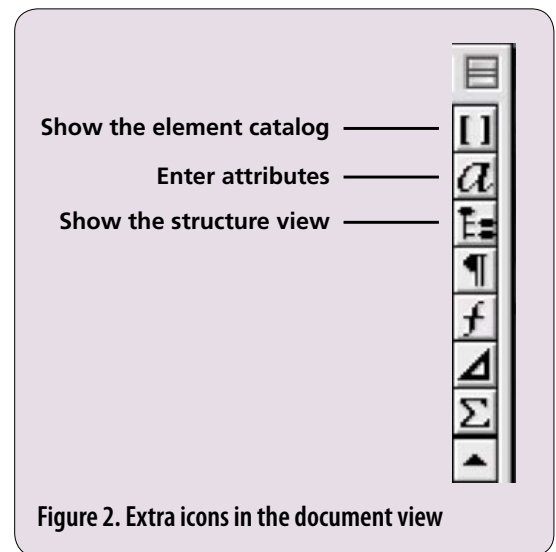


Figure 2. Extra icons in the document view

There is a strong correspondence between FrameMaker's EDD and the *document type definition* document (DTD) used in SGML — indeed, FrameMaker can create a DTD from an EDD and vice versa. The significant difference is that while a DTD defines only structure, an EDD can also include additional rules to control formatting and other issues in the target document.

Structured documents in FrameMaker

When you first start FrameMaker Version 7.x, it asks you whether you want to work in structured or unstructured mode. If you choose unstructured, structure commands are excluded from FrameMaker's menus. In fact there's little point to this, as it's perfectly possible to work with unstructured documents in structured

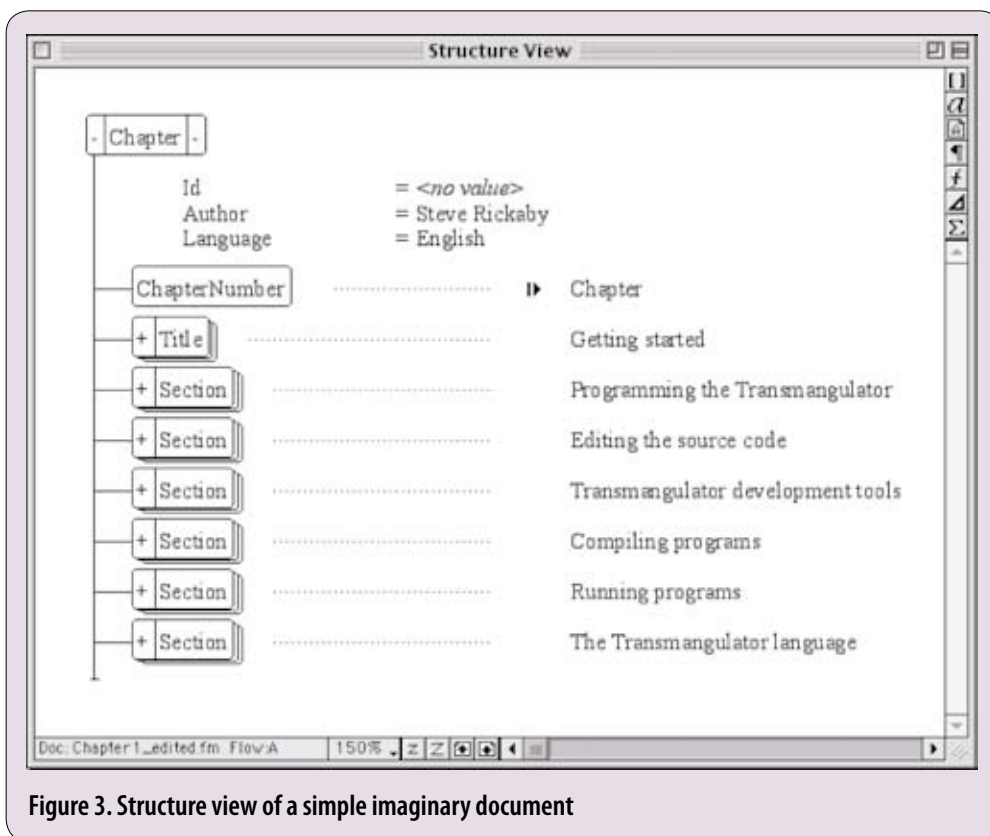


Figure 3. Structure view of a simple imaginary document

FrameMaker and just ignore the structure features. (The converse is not true.)

The most significant thing you notice when first invoking the structured interface is that the top right of the document view sprouts a few extra icons, as Figure 2 shows. Of these, the most interesting at this stage is the icon to display the structure view.

The structure view

FrameMaker's structure view, which is independent of the document view, shows a document in terms of its elements and their relationships.

Figure 3 shows an imaginary example. Here a chapter consists of a **Chapter** element, **Chapter Number** and **Title** elements, and six **Section** elements. The presentation of the elements indicates that all except the **Chapter** and **Chapter Number** contain child elements that are hidden from view. Clicking on the '+' toggles the expansion and collapse of these child elements, enabling you to 'zoom' in and out on details of the document's structure. The view also shows the titles of each section, while **Id**, **Author** and **Language** are attributes of the **Chapter** element.

You can use the structure view both to select elements and to edit the document's structure. Clicking on an element selects it and all its contents, while dragging a selected element in the structure view moves its physical position in the document. For the simple example shown in Figure 3, this would provide an easy way to re-order sections.

The structure view also shows missing elements and invalid structure. If an element is required by the EDD but is not present in the

document, the structure shows a red square.


Equally, if the existence or location of an element makes the document's structure invalid with respect to the EDD, the structure shows a dotted red line. Figure 4 illustrates these.

Future articles

The next article in this series will look at working with structured documents more closely, at the advantages and disadvantages of using off-the-shelf EDDs, and will describe the construction of a simple EDD. Later articles will go into more detail about element definitions, rules and how they are used.

For more information

The FrameMaker *User Guide* does not attempt to cover anything other than merely working with structured documents, so you need extra material to help you get to grips with EDD creation. FrameMaker is supplied with an online guide, the *Structure Application Developer's Guide Online Manual*. This runs to over 500 pages of densely written technical information, and although comprehensive, is quite enough to put anyone new to structure off it for good.

Sarah O'Keefe and Sheila Loring's book *Publishing Fundamentals: FrameMaker 7*, mentioned here before, neatly plugs the gap between these two extremes. Scriptorium Press also offers a range of self-training material that uses this book: the most relevant here is probably *Advanced Structured FrameMaker: Building EDDs*. For details, visit www.scriptorium.com/books/frametrainingseries.html. 

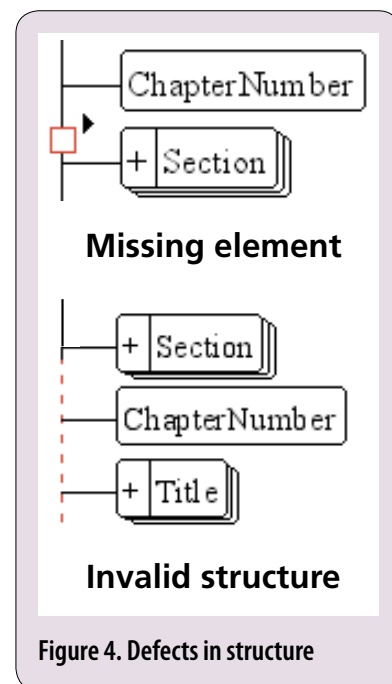


Figure 4. Defects in structure

Steve Rickaby BSc MISTC has been a freelance technical author and editor for 16 years, and has used FrameMaker for most of that time.
E: srickaby@wordmongers.com
W: www.wordmongers.com

First steps in structure: EDDs

Steve Rickaby continues his beginner's guide to structured working in FrameMaker by looking more closely at element definition documents.

The previous article in this series (*Communicator* Spring 2007) introduced the concept of structured working in FrameMaker and described some of its advantages. Now we can look in more detail at how editing operates in structured documents, and at defining a simple element definition document (EDD), which holds the structure definition for a document or document family.

Working in structured documents

The process of writing and editing structured documents in FrameMaker is substantially different to that for unstructured documents. In the latter case, writers often go through a repeated cycle of entering text, applying mark-up from the paragraph or character palettes, more typing, applying more mark-up, and so on. Thus there is permanent division of labour, and concentration, between the writing task and the typesetting task.

In structured documents, in contrast, the writer chooses the required element from the Element Catalog, and just writes: mark-up is applied automatically under the control of the EDD. A carriage return creates a new element of the required type. When a different element type is required, the Element Catalog presents only those elements that are valid in the current context. No interaction with the paragraph or character palettes is necessary, let alone with the Paragraph or Character Designer dialogs.

Figure 1 illustrates this. As this picture was captured when the insertion point was in a body paragraph, only character and marker elements are shown. <TEXT> means that free text is also allowed. As the figure shows,

FrameMaker provides controls for inserting an element, wrapping selected text in an element, and changing a selected element's type via this palette.

As well as the document view, you can use the structure view, described in the previous article, to set the insertion point and to cut, paste and drag elements. In fact, it is often preferable to use the structure view to position the insertion point: although FrameMaker offers options to make element boundaries visible in the document view, they are much more visually disruptive than the **View>Text Symbols** option, as Figure 2 demonstrates.

As you work with a structured document, FrameMaker synchronises the document view and the structure view, so that:

- Selecting an element in the structure view highlights it in the document view.
- Selecting any text range in the document view highlights its lowest-level parent element that is currently visible in the structure view.

FrameMaker also uses different icons in the structure view to show whether the insertion point is at the start of, in the middle of, or at the end of an element, illustrated by Figure 3.

Designing an EDD

You cannot create structured documents without an EDD. How you get hold of a suitable EDD, however, is fenced about with a small forest of decisions. The first of these you can think of as...

'BigSmall'

...to borrow a phrase from recent Toyota advertisements. The first decision you have to



Figure 1. The Element Catalog

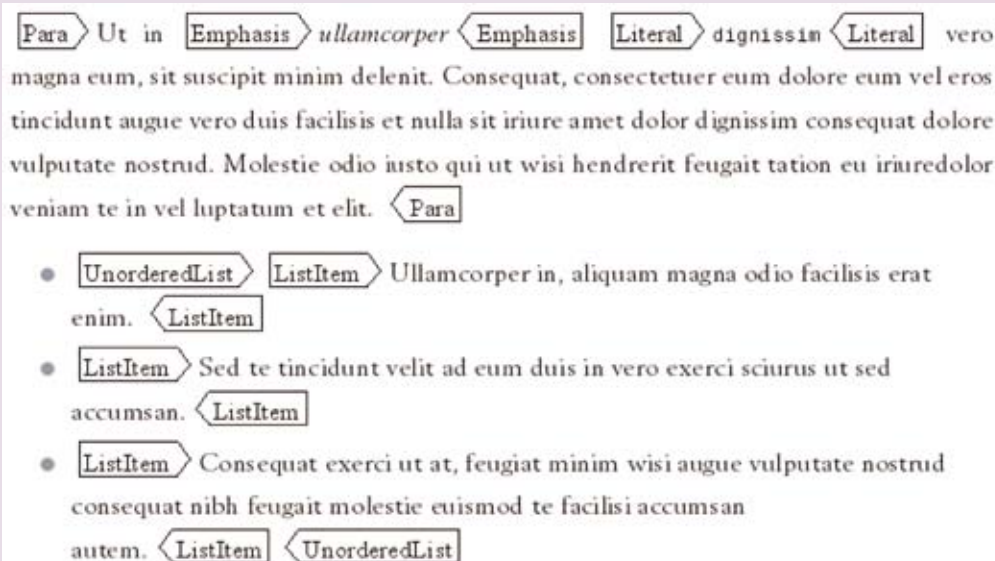


Figure 2. Exposed element boundaries in the document view

make is whether to use an ‘off the shelf’ EDD (the ‘big’ approach), or to create your own (the ‘small’ approach). There are strong arguments for and against both:

- If your project must conform to an existing standard, such as DITA (Darwin Information Typing Architecture), you have little choice, and must specialise the DITA DTDs for your purposes. Adobe has been doing substantial work on DITA recently, and it seems likely that FrameMaker Version 8, due this year, will include specific DITA support.
- General-purpose EDDs such as DocBook (supplied with FrameMaker) are quite relaxed in their application of *context rules*, which can result in poor document structure, and may also contain many element definitions that are irrelevant to you.
- If you are new to structure, as we assume here, adapting an existing EDD like DocBook to your needs is a large-scale and rather daunting task.
- Conversely, developing your own small-scale EDD is a great way to learn about structured documents. For the examples here, we’ll assume that you plan to develop your own small-scale EDD.

approach confers the substantial advantage that all the structure information is in the EDD and all the presentation information is in the document template. This means that the visual appearance of content can be changed without any need to change the structure definition.

Document analysis

The next step in designing your own EDD is a rigorous analysis of the documents you intend to structure. As the EDD will both represent and control their structure, clearly a full knowledge of permissible structures is a prerequisite. Modelling document structure with a DTD is a good approach, as FrameMaker allows you to import a DTD and create an EDD from it.

A simple example will be enough to illustrate the principles here. We will assume that you are creating an EDD to represent documents that consist of a sequence of sections containing headings, body paragraphs, and bullet lists. From that, you should be equipped to extend the EDD to include other document structures up to the book level.

Defining the structure

What does an EDD contain? A FrameMaker EDD is itself a structured document, and typically contains a sequence of element definitions and optional comments, as well as other controls. Each element definition consists of:

- The element’s name
- The element’s type
- A *general rule* (for container elements, described below)
- An optional attribute list
- A sequence of formatting rules

...as well as many other optional controls. The essential aspects are described below: you can find complete documentation on element definitions in the *Structure Application Developer’s Guide Online Manual* supplied with FrameMaker.

Element names

When selecting names for your elements, try to make them short and meaningful, as you would for paragraph and character tag names. If your application needs to support SGML or XML at any stage, bear in mind that their rules for element names are more restrictive than FrameMaker’s. The relevant standards contain full details, but a good approach is to start element names with a letter, stick to alphanumeric characters, and avoid spaces.

Element types

FrameMaker supports some sixteen element types, but by far the most common is **Container**. A container element, as its name implies, contains other elements and/or text. Other element types support graphics, equations, cross-references, footnotes, variables and the various parts of tables.

In the simple document structure we’ll use to illustrate EDDs in these articles, we’re going to include sections, headings, body paragraphs, and

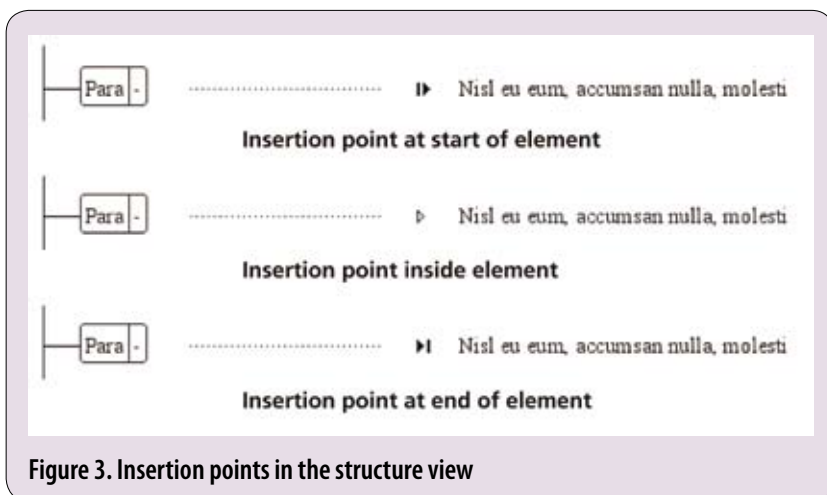


Figure 3. Insertion points in the structure view

Separation of concerns

Once you’ve opted to create your own EDD, you are faced with a second major decision: where to control formatting.

The principle from software engineering known as *separation of concerns* states, in brief, that each entity should have only a single purpose. When applied to EDDs, this can refer to how document formatting should be applied. In structured FrameMaker it is possible to include formatting instructions in the EDD itself, to refer out of the EDD to a document’s paragraph and character tags, or to use a combination of both.

There is not room here to elaborate the advantages and disadvantages of these approaches, and in any case opinions differ amongst experienced practitioners. A reasonable starting position is to keep all formatting within the FrameMaker document template by referencing paragraph and character tags from the EDD, but to remain aware that some formatting operations are simplified by including formatting instructions directly in the EDD. This

Element (Container): Section
General rule: Heading, (Para | UnorderedList | Section)*
Automatic insertions
Automatically insert child: Heading

Figure 4. A (very) simple element definition

Element (Container): Section
General rule: Heading, **Para**, (Para | UnorderedList | Section)*
Automatic insertions
Automatically insert child: Heading

Figure 6. A modified Section element definition

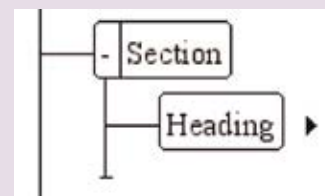


Figure 5. Results of creating the Section element defined in Figure 4

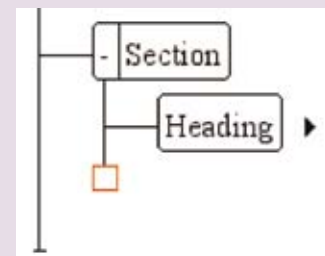


Figure 7. Results of creating the Section element defined in Figure 6

bullet lists. All these are modelled using container elements, as is the top-level element, which we'll call **Chapter**. In a later article we'll flesh out the structure with *object elements*, a term used collectively to describe elements that support only one object, such as a marker, cross-reference, variable or graphic.

It's worth noting that the distinction that FrameMaker makes between paragraph and character *tags* does not exist for elements: both paragraph and character entities are modelled using container elements, although the details differ.

Rules in the structure definition

Rules are the essence of the power of structured FrameMaker. There are many types of rule, but here we will mainly consider three:

- **General rules.** Every container element definition must include a general rule, which specifies the element's permitted contents. FrameMaker uses the same terse but powerful syntax as SGML for general rules, described below.
- **Formatting rules.** Any formatting control that is accessible in unstructured FrameMaker can be included in an EDD and referenced in an element definition.
- **Context rules.** You can insert rules that make an element behave differently depending on its context within the document's structure. Context rules can check the identity of parent elements, the level of the element in the document, or its ordering (first, last and so on).

Figure 4 shows what an actual element definition might look like in the document view of the EDD. We'll describe what this means below.

General rules

The syntax used to define the permissible child elements of a container in general rules are:

- **Element** — the element is required and must occur only once.
- **Element+** — the element is required and may

occur more than once.

- **Element?** — the element is optional but may occur only once if used.
- **Element*** — the element is optional and may occur more than once if used.

In addition, you can group element declarations in general rules using braces and these connectors:

- **,** (comma) — elements must occur in the order given.
- **&** (ampersand) — elements may occur in any order.
- **|** (vertical bar) — elements are alternatives to each other.

These correspond to the syntax of an XML DTD content model.

We are now in a position to make sense of the element definition in Figure 4. The general rule specifies that a **Section** element must contain only one **Heading** element, which must occur first, followed by zero or more **Para**, **UnorderedList** or **Section** elements, which may occur in any order. **Automatic Insertions** is an additional rule that inserts an empty **Heading** element whenever a **Section** element is created, giving the structure shown in Figure 5. Here FrameMaker has created the **Section** element, automatically inserted a **Heading** element, and is waiting for further input (the heading text).

If, instead, you define the **Section** element as shown in Figure 6, creating a **Section** element in a document results in the structure shown in Figure 7. It is showing a red square to indicate that the structure is incomplete, because the **Para** element made mandatory by the modified general rule (shaded in Figure 6) has not been created.

The element definitions in Figure 4 and Figure 6 are greatly simplified: in practise you might use a more complex general rule to specify, for example, that a section cannot start with a list. They are not a complete EDD: definitions of the **Chapter**, **Heading**, **Para** and **UnorderedList** elements are also required. We'll see how these are defined in the next article. **C**

Steve Rickaby BSc MISTC

has been a freelance technical author and editor for 16 years, and has used FrameMaker for most of that time.

E: srickaby@wordmongers.com
 W: www.wordmongers.com

First steps in structure: rules

Steve Rickaby continues a beginner's guide to structured working in FrameMaker to cover further rule types supported in an EDD.

The previous article in this series (*Communicator* Summer 2007) looked at how editing works in structured documents, and made a start on defining a simple EDD (element definition document). This article goes into more detail about rules in the EDD and how you can use them to automate your writing environment.

Rules in the structure definition

As we saw in the last article, rules convey the power of structured FrameMaker. The following three rule types are essential:

- **General rules.** Every container element definition must include a general rule, which specifies the element's permitted contents. The previous article illustrated simple uses of general rules.
- **Formatting rules.** Any formatting control that is accessible in FrameMaker can be included in an EDD and referenced in an element definition.
- **Context rules.** These make an element behave differently depending on its context within a document's element hierarchy. Context rules

can test the identity of ancestor elements, the level of the element in the document hierarchy, or its ordering.

Formatting rules

When designing an EDD, you have to decide whether to control document formatting in the EDD, in the document using paragraph and character tags, or a combination of both.

Formatting rules enable you to use the first and last of these options. However, it's important to note that in an EDD — in the absence of other controls — formatting is inherited by child elements from their parents. This allows, for example, nested lists to be formatted easily, by inheriting paragraph formatting and adding only the required indent.

To define explicit formatting in the EDD, you create a named *format change list*. A format change list is a little like an element, but does nothing by itself, and is referenced by other elements. It can access any of the properties specified by the Paragraph Designer. Figure 1 shows what a format change list looks like.

The figure also illustrates something unique about format control in an EDD: in contrast to paragraph tag definitions, format change list specifications can be *relative*. The example shows how you could increase the indent by 0.33 inch, or 2 pica.

Figure 2 shows how the format change list defined in Figure 1 can be used in an element definition. It also shows our first *context rule*, an all-contexts rule. We'll return to context rules later.

Using format change lists and inheritance enables you to keep an EDD relatively simple. It is also possible, however, for an EDD to apply paragraph and character tags from the document. Figure 3 shows a simple example of this by expanding on the definition of the **Section** element given in the previous article. Here the paragraph tag **Body** will be applied to a **Section** element's contents *and that of its descendants* unless overridden.

If, when an EDD is imported into a document, any of the paragraph tags that it references do not exist in the document, FrameMaker creates default definitions for them. If this is unexpected, it's sometimes a hint that you've misspelled a tag name in the EDD.

Context rules

Context rules could be said to be where the fun starts with structured FrameMaker. They are typically where the greatest complexity of an EDD lies. Using a context rule, you can both enforce and automate formatting in a document

```
Format change list: SetIndent
Basic properties
  Indents
    Move first indent by: +0.33"
    Move left indent by: +0.33"
```

Figure 1. A format change list to set indents

```
Element (Container): IndentedPara
General rule: <TEXT>
Text format rules
  1. In all contexts.
    Use format change list: SetIndent
```

Figure 2. Referencing a format change list

```
Element (Container): Section
General rule: Heading, Para, (Para | UnorderedList | Section)*
Automatic insertions
  Automatically insert child: Heading
Text format rules
  Element paragraph format: Body
```

Figure 3. Specifying a base paragraph format

Element (Container): Heading
General rule: <TEXT>

Text format rules

1. **Count ancestors named:** Section
 - If level is:** 1
 - Use paragraph format:** Heading1
 - Else, if level is:** 2
 - Use paragraph format:** Heading2
 - Else, if level is:** 3
 - Use paragraph format:** Heading3

Figure 4. A simple level context rule

Element (Container): OrderedList
General rule: Para+

Automatic insertions
Automatically insert child: Para

Element (Container): Para
General rule: <TEXT>

Text format rules

1. **If context is:** OrderedList
 - 1.1. **If context is:** {first}
 - Use paragraph format:** NumericalList1
 - Else**
 - Use paragraph format:** NumericalList

Figure 5. A simple positional context rule

during writing and editing. Context rules enable FrameMaker to test:

- The level of an element in the document's element hierarchy
- The position of an element relevant to its siblings: first, last, only, middle, not first, not last and so on
- The types of an element's ancestors.

They also enable FrameMaker to apply formatting to the relevant elements automatically based on the above, using the methods described in the previous section. Context rules are checked and re-applied every time the structure of a document is changed.

Figure 4 shows a level rule that applies the relevant paragraph tag (**Heading1**, **Heading2**, **Heading3**) to the contents of a **Heading** element depending on the nesting level of its parent **Section** element. Although simple, this rule ensures that if sections are moved within a document's hierarchy—which you can do merely by dragging them in the structure view—the section headings are reformatted automatically.

Note that in all these examples the various parts of element definitions are not entered by hand, but are themselves predefined elements built into FrameMaker itself and inserted using the Element Catalog. We'll see how this works in

the next section.

Figure 5 shows just one way in which you could apply context rules to paragraph tags depending on the ordering of an element. The two element definitions specify **Para** and **OrderedList**, in which each child element is of type **Para**, inserting the first list element automatically. In the definition of the **Para** element, a context rule first checks to see if the element is a child of an **OrderedList**. If so, it uses a nested *subrule* (1.1) to check whether it is the first child of the list element. If it is, FrameMaker applies the paragraph tag **NumericalList1** (which should reset the list counter): if not, it applies the paragraph tag **NumericalList**, which just increments it.

Context rules are particularly useful for handling formatting issues such as this. Notice how the two element definitions work together to create the desired result. In the target document, the writer inserts an **OrderedList** element from the Element Catalog, then simply uses carriage returns to add more list elements: FrameMaker applies the paragraph formatting automatically.

As well as checking the immediate context of an element, as shown in Figure 5, a rule can also check the ancestors of the current element. FrameMaker offers the notations shown in Table 1

Table 1. Context rule notations

Notation	Interpretation
ElementA < ElementB	ElementA is the parent, and ElementB is its parent.
ElementA < * < ElementB	ElementA is the parent, and one of its ancestors is ElementB .
ElementA < (ElementB ElementC)	ElementA is the parent, and either ElementB or ElementC is its parent.
You can combine positional indicators with ancestor rules, too. For example:	
{first} < ElementA	The current element is the first sibling of ElementA .
ElementA {after ElementB } < ElementC	ElementA is the parent, and immediately follows an ElementB whose parent is ElementC .

for this. The *Structure Application Developer's Guide Online Manual*, supplied with FrameMaker, elaborates the entire syntax, with examples.

In reality, an EDD is likely to be far more complex than the simple examples shown here. However, they demonstrate the principles, and the rules shown do work, despite their simplification.

Creating the EDD

So far we've talked about analysing document structure and defining an EDD, but we've not looked at how it's actually done.

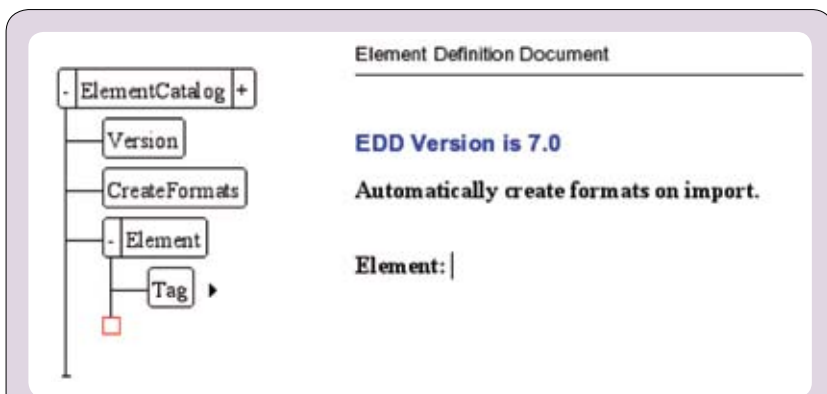


Figure 6. The contents of a new EDD, structure and document views

Element (Container): Chapter
 General rule: Section*
 Valid as the highest-level element.

Figure 7. A Chapter element definition

You create a new EDD in FrameMaker by selecting **File>Structure Tools>New EDD**. You will notice when you do this that FrameMaker has a whole host of structure-specific commands collected under **File>Structure Tools** and **File>Utilities**. The **New EDD** command creates a blank EDD as a structured document that contains only the declarations shown in Figure 6, which shows both the structure and document views.

The default declaration **Automatically create formats on import** refers to FrameMaker's ability to create format definitions (paragraph, character, table or cross-reference) when you import an EDD that refers to such formats into a document that does not already contain them. If you do not want this to happen, you can select the **CreateFormats** element and, using the Element Catalog, change it to a **DoNotCreateFormats** element. This is the better option if you are creating formats manually.

FrameMaker has also created the first, blank, element declaration. The process of defining elements is guided by the choices presented in the Element Catalog. To take a very simple example, the sequence required to create the **Chapter** element definition shown in Figure 7 is as follows:

1. Position the entry point in the **Element** field

that FrameMaker has created by default. The Element Catalog shows **<TEXT>**, indicating that only text is allowed here.

2. Enter 'Chapter' and press **Return**. The Element Catalog changes to show all the elements that are now valid—these are the element type definitions.
3. In the Element Catalog, double-click on **Container**. FrameMaker automatically inserts a **GeneralRule** element and places the entry point in its text field. Once again, the Element Catalog shows **<TEXT>**.
4. Enter the general rule, 'Section*' and press **Return**.
5. Again the contents of the Element Catalog changes to show valid elements. Double-click on **ValidHighestLevel**, and the **Chapter** element definition is complete.


This demonstrates how a structure definition guides and enforces valid document structure—in this case the structure definition for EDDs that is built into FrameMaker itself. Of course, the example is a very simple element: in reality some iteration is often required to complete element definitions.

Future articles

The next article in this series will flesh out the simple EDD examples shown so far with other essentials such as text range (character formatting), marker and cross-reference elements, and look at testing EDDs. Further articles will describe how to apply structure to unstructured documents, and how to set up bidirectional interchange between FrameMaker and XML.

For more information

For information on working with structured documents and creating EDDs, Sarah O'Keefe and Sheila Loring's book *Publishing Fundamentals: FrameMaker 7* is recommended. Scriptorium Press also offers a range of self-training material that uses this book: the most relevant here is probably *Advanced Structured FrameMaker: Building EDDs*. www.scriptorium.com/training/frame7train.html. There is also a useful collection of white papers at www.adobe.com/products/framemaker/indepth.html.

For wider information on structured documentation, see *A Gentle Introduction to SGML*, www.isgmlug.org/sgmlhelp/g-index.htm. 

Steve Rickaby BSc MISTC has been a freelance technical author and editor for 16 years, and has used FrameMaker for most of that time.
 E: srickaby@wordmongers.com
 W: www.wordmongers.com

First steps in structure: elements

Steve Rickaby continues a beginner's guide to structured working in FrameMaker by looking at different types of elements.

The previous article in this series (Autumn 2007 *Communicator*) described rules in the Element Definition Document (EDD) and how you can use them to automate your writing environment. This article explains how other document objects such as character formatting, marker and cross-reference elements are defined in an EDD, and discusses testing EDDs.

Types of elements

Structured FrameMaker supports the following element categories and types:

- **Container elements.** We have already looked at container elements in previous articles in this series. Container elements can hold other elements and/or text, and can be recursively nested.
- **Object elements.** This is the collective term for elements that contain only one object. Object element types exist for cross-references, equations, graphics, markers and system variables.
- **Table elements.** As their name implies, table elements define the various parts of a table. Table element types exist to model the whole table, table body, table cell, table footing, table heading, table row and table title.

You may notice that this list makes no mention of character formatting elements. In structured FrameMaker, a character formatting element is also a container element, but one that specifies the **Text Range** property. Thus the distinction between paragraph and character formatting that exists at the document level is removed in the structure definition. However, the context-sensitive nature of the Element Catalog ensures that only valid elements are presented in any specific context.

Figure 1 shows an example of a character formatting element, defining an element type **Emphasis** that can be used to wrap a range of text and automatically apply the *character tag* **Emphasis** to it. (If you find such duplication of names confusing, you can of course use different names for your elements and the corresponding tags.)

This definition is over-simplified, as the `<TEXT>` general rule means that no other elements, such as marker elements, can be nested inside an **Emphasis** element, which would not be the case in practice.

Adding object element definitions

The process of defining object elements is the same as previously outlined in this series for container elements: the context rules built into FrameMaker itself guide what is offered in the Element Catalog while you build each element definition. However, some object elements have additional properties that allow you to automate the target environment further. For example,

```
Element (Container): Emphasis
General rule: <TEXT>

Text format rules

1. In all contexts.
   Text range.
   Use character format: Emphasis
```

Figure 1. A character formatting element

```
Element (Container): Graphic
General rule: (AnchoredFrame | ImportedGraphic)

Text format rules

1. In all contexts.
   Use paragraph format: FigureAnchor

Element (Graphic): AnchoredFrame
Initial graphic element format

1. In all contexts.
   Insert anchored frame.

Element (Graphic): ImportedGraphic
Initial graphic element format

1. In all contexts.
   Insert imported graphic file.
```

Figure 2. Object elements: graphics element definitions

cross-reference elements can specify a default cross-reference format, while marker elements can specify a default marker type. A graphic element can specify whether a blank anchored frame should merely be inserted into a document, or the Import File dialog also displayed.

Figure 2 shows some simple element definitions to support graphics. The element **Graphic** offers the choice of **AnchoredFrame** or **ImportedGraphic** child elements, while applying the paragraph tag **FigureAnchor** to the paragraph that contains the graphic frame's anchor.

Attributes

Attributes are separate items of information that are associated with a specific element. They have three main purposes:

- They can be used to track extra information about a document or element, such as author, version, dates and so on.
- They can be used to control special processing applied to elements when content is saved as XML or another tagged format.
- Read-only attributes are required to support cross-references if content is to be saved to XML or SGML, as these formats only support cross-references in this form.

Figure 3 shows the typical appearance of attributes

Note

This series of articles on structured working refers to the menu options available in FrameMaker versions 7 to 7.2. There is a new **Structure Tools** menu in FrameMaker 8 (see page 16 of this issue).

in the structure view. Here **Id** is a read-only attribute used to support structured cross-references to the chapter, **Author** is a free text attribute and **Language** is a multiple-choice attribute with predefined values. When a new chapter element is created, FrameMaker displays the dialog shown in the lower part of the figure. The widget on the right displays option values for multiple-choice attributes.

Testing a structure definition

The process of creating and testing a 'real' EDD from scratch is usually iterative:

1. Create a new blank EDD.
2. Create a simple test document.
3. Create or modify one or more new element definitions.
4. Import the EDD into the test document.
5. Optionally, create or refine the required paragraph or character tags in the test document.
6. Test the behaviour of the new elements.
7. Return to step 3 until done.

FrameMaker does its best to help you: when you import element definitions into a document it checks all the element definitions in the EDD. If it finds errors, it displays them using the Element Catalog Manager Report dialog, such as the example in Figure 4. The element names in this report—in this case **Emphasis** and **IndentedPara**—are hyperlinks to the respective element definitions in the EDD being imported. The report shown in the figure indicates that the EDD defines these elements but they are not referenced in the general rules of any container element definitions (this is not really an error, just a warning). The opposite of this—elements that are referenced in a general rule but not defined in the EDD—definitely is an error.

As is often the case with compiler errors, if importing an EDD creates pages and pages of error messages, it is often only the first few that are relevant. A common cause of errors is mistyping an element name in a general rule. If the Element Catalog Manager Report indicates that FrameMaker has created one or more new tags in the target document, again, this is often caused by mistyping the respective tag name in a context rule in the EDD. For example, if your document defines a paragraph tag called **Body**, but a context rule in the EDD instead refers to **body**, FrameMaker will create a default **body** tag when you import the EDD because case is significant.

FrameMaker offers other tools for debugging structure definitions. While testing an EDD, there will be times when you think that your test document should be valid, but it may not be due to errors in the EDD. You can validate a document using **Element>Validate**, which produces the dialog shown in Figure 5. As Sarah O'Keefe helpfully points out in *Publishing Fundamentals: FrameMaker 7*, if you think of this as the structural equivalent of a spell-checker, its operation becomes clear.

The most complex part of any EDD, particularly one designed to provide a high degree of control

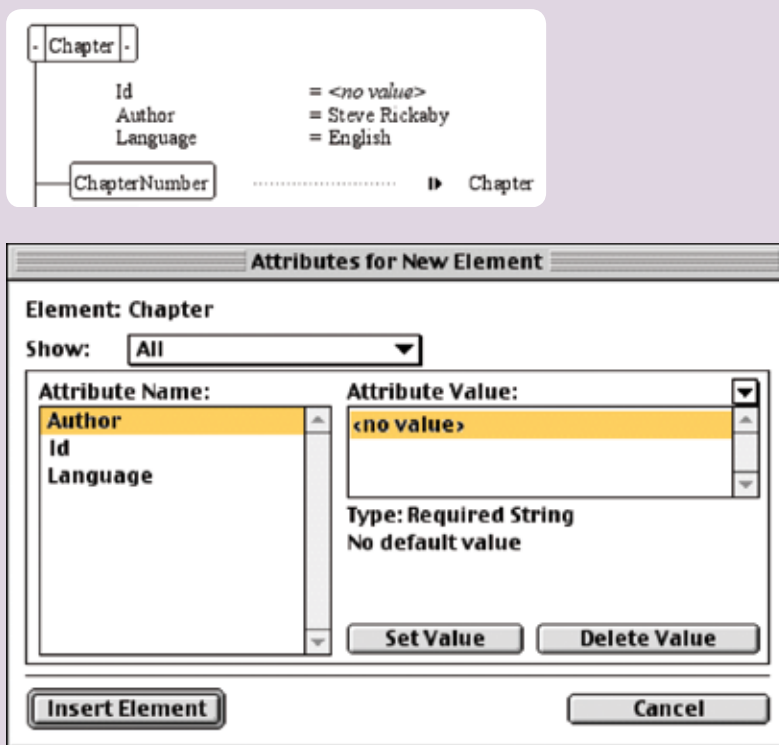


Figure 3. Attributes for a chapter element, and their entry dialog

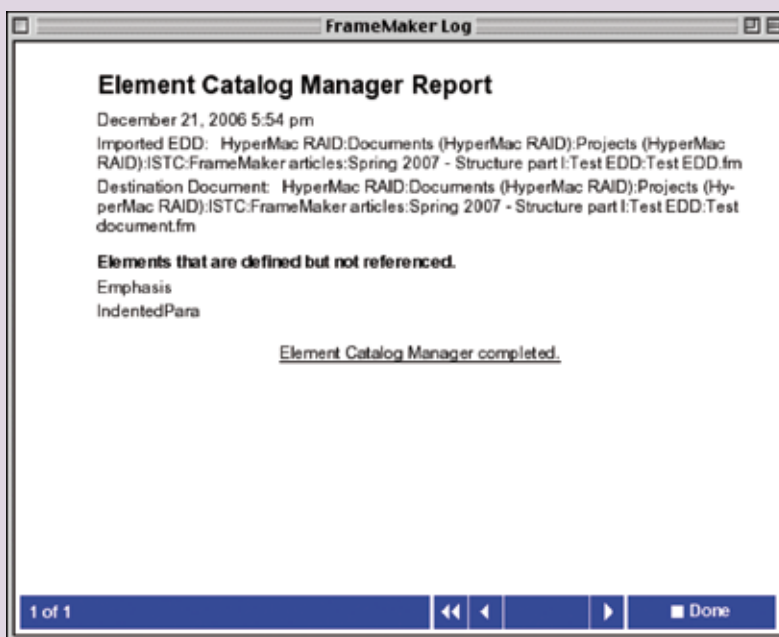


Figure 4. An Element Catalog Manager Report

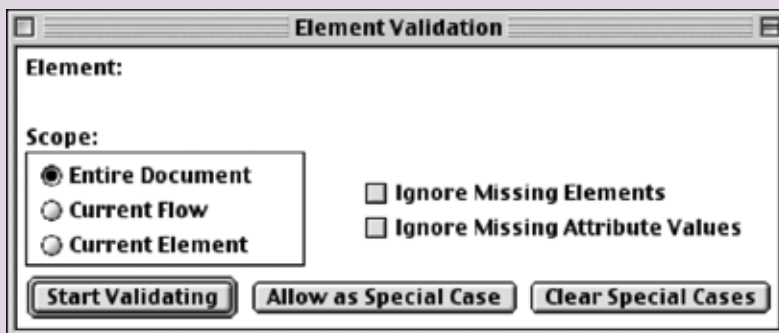


Figure 5. The Element Validation dialog

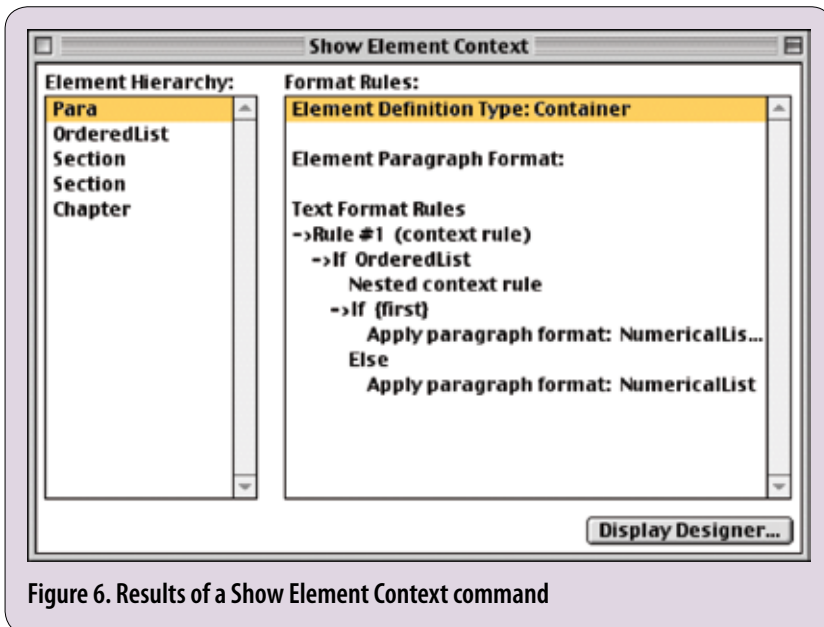


Figure 6. Results of a Show Element Context command

and formatting automation, is going to be its context rules. Their programmatic if/elseif/else structure can lead to unexpected behaviour in the target document, and some sort of debugging tool is vital.

FrameMaker provides the **File>Structure Tools>Show Element Context** command for this. For any selected element, this displays the element's ancestors and the context rules 'fired' by the element. In Figure 6 the insertion point is in the first **Para** element of an ordered list that uses the element definitions shown in Figure 5 of the Autumn 2007 article in this series. The element hierarchy is shown on the left, while on the right-hand side of the dialog the arrows show that the first context rule—the **If OrderedList** rule—and the **If {first}** rule have both fired. The dialog obscures it, but this has resulted in the paragraph tag **NumericalList1** being applied, resetting the list counter.

Steve Rickaby BSc MISTC

has been a freelance technical author and editor for 16 years, and has used FrameMaker for most of that time.

E: srickaby@

wordmongers.com

W: www.wordmongers.com

Structuring unstructured documents

Importing element definitions into a structured FrameMaker document reapplies the structural definitions to the document. Sadly, importing element definitions into an *unstructured* FrameMaker document does not magically apply structure. FrameMaker does, however, offer tools to do this, in the form of *conversion tables*. The process of applying structure consists of successively *wrapping* paragraph tags in elements, and elements within other elements, in a 'bottom up' fashion, until an entire document is structured.

A conversion table is like any other FrameMaker table, and exists within its own document. It has columns to control the paragraph tag or element to be wrapped, the element used to wrap it, and an optional qualifier that is used to indicate temporary elements. You apply a conversion table to an unstructured FrameMaker document using the **File>Utilities>Structure Current Document** command. The next article will look at conversion tables in detail.

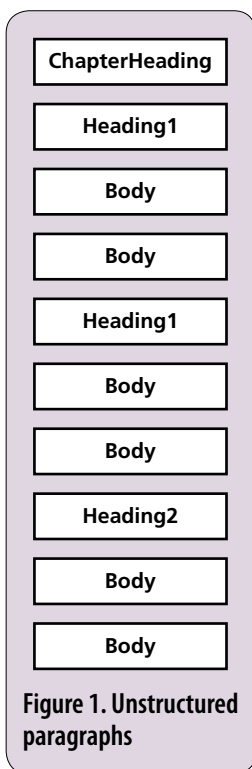
For more information

For information on working with structured documents and creating EDDs, Sarah O'Keefe and Sheila Loring's book *Publishing Fundamentals: FrameMaker 7* is recommended. Scriptorium Press also offers a range of self-training material that uses this book: the most relevant here is probably *Advanced Structured FrameMaker: Building EDDs*. www.scriptorium.com/training/frame7train.html.

FrameMaker is supplied with a comprehensive online guide, the *Structure Application Developer's Guide Online Manual*, which is useful for reference. **C**

First steps in structure: wrapping up

Steve Rickaby completes a beginner's guide to structured working in FrameMaker by looking at conversion tables and round-tripping.



The previous article in this series (*Communicator* Winter 2007) looked at the various types of elements you can define in a structured document, and touched on how to apply structure to unstructured documents. This article completes the series by going into more detail about structuring unstructured documents, and finishes with a brief overview of 'round-tripping'—setting up a FrameMaker <-> XML or SGML environment.

Structuring unstructured documents

In the previous article, we saw that importing element definitions into an unstructured FrameMaker document does not magically apply structure. FrameMaker does offer a way to do this, in the form of *conversion tables*. Conversion tables, also known as *wrapping tables*, allow structure to be applied to an unstructured FrameMaker document by successively wrapping named unstructured objects in elements, and elements within other elements, in a 'bottom up' fashion, until the entire document is structured.

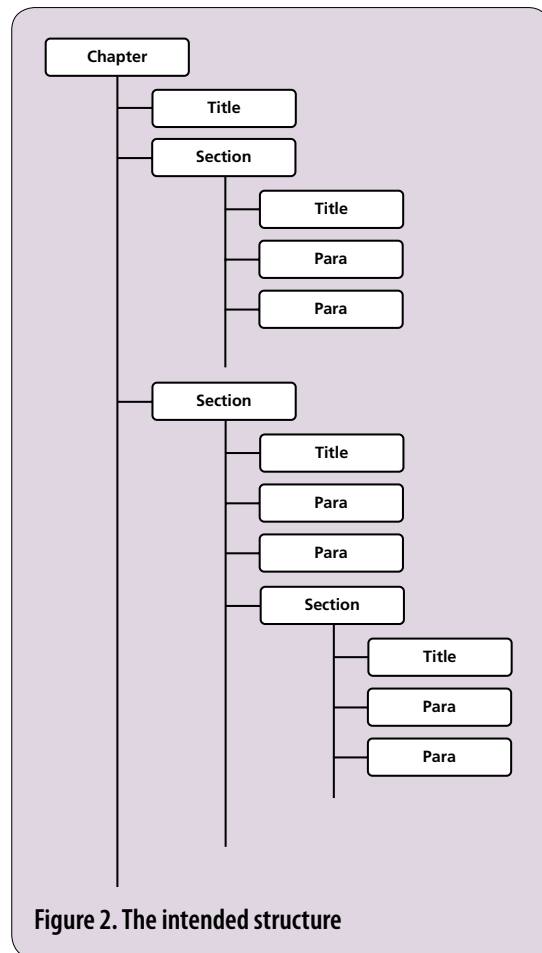
A conversion table is like any other FrameMaker table, and exists within its own document. FrameMaker can create a default conversion table for you with the **File>Structure Tools>Generate Conversion Table** command. When you do this in an unstructured document, FrameMaker populates the table with all the paragraph, character, and table tags used in the document.

A conversion table has a minimum of three columns:

- **Wrap this object or objects.** This column lists the object to be wrapped. FrameMaker uses a prefix to indicate the object type, for example 'C' for character tag, 'P' for paragraph tag, 'E' for element and so on.
- **In this element.** For each object listed in the first column, this column lists the element in which the objects should be wrapped.
- **With this qualifier.** The final column enables you to apply an optional, temporary qualifier. FrameMaker tags the wrapping element with this qualifier, but only uses it internally. Its purpose is to allow two-stage wrapping: we'll look at this in more detail later.

You can also add additional columns for explanations (probably a good idea).

Let's see how this works in practice. Consider a (very) simple unstructured FrameMaker document that consists only of body paragraphs—paragraph tag **Body**—and two levels of heading, **Heading2** and **Heading1**, plus a chapter heading, as shown in Figure 1. To keep things simple we'll exclude lists from the example. Assuming a suitable Element Definition Document (EDD), we want to end up with:



- **Body** paragraphs wrapped in **Para** elements
- **Headingx** paragraphs wrapped in **Title** elements
- **Title** elements and all their child **Para** elements wrapped in **Section** elements
- **Section** elements connected with the correct hierarchy
- All **Section** elements, plus the **ChapterHeading**, wrapped in a **Chapter** element.

This will give the structure shown in Figure 2. For clarity, the element names differ from the paragraph tag names, although this is not mandatory.

The key is to think of this as a sequential process that builds the element structure from the lowest level of detail to the highest—hence *bottom-up*—as FrameMaker processes the conversion table row by row. Here is what we need to do:

1. Wrap each **Body** paragraph in a **Para** element.
2. Wrap each **Heading2** paragraph in a **Title** element.
3. Wrap each **Heading1** paragraph in a **Title** element.
4. Wrap each **Title** element and one or more **Para** elements in a **Section** element.
5. Wrap (sub)Section elements in **Section** elements.
6. Wrap the **ChapterHeading** paragraph in a **Title** element.

7. Wrap all **Section** elements and the chapter's **Title** element in a **Chapter** element.

Figure 3 shows a first attempt at what a conversion table might look like.

Wrapping in stages

Once you have imported the element definitions from the EDD into an unstructured document using **File>Import>Element Definitions**, you apply a conversion table using the **File>Utilities>Structure Current Document** command. When you do this, FrameMaker creates a copy of the unstructured document and applies the structure defined by the conversion table to it, processing the conversion table row by row from the first to last rows.

Unfortunately, the conversion table shown in Figure 3 will not produce the required results. In fact, what happens is that the (unstructured) subsection with the **Heading2** heading appears in the structure as a third top-level section. Why? Let's run through the table line by line:

- The first line wraps **Body** paragraphs in **Para** elements.
- The second line wraps **Heading2** paragraphs in **Title** elements.
- The third line wraps **Heading1** paragraphs in **Title** elements.
- The fourth line wraps all **Title** elements that are followed by one or more **Para** elements in a **Section** element.
- The fifth line wraps all **Title** elements that are followed by one or more **Para** or **Section** elements in a **Section** element.
- The sixth line wraps the **ChapterHeading** paragraph in a **Title** element
- The last line wraps the chapter's **Title** element and all **Section** elements in a **Chapter** element.

Because the fourth line wraps all **Title** and **Para** element sequences in **Section** elements, it will apply this process both to sections having a **Heading1** heading and sections having a **Heading2** heading. Thus the fact that the **Heading2** section is a subsection is lost, and the fifth line of the conversion table is redundant; all **Title** elements have already been wrapped.

The trick in this case is to preserve the fact that the **Heading2** section should be nested in its parent **Heading1** section when structure is applied. This is where the third column of the conversion table comes in. It allows the element applied by a specific row of the conversion table to be tagged with a qualifier, so that it has the same element, but a temporarily different element name, later in the conversion process. The element qualifier has no existence outside the conversion process.

Consider the conversion table shown in Figure 4. As before, the first line wraps all **Body** paragraphs in **Para** elements. However, the second line uses an element qualifier **nested** to tag the **Title** elements that wrap **Heading2** paragraphs, such that their identity becomes **Title[nested]**. The third line of the table similarly

creates a **Title[outer]** element, while the fourth line wraps only the nested **Title** elements and their child **Para** elements in **Section** elements, and again applies the **nested** qualifier to give **Section[nested]**. Now the fifth line of the conversion table has something to do, namely wrapping the remaining unwrapped top-level **Title** elements with their child **Para** elements and **Section[nested]** elements in a top-level **Section[outer]** element. This table has the desired effect, producing the structure shown in Figure 2. You can extend this method to deeper levels of subsection nesting if required, and also use as many different element qualifiers as you need.

Another example of a situation in which you need to use element qualifiers is in wrapping different types of list. Your EDD may—in fact, probably should—use the same **ListItem** element for all list items. However, you will probably have both bulleted lists and numeric lists, making it essential to be able to distinguish between the two types of list item during the wrapping process, so that they can be wrapped in the correct parent element for the list type.

Wrap this object or objects	In this element	With this qualifier
P.Body	Para	
P.Heading2	Title	
P.Heading1	Title	
E.Title, (E.Para)+	Section	
E.Title, (E.Para E.Section)+	Section	
P.ChapterHeading	Title	
E.Title, (E.Section)+	Chapter	

Figure 3. Conversion table: first try

Wrap this object or objects	In this element	With this qualifier
P.Body	Para	
P.Heading2	Title	nested
P.Heading1	Title	outer
E.Title[nested], Para+	Section	nested
E.Title, (Para Section[nested])+	Section	outer
P.ChapterHeading	Title	chapter
E.Title[chapter], Section[outer]+	Chapter	

Figure 4. Conversion table: second try

Testing conversion tables

For very simple structuring such as the example here it is possible to debug a conversion table by letting it run to completion, examining the results, applying a fix, and trying again. However, for 'real' applications in which a conversion table may have many tens of lines, it can be hard to guess the problem when the results are not what you expect.

A useful technique is to take advantage of the fact that table rows can be conditionalised. Because FrameMaker executes a conversion table row by row, this enables you to selectively 'switch off' lower rows of a conversion table so that you can examine intermediate results of the process

and pin down where an error is occurring.

Creating and testing a conversion table requires substantial effort when moving unstructured FrameMaker documents to a new EDD. However, provided that the unstructured documentation makes consistent and controlled use (that is, no overrides) of a stable template, it should only have to be done once.

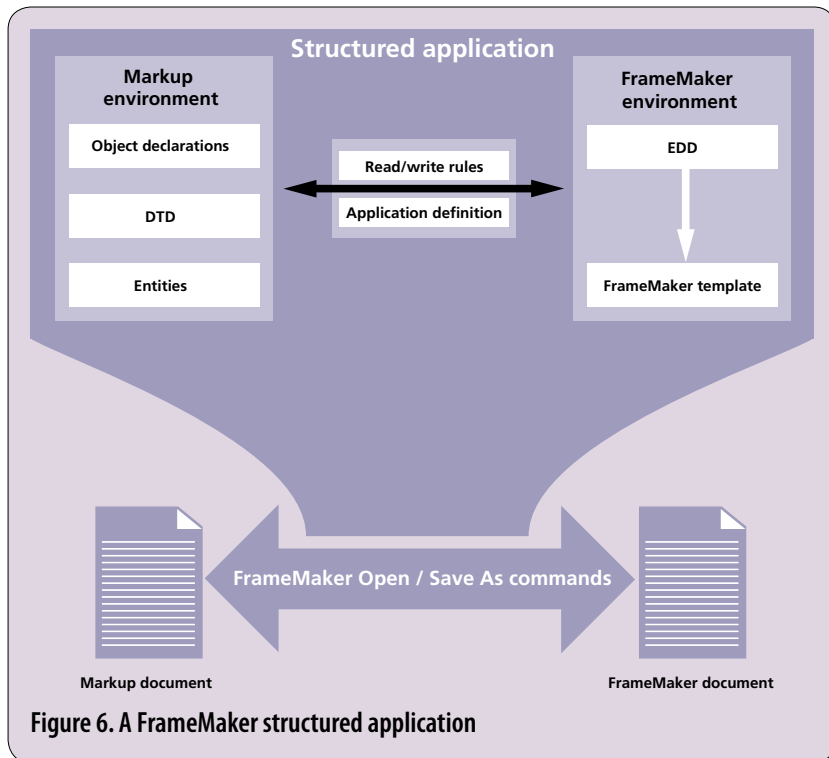


Figure 6. A FrameMaker structured application

Setting up round-tripping to other formats

'Round-tripping' refers to the bidirectional interchange of content between FrameMaker-native format and a markup language such as XML or SGML. This section can only scrape the surface of this subject, but the key point for those new to structured FrameMaker is that it is entirely possible to store FrameMaker content in, say, XML, but have it appear and behave as conventional structured content in FrameMaker when the XML file is opened or saved. Once implemented, FrameMaker applies the appropriate conversions entirely behind the scenes.

But what is 'implemented'? To be able to open a markup format file and convert it to FrameMaker structured content, FrameMaker needs extra information in the form of an EDD, a valid template, and optionally a set of read/write rules. When opening a markup file, FrameMaker applies the read/write rules, the EDD and the template to convert the markup data to FrameMaker structured content.

When saving to a markup language, this process is reversed, except that instead of using the EDD, FrameMaker uses a corresponding Document Type Definition (DTD) to validate the markup data produced.

From this it is possible to see that to handle conversion to and from markup format,

FrameMaker needs the following files:

- An EDD
- A DTD
- A document template
- An optional set of read/write rules.

Collectively, FrameMaker refers to this group of files as a *structured application*. Structured applications—that is, the lists of such groups of files—are stored as named entities in FrameMaker's **structapps.fm** file. When you ask FrameMaker to open a markup file, or save to a markup format, FrameMaker asks you which structured application you want to use (Figure 5). The overall process is illustrated diagrammatically in Figure 6.

However, markup languages such as XML and SGML differ from FrameMaker in the way in which they represent content, so the full development and refinement of a structured application is likely to be considerably more complex than is described here. If starting with an EDD and creating a DTD from it, for example, some simplification of the DTD may be required, particularly for XML. This is because FrameMaker's EDDs support a richer object model than those of many markup languages. Figure 7 illustrates this: the first element definition contains

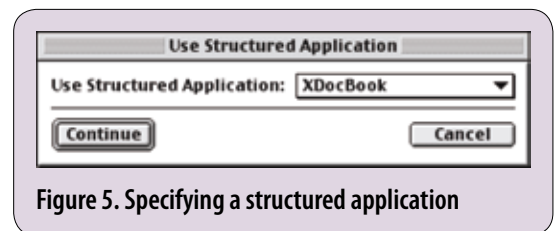


Figure 5. Specifying a structured application

a general rule that specifies a single graphic element. This is not valid for XML: the closest valid general rule for an XML DTD is shown below it, which allows any number of **Graphic** elements. This type of variability can be encoded in an EDD conveniently using conditional text.

If round-tripping is not required—perhaps for a workflow that must combine some XML content with plain text, format it in FrameMaker and publish to PDF—the full complexity is not required and the job of developing the structured application is simplified. The online *Structure Application Developer's Guide* contains all the information required for these tasks.

Read/write rules

Read/write rules are written in plain text and contained in a FrameMaker document that forms part of your structured application. They have a simple syntax not unlike C. For example:

```
element "xyz" {
  is fm element "pdq";
  attribute "aa" is fm attribute "bb"
}
```

defines a bidirectional rule for translating an element name and one of its attributes between the markup and the FrameMaker domains.

FrameMaker offers a default application that translates markup elements to FrameMaker

elements of the same name and markup attributes to FrameMaker attributes, and vice versa. Read/write rules enable you to modify this behaviour, for example in the case in which markup attributes correspond to FrameMaker formatting properties.

There are several situations in which you may require read/write rules:

- Constructs such as tables, graphics or cross-references may require special translation between markup and FrameMaker that cannot be specified in a DTD. When translating such constructs, FrameMaker makes assumptions about them that may not be what is required.
- Elements and attributes may need to be renamed in the markup domain. FrameMaker allows quite luxurious naming, which can conflict with the stricter rules of XML and SGML.
- Translating entities in XML or SGML that represent special characters into the actual characters in the FrameMaker domain.

through its API (application programming interface). FrameMaker's API enables you to exert arbitrary control over its processing, although using it does require programming skills. We hope to make this the subject of a future article.

Read/write rules, unlike element definitions, are not context-sensitive: an element is treated the same way irrespective of its parents.

For more information

For information on working with structured documents and creating EDDs, Sarah O'Keefe and Sheila Loring's book *Publishing Fundamentals: FrameMaker 7* is recommended. Scriptorium Press also offers a range of self-training material that uses this book: the most relevant here is probably *Advanced Structured FrameMaker: Building EDDs*. www.scriptorium.com/training/frame7train.html.

Finally, in the long journey to get to here with structure, I have had sterling help and support

```

Element (Container): Figure
General rule: (<TEXT> | IndexTerm | Literal | XRef)*, Graphic

Attribute list
1. Name: Id                Unique ID                Optional
   Control flags: Read-only

Text format rules
1. In all contexts.
   Use paragraph format: Figure

```

Figure 7a. Element definition valid for EDD but not for XML

```

Element (Container): Figure
General rule: (<TEXT> | IndexTerm | Literal | XRef | Graphic)*

Attribute list
1. Name: Id                Unique ID                Optional
   Control flags: Read-only

Text format rules
1. In all contexts.
   Use paragraph format: Figure

```

Figure 7b. Element definition valid for XML

- Making structural information that is implicit in XML or SGML explicit in FrameMaker, such as the number of columns in a table.
 - Flattening structural hierarchies that are required in one domain but not in the other.
- FrameMaker read/write rules form a comprehensive 'language' of over 70 commands that cover the translation of elements, attributes, books, cross-references, entities, equations, footnotes, graphics, markers, tables, text, text insets and variables. Where read/write rules cannot handle the differences between markup format and FrameMaker format, you can write a structured client that controls FrameMaker

from the assorted gurus on the FrameUsers group (www.FrameUsers.com). I cannot commend or thank them too highly. Special mention should be made of my stalwart and patient reviewers: Jane Dards, Marcus Carr of Allette Systems, Sydney, and Lynne A Price, of Text Structure Consulting, Inc, California. **C**

Steve Rickaby BSc MISTC has been a freelance technical author and editor for 17 years and has used FrameMaker for most of that time.
E: srickaby@wordmongers.com
W: www.wordmongers.com

Training in structured FrameMaker

Reviewer Lynne A Price is president of Text Structure Consulting Inc, which specialises in training, consulting and application development in structured FrameMaker. See www.txstruct.com for tips and free tools dealing with EDDs, read/write rules and structure application definitions, as well as descriptions of training and consulting services. Users new to structure may be interested in the self-study course on *Editing Structured Documents: A Course for Experienced FrameMaker Users* (details from Lynne at lprice@txstruct.com).

Lynne is willing to run training in the UK if there is enough demand. Interested readers should contact Marian Newell at journal.editor@istc.org.uk with details of the type of training that would be useful to them.