# Software we have lost — the mortar that held the bricks together

Peter Flynn

Abstract

Since the first days of SGML, there has been a variety of software to parse, validate, analyse, format, store, search, and extract the information. Some of this was what we now call 'Open Source', particularly the smaller utilities, but the majority of applications were conventional commercial offerings.

In the course of time, many of these have become unavailable, for assorted reasons, with the result is that some very useful systems have been lost, and replacements are not always as effective.

This research attempts to catalogue and analyse a collection of XML and SGML software that is either off the market, or only available within a different product, and thus not accessible to users. The objective is to see if there are still ways to 'shorten the distance between the bricks' that are not otherwise provided for.



## Contents

I am grateful to the numerous people in University College Cork and elsewhere who stepped up with offers of Windows and other installer CDs when my carefully-preserved originals went missing, including (alphabetically) John Barrett, Roy Cummins, Stephen Dineen, AV Drepe, Martin Fleming, Nick Hogan, Sinead Horgan, Steve King, Margaret Lantry, Piaras MacEinri, Neil Nash, John O'Connell, Michael O'Halloran, Billy O'Rourke, Bereniece Riedewald, Joel Walmsley, and Frank van Pelt. Thank you also to the SGML-era veterans who prompted me with the names, details, or disks of long-forgotten products, especially Debbie Lapeyre, Lauren Wood, and Michael Sperberg-McQueen

# 1. Background

Before we had XML, we had SGML, which is the ISO standard (ISO JTC 1/SC 34, 1985) on which XML is based. By default, SGML is pointy-bracket markup like XML...but in SGML almost everything can be re-defined, including the markup characters themselves, and there are numerous options for additional features and for abbreviations and markup shortcuts to minimise typing. Any changes to syntax or to the configuration value limits have to be made in a Declaration file, and a DTD is compulsory on every document.

Notice the words 'to minimise typing'. In the beginning, there was no software with an editing interface that could use the DTD to provide a contextual menu of available element types; and the idea that the markup would be hidden from the user was counterintuitive — if you couldn't see the tags, how could you know what was marked?

In these early stages, therefore, markup was applied by typing it in a plaintext editor, so *the* essential piece of software to begin with was the parser, not the editor, so that you could check that you hadn't got something wrong. The term 'parsing' was often used to mean 'parsing-and-validating';[1] as there was no concept of well-formed 'tag validity' in the sense introduced with XML. There were many early parsers; among the most significant were:

- *ARC SGML* (Almaden Research Center) originally by Charles Goldfarb; later developed by James Clark into *sgmls* (see below);
- *ASP SGML* (Amsterdam SGML Parser), still available (Warmer & Egmond, 1989);
- *Exoterica* by Sam Wilmott (later included in *Omnimark*);
- the parser in *Framemaker+SGML* by Lynn Price;
- a parser for Boeing (internal only) by Greg O'Connell and Debbie Lapeyre;
- *Mark-It!* by Jean-Pierre Gaspard;
- *sgmls* by James Clark, the only one still in widespread use; redeveloped as *nsgmls* for *SP*, and now as *onsgmls* to handle XML for *OpenSP*.

Other software developed rapidly, spurred partly by the adoption of SGML for some military documentation in the US and elsewhere, and partly by its growing use in publishing, research, and academia. Editing software included:

- Arbortext *ADEPT* (through several name changes (eg Epic), now *PTC Arbortext Editor*);
- SoftQuad *Author/Editor* and the editors based on it, *HoTMetaL* (for HTML) and later, *XMetaL* (for XML);
- STiLO *Document Generator*, with *Arbortext* one of the few to handle mathematics in a general-purpose SGML editor;
- *Emacs* with *psgml-mode*;
- *epcedit*, a free SGML and XML editor from tkSGML;
- the *Euromath Editor*, an EU project built on the GriF editor[2]
- Siemens Nixdorf *InContext*;
- Citec *MultiDoc* Translating Editor;
- Microstar *Near&Far Author for Word* and *Near&Far Designer*, a graphical DTD editor;
- GriF *SGML Editor*;
- Richard Light's *SGML Tagger* (OUP), a memory-resident monitor for MS-DOS editors.

[1] In fact, in the authors' description of the *Amsterdam SGML Parser* (Warmer & Egmond, 1989), the only instances of the term 'validation' are in the formal references to 'validation *services*' in the SGML standard itself.

[2] The editor is reputedly being resuscitated and rebuilt using INRIA's *Thot* structured editor.

- Corel *WordPerfect* had a built-in SGML editor;
- Sema *Write-It!* (using *Mark-It!* as the parser).

Documents also need processing in some way: adding to a database, putting on the Web, mining it for data, or converting it for a formatting system for publishing. Conversion or processing (transformation) systems included:

- AIS Software *Balise*;
- DFN *DAPHNE* (VMS only; converted to TEX);
- EBT (later Inso) *DynaText* trainable converter from *Word* to SGML;
- James Clark's *Jade* (using DSSSL) can convert to TEX and other formats;
- Exoterica *Omnimark* (XTRAN);
- Microsoft *SGML Author for Word*, despite its name, this was *not* an editor, but a converter into and out of *Word*.

There were several standalone viewers, especially for vertical-market applications, but few general-purpose browsers. As with editors, some used SGML-syntax stylesheets to format the display; others used proprietary stylesheet syntax. Formatting systems for printed output typically produced *Postscript* (pre-PDF days). Some handled SGML input direct, others via an established conversion route; output was formatted using TEX or a proprietary typesetting engine.

- Advent *3B2* typesetter;
- EBT *DynaWeb* NT server for documents converted with *DynaText*;
- Adobe *Framemaker+SGML* typesetter (FTC's original had no SGML support);
- LATEX, typesetter, usually via transformation through *Omnimark*, *Balise*, *Jade*, or similar;
- Citec *MultiDoc Pro Publisher* standalone browser;
- *Panorama Viewer*, an SGML plugin for the *Mosaic* and *Netscape* browsers; also the standalone *Panorama Publisher*;
- Arbortext *Publisher* typesetter.

There was far more software available which is outside the scope of this report — some of it is now either uncompilable or uninstallable, or was in any case incomplete or experimental at the time. A significant amount was normal commercial software which has suffered the conventional fate of being superseded, falling out of use, or being abandoned when the company failed or was taken over. There are extensive lists of both free and commercial applications in Robin Cover's SGML/XML Web Pages, and some of the SGML Conference CDs have a considerable amount of freely-distributable and commercial-sample software in subdirectories..

Other categories not covered here include design tools, search engines, and databases. The only three of these of which this author has direct experience (noted below) were Microstar's *Near&Far Designer*, Tim Bray's *PAT* search engine in section 2.5.2 on page 16, and the *SGML DARC* document management database in section 2.5.3 on page 16.

# 2. Software

The sections below refer to those programs and systems which were *either* installed and [re-]tested for this paper *or* were tested and documented in the author's *SGML Toolbook* (Flynn, 1998). The following symbols are used:

☑ a checkmark beside an item denotes that it installs and executes correctly;

☒ an X denotes that the software exists but cannot be compiled or installed correctly, so testing it was not possible;

◯ a circle denotes that the software could be installed but either would not execute, or executed but with unresolvable errors;

☐ an empty box denotes the software is no longer available.

The platform used for testing Windows and MS-DOS software was Windows XP SP2 running on a Dell Optiplex 745.[3] The objective was to emulate as reasonably as possible the office environment *circa* 1998–

[3] An attempt was made to use Windows 95 but this satisfied the requirements for only the oldest programs.

2002. A modern Linux distribution (Mint 19) was used for the few UNIX or GNU/Linux utilities. The procedure for testing was:

1. Install the software from original media where possible, or from zip archives from network repositories;
2. Run the relevant program[s] from the Start menu or from an installed icon (a few command-line procedures were run from the Windows Command terminal);
3. Open or otherwise invoke the sample SGML document (see Appendix A on page 19), performing any necessary prerequisites such as making the DTD or SGML Declaration available to the program;
4. Exercise the features or functions of the software to check they operate correctly (eg `Insert Element`, `Edit Attribute`, `Validate`, etc);
5. Record details of success or failure.

## 2.1. Parsers and validators

Three of the products listed in the list on page 2 were tested.

### 2.1.1. *ARC SGML* ◯

This software was installed from the copy distributed on the SGML'97 CD-ROM. However, the `INSTALL` and `readme` files referenced binaries that were not included, and attempting to parse any of the test suite resulted in references to components that were not recognised. However, the `vm2` program did appear to execute, but failed to parse the sample file, returning error messages related to capacities, and it was not clear how the values from the SGML Declaration could be implemented within the time available.

### 2.1.2. *ASP SGML* ☒

Several attempts were made to compile this parser from the source code available in `Sgml.tar.z` but the changes in the C libraries over the years mean that significantly more work is needed to recode those parts which currently generate errors. While this would be an interesting excursion for a student coding project, there is probably little to be gained by resuscitating the software for production use when *onsgmls* fulfils the same function.

### 2.1.3. *sgmls* ☑

Derived from *ARC SGML*, the parser was rewritten as part of the *Jade* DSSSL processor (now *OpenJade*). This is a standalone (commandline) parser with options (among others) for:

- suppressing the default ESIS output (a markup-and-data stream) when just a validity check is required;
- limiting the number of error messages;
- switching to XML mode (*onsgmls* only);
- specifying the SGML Declaration to use.

*sgmls* is also used by the *sgmlnorm* normalizer to expand SGML shortcuts; and by the *Emacs/psgml* editor for validation. The 32–bit Windows binaries used in this test were installed from the SGML'97 CD-ROM.

```
C:\SGMLS\> sgmls -s sgml.dec recipe.sgml
```

In the case of this, and most command-line parsers, a null return means no errors were found in the DTD or document.

The companion *sgmlnorm* utility was also exercised: the normalization fills in all the missing parts of the SGML document by applying the rules from the SGML Declaration on what end-tags may be omitted and if attributes may be minimized (and much else). In this case the following output was obtained:

```
C:\Documents\> sgmlnorm sgml.dec recipe.sgml
<RECIPE>
<TITLE>Chocolate fudge</TITLE>
```

```
<COMMENT>My mother's recipe</COMMENT>
<INGREDIENTS>
<INGREDIENT QUANT="1" UNITS="LB">sugar
    </INGREDIENT>
<INGREDIENT QUANT="4" UNITS="OZ">chocolate
    </INGREDIENT>
<INGREDIENT QUANT="?" UNITS="PT">cream
    </INGREDIENT>
<INGREDIENT QUANT="1" UNITS="OZ">butter
  </INGREDIENT>
</INGREDIENTS>
<METHOD>
<LIST>
<ITEM>Mix the ingredients in a pan
      </ITEM>
<ITEM>Heat to 234?F, stirring constantly
      </ITEM>
<ITEM>Pour into greased flat tin
      </ITEM>
<ITEM>Allow to cool before cutting
    </ITEM>
</LIST>
</METHOD>
<SOURCE>Adapted from the Good Housekeepings cookbook</SOURCE>
</RECIPE>
```

Note that by default, SGML is case-*in*sensitive, and that the pretty-print indentation of the original document means that normalizing the omitted end-tags has introduced a white-space node into the character data content of the `ingredient` and `item` elements.

## 2.2. Editors

Of the editors in the list on page 2, *Document Generator*, *SGML Tagger*, *Euromath Editor*, and *Write-It!* were not reviewed either due to lack of software or incompatibilities.

### 2.2.1. ADEPT ☒

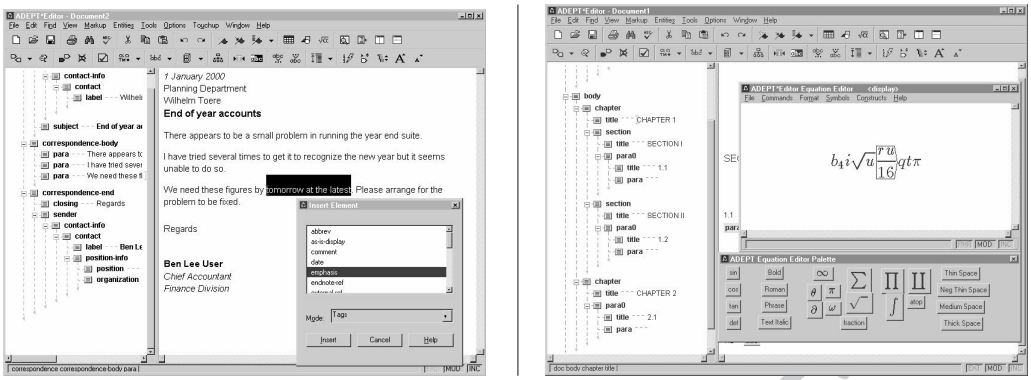
The Arbortext editor is perhaps the best-known industrial SGML and XML editor. It was (is) a very large suite, including the *Document Architect* program for compiling DTDs and FOSI stylesheets, and *Publisher* which output typeset-quality formatting (printing from the editor itself was more office-quality). Setting up the suite takes a long time, especially if many custom DTDs and stylesheets need to be installed: this is not a simple editor but a fairly complete document production system — the *Author/Editor* and *Panorama Publisher* setup had similar capabilities, but the editor could also be installed and running in a few minutes, which is not the case with *ADEPT*. It was also *expensive*: in 1992, the present author was quoted $5,000 per seat and no discounts.

A strong feature of *ADEPT* was mathematics (in an earlier incarnation, Arbortext also sold a commercial version of TEX, and the TEX engine remained inside their products for many years). The graphical interface to mathematical editing (see Figure 1 on the following page) was a *de facto* industry standard which was only challenged by LyX and more recently *Word*'s *Equation Editor*.

As with *Author/Editor* and *RulesBuilder*, the principle was to separate the business of editing — the job of professional technical writers — from the business of maintaining the DTDs, many of which were from industrial vertical markets and subject to strict data controls over who could change what and when and how. One interesting approach, not seen in any other product of the era, was that after a DTD was successfully compiled for the first time, the system would ask for the element type names used for:

- the document title, title block, and title page;
- normal paragraphs;
- graphics, and attribute names for assorted graphical manipulation features;

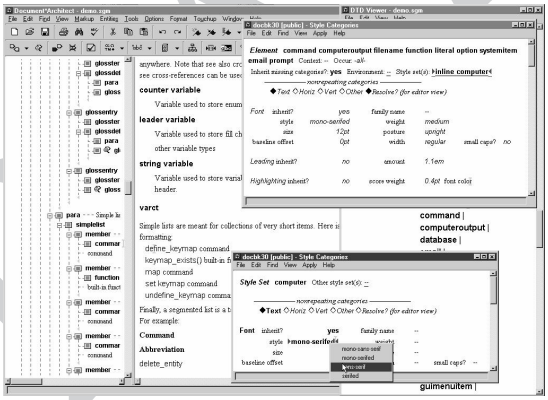
Figure 1: *ADEPT* showing the Insert Element dialog (left); and editing mathematics (right)



- divisions (chapter, section, subsection, etc);
- lists (numbered/bulleted/definition);
- figure blocks and page breaks.

This was then used to construct an initial FOSI stylesheet so that when a new instance of the document type was created or opened, it would at least be styled with the basic structure rather than presented as an amorphous slab of markup and text. The stylesheet could then be edited to refine the formatting (see Figure 2): the styling interface was comprehensive but required substantial training to use.

Figure 2: *Document Architect* creating styles for a document type



### 2.2.2. *Author/Editor* ☑ / *RulesBuilder* ◯

This was one of the most widely-used editors: easy to use and fuss-free in operation. Its interface later became familiar to a much wider audience as it formed the basis for the HTML editor *HoTMetaL* and the XML editor *XMetaL*. Installation was from CD.

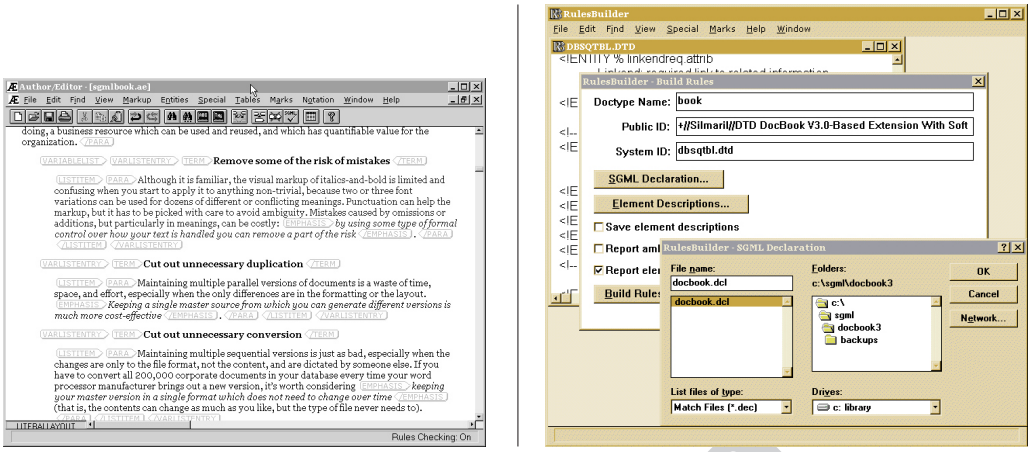
*Author/Editor* would immediately open an arbitrary SGML file, but would only enter full synchronous typographic mode if the DTD was known and precompiled. For documents using previously unencountered document types, it acted as a plaintext editor until provided with a compiled DTD.

For full use, the separate *RulesBuilder* program was needed. The assumption was that the DTD was a corporate asset that would not be available to an end-user, only to an administrator; and even outside that type of managed framework, it would be undesirable for arbitrary users to be able to modify what should be a stable DTD. The administrator would compile the DTD to a proprietary `.rb` file which *could* be given to an end-user.

### 2.2.3. *Emacs + psgml* ☑

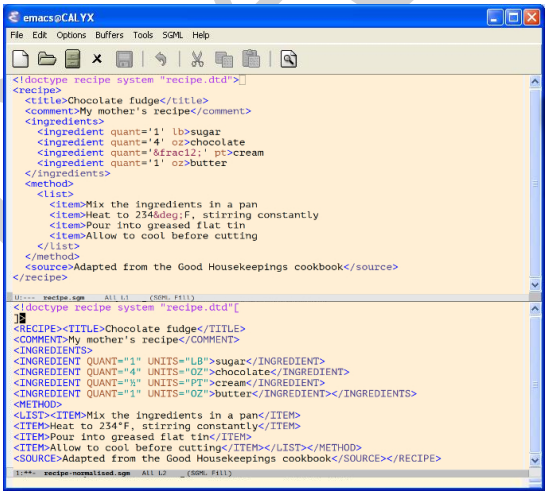
GNU *Emacs* is a plain-text editor and work environment by Richard Stallman, with a macro and programming facility using a dialect of Lisp, used extensively for the creation of add-on packages. *Emacs* 20.7 for Windows 95 and NT was installed from the GNU archive.

Figure 3: *Author/Editor* editing a *DocBook* document (left); *RulesBuilder* compiling a DTD (right)




The *psgml-mode* package by Lennart Staflin is an *Emacs* major mode for SGML and XML which parses the DTD (a requirement in SGML). At the time it provided the only free and unencumbered fully-featured SGML editor.[4] Installation of various versions of *psgml* were tested to find one which was compatible with the 20.7 version of *Emacs*, as at that time there was no package library synchronisation. Version 2.12 worked with the removal of the compiled form of the `psgml-other.elc` file, which was the only code mismatch. Installation would normally be via the *Emacs* package system,[5] or by using the `Makefile` in a downloaded *psgml* distribution, or (in the test case under Windows) by copying the files to a suitable directory manually.

Figure 4: *Emacs* with the sample document before and after normalization




A setting in the user's `.emacs` configuration file can make *Emacs* invoke *psgml-mode* automatically for files ending in `.sgml` or other extensions. The DTD is tokenised and the result used to guide the user's selection of element types via a menu or with TAB completion, and the use of attributes via a subsidiary window. Manipulation of markup in context as with any other syntax-directed editor is done from the menus or with keystroke abbreviations.

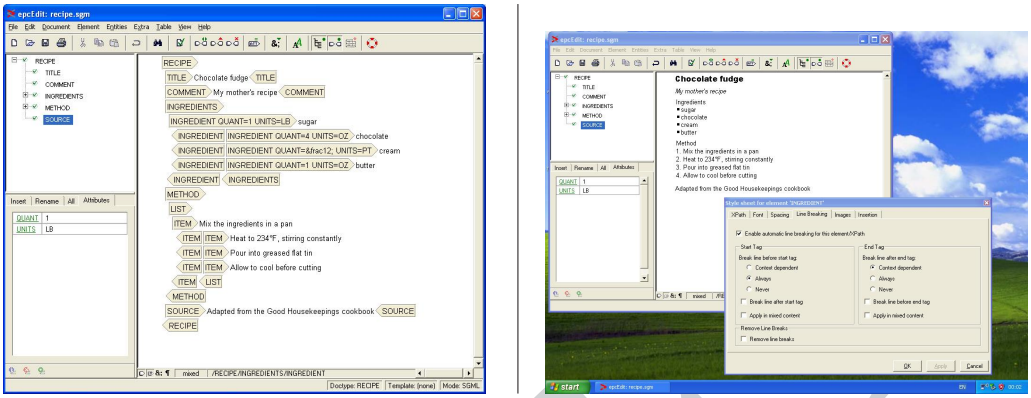
[4] Arguably, it still does, and for XML as well.

[5] The package has recently and inexplicably been withdrawn from the current repositories.

### 2.2.4. *epcedit* ☑

This is an SGML (and later, XML) editor from tkSGML by Heinz Detlev Koch and Roman Halstenberg. While it is no longer under development, it can still be downloaded but it is a 32–bit system only and requires a licence key. The program uses the Tcl/tk language (v8.3) which is included in the distribution. The sample document was opened without problems (see Figure 5)

Figure 5: *epcedit* with the sample document and with the sample stylesheet

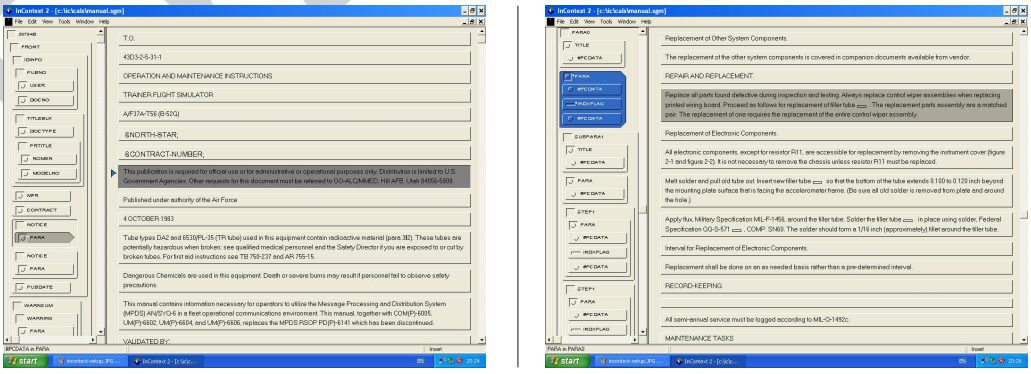


There is a built-in stylesheet system which is reasonably comprehensive except that it does not allow for the formatting of generated content, and attribute values cannot be included in generated content. The result of styling the sample document is also shown on in Figure 5

### 2.2.5. *InContext* ☑

*InContext* is one of a small number of SGML editors more suited to 'data' applications, with the interface using boxes for each element (see Figure 6). Navigation is provided in a side-panel, using a 3D arrangement of stacked, nested buttons to represent the document tree. This is a good way to display documents where 'data' elements occur frequently and text is minimal, but its usability as an editor for writing continuous text is severely limited. As the only available disk was an evaluation copy, it was not possible to open the sample document, only the samples provided by the vendor. Microsoft Office (specifically, Excel, to handle tables) must be installed before installing *InContext*.

Figure 6: *InContext* editing a CALS manual



Access to editing in Mixed Content *is* possible, despite the box design: selecting an element type like `para` lets you split the content into nodes, and empty elements are shown as symbols.

Other software with a related interface includes Microsoft Office *InfoPath* (2003), and the Citec's *MultiDoc Pro Translating Editor*

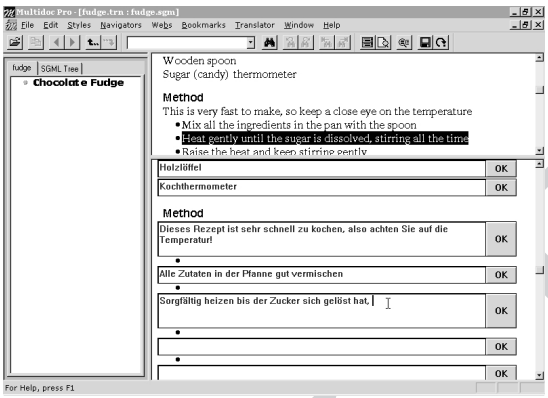
### 2.2.6. *MultiDoc Pro Translating Editor* ☑

This is a companion program to the *MultiDoc Pro Browser* and *Publisher*, sharing the same interface and the same style files. The objective is to let a translator fill in a parallel document structure with the translated text

('target') occupying the same element types in element content as the original document ('source').

Installation is from CD, with a choice of this program, the browser/publisher, and some ancillary programs. Opening the sample recipe document meant using the normalized version (created with *sgmlnorm* or *epcedit*) because MDP does not handle files using missing end-tags or attribute minimization.

Figure 7: *MultiDoc Pro Translating Editor* editing a recipe (original formatted above; translation in boxes below)

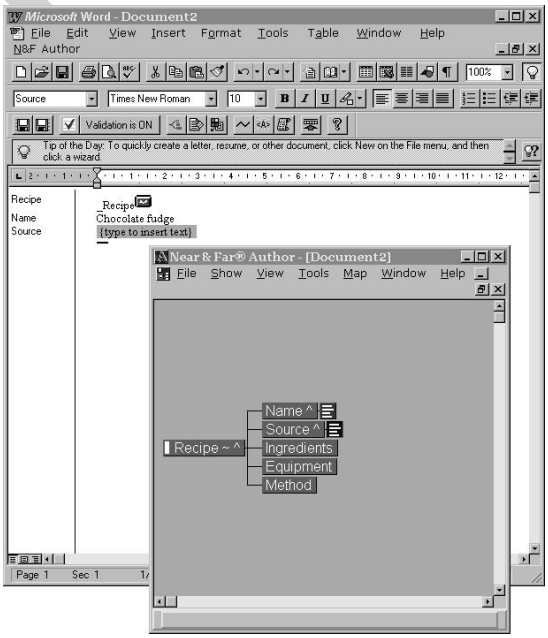



The similarity between the target interface and the one used by *InContext* is immediately apparent in Figure 7. The advantage of using their existing browser and stylesheet apparatus to present the source document presumably meant that adding the parallel hierarchy below was unproblematic, except that in Mixed Content, any subelements may be in a different order in the target because different languages have different ways of expressing things.

### 2.2.7. *Near&Far Author for Word* ☒

From the same stable as *Near&Far Designer* (see section 2.5.1 on page 15), this is not a standalone editor but an add-on for Microsoft *Word*. It adds 'Import' and 'Create' menu entries for the *Word* interface which parallel the 'Open' and 'New' operations. It uses whatever DTD you select, without the need to precompile it, and it uses the same graphical navigation as the companion *Designer* program (see Figure 8).

Figure 8: *Near&Far Author for Word* editing a *Word* document with named styles




However, as with every other markup tool that interfaces with Microsoft *Word*, there is a prerequirement that the *Word* document uses exclusively Named Styles, as this is the only way in which element type names can be bound to recognised spans of text in the document.

As can be seen from the screenshot, the floating window with the document structure from the DTD can be used to guide formation of the document, with the selected element type mapping to a named style, which in turn provides the formatting...which can of course be changed by the author without affecting the markup, although the formatting toolbar is usually disabled to prevent meddling.

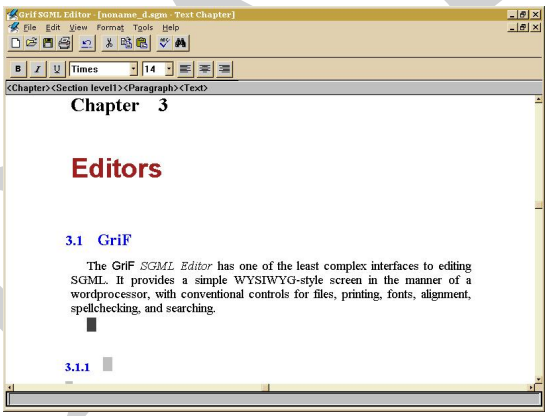
An export function saves the document as SGML. In this way a circular conversion can be obtained, much like Microsoft's *SGML Author for Word* but with the benefit of executing from within the editor.

### 2.2.8. GriF *SGML Editor* ☒

The GriF *SGML Editor* has one of the least complex interfaces to editing SGML. It provides a synchronous typographic screen in the manner of a wordprocessor, with conventional file management and editing controls. In addition to being popular in business and publishing applications, GriF was the interface used in the Euromath editor.

As with most editors in this class, compiling the DTD and creating a stylesheet are tasks administered separately from the editor, using the *Application Builder* program. However, (so far as this author is aware) uniquely to GriF, the screen controls still allow the user to impose local (non-element-based) styles on a document. This approach allows almost complete word-processor-style control with independence from the markup: a stylesheet may cause a particular element to default to bold type, for example, but you can override that manually by using a different font, for example, or changing the size, on an entirely independent basis, without affecting the *element* markup in any way. The effect is achieved internally by storing Processing Instructions to record the *ad hoc* styles within the markup, so that these operator-controlled styles are not tied to an element.

Figure 9: GriF's *SGML Editor* showing text-entry location for the title of section 3.1.1




By default, locations on the screen where character data content is required are identified by gray squares, so text entry for very prescriptive DTDs can be made almost as simple as a form-fill application (many other editors also do this but require scripting).

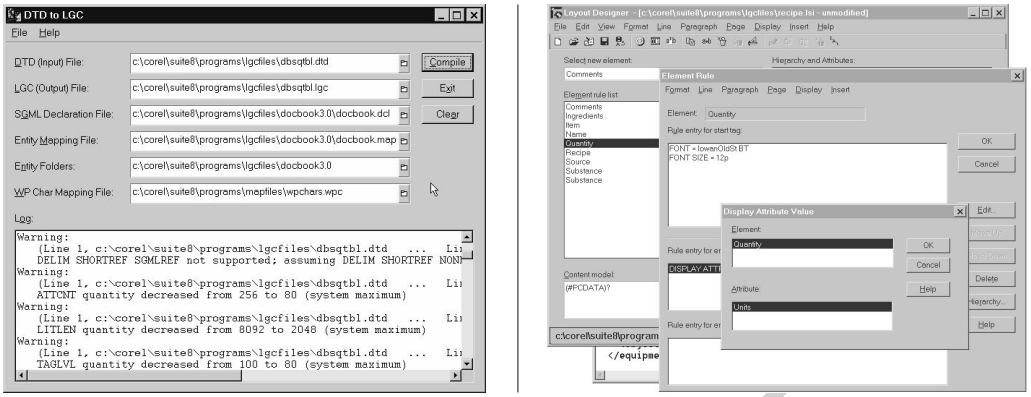
GriF was an early implementer of the feature during text entry that the Enter key performs an element split; that is, it creates a new instance of the current element in element content, if the DTD permits this — for example, pressing Enter at the end of a `para` would create another `para`. If the current content model is at an end (all required elements are present, and no further optional elements are wanted), the TAB key moves to the next location in the document model where input is possible.

### 2.2.9. WordPerfect+SGML ☑

*WordPerfect* was for many years *the* dominant wordprocessor, especially in the MS-DOS period. Reputedly even WP's own engineers regarded the character-cell version as the 'real' *WordPerfect*, and superior to the GUI version. However, embedded inside the Windows version (8) was a real, fully-fledged SGML editor and stylesheet system, providing synchronous typographic editing of SGML documents for the price of a wordprocessor, well below the entry-point for the larger SGML-only editors.

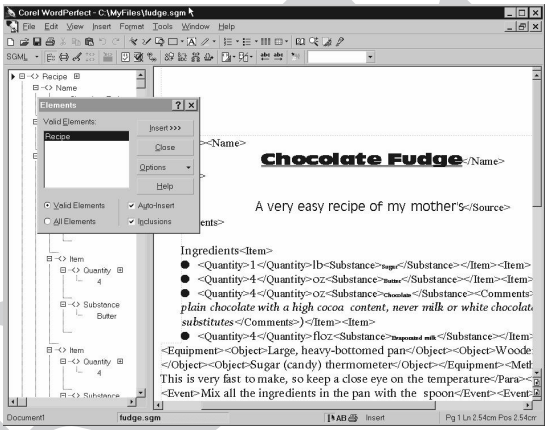
Compilation requires choosing the correct character map files for whatever is expressed in the SGML Declaration, but otherwise it is fairly tolerant of the settings, and adjusts them to suit itself. This produces what WP calls a 'logic' file, which can then be used to create a stylesheet (template, see Figure 10 on the next

Figure 10: *WordPerfect* compiling a DTD and creating a stylesheet




page) which becomes available along with the document type of the DTD and appears in the wordprocessor's SGML menu (the normal Open/New functions only work for wordprocessor documents).

Figure 11: *WordPerfect* editing SGML




## 2.3. Processors

Of the systems in the list on page 3, *Jade* is not covered here.

### 2.3.1. *Balise* ☒

*Balise* was known as the 'Swiss Army Knife' of SGML. It can act as a parser and validator, transformation programming language, document manipulator, outliner, pretty-printer, and much else. It consists of a high-level programming language which in effect acts as an API to the document. The interface is the command-line compiler/interpreter.

Unusually (uniquely) it also provides access to the DTD, both logically and syntactically, so that the requirements of the DTD can be queried during document processing; for example finding what other element types are valid at the location of the current one. It also provides access to some non-ESIS parts of the document structure, such as the start and end locations of marked sections.

### 2.3.2. *DAPHNE* ☐

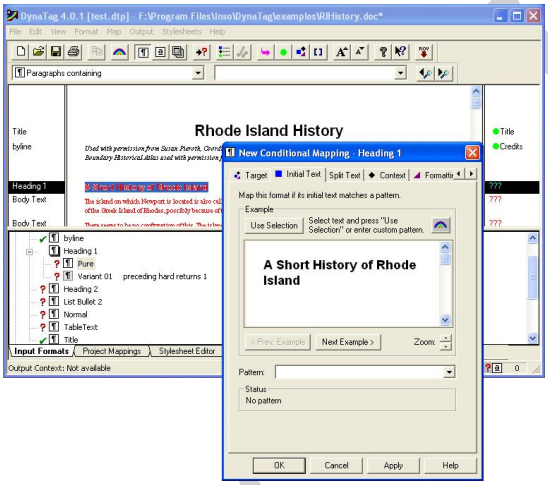
SGML-to-LaTeX converter for VAX/VMS from the German Research Network (DFN) based on the QWERTZ DTD created by the Institute for Applied Information Technology (FIT) at the German National Research Centre for Computer Science (GMD). As far as is known, this is no longer available.

### 2.3.3. *DynaText* ☑

There are three components to the *DynaText* system: *DynaTag*, which models conversions from *Word* to SGML; *DynaText* proper, a system for creating eBooks; and *DynaWeb*, a web server for dynamic publishing of the eBook documents.

*DynaTag* displays a *Word* file with named styles, and graphically lets the operator give a mapping to the desired SGML element, as shown in Figure 12. Text can be split, combined, omitted, and grouped (providing containment for lists, for example). A common use was to create an intermediate SGML file which would then be transformed by (eg) *Omnimark* or *Balise* to the final form. Once the initial mapping was established on one document, additional documents that followed the same style could be added, and the mapping refined. Eventually, the accumulated 'knowledge' could be used on entire directories of documents of one pattern for bulk conversion.

Figure 12: *DynaTag* configuring a *Word* document for conversion to SGML




*DynaText* itself could take arbitrarily large SGML documents (perhaps produced by *DynaTag*), and use an SGML-based stylesheet to render them on-screen as dynamic eBooks, with full-text searching, including Boolean operators and the use of the markup to guide the search. The system was in widespread use in industry and in technical, literary, and academic publishing, and was very influential on later developments (DSSSL, CSS, XSL:FO).

*DynaWeb* was a HTTP server for Windows NT, performing a similar function to *DynaText* but serving HTML generated on-the-fly. This meant that fast-changing documents (sourced in *Word*) could be pushed through a workflow starting with conversion in *DynaTag*, and once validated, served immediately to the next request.

### 2.3.4. *Omnimark* ☑

Until the appearance of XSLT, *Omnimark* was a frequent choice for SGML conversions. It contained a pattern-matching language which enabled transformation between SGML and non-SGML formats ('down-translation'), between non-SGML and SGML formats ('up-translation'), and between arbitrary text formats ('cross-translation').

In the example in Figure 13 on the facing page, the `%c` emits the element content (equivalent to XSLT's `apply-templates`); `%n` is a newline; `%g` dereferences a buffer; `%v` dereferences an attribute. A streaming feature meant the document did not have to be read into memory in its entirety; a value not yet encountered in document order (but known to occur) could be referenced, but not dereferenced until the end of processing, by which time the desired value would have been encountered and set as a 'referent').[6]

[6] Very large documents with very high numbers of such referents typically caused a brief but audible rattling at the end of processing as the disk drive actuator arm repeatedly sought and wrote the data from whatever temporary location had been created during processing.

Figure 13: Fragment of *Omnimark* code showing transformation to LaTeX

```
down-translate

global stream temp

element ABSTRACT
        output "%n\begin{abstract}%n%c%n\end{abstract}"

element ACRONYM
        set buffer temp to "%c"
        output "%g(temp)\index{M}{%g(temp)}"
            when attribute remap isnt specified
        output "\acro{%g(temp)}{%v(remap)}"
            when attribute remap is specified
```

For a brief period in the late 1990s the product was made available without charge, but this was later abandoned. The software is still available in a much more advanced version for XML and is widely used in publishing workflows.

### 2.3.5. Microsoft *SGML Author for Word* ☒

Despite the name, this is *not* an authoring editor. It was (perhaps still is) a plug-in converter from *Word* to SGML *and back*. In tests conducted for an earlier review, it was able to convert circularly: from *Word* to SGML, edit the document, convert from SGML back to *Word*, edit the document, lather, rinse, repeat...losslessly (Flynn, 1998).

Admittedly it took considerable configuration, but astonishingly, it worked. This was intended to allow a non-SGML person to author a document, have the publications staff convert it and edit it, and then convert back to *Word* and hand it back to the author to carry on writing or editing, again and again until ready. Microstar's *Near&Far Author for Word* provides a similar facility, but embedded within the *Word* interface itself.

## 2.4. Formatters, including browsers and servers

Of the items in the list on page 3, *3B2* and Arbortext *Publisher* are not covered here.

### 2.4.1. *Panorama Publisher* and *Viewer* ☑

The *Panorama Free* plugin for *Netscape* was many end-users' first sight of SGML, although if properly set up, they would hardly notice the difference except for the superior formatting. When a Panorama user browsed or followed a link to an SGML document, the plugin (or standalone version) would expect to find two files called (exactly) `catalog` and `entityrc` on the server in the same directory as the SGML file. The `catalog` contained entries resolving the Formal Public Identifier in the document's `DOCTYPE` declaration to a downloadable DTD in the normal way of catalogs:

```
PUBLIC "+//Silmaril//DTD Recipes//EN" "../recipe.dtd"
```

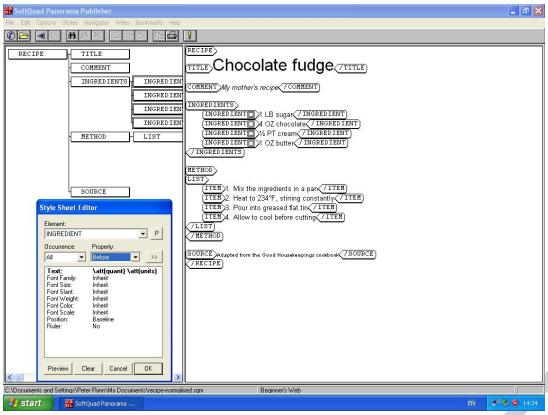
This allows *Panorama* to download the DTD. The `entityrc` file contains entries matching the FPIs in the `catalog` and providing the names of the stylesheet[s] and navigator[s].

```
PUBLIC "+//Silmaril//DTD Recipes//EN"
  DOCTITLE "recipe,title"
  STYLESPEC "Standard" "recipes.ssh"
  NAVIGATOR "Contents" "recipes.nav"
```

Per-document variants of the stylesheets and navigators can be specified in the instance by using Processing Instructions:

```
<?STYLESHEET "NewStyle" "cookbook.ssh">
<?NAVIGATOR "WebVersion" "webrecipes.nav">
```

Figure 14: *Panorama Publisher* creating styles for a document



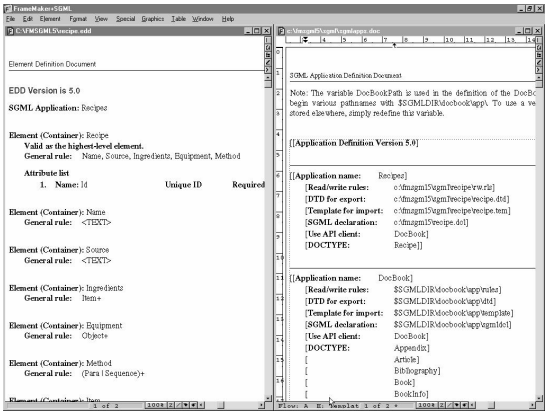

For a large DTD (eg *DocBook*, TEI, and many industrial schemas) the process of creating the stylesheet is lengthy, the same as it would be for CSS today, but the stylesheet interface is entirely graphical and very easy to use (see Figure 14).

Opening SGML either locally or over the web offered a significant advantage: the browser obeyed the stylesheet formatting, so you were no longer at the mercy of the web browsers' feeble implementations of CSS; the downside was that in-browser scripting (eg *VBScript* or *Javascript*) were not available within Panorama, so it was restricted to classical document-server applications (nonetheless extensive). The biggest advantage, however, was never really taken up: hypertext. *Panorama* implemented *HyTime* (ISO JTC 1/SC 34, 1992), so it could handle bidirectional linking, multi-headed (drop-down) links, and — most importantly — you could apply links via the browser without needing write-access to the document because they were stored in your own local file which you could publish on the web, so that other people opening the document could reference your file and see all the links take effect.

### 2.4.2. *FrameMaker+SGML* ☒

*FrameMaker* was another long-time standard for publishing, and had the ability to work with SGML documents since Adobe took it over in the mid 1990s, when it was aimed at the industrial structured-document formatting market. As with other editors, the DTD has to be compiled, in this case to an EDD file, and then a stylesheet created, before any editing or formatting can take place. The early interface for this was forbiddingly complex (see Figure 15).

Figure 15: *FrameMaker+SGML* creating styles for a newly-compiled DTD

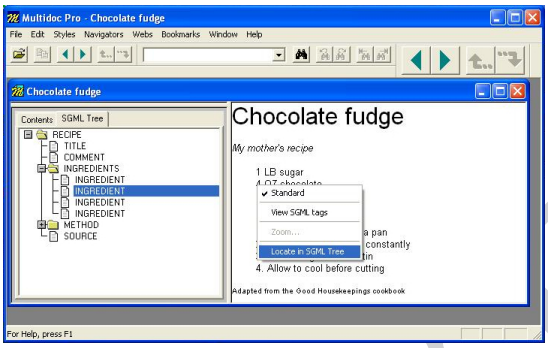



While the system was popular with typesetters, its use with SGML was troublesome for many, with tweaks and adjustments required, and concerns about the way in which SGML was exported, especially as its own (MIF) compatibility export format was very successful. Like *ADEPT*, it had the ability to apply non-structural 'tweaks' after formatting, before pre-press, to adjust visual details not provided for in the code.

### 2.4.3. *MultiDoc Pro Publisher* ☑

*MDP* used the same mechanisms and file structures as *Panorama* (see section 2.4.1 on page 13), so files prepared for one system could be used in the other.

Figure 16: *MultiDoc Pro Publisher* viewing a document created in *Panorama Publisher*



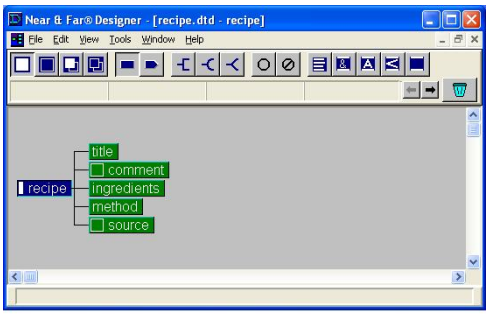

It also implemented the Occurrence Density Display of search results (familiar to modern users as the right-hand bar in a search in some web browsers, giving fine horizontal lines at proportional locations in the height of the window to the length of the document) for extensive search features including the TEI Pointer syntax.

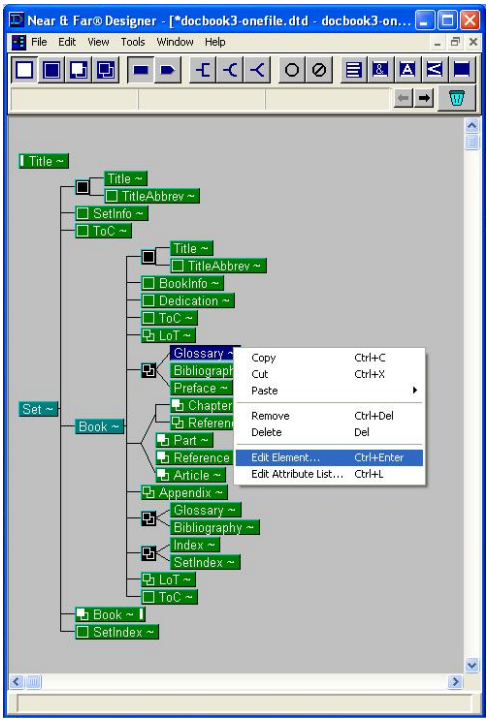
## 2.5. Other software

### 2.5.1. *Near&Far Designer* ☑

A companion to *Near&Far Author for Word* (see section 2.2.7 on page 9), *Designer* was a graphical interface to DTD creation and management. It had a simple and effective drag-and-drop paradigm to create element types, add attributes, and establish content models, using symbols to represent the syntax of declarations (eg `EMPTY`, `#REQUIRED`, and the punctuation of content models). Whole chunks of element content could be clicked and dragged around the document model while working to find an optimal way of representing the document.

Figure 17: *Near&Far Designer*'s view of the sample Recipe DTD and the *DocBook3* DTD



There were a few drawbacks: importing a DTD meant it had to be 'flattened' to a single file — problematic for a heavily modularised DTD like the TEI; it also meant that re-exporting it would lose any conditional parts that had been excluded by the use of Parameter Entities. While it worked well for smaller structures, it was not generally used for industrial or technical DTDs — with the exception that document type designers used it very extensively (and some still do) for the quality of the display, rather than its constructional modelling abilities, partly because the way in which the tree was represented seemed to be recognised by otherwise non-technical clients as immediately comprehensible.

### 2.5.2. *PAT* ☐

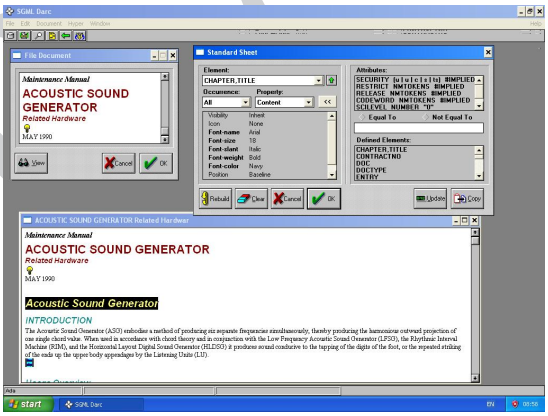
At the time, *PAT* was claimed as the only native SGML database product (Seybold, Inc, 1996). It was created to provide indexing search for the Oxford Dictionaries project at the University of Waterloo, and later commercialised by OpenText Corporation. PAT was available for SunOS 4.1.3 (this author's platform in those days) and was installed for the CELT project for searching their TEI corpus of Irish writing. It continued in use until the platform failed in a lightning strike in 2003 (resuscitation is currently ongoing, as this was the world's ninth web server).

PAT's main advantages were the ease of ingestion of a new corpus (basically a single vast, monolithic SGML document), and the speed of a search. Significant scripting was needed at CELT to turn the KWIC format output into a fully-referenced page for the web, as it meant revisiting each hit to look up the element types needed to compose a reference, and then again to retrieve the reference points themselves, resulting in a slower-than-optimal return, but acceptable in those days.

### 2.5.3. *SGML Darc* ☑

The SGML Document Archive from the KTH in Stockholm is perhaps not strictly a native SGML database, but provides some search and extraction facilities. The system was installed from two 3½" floppy disks without problems.

Figure 18: SGML *DARC* searching a document




The system requires preparation by compiling the DTDs and the stylesheets to be used for the documents to be stored. Ingesting the documents also requires details of which items are to be indexed, and how, so it can be a lengthy task. The indexing, however, is very robust, and retrieval is fast. For a large repository ('archive') of documents, having them dynamically formatted is a big advantage, because incoming additions to the same DTD would appear completely consistently.

The display engine was later developed by Synex and SoftQuad as *Explorer* and later *Viewport*, which in turn informed Softquad's *Panorama* (viewer and publisher).

## 3. Conclusions

The structured-document software world has moved on significantly since the virtual replacement of SGML with XML. Some of this is due to improvements in hardware, especially in speed and capacity, and in software capability and compatibility (or at least interoperability), and in language development, particularly *Java* and *Javascript*. XML deliberately cut out a lot of facilities from SGML which were underused or added complexity

for little gain — the Design Goals of the XML Specification emphasise ease of use and simplicity (Bray, Paoli & Sperberg-McQueen, 1998). The number of people (and companies) using XML is much larger than it ever was for SGML, so there is probably more software available to meet the demand. With better frameworks and raised awareness, vendors, developers, and programmers have generally been paying more attention to usability, so installing and using current software is easier and more reliable than it was in the days of Windows 95/XP and SGML. Modern applications tend to make less fuss, a lesson learned from the so-called 'Web 2.0' paradigm which emphasises obviousness. So have we learned anything else?

- *Reports of the death of the command line are greatly exaggerated.* People still use it, and its availability in OS X and Windows 10 means that programs originally restricted to UNIX or GNU/Linux are now available on any platform. Most users may not need it, but developers, administrators, and other technical users do, especially for scripted document management functions and for the bulk processing of documents in a workflow.
- *People do still use SGML.* Several consultancies, including the author's, have publishing clients still maintaining SGML systems, for a variety of reasons.
- Lots of the software did still install and execute, which was a surprise. Of those which failed, some were due to faulty media (having been kept for several decades) and some to the OS environment. The clock had to be reset to 1998 for some of the MS-DOS utilities, and there was one unresolved oddity in executing *RulesBuilder*, which consistently gave a Windows `Divide by zero or Overflow` error.

## 3.1. Some stuff has been gained.

1. *DTD/Schema resolution has improved.* SGML applications tended to be rather helpless about where to look for the DTD, with each vendor having a different idea of where 'the right place' was. Catalogs fixed most of that problem, but required care and feeding, and Owners were sometimes careless about naming, spelling, and punctuation. XML Catalogs are an improvement, and with the Public Identifiers now less used, most software seems to look in the document folder for the System identifier unless otherwise instructed; or at least it *asks* for it instead of crashing with an error message.
2. *There is more consistency.* The web interface paradigm, which was still in its infancy in 1995, is now the dominant method of interacting with general users — people, even non-computer-users, are expected to know that you click on things to see more — in the same way that office software is expected to work like *Word* and *Excel*. Breaking dominant patterns like these needs extraordinary changes and extraordinary benefits, and the new paradigm of everything being clickable, and not necessarily coloured blue and underlined, is an opportunity to ensure that the underlying XML is used for consistency.
3. *The move to using XML* as the storage format for both *Open/Libre/Neo Office* and *Word* has been a sea-change in making documents programmatically accessible
4. *The creation of XPath and XSL* (both T and FO) has brought about huge improvements in expressing addressing and programming transformations
5. *The lessons learned in usability* from end-user interfaces like the *Panorama/Synex/Citec*-type navigation and stylesheet creation windows mean we now have much better facilities for creating in-app navigation and styling tools
6. *There seems to be far less reinvention of the wheel* now, in that many more applications re-use, or build on the shoulders of, existing schemas and DTDs, rather than inventing new ones every time. Writers and speakers have constantly warned about the risks of corporate hubris in writing everything from scratch rather than adopting or adapting a close-match common vocabulary and structure — while at the same time extolling the virtues of modularity and extensibility (Maler & el Andaloussi, 1999; W3C QA Group, 2005; Flynn, 2017).

## 3.2. Some stuff has been lost.

1. *Deprecating the Formal Public Identifier* was probably a good move, but using a web URI is nearly as bad. If the GCA had realised what they had in the ISO 9070 Registry, they could have made a big difference. Formal ownership of Names is important.
2. psgml *risks being lost* because of the introduction of *nxml-mode*, which handles only RNG, and has a very limited control set, making it virtually unusable as a text-document editor

3. Although the *'family-tree' hierarchical box diagram representation* of the schema or DTD tree is common in many XML editors, none of them yet appears to match the design, clarity, and ease of use of *Near&Far Designer*

4. *The use of XML in the web browser* has never properly been supported by the browser-makers, for the exact same reasons as the original HTML wasn't. It has to some extent been saved by *Saxon/CE* and *Saxon/JS*, but the browsers themselves are a lost cause

5. *Open Source software* (then usually just called 'free') is no better at surviving three decades than commercial software. In fact, commercial software may have the edge, in that it came in boxes, with manuals, CDs, dongles, licence keys, and other stuff, so it got put on a shelf or into a cupboard.

   However, most (but not all) web sites acting as repositories for the 'free' software have long since disappeared; but so have almost all of the corporate web sites of the commercial software.

   What is particularly pernicious is that when the owner[s] of a popular and much-used commercial product diversify (or, sadly, die), and it is then sold to another company, the buyers usually knows roughly what they have bought — the first time it happens. But when those buyers are themselves bought, the third owner has less knowledge and interest. By the time it happens again, and maybe again, a once-reputable product is now owned by a manufacturer of children's toys, and has no idea why it also sells a structured-document editor. 'Let the market decide' only works in the textbook economic circumstances of perfect knowledge. In the Real World™ where everything is kept under wraps, nothing is safe.

6. *Hypertext linking* in the *HyTime* sense never took off. The *Panorama*-style browsers demonstrated that it was not only possible but easy to use, and anyone who has taught HTML or had to deal with novice designers will know that there is demand for multi-headed and bidirectional links.

# A. Sample SGML document

This was file `recipe.sgml`.

```
<!doctype recipe system "recipe.dtd">
<recipe>
  <title>Chocolate fudge</title>
  <comment>My mother's recipe</comment>
  <ingredients>
    <ingredient quant='1' lb>sugar
    <ingredient quant='4' oz>chocolate
    <ingredient quant='&amp;frac12;' pt>cream
    <ingredient quant='1' oz>butter
  </ingredients>
  <method>
    <list>
      <item>Mix the ingredients in a pan
      <item>Heat to 234&amp;deg;F, stirring constantly
      <item>Pour into greased flat tin
      <item>Allow to cool before cutting
    </list>
  </method>
  <source>Adapted from the Good Housekeepings cookbook</source>
</recipe>
```

## A.1. The DTD used in the sample document

This was file `recipe.dtd`.

```
<!ELEMENT recipe - - (title,comment?,ingredients,method,source?)>
<!ELEMENT title - - (#PCDATA)>
<!ELEMENT comment - - (#PCDATA)>
<!ELEMENT source - - (#PCDATA)>
<!ELEMENT ingredients - - (ingredient+)>
<!ELEMENT ingredient - o (#PCDATA)>
<!ATTLIST ingredient quant CDATA #REQUIRED
                     units (g|Kg|dl|l|oz|lb|pt|cup|0) #REQUIRED>
<!ELEMENT method - - (para|list)>
<!ELEMENT para - - (#PCDATA)>
<!ELEMENT list - - (item+)>
<!ELEMENT item - o (#PCDATA)>
<!ENTITY deg CDATA "&#176;">
<!ENTITY frac12 CDATA "&#189;">
```

## A.2. The SGML Declaration used for the sample document

This was file `sgml.dec`.

```
<!SGML  "ISO 8879:1986"
--
        Document Type Definition for the HyperText Markup Language
        as used by the World Wide Web application (HTML DTD).

        NOTE: This is a definition of HTML with respect to
        SGML, and assumes an understanding of SGML terms.

        If you find bugs in this DTD or find it does not compile
        under some circumstances please mail www-bug@info.cern.ch
```

```
--

CHARSET
         BASESET  "ISO 646:1983//CHARSET
                   International Reference Version (IRV)//ESC 2/5 4/0"
         DESCSET  0   9   UNUSED
                  9   2   9
                  11  2   UNUSED
                  13  1   13
                  14  18  UNUSED
                  32  95  32
                  127 1   UNUSED
     BASESET   "ISO Registration Number 100//CHARSET
                ECMA-94 Right Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"
     DESCSET   128 32 UNUSED
               160 95 32
               255  1 UNUSED


CAPACITY        SGMLREF
                TOTALCAP        150000
                GRPCAP          150000

SCOPE    DOCUMENT
SYNTAX
         SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
                           19 20 21 22 23 24 25 26 27 28 29 30 31 127 255
         BASESET  "ISO 646:1983//CHARSET
                   International Reference Version (IRV)//ESC 2/5 4/0"
         DESCSET  0 128 0
         FUNCTION RE          13
                  RS          10
                  SPACE       32
                  TAB SEPCHAR  9
         NAMING   LCNMSTRT ""
                  UCNMSTRT ""
                  LCNMCHAR ".-"
                  UCNMCHAR ".-"
                  NAMECASE GENERAL YES
                           ENTITY  NO
         DELIM    GENERAL  SGMLREF
                  SHORTREF SGMLREF
         NAMES    SGMLREF
         QUANTITY SGMLREF
                  NAMELEN  34
                  TAGLVL   100
                  LITLEN   1024
                  GRPGTCNT 150
                  GRPCNT   64

FEATURES
  MINIMIZE
    DATATAG  NO
    OMITTAG  YES
    RANK     NO
    SHORTTAG YES
  LINK
    SIMPLE   NO
    IMPLICIT NO
```

```
    EXPLICIT NO
  OTHER
    CONCUR   NO
    SUBDOC   NO
    FORMAL   YES
  APPINFO    NONE
>
```

# B. Software and documentation available

This is a list of all the material accessible. It is being made available to those who are prepared to act as custodians of what — in some cases — may be the world's last executing or readable copy.

- Advent *3B2 SGML* v.2 manual only;
- Arbortext *ADEPT* and *Document Architect*, 5.4.1 3½" diskettes and docs, trial licence;
- SoftQuad *Author/Editor* 3.5 with *RulesBuilder* (DTD compiler), manuals and CDs;
- AIS Software *Balise* 3.1 Reference Manual, Installation Note (2), Tutorial (2), Programmer's Guide, 3×3½" diskettes, parallel-port dongle (required);
- DFN *DAPHNE* User Manual 3.0 (in German) Deutsches Forschungsnetz Report 51 (April 1988).
- KTH SGML *DARC* on 2×3½" diskettes, experimental version with manual;
- Arbortext *EPIC*, boxed, CDs and docs, CPUID licence;
- EBT *DynaText*, *DynaTag*, and *DynaWeb* Document Preparation, Features, Introduction, Publisher's Guide, Customising, Server, Online Publishing Guide, Publishing Setup, InSted Users Guide; CDs;
- EMT *EuroMath* Users Guide v.2;
- Adobe *FrameMaker+SGML* 5.1.1, boxed, manuals, evaluation copy. This list is alphabetical by product name.
- *GriF* manuals and 3½" diskettes;
- InContext *InContext* 3½" installation evaluation disk only, no licence key;
- Citec *MultiDoc Pro* Publisher 2.5;
- Microstar *Near&Far Author* 2.0 on 3½" diskettes (needs *Word* 6 or 7 *only*);
- Microstar *Near&Far Designer* on 3½" diskettes for Windows 3.1 or 95/XP;
- *Omnimark* MS-DOS V2R5 manuals and 3½" diskettes;
- SoftQuad *Panorama Viewer* and *Publisher* manuals, 3½" diskettes and CD;
- *PAT* 3.3 Users Guide (Heather Fawcett) New Oxford English Dictionary Centre, University of Waterloo, Canada;
- *PAT* 3.4 Release Notes (OpenText Corporation);
- *PAT* Workstation Guide (Heather Fawcett) New Oxford English Dictionary Centre, University of Waterloo, Canada;
- Quicksoft *PC-Write* 3.02 manual and 5¼" diskettes;
- Microsoft *SGML Author for Word* documentation and 3½" diskettes;
- OUP *SGML Tagger* documentation and 3½" diskette;
- *WordPerfect* 8 with SGML, box only, but includes software on the CD from *Understanding SGML and XML Tools*.

The following are books or other documents, not documentation, but the *SGML Buyer's Guide* has a CD with a collection of 'free' software.

- Barron, D. & Rees, M. (1987). *Text Processing and Typesetting with Unix*. Reading, MA: Addison-Wesley

- Goldfarb, C., Pepper, S. & Ensign, C. (1998). *SGML Buyer's Guide*. Upper Saddle River, NJ: Prentice Hall PTR
- GCA *SGML '91 Conference Proceedings* Providence, RI;
- GCA *SGML '95 Conference Proceedings* Boston, MA;
- GCA *SGML '96 Conference Proceedings* Boston, MA;
- GCA *SGML/XML '97 Conference Proceedings* Washington, DC;
- GCA *SGML/XML Europe '98 Conference Proceedings* Paris, France;
- Mulberry *The SGML Hornbook*, paper, 8pp.

Some CDs are just conference papers, others are mixed software

- SGML '97 Conference
- SGML/XML '98 Conference
- XML '99 Conference
- XML 2003 Conference
- Extreme Markup 2002
- Markup Technologies '99
- SGML '96 Power Tools
- SGML '97 Power Tools
- SGML '97 Power Tools
- XML '99 Power Tools

## References

Barron, D. & Rees, M. (1987). *Text Processing and Typesetting with Unix*. Reading, MA: Addison-Wesley.

Bray, T., Paoli, J. & Sperberg-McQueen, M. (1998). *Extensible Markup Language Version 1.0* [REC-xml-19980210]. World Wide Web Consortium. Cambridge, MA. Retrieved from https://www.w3.org/TR/1998/REC-xml-19980210

Flynn, P. (1998). *Understanding SGML and XML Tools*. Boston: Kluwer.

Flynn, P. (2017). Your Standard Average Document Grammar: Not just not your average standard. In *Proc. Balisage 2017*, 2017. Rockville, MD: Balisage Series om Markup Technologies. Retrieved from https://www.balisage.net/Proceedings/vol19/html/Flynn01/BalisageVol19-Flynn01.html

Goldfarb, C., Pepper, S. & Ensign, C. (1998). *SGML Buyer's Guide*. Upper Saddle River, NJ: Prentice Hall PTR.

ISO JTC 1/SC 34. (1985). *Standard Generalized Markup Language* [ISO 8879]. International Organization for Standardization. Geneva.

ISO JTC 1/SC 34. (1992). *Information technology – Hypermedia/Time-based Structuring Language (HyTime)* [ISO 10744]. International Organization for Standardization. Geneva.

Maler, E. & el Andaloussi, J. (1999). *Developing SGML DTDs: From text to model to markup*. Upper Saddle River, NJ: Prentice-Hall.

Seybold, Inc. (1996). DBMS Support of SGML Files. snee.com: Bob DuCharme. Retrieved from http://www.snee.com/bob/sgmldbms.html

W3C QA Group. (2005). More About Custom DTDs. *A List Apart*. Retrieved from https://alistapart.com/article/customdtds2/

Warmer, J. & Egmond, S. V. (1989). The Implementation of the Amsterdam SGML Parser. *Electronic Publishing*, *2*(2), 65–90. Retrieved from cajun.cs.nott.ac.uk/compsci/epo/papers/volume2/issue2/epjxw022.pdf