The FrameMaker Document Model

David J. Murray

A FrameMaker document consists[1] of an ordered set of Page objects, each of which can contain Frames (containers), that in turn can contain any/all of the following types of Content objects: Vector Graphic objects, Text objects, and Frames.

There are 3 types of Pages in a FrameMaker document file: 1. Body pages, which contain the objects and text that directly correspond to the printed (or electronically viewed) document represented by that file. 2. Master Pages, which serve as templates for the layout of Body pages created manually by authors, and for Body pages created automatically by the software to make space for new content added to the text streams on Body Pages. Master pages also provide background objects for Body Pages. That is, they create page elements that repeat on multiple Body Pages, such as headers, footers, page numbering, and graphic page elements such as lines and boxes to separate or surround text columns. Every document contains two undeletable master pages named Left and Right, and users can add an unlimited number of additional named master pages. 3. Reference Pages, which are catch all locations for named (tagged) objects, used by reference, in ways that we'll cover below.

Each Body Page has a property that identifies its associated Master Page. The options are None, Default, or a specific named master page. Default means use the Left or Right master page, depending on the body page's parity, which can change as pages are added or deleted. There is a document file level property that controls whether the document is treated as a single or double sided one and, for double sided cases, whether the first page is a left or right page. When a body page is rendered, drawing begins with the objects on the associated master page (except for the template column layout objects, which we'll cover below), followed by the graphics and text on the Body Page itself.

As mentioned above, Master Pages can contain template text column objects that determine the normal location for text columns on corresponding Body Pages. Users can directly manipulate the layout of those template columns on Master Pages, or modify their placement indirectly using various layout modification commands. In either case, when the user returns to viewing Body Pages after modifying Master Pages, FrameMaker automatically applies the changed template column layouts to the columns on all of the corresponding Body Pages. For Body Pages whose columns matched the layout of their master page before it was edited, the new column layouts are applied automatically. But for any body pages whose column layouts already differed from their master pages FrameMaker informs the user, and provides the option to retain those overrides or remove them.

---

[1] I will use present tense to describe Frame's document model, even though I will actually be describing the model supported in the first few versions of the product. I do this partly for simplicity, and partly because the description still holds, although it is incomplete compared to more recent versions of the product.

Every page automatically contains a Page Frame that completely covers its surface, so that users can place content objects onto pages without having to first manually add a frame to put them into. Each frame defines its own coordinate system. Content objects placed within a frame have coordinates relative to that system, so that, if a Frame is moved, all of the objects inside the frame move too, retaining their relative position within the frame.

The graphic objects supported by FrameMaker are vector graphics whose geometry and properties can be manipulated directly with the mouse and with graphics modification commands at any time. They include rectangles, rounded rectangles, ovals, simple lines, open and closed polygons, multi-point open and closed bezier curves, embedded bitmapped images (originally B&W, and later in the product's evolution, color), which can be manipulated like rectangles whose "fill" property is the image, and references to external objects (primarily to external bitmapped images in various formats over time, ranging from Sun raster files and Mac Picts to TIFF, and later JPEG images). All of these objects have properties which define how they are drawn, such as the thickness, color, and pattern of their border/path, and the color and pattern of their fill, for objects where Fill is relevant. Unclosed objects (lines, open polygons, and open bezier curves) have properties that define how their end caps are rendered: butt, square, rounded, and arrows. All of these graphic objects, as well as the textual objects described next, can be grouped together into hierarchical (nested) group objects, so that the user can move, scale, copy/paste or modify them interactively as a unit.

FrameMaker supports three types of text objects: TextLines, TextRects and their associated TextFlows, and Equations.

TextLines are simple text objects that contain a character string presented on a single line, using any mix of character attributes (font, face, style, position shifts, etc.). If the user presses the Return key while typing into a TextLine, the software will create a second, independent TextLine, spaced below the original one based on the current font size, but not connected to it in any way. TextLines are used primarily for things like callouts and labels in diagrams. In early versions of FrameMaker the width and height of the characters in a TextLine were determined solely by their Font Size. But starting in FrameMaker 3 (I believe) the width and height of a TextLine's bounding box could be independently scaled, which caused the characters in the TextLine to be stretched or compressed accordingly.

The second, and most important type of text object supported by FrameMaker is known as a TextRect, which is a standard graphic rectangle object (usually with its border and fill properties set to None) that contains a TextFlow. The most common use of a TextRect is to hold a column of paragraph oriented text on a page, and multiple TextRects are usually linked together, so that textual content can flow between them as the content is edited and formatted. The textual content that appears in a single standalone TextRect, or in a set of linked TextRects that may span multiple pages, constitute what is known as a TextFlow. Fundamentally a TextFlow is an ordered set of characters grouped into paragraphs, much like the text that makes up the main text stream in a typical word processing document. FrameMaker supports an arbitrary number of TextFlows (each of which is associated with a

linked set of one or more TextRects). Each TextFlow can have a Tag (a name) and has an Autoconnect property that defines what happens when the formatted text in the TextFlow is too long to be completely visible in the TextRect(s) containing the flow.

We'll dive more into the properties of TextFlows in a moment. But we have now introduced enough of FrameMaker's document model to explain how it mimics the automatic layout capabilities and interactive style of a traditional word processor, even though it supports completely arbitrary direct manipulation of the objects and text columns on pages too.

When a user creates a new blank document that's not based on a pre-defined template, the program presents a dialog box in which the user can choose parameters that define the document's basic layout, including its page size, the number of columns on each page (for text), and the size of the margins for those columns. There are other options, including whether the document is to be single or double sided. But for simplicity let's assume a double sided document that starts with a Right hand page, and ignore those other options for now. Using this information, FrameMaker creates an initial document containing a single Body page set to use the Default Master page (and therefore it uses the Right Master Page), Left and Right Master pages, and a single Reference page, all having the specified page size. It places TextRects on the Body and Master pages to match the specified number of columns, with the specified margins and spacing. If there is to be more than one column on a page, it links the columns within each page, left to right, and places a TextFlow into the TextRect(s) on each page. It sets the Tag of the TextFlow to "A", and turns on the TextFlow's Autoconnect property. The document opens displaying the first body page, which is empty, with the insertion point blinking at the start of the first empty column, ready to accept the user's typing.

As the user adds and edits the TextFlow's content on Page 1, FrameMaker formats that content in real time. When the formatted content becomes too long to fit on Page 1, FrameMaker automatically adds a second Body page, containing TextRects whose layout is drawn from the template TextRect objects on the Left Master Page. It then links the overflowing TextRect on Page 1 to the appropriate TextRect on the newly created Page 2, based on the Tag(s) of the TextFlows involved. In our sample case, the only TextFlow to connect up is TextFlow A. The TextFlow is then reformatted to use the newly added space on Page 2. All of that happens in a fraction of a second, so the user is able to just keep typing/editing, without having to worry about manipulating objects. As the user adds more and more content, this process is repeated, and the document automatically grows to accommodate the content, just like in a simple word processor. FrameMaker does not remove extra pages while the user is editing. But each document has an *Automatically Delete Trailing Empty Pages* property. When enabled, FrameMaker will automatically delete unused empty pages before saving or printing the document.

Now let's explore the properties of a TextFlow. As in the main flow of a standard word processing document, the characters and paragraphs in a TextFlow have a rich set of formatting properties.

The character formatting properties supported by FrameMaker included all of the usual ones (Font Face, Size, Style, superscript, small caps, etc), plus optional automatic pair kerning based on font-specific kerning tables, and tracking.

Character Formats also included a Tag (name) that can be used to indicate the purpose or structure of the formatted text, and used later to apply global formatting changes to a document based on format catalogs that implemented various document designs. More on this later.

In a bit of a kludge, the Character Format applied to text can also specify which human language dictionary is used to spell check and optionally hyphenate that text. Options initially included US English, British English, French, Spanish, German, and Italian, with more languages added over time. This ability to create documents containing multiple languages, without having to deal with erroneous spell checking feedback and inappropriate hyphenation was crucial in winning some important early customers, including the US National Security Agency.

Paragraph Formats each have a tag, and a set of formatting properties that include the usual ones such as first, left, and right indent amounts (relative to the containing TextRect), justification (left, center, right, or full), line spacing, default character format (supporting all of the Character Format attributes described above, including Language), and tab stops (Left, Center, Right, and Decimal-which could actually align on any user definable character). Paragraph formatting also supports optional automatic hyphenation. FrameMaker's hyphenation is dictionary based, although it falls back on an algorithmic approach if it needs to hyphenate a word that is not in the appropriate language-specific hyphenation dictionary and that does not contain user-supplied hyphenation points.

There are advanced paragraph formatting properties to tune the way line breaking decisions are made, especially in the case of full justification. These specify the minimum, maximum, and optimum sizes for interword spacing and for intercharacter squeezing and stretching. FrameMaker uses these parameters in a way that favors the optimum settings whenever possible while minimizing truly awkward line breaks; a design that won praise from the Seybold Report editors when it was released. More importantly, it produced line breaking results that were on a par with those produced by much more expensive, dedicated, high end compositing systems, and significantly better than most of our peers back then, including Interleaf (although they responded to this in a later release).

Paragraph Formats also contain properties to control where column and page breaks could occur: Widow and orphan line control, and whether to keep a paragraph in the same column as its previous or following paragraph.

Paragraph Formats also support a rich Paragraph Numbering scheme based on a fairly simple textually defined numbering specification that was one of the few aspects of FrameMaker's user interface that harked back to markup languages. The feature is flexible enough to support all of the popular ways in which various types of headings (Chapter,

Section, etc.), Figures, Tables, Equations, and other entities were numbered in technical documents of that era.

A paragraph's numbering specification consists of an optional Series Identifier such as "a:", positional counters and counter references indicated by "#", "+", "nn" (where nn is any positive integer), tab and carriage return tokens "\t" and "\r", and other plain text. The numbering spec also can control whether the number is added as a prefix or suffix to the paragraph's text. To see how these elements work, consider a document containing numbered Chapter, Section, and Subsection headings, and also numbered Figure and Table headings. Each of these would have associated Paragraph Formats with those same names, having numbering formats as follows:

Paragraph Format Name	Numbering Specification String
Chapter	+.\t
Section	#.+\t
Subsection	#.#.+\t
Figure	f:Figure +\t
Table	t:Table +\t

The Chapter, Section, and Subsection specifications do not begin with a series identifier, so are part of the "default series". When determining the numbering to display for a paragraph using the Chapter format, FrameMaker would look backwards through the containing TextFlow for the closest prior paragraph that also used the default numbering series, take the value of the first counter from that paragraph, increment it, and display the result. If no prior paragraph was found it would use a value of zero as the prior value. So if the first paragraph of a TextFlow happened to use the Chapter format, its paragraph numbering string would display as "1." followed by a tab, after which the actual text of the paragraph would appear. So, of course the tab stop setting and margin settings for Chapter Paragraphs would need to be configured so that the resulting chapter heading was formatted sensibly. Similarly, the numbering string for a paragraph using the Section format above would be determined by looking backwards for the closest prior paragraph using the default numbering series. The first counter, #, would pick up and use the value of the first counter found in that prior paragraph. If no prior paragraph using that series was found, it would display a zero. But normally a Section Heading would follow a Chapter Heading, so the first Section paragraph in a chapter would find the prior Chapter heading, and pick up the value of its first counter; i.e. the Chapter number. The second counter in the Section format, +, would similarly look backwards, pick up the value of the second counter from the prior numbered paragraph in the series, increment it, and display it. Again, the first Section paragraph following a Chapter paragraph will find no second counter in the Chapter paragraph's number, so it will pick up a zero, increment it, and display the result, or "1.1" followed by a tab, The next Section heading, assuming it follows Section 1.1, will find the

value 1 in the second counter, and increment and display that, to yield 1.2, and so on. But if a later Section heading paragraph finds that the closest prior paragraph in the series is the second Chapter paragraph, the value it finds in the first counter will be 2, and it will not find a second counter in that paragraph's number, so it will pick up a zero and increment that. The result will be to display the number 2.1. You can think of this as the intervening Chapter format resetting all the counters to the right of the first counter. The subsection format works using the same logic as already described. The Figure and Table counters use the same mechanism too, except that, since they each reference their own series (i.e. series "f" or series "t") the values they find and increment when they look back are based only on the prior paragraph in that series. And these formats also display the literal text "Figure " or "Table " prior to the computed number.

The final unusual Paragraph Formatting attributes that I'll mention are Frame Above and Frame below. These attributes can reference named Frames placed on Reference pages in the document, that are typically used to insert graphic elements like lines or filigrees above or below section headings. The height of the frame determines how much vertical space it occupies, and the contents of the frame are simply displayed (like an anchored frame, described below) above and/or below the paragraph text.

Anywhere that a standard character can appear in a TextFlow there can also be any of several specialized embedded objects, including:

- Variable References. FrameMaker documents include a collection of user definable String Variables that can include simple characters, contextually-evaluated place holders for things like "current page number", "current date", or "document modification date", and optional formatting represented by references to named Character Formats. The contextually evaluated value of a variable appears within the TextFlow at the point where the Variable Reference token is inserted. Clicking anywhere within a variable's text selects the entire string, and double clicking anywhere in the text brings up the *Variables* dialog, where users can edit existing variables or create new ones. As you'd expect, changing the value of an existing variable causes that change to be reflected in all references to that variable. Variables are typically used for things like Product Names when documenting a new product whose final name is not yet known, or to insert page numbering and section or chapter names into page header and footer text on Master Pages.
- Cross References and their associated Source Markers. Cross references are much like Variable References, except that their value comes from contextual information drawn from the paragraph containing their associated Source Marker, as driven by a named Cross Reference Format. As an example of how these things play together, tech pubs departments typically have style guides that specify the wording of cross references for different types of things. A style guide might specify that cross reference to a Figure should be of the form "See Figure N on page P in Section S". Or the form for the reference might just be "See Figure N." So at the point where the cross reference appears in the document, the author inserts a Cross Reference entity that points to a specific named marker, and that has an associated cross reference format that might be called something like FigureCrossReferenceFormat.

The user also inserts the uniquely named source marker in the caption text of the Figure being referenced. The user can edit the definition of FigureCrossReferenceFormat if the style guide changes, and all cross references that use that format would change accordingly. A cross reference format can pick up a rich set of information pertaining to the paragraph containing the associated source marker, including the formatted or unformatted paragraph text, its numbering, the numbering of the closest preceding paragraph with a specified paragraph tag (allowing the reference to pick up the number of the Section containing the Figure, for example), and of course the page number on which the marker occurs.

- Anchored Frames, consisting of the anchor point (a zero-width/height marker character that can be placed between any other characters in a text flow), the graphical frame itself (that can contain all of the graphical and text object types described above, including TextRects containing TextFlows), and properties that determined border and fill of the frame, and the placement of the frame relative to its anchor point. Placement options included “in line” (i.e. like an oversized character occuring at that point), below the current line, at the bottom or top of the TextRect containing the anchor point, and to the left or right of the containing TextRect (i.e. in a margin). FrameMaker documents also supported the notion of Left and Right pages, and the AnchoredFrame’s placement could be on the Outside or Inside edge of the containing TextRect, depending on whether the automatic layout of the TextFlow caused the anchor point to fall on a Left or Right page.
- Tables, which consist of cells arranged in rows and columns, and a set of formatting properties that control how the table is paginated (really how it is laid out across multiple TextRects). Each cell in a table is represented by a TextRect containing a TextFlow (with AutoConnect OFF), so each cell can contain all of the elements that we are describing here, including richly formatted paragraph oriented text, variables, cross references, anchored frames, and yes, tables. Anchored frames in table cells that call for placement outside their containing TextRect are “promoted” up the nested objects, to be placed outside the first TextRect encountered that is not in some way nested within a containing TextRect, either directly or indirectly, although users rarely create complex nesting of this type. FrameMaker’s Table objects are designed to handle the automatic layout of long tables that span multiple pages. It can automatically repeat header rows (to repeat the table’s title and column headings) and footer rows, and the user can ensure that page breaks do no occur between specific rows if needed. Table row shading patterns can be specified so that a row’s shading changes appropriately if it moves to a different position relative to the header (i.e. if rows are added or deleted above it, or if a table’s starting location moves due to changes in the text above it, causing a row to shift to a different page. For many of our early customers, these long table features were the key reason that they used FrameMaker over popular and less expensive word processors like MicrosoftWord, which supported fairly complex tables, but, at the time, had no ability to intelligently paginate multipage tables.

Any range of text or other entities in a TextFlow can be optionally hidden or shown based on an arbitrary number of user defined Conditional Text flags.

Unlike a word-processing-oriented document model, FrameMaker supports an arbitrary number of independent TextFlows. A typical office document or novel written with FrameMaker would, of course have one text flow that the author thinks of as the Main flow, but there is no formal concept of a main flow in FrameMaker's model. So it is possible for each of the multiple stories in a newspaper/newsletter/magazine-style FrameMaker document to be stored in separate TextFlows, allowing their pagination to be independent. Isolated TextFlows in single TextRects can also be used for small paragraphs as captions for diagrams placed directly on pages, or within anchored frames.

Now that we've introduced the basic elements of FrameMaker's document model, it should be apparent how FrameMaker readily supports hand-crafted document layouts: Users can simply place graphic objects and TextRects on the page, to create any desired layout, and fill in the TextRects with the desired text. To allow text to flow from one handcrafted page to the next, users can manually add pages, add TextRects, and link them to TextRects on prior pages, to allow the TextFlow(s) to span the connected TextRects.

A weakness that early versions of FrameMaker had when creating handcrafted layouts, compared to programs like PageMaker, was in the area of automatic irregular text runarounds. In those other programs, if you placed a graphic so that it overlapped a column of text, the text would automatically shift to avoid overwrapping the graphic. Early FrameMaker had no such automatic run around feature. It did have a work around feature that could automatically split an existing TextRect into multiple linked TextRects that were each one line high. Users could then use their mouse to drag the left or right sides of each single line TextRect, to avoid overlapping a graphic, and the text would re-wrap appropriately, and look ok, as long as all of the lines of text had the same height. But if the text being shaped in this way included different line heights (such as a a mix of body text and larger title text) the heights of the linked TextRects would not adjust, and the results became unreadable. It took us several years to get around to fixing this, but we eventually added proper support for that feature.

The final type of text object supported by FrameMaker is an equation object (which often includes a mix of textual characters using the Postscript Symbol font, and lines to create things like the line separating the numerator and denominator of a fraction). A FrameMaker equation object contains a hierarchical tree that represents the terms of an equation, in a form that allows FrameMaker to automatically format the equation. FrameMaker treats an equation as an object that can be placed in any Frame, so equations in body text must be placed in anchored frames. To ease the burden this causes, FrameMaker includes a keyboard command that automatically insert an inline anchored frame containing a centered empty equation object, and commands that automatically open up the frame to make more room as terms are added to the equation, and that shrink it down so that it is just large enough to hold the objects it contains (the Shrinkwrap Frame command). In fact every command used to edit an equation has a keyboard equivalent, and the graphical user interface for FrameMaker's equation toolbox was implemented as a small locked FrameMaker document containing hypertext links that

sent the appropriate keyboard commands to the current selection in the active FrameMaker document.

FrameMaker documents also include a Character Formats catalog and a Paragraph Formats catalog. These catalogs each contain sets of named character and paragraph formatting properties, that are not themselves tied to any instances of characters or paragraphs in the document, but which can be applied to those instances individually, or en masse based on name matching. Think of them as representing the default settings associated with each named format. Users can directly format instances of paragraphs or characters, and at the same time optionally update the corresponding Catalog entry and/or all other instances using the same Tag. And users can apply any named format from the catalog to any selected instances in the document.

Users can also, in one step, apply several types of formatting and other properties from one document (such as a template) to any other document or set of documents based on name matching where needed, and the document is reformatted accordingly. Formatting attributes that can be applied in this way included:

- Page size
- The layout and contents of named master pages, which would then be applied to their corresponding body pages
- Paragraph and Character Formats
- Table Formats
- Variable Definitions
- Cross Reference formats
- Conditional text settings.