FrameMaker Tutorial
& Quick-reference Guide

NOTE on this sample:

Encyclopædia Britannica's intellectually sophisticated editorial staff, possessed of widely varying computer experience & receptivity, needed tailor-made documentation for radically customized publishing software.

The introduction and first chapter that follow are part of the training document I researched, wrote, and desktop published for the company. It brought the editorial division quickly and comfortably up to speed—also saving the company some $30,000 in outsourcing to create documentation. The document was still in use six years after it was created, despite massive modification of the original program.

— Judith West

Table of Contents

Introduction

FrameMaker+SGML in Context

FrameMaker+SGML is one part of the New Publishing System being developed to produce The Publishing Company's books and electronic publications. For Core products and projects, FrameMaker+SGML is used to create and modify articles.

- The Editorial and Copy departments will use FrameMaker+SGML in similar ways as they handle text for these products.
- The Composition Department's text-related duties require that they use many of the same FrameMaker+SGML tools (as well as others related to page layout and special formatting for printed products, such as yearbooks).

Ancillary electronic tools that interact with FrameMaker+SGML at certain stages have been developed and continue to be refined:

- Information Management is developing an Index utility, Spectrum, and other tools.
- The Art Department maintains their graphics files outside of FrameMaker+SGML in separate electronic applications.

A separate workflow and document management system will soon be implemented to determine how articles and projects are accessed, tracked, and promoted from stage to stage. This system will be integrated fairly seamlessly with FrameMaker+SGML, as an adjunct rather than a noticeably separate program.

Purposes of the Tutorial and Reference Guide

This Tutorial and Reference Guide focuses on the aspects of FrameMaker+SGML that editors and copy editors will use most frequently. Its chapters will introduce you to basic concepts and physical mechanisms and provide extensive "hands-on" exercises and practice.

The examples cited and exercises offered are based on EB Core files, to provide you a familiar and pragmatic context for learning FrameMaker+SGML's features and operations. It's also hoped that you'll get a sense of how you'll actually use FrameMaker+SGML in your work.

A further aim of this document is to provide a concise reference tool for basic FrameMaker+ SGML operations. To this end, the Table of Contents, an Index, and a Glossary and other Appendixes have been compiled.

Pain and Gain…

In many ways, FrameMaker+SGML is not "better" or "easier" than its Psedit predecessor. It's just different — and new. The transition from Psedit to FrameMaker+SGML will entail considerable frustration. Many editors and copy editors have to relearn a large part of how to do their jobs. To compound the frustration, you'll be trying to learn a new publishing tool at the same time that you have to produce "real work."

No one can make this an easy process. But a basic knowledge of FrameMaker+SGML and some of patience should help you along.

On the "Gain" side, FrameMaker+SGML, as a Windows-based program, provides a widely used, generally familiar electronic venue (or "platform") for your work. Further, it provides more ways of doing various tasks, allowing you to "customize" your work methods in some degree. And finally, since FrameMaker+SGML *is* "SGML-compliant," the products we create can easily be transplanted to such mediums as CD-ROMs and the Web with a greater degree of flexibility than heretofore.

Procedural Notes

- If you don't know basic Windows and word-processing procedures and operations (e.g., creating, opening, and saving documents; using scrollbars and menus; inputting, copying, cutting, and pasting text), see your supervisor about training possibilities. If you're a little rusty and need a refresher, check the Editorial Library.
- Important terms in FrameMaker + SGML are highlighted and underscored here on their first occurrence and can be found in the Glossary in Appendix A (p. **Error! Bookmark not defined.**).

Accessing Tutorial Materials

Before you begin the Tutorial, you'll need to copy the exercises and practice files to your own computer.

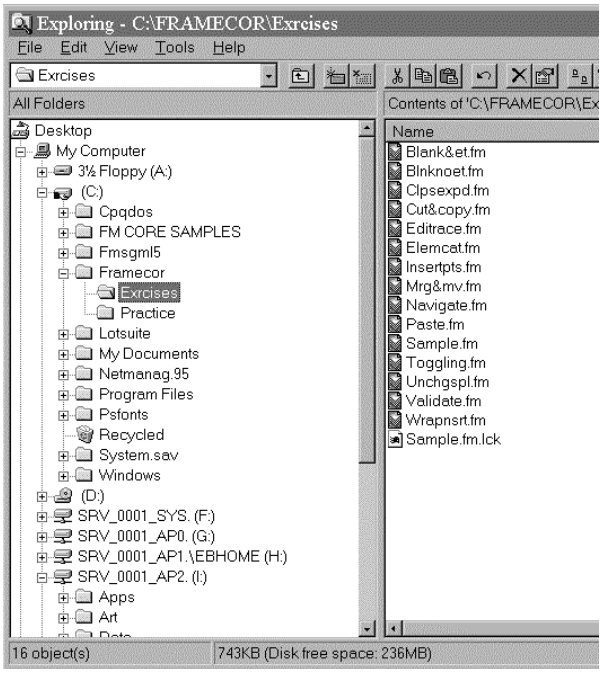
If you know your way around Windows, copy the FRAMECOR folder from the "i" drive (pathway "i:\apps\training\framecor") to your "c" drive.

If you need help, follow the steps in Exercise 1.

EXERCISE 1: MOVING MATERIALS TO YOUR COMPUTER

1. **Click the Start button at the bottom of your screen, go to Programs, and click Windows Explorer.**

   The Figure at right illustrates the main drives and many of the folders you'll be dealing with. Your Windows Explorer may look somewhat different.



2. **In the left-hand pane, double-click the "i" drive, then the APPS folder, and then the TRAINING folder.**
3. **In the right-hand pane, click the FRAMECOR folder and copy it (*SHORTCUT*[✕]: Control+c).**
4. **In the left-hand pane, click the "c" drive and paste the contents of your Clipboard to it (*SHORTCUT*: Control+v).**

   The Tutorial materials are now on your hard drive in a special FRAMECOR folder.

   You're now set up for the Tutorial.

Exercise Files

Exercises for this Tutorial are in the EXRCISES subfolder of the FRAMECOR folder. The following explains how to open the exercise files when you're asked to do so during the Tutorial.

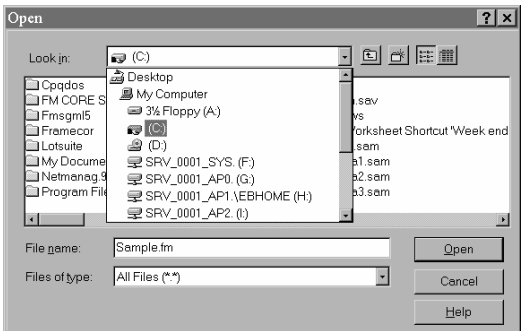
[✕] The + (plus sign) in a ***SHORTCUT*** indicates that you must hold down the first key while pressing the next key (or while carrying out the next action).

Experienced users: open the FrameMaker+SGML application from your desktop shortcut icon or from the Windows Start button. The path to the exercise files is c:\framecor\exrcises*.

The rest of us, do the following:

EXERCISE 2: OPENING FRAMEMAKER+SGML EXERCISE & PRACTICE FILES

1. **Click the FrameMaker+SGML icon on your desktop, or open FrameMaker+SGML from the Windows Start button (go from Programs to Adobe to Adobe FrameMaker+SGML 5.5).**
2. **Under the File menu, select Open (or you can click on the toolbar icon  for "Open File").**



3. **In the dialog box that appears, click in the box to the right of the "Look in:" message. Scroll until you find your "c" drive, then click on it.**
4. **Double-click on the FRAMECOR folder, and then on the EXRCISES folder.**
5. **Double-click the filename specified in the exercise you're working on.**

If you need to stop during an exercise, save your changes in the document you're working on and mark the spot in the Tutorial and Reference Guide where you stopped. You can pick up again later.

Practice Files

At step 4 above, you may also have noticed a PRACTICE folder.

The files in this folder are:

1. *NOTAGS.FM* — an untagged sample Core article (with Edit Trace enabled),
2. *TAGSAMP.FM* — a tagged sample Core article (with Edit Trace enabled), and
3. *BLNKPG.FM* — a blank page (without Edit Trace), ready for you to create your own original Core article.

You should use these anytime you want to experiment further with any of the features and operations presented to you during the tutorial. Occasionally, a 

suggestion will appear as a reminder of these opportunities. You're strongly advised <u>not</u> save your changes to these Practice files. That way, using them for later practice sessions will be a lot easier and less confusing.

Should the EXRCISES or PRACTICE subfolder "disappear" from your dialog box at any time, just use the Up One Level button  to display the FRAMECOR folder, and "drill down" to the subfolder from there.

Chapter One:
How FrameMaker+SGML Works

Some Working Definitions

document, or **text document:** a unit of text — e.g., an article, a section of an article, a bibliography — designated for electronic handling.

file: the electronic container for the text document**.**

Core: the data (articles, associated text and art, indexing, etc.) from which all electronic products are drawn. FrameMaker+SGML handles the text portion of Core data for works-in-progress.

Introduction

This chapter provides an overview of the FrameMaker+SGML publishing system. It describes *what* FrameMaker+SGML does and some of its features, and it introduces important terminology. You'll learn the *how-to* later, in Chapters Three through Five.

If some of the concepts presented here are less than perfectly clear, don't be too concerned. They'll be reintroduced in different ways as you proceed through the Tutorial, and you may find some of these later approaches easier to grasp.

There's a lot to present and assimilate, so be patient with yourself and the whole process.

Markup

Markup is usually some kind of code that conveys extra, nonsemantic information for processing a text.

Different kinds of markup accomplish different kinds of processing.

- Punctuation is one familiar system of basic markup for a written work.
- Editors classically think of markup as coded instructions (such as ¶, ital., 9 pt. Helvetica, 6 pts b/b, indent one em) for typesetting and printing a text:

- With the introduction of electronic publishing, editors have become familiar with "markup languages" — i.e., coding that directs how a computer should process a text. The following table provides some examples of electronic markup with which you may be familiar.

Early electronic composition systems could be quite heavily coded with markup, requiring specialized knowledge.	**EB's previous PSEDIT system provided user-friendly handles for some — though not all — markup coding.**
❖ Changing from regular roman font to boldface, then back again, required markup at each step: <RM>…<BO>…<RM> ❖ Several commands — to generate, for example, proper type size, typeface, and indentation — could be amalgamated into one set of start and end markup, masking the more extensive underlying programming (e.g., the title coding between solid triangles, below). ❖ An article title, fully marked up, looked like this: <RM>◣UFT◥ <DX>MT<BO>iron<DX>TM<RM>◣UFET◥	❖ Shift+PF19 (a function key) generated underscored text on-screen, which printed as italic text. ❖ Shift+PF21 generated blue characters to signify boldface text. ❖ Coding governing vertical and horizontal spacing could still be rather dense — for example, <ju>  <pv .5pt> was required to add ½ point of leading between lines.

Some markup coding doesn't specify *how* things should look in a document but instead describes *what* is in the document (i.e., its content).

In many word-processing systems…	**In PSEDIT…**
❖ "styles" — like Title, Heading1, Heading2, etc. — are markup that not only dictate appearance but also let you classify organizational components of text. (Outline View in most systems shows you this hierarchical facet.) ❖ notes and cross-references based on complicated coding and programming are invisible to the user, who simply presses buttons and answers dialog box questions.	❖ apart from their visual effects, markup like <ti><eti>, <h1>$^{U}_{D}$, <quote><endquote>, and <biblio><endbb> labeled different functional components of an article. ❖ index hooks and art tags were markup that acted as electronic markers or pointers. ❖ turquoise characters were used to code electronic-only text.

FrameMaker+SGML is a publishing tool that uses the rules of a content markup system called SGML — Standard Generalized Markup Language — to handle and maintain complex, tightly structured publications like those that EB produces. In fact, FrameMaker+SGML's markup is based on SGML concepts.

To be used to produce EB's various books and electronic products, FrameMaker+SGML requires that an EDD — an Element Definition Document — be written for each product. The Core EDD is a set of "rules" that describe and classify every part of the text of an article, the basic Core unit of production. The EDD also accounts for the correct order of all these parts and, in a very few cases, aspects of their appearance.

This Tutorial and Reference Guide introduces the basics of FrameMaker+SGML in light of our electronic Core products.

The “Frames” of FrameMaker+SGML

FrameMaker+SGML is designed to separate various flows of printed page components that need to be manipulated independently — e.g., text, art, tables, captions. It designates rectangular boxes, called frames, for each of them. This allows FrameMaker+SGML to provide a WYSIWYG effect (*W*hat *Y*ou *S*ee *I*s *W*hat *Y*ou *G*et) to facilitate the printing of books.

For Core articles, however, there is only one frame containing one text flow. Without the spatial constraints of the printed page, Core files are simplified.

Associated non-text components of Core articles — specifically, art and maps — are viewable within a FrameMaker+SGML article as a separate window, reached by a special typed command.[✂]

Structure and Formatting

Structure and formatting represent two different aspects of a text document: its organizational content and its appearance.

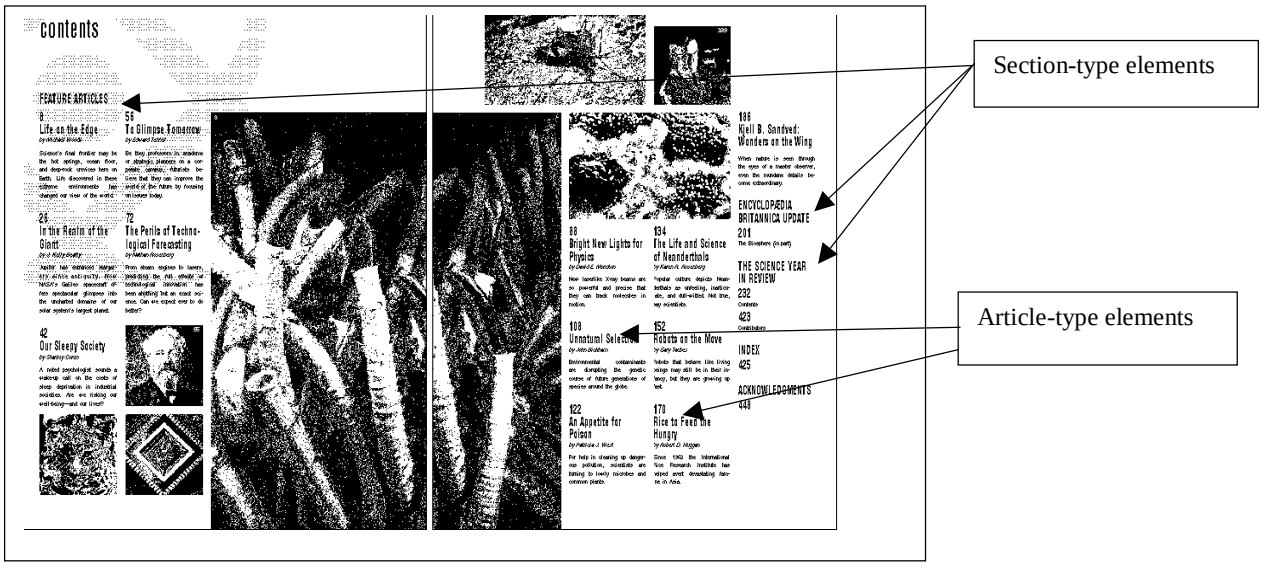
Though these two aspects may overlap, FrameMaker+SGML treats them independently of each other.

Structure

Structure refers to text in terms of the organization of its content.

Using a product’s EDD, FrameMaker+SGML structures text documents based on units of content it calls elements. Elements categorize different kinds of text by recognizing and labeling the function that each serves within a written work.

For printed publications, the most broadly defined of these elements make up what we recognize as a book. For example, a book’s table of contents reflects the structure of that book’s highest-level elements.



[✂] The ability to access art images and captions/credits from within a FrameMaker+SGML text document will not initially be in place for Core articles, though those files *will* be accessible. As information on these means becomes available, you’ll be informed.

For Core, however, the breakdown is more fundamental. All the elements used to structure Core data are based on the functional parts of an article. The highest-level elements you'll work with in a Core document, then, correspond to articles — though you might at times work on a document constituting one or more consecutive parts of a complex article, or a bibliography only, rather than an entire article.

FrameMaker+SGML continues to refine structure down to the paragraph level, and to even smaller elements within paragraphs. (On the "molecular level," so to speak, fractions and super- and subscripts are designated as elements.)

For example, when you work on a file containing a short Core article, FrameMaker+SGML treats "article" as the highest element and determines all its possible subelements (title, paragraphs, cross-reference links, etc.). For each of these elements, FrameMaker+SGML provides a descriptive label called an element tag (see Figure 1, below).

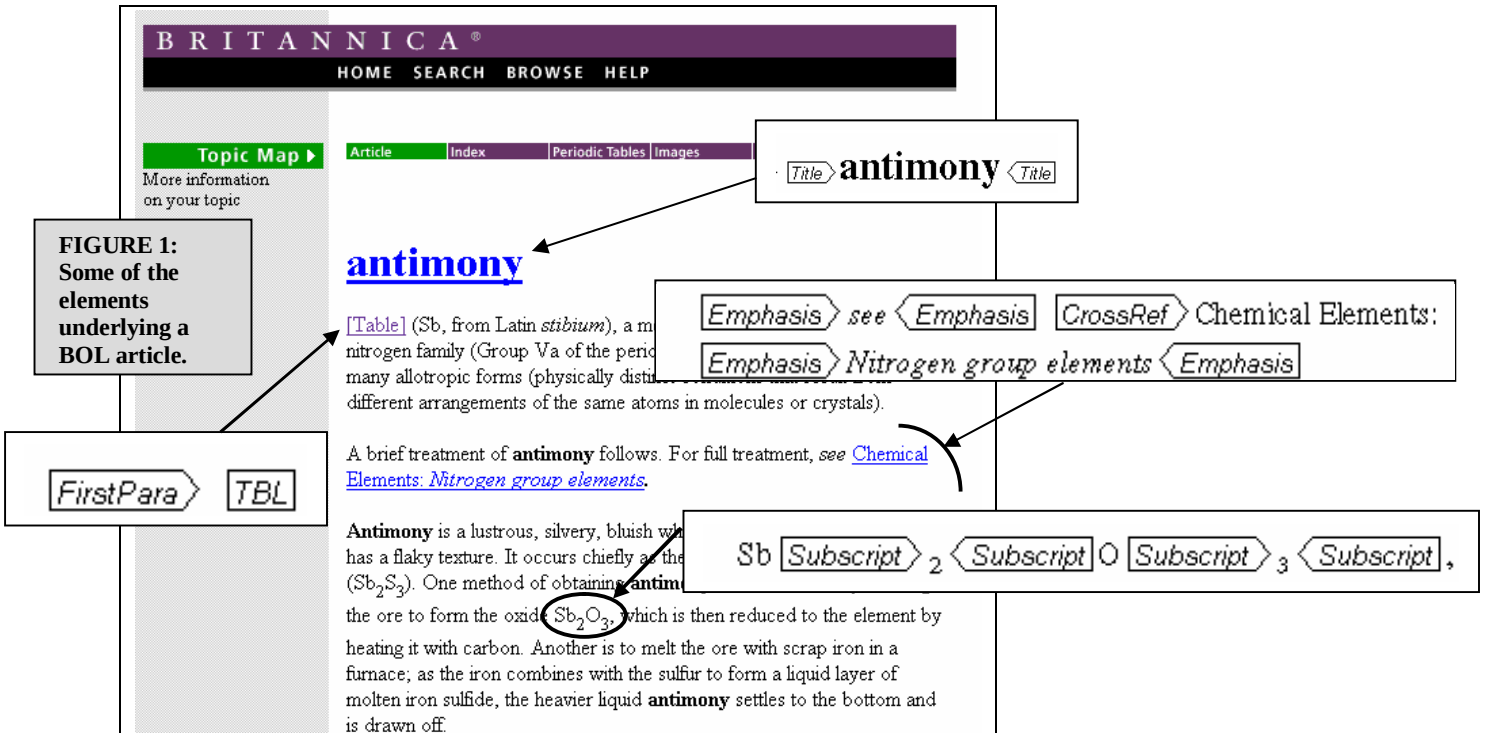


FIGURE 1: Some of the elements underlying a BOL article.

FrameMaker+SGML then organizes these elements according to their prescribed sequence. Through this organization, it guides you to "build" a structured article according to the established rules of what makes a valid Core article.

NOTE: For a list of the element tags available for Core at this writing, see Appendix B, p. **Error! Bookmark not defined.**.

Two Representations of Structure in FrameMaker+SGML

FrameMaker+SGML gives you two ways to view and work with the elements of an article's structure: the Document Window and Structure View.

Figure 2 (p. 12) shows a Core article as it appears in FrameMaker+SGML's Document Window. This window presents the article as a nested structure of element tags that contain either text or other element tags.

Nested within parent elements (for example, Article) are child elements (e.g., Title, TitleVariant, FirstPara, IndexRef). As Figure 2 illustrates, some child elements may themselves become parent elements containing text and further child elements. For example, the FirstPara is a child of the Article element but a parent of an Emphasis element and some IndexRef elements.

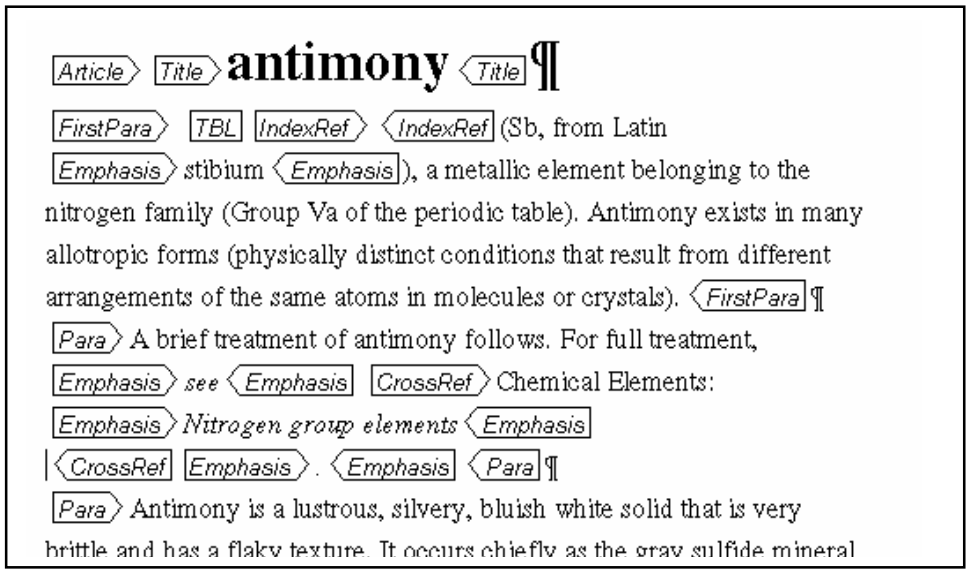


FIGURE 2: Tags in the Document Window

Higher-level parent elements in a text document are ancestors to all the child elements subordinate to them; the latter are called descendants. In Figure 2, all elements are descendants of the ancestor Article element. (You can see only the Article start-tag here; its end-tag is the very last element of the text document.)

Some elements, called empty tags, consist of only one object (for example, RefPoint and ARTX). These elements function most often as placeholders (e.g., for the "destination" end of a cross-reference) or pointers (e.g., to art stored elsewhere).

Structure View (Figure 3) graphically displays an article's structure as a family tree, borrowing the indentation conventions of outlining to represent one element's subordination to another.

FrameMaker+SGML's Structure View tool focuses on showing the organization of your article's parent and child elements, with only enough of a text snippet to orient you to the article itself.

Structure View is perhaps most useful as a navigation tool, to let you see where you are in an article and get where you want to go.

During FrameMaker+SGML's validation operation, Structure View helps you determine that you've included in your article all the required elements in the proper order.

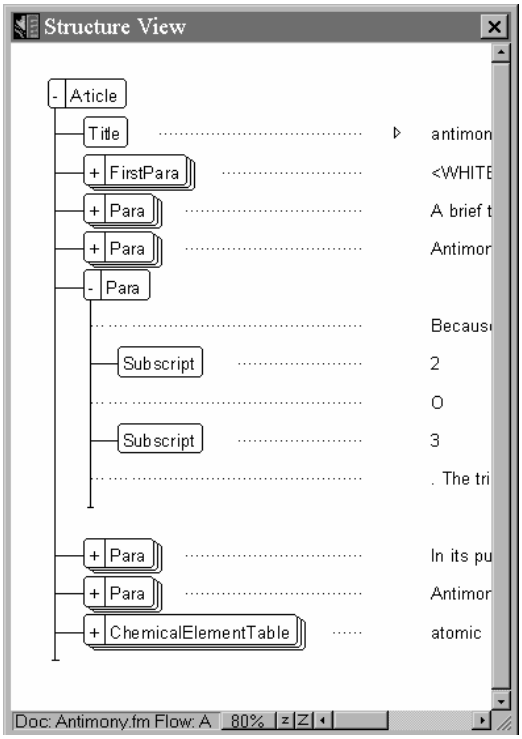


FIGURE 3: Elements in Structure View

Attributes (Part One)

Some elements also have properties called attributes. These attributes carry information about their elements apart from content.

Each IndexRef element, for example, has a Type attribute (indicating what variety of index link it is) and a Reference attribute (a numerical identifier relating it to the Index utility). CrossRef elements also have a Reference attribute. These attributes serve administrative and maintenance purposes — in fact, that basically defines their existence in FrameMaker+SGML.

Another kind of attribute connects an article's content with its formatting and is discussed below.

Formatting

Formatting refers to text in terms of its appearance.

In editing and copy editing, formatting instructions designate the visual properties of text — e.g., indentation, font, point size, leading, typeface, underscore, and gutter width. The electronic mechanisms that generate those properties are also called formatting.

But because Core articles are not being prepared for a specific print publication — or even a *single* electronic publication — Core article elements carry relatively little by way of formatting.

All elements in FrameMaker+SGML, as it's used for Core data, classify or have some direct relationship to an article's content. Most element tags (except for empty tags like Index Ref) also contain format rules, which allow visual distinctions to be made between different elements while you edit. In Figure 2 (p. 12), for example, most of the elements you see have the same very basic format rules to produce a "generic" text — though a few (e.g., Para and Title) also create paragraph breaks, and even fewer (e.g., Title) provide typeface distinctions.

FrameMaker+SGML's Core template provides a base document for Core data that regulates your editing environment. It:

- lays out your wide-column manuscript page,
- activates the formatting potential (i.e., format rules) of various elements, and
- makes available all the element tags you'll need to structure any Core article's text.

Don't, however, get too attached to the formatting you see as you're editing. While it apes some of the formatting you're used to seeing (e.g., boldface titles) to give you a familiar, orderly working environment, it has absolutely no *causal* relationship to the final "look" of any Core article. It's the *tags* that are manipulated to evoke an article's ultimate appearance. As far as SGML-based electronic publishing is concerned, your article could be produced just as effectively from a continuous mass of text characters interrupted by tags. It's FrameMaker+SGML, in a sort of diplomatic capacity, that makes sure SGML gets what it needs and that you get (basically) what you need to do your work.

Formatting for Core articles, therefore, is not designed on the WYSIWYG principle (*W*hat *Y*ou *S*ee *I*s *W*hat *Y*ou *G*et), as print-oriented yearbooks are. As long as Core data is tagged for structure and function, its visual dimension can be manipulated for many different venues. For example, as long as an article's title is tagged as such, it might be formatted with different fonts for BOL and for a Web site without any manipulation of the article itself.

There's one important exception to some of the above generalizations about formatting…

Attributes (Part Two)

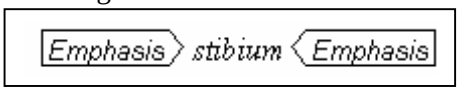


FIGURE 4:
Emphasis tags with italic attribute

A few elements require that you provide more integral formatting information — that is, information about the way some kinds of material should *always* appear. For example, published book titles — regardless of their venue, should always be italicized in EB Core products. And certain components of scientific and mathematical notations must appear in italic or boldface text.

You embed this kind of styling by defining format-related attributes for these elements.

Format-related attributes are based on style decisions, which in turn reflect distinctions in the *functions* of certain elements. Thus, format-related attributes dictate general, function-based aspects of appearance (e.g., italic text) rather than enforcing detailed, design-based specifications (7 point Times Roman on 8.3 pt leading). This is in keeping with the content-centered focus of SGML — and, by extension, of FrameMaker+SGML: the venue that "receives" one of our SGML documents (prepared in FrameMaker+SGML) has a wide latitude as to how the various elements will actually look, beyond certain basic (attribute-defined) qualities.

The most common format-related attributes are those associated with Emphasis tags. A special attribute dialog box requires that you specify what kind of emphasis you want: italic, boldface, or small caps.[⌘] Other format-related attributes come into play with such elements as Fraction (type of stacking) and ChemicalElementTable (number of columns).

In sum, the FrameMaker+SGML mantra is: "All formatting is illusion. Only tags have reality."

[⌘] Eventually the general Emphasis element may be further refined as more content-specific elements. For example, BookTitle, Film, LinguisticEmph, and ForeignWord might be element tags that each represent a different functional component of text distinguished by italics. Such a breakdown would certainly be more useful for electronic identification and retrieval purposes and could allow an even greater range of text design options for various electronic venues.

Edit Trace: An Addition to FrameMaker+SGML

Edit Trace is a tool developed in house to augment FrameMaker+SGML by creating the electronic counterpart of a "paper trail." It provides a means of accountability and preserves the systematic checking and incorporation of text changes that we rely on for quality control.

Edit Trace is "enabled" and "disabled" (i.e., turned on and off) at designated points in an article's handling. Its system of color coding tracks additions and other changes by department and by the individual who made the changes. Deletions (which are preserved as strike-through text) can be hidden for easier on-screen reading.

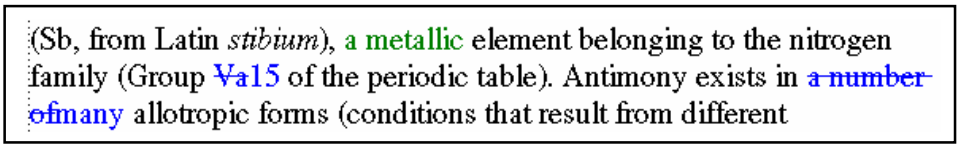


FIGURE 5:
Deletions (as strike-through) in Edit Trace

When Edit Trace is disabled, the changes made since it was turned on are accepted and incorporated as regular black text. No further text corrections can be made when Edit Trace is disabled (unless, for some reason, Edit Trace is re-enabled).

Because Edit Trace is new and is being tailored to meet Core editing needs, it will be undergoing considerable changes as our use of and familiarity with FrameMaker+SGML develops.

How SGML Fits In

Early in this section, we described the following kinds of text markup:

- **Basic markup,** such as punctuation;
- **Editorial markup,** the classic proofreading and printing symbols and instructions;
- **Markup languages,** which augment (and sometimes replace) editorial formatting markup with computer coding; and
- **Generalized markup,** specifying (with "tags") not *how* a text should look but *what* constitute the pieces of its content and their relationships.

All of which lead up to…

- **SGML —** Standard Generalized Markup Language.

SGML itself is not so much a generalized markup language as it is a "metalanguage." It provides a set of very basic rules or conventions that can allow vastly different applications to understand each other — a kind of "universal translator" for electronic text programs.

SGML lets people using different applications and electronic systems exchange text and information about their documents without losing or garbling information because of their individual differences.

SGML's tagging system is the basis for the way that FrameMaker+SGML tags content with element tags.

Furthermore, anything beyond a certain *very* basic set of characters that you want to be recognized outside of your own electronic environment must be accounted for in SGML-style markup. This is one reason for the large number of elements in FrameMaker+SGML. For example, if you want your text to read "$13\,{}^{15}/_{26}$" rather than 1315/26, you have to account for that with SGML markup.

- Fortunately, **FrameMaker+SGML** handles and maintains SGML structures for you. So you don't really have to "learn" SGML!

FrameMaker+SGML is "SGML aware." But its system of element tags based on SGML conventions have been made much more user friendly. When a Core article has been completely edited, proofread, illustrated, and indexed, it's returned to the Core database and stored as an SGML document until it's wanted again for revision.

If you *really* want to know about the workings of SGML, take a look at:

- "A Gentle Introduction to SGML" on the Net at http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG
- the introductory material in the online manual called *FrameMaker+SGML Developer's Guide* available from FrameMaker+SGML's online Help window.

NOTE: You'll sometimes hear talk about DTDs, or Document Type Definitions. These are the SGML instruction documents that spell out the rules for constructing an SGML text document (e.g., a Core article). DTDs can be very intricate — and *very* challenging to read — since they must account not only for structural elements but for any special characters and other important peculiarities of your document. The SGML text documents built using the DTDs and coded for SGML (especially complicated EB text documents) are extremely dense.

FrameMaker+SGML's Core EDD (see p. 2) is developed based on the Core DTD. But FrameMaker+SGML, for all its extensive tagging, is much easier to deal with.

The Transition to FrameMaker+SGML

Stepping into a new publishing system is inevitably unsettling. FrameMaker+SGML is not innately better or worse than other systems. Different people will find different aspects valuable or annoying, and you'll doubtless have heard a little of each. Here are some pros and cons.

Some Good Things…	Some Challenges…
• Combining structure and formatting into tags makes markup less abstruse. • The many different navigation, view, and text-handling options answer to a range of different circumstances, needs, and preferences. • SGML-tagged text translates to different electronic venues (e.g., BOL, BCD, Websites) without any change to the original text documents. • Specific elements can be easily retrieved, based on their tags, for various administrative purposes (e.g., title lists, contributor lists). • Guided Editing and validation help ensure that all the parts of a publication are present and appropriately placed.	• FrameMaker+SGML is so different from Psedit — and from most word-processing applications — that it takes a while to learn and adjust to it. • To meet EB's specialized needs, FrameMaker+SGML must be extensively customized and enhanced. Since this is happening *while* we're using it, a degree of disruption and frustration can be anticipated during transition. • Editors and copy editors will now do many of their markup duties online. This means added work steps, in some cases. • Many EB products are so highly structured that initial tagging may be a lengthy process.