The GELLMU Manual

Version 0.8.4 Release, July 2007

William F. Hammond

Dept. of Mathematics & Statistics
University at Albany
Albany, New York 12222 (USA)

`hammond@math.albany.edu`

Revision of July 6, 2007

Abstract

This is the manual for Generalized Extensible LaTeX-Like Markup (GELLMU). The central focus in the GELLMU project is to tie LaTeX to the worlds of SGML and XML by providing LaTeX-like markup for writing documents under SGML and XML vocabularies (formally known as document types).

The Manual explains the distinction between *basic* and *advanced* use, provides a description of *regular* GELLMU as an instance of advanced GELLMU, and discusses the use of the didactic production system, which is the project's suite of processors for working with regular GELLMU.

The Manual also deals with the metacommands available when writing GELLMU markup. One of these metacommands is the project's emulation of LaTeX's *newcommand*, which makes it possible to have macros taking multiple arguments while writing for an SGML or XML vocabulary.

Table of Contents

1 Introduction

GELLMU is an acronym for "Generalized Extensible LaTeX-Like MarkUp", which is the author's concept for using LaTeX-like markup to write consciously for SGML document types such as HTML, DocBook, TEI, or GELLMU's own didactic LaTeX-like document type called *article*.

It evolved from earlier thought about delineation of a coherent subset of LaTeX commands with the property that if a LaTeX document used only those commands then it could be translated with full reliability to other formats including HTML so that documents could be prepared both for print and for the web from a single source.

Problems with this early thought during the years 1996–1997 included the fact that there did not seem to be a community of LaTeX users willing to focus on a narrow vocabulary and the fact then of a legacy practice that mixed LaTeX commands freely with non-LaTeX TeX commands.

The present idea was crystalized in the summer of 1998 while the author was looking at Ulrich Vieth's LaTeX markup for the TeX Directory System (TDS) specification from the TeX User Group (TUG), which now is physically realized in TUG's TeXLive series of TeX-related software distributions on CDrom. The HTML version of that specification was derived through an intermediate ad hoc translation from LaTeX to *Texinfo*, the language of the GNU Documentation System, which is a robust hypertex system pre-dating HTML driven by TeX the Program.

In thinking about generalizing Vieth's ad hoc translation, which used GNU *Emacs* Lisp (*Elisp*), one of the most widely available free[1] cross-platform programming languages for which there is a free robust engine, the same engine that underlies the interactive editing interface of *Emacs*, the author realized that the structure of *Texinfo* is very much like that of an authoring level SGML document type. From that idea it was a small step to decide that one might profitably write *Elisp* code to use LaTeX-like markup for the conscious writing of a new LaTeX-like *article* document type.

[1]URI: http://www.gnu.org/

The idea, which by itself saves keystrokes, gains significant power with the emulation of LaTeX's *newcommand*. Although SGML markup offers both subdocument inclusions and macro substitutions using the notion of *entity*, there is no SGML-standard macro facility that takes arguments. Moreover, GELLMU's *newcommand*, which is, unlike LaTeX's, a simple macro substitution facility, also provides a base for experimenting with document type extensions in a way that the SGML notion of *entity* does not since SGML entities are invoked with a notation that is apart from that used to markup with SGML *elements*, which are the objects corresponding in the GELLMU scheme with LaTeX-like *commands*.

Software associated with the GELLMU project falls into two parts:

1. **The syntactic translator.** This is a purely syntactic layer for converting LaTeX-like markup in configurable ways to SGML markup. Its output may be handled in standard SGML or XML systems.

2. **The didactic production system.** This component is a sketch, which might usefully serve as a base for further development, of an SGML production system for an authoring environment that consists of

   (a) An SGML document type called *article* that is accompanied by a corresponding XML document type.

   (b) A package of extensible translating utilities written in the language *Perl*.

   As its name suggests, these materials are didactic and should be regarded as unfinished for production work.

There are two overall concepts: *basic* mode and *advanced* mode. The basic mode may be used to write consciously for any standard document type such as HTML, DocBook, or TEI, and the syntactic translator is for that mode the only software under this project that might be relevant. The advanced mode incorporates a configurable broader array of LaTeX-like markup features, mostly for brevity, in the syntactic translator. This mode is fully developed only for use with a LaTeX-like document type such as the didactic *article* document type that is part of the didactic production system.

One may use any SGML or XML processing framework in working with the *article* document type. The didactic production system includes what is needed to produce both HTML and regular LaTeX forms of an *article* instance. Consequently, one is able to produce both DVI using the LaTeX format for TeX, the program, and PDF using the LaTeX format for PDFTeX, the program. Moreover, one may tune the PDF in various ways using small alterations of the *Perl* code for translating the XML version of *article* to regular LaTeX.

2 Basic GELLMU

Neither the basic nor the advanced mode involves in any way adoption of the language of LaTeX. (But many command names under the didactic *article* document type, mimic LaTeX command names.) There are two fundamental ideas:

1. A LaTeX-like command "`\foo`" corresponds to an SGML element "`<foo>`".

2. The syntactic translator is almost entirely ignorant of vocabulary and a name like *foo* need not have meaning in it although it must have meaning in the document type for which one is consciously writing.[2]

To use the basic mode one must be familiar with the SGML document type for which one is writing. Ordinary HTML is an example. Very few of the features in the advanced markup not also part of the basic markup[3] make any sense for use in the direct preparation of HTML with LaTeX-like input.

GELLMU uses LaTeX special characters such as '`\`', '`{`', and '`}`' along with LaTeX-like argument/option syntax, where braces immediately following a command name indicate command arguments and square brackets, i.e., '`[`' and '`]`', indicate command options. A command corresponds to an SGML element, and in basic mode a command may have at most one argument, the content of which corresponds to SGML element content, and at most one option, the content of which corresponds to a list of SGML attribute specifications. Thus for example, in basic mode for HTML one may use the markup

```
\a[href="http://www.w3.org/"]{The World Wide Web Consortium}
```

to form the HTML anchor:

```
<a href="http://www.w3.org/">The World Wide Web Consortium</a> .
```

(The formation of anchors with the didactic *article* document type in advanced mode is slightly more complicated because the characters '`=`' and '`/`', which may acquire special (and "overloaded") semantic significance in mathematical contexts, are held for delayed evaluation as empty elements and because the syntactic translator, which does not recognize command names, regards this usage in advanced mode as *multiple* argument/option syntax (section 4.2), which is not part of basic mode.)

An example of the distinction between basic and advanced GELLMU is that in advanced mode it is possible and easy to arrange to have a blank line, as in LaTeX, represent the beginning of a paragraph. In basic mode for HTML one must[4] use "`\p`" to begin a paragraph, and for the XML version of HTML one must also provide markup for the end of every paragraph, which may be done in several ways.

For some of the details on using the basic markup with HTML see *Using the* GELLMU *Syntactic Translator to Write* HTML[5]. It will be instructive to have the parallel source markup[6] available at the same time.

[2]The syntactic translator does, however, have some facilities for classifying the names in a list in regard to common syntactic behavior. See, in particular, the *Elisp* variables *gellmu-autoclose-list* and *gellmu-parb-hold*, both of which are not significant in basic GELLMU.

[3]With several minor exceptions, one related to the direct writing of SGML attributes (which cannot contain markup and which do not have many parallels in LaTeX) and another related to the way of escaping the character '`\`', everything about basic mode also applies to advanced mode.

[4]There is a way, with the setting of several variables for the syntactic translator in advanced mode, to have blank lines begin new paragraphs in basic input for HTML

[5]URI: ghtml.html

[6]URI: ghtml.glm

3 Metacommands

A metacommand is a LaTeX-like command that does not correspond to an SGML element. Each metacommand is handled internally by the syntactic translator.

3.1 \documenttype

A document prepared in GELLMU source usually begins with a *documenttype* command. For example,

```
\documenttype{html}
```

is used to begin a document prepared for the most common form of classical HTML.

The syntactic translator has two public variables *gellmu-doctype-keylist* and *gellmu-doctype-info*, which are *Elisp* associative lists, that enable one to match XML or SGML `"<!DOCTYPE ... >"` declarations with LaTeX-like

`\documenttype[`*my-optional-key*`]{`*my-doctype*`}`

commands, where *my-optional-key* is available to override a default key for *my-doctype*. Thus, for example, "`\documenttype{html}`" points to the default key for `"html"`, which is `"html-4.01"` and which points to the *W3C HTML 4.01 Transitional* document type, while

```
\documenttype[xhtml-1.0s]{html}
```

indicates *W3C XHTML 1.0 Strict*.

A user may configure these variables without modifying the source code for the GELLMU syntactic translator, but minimal knowledge of *Elisp* will be required. A future release might provide a configuration file for this purpose.

A second option for the *documenttype* metacommand, which must follow the single required argument, is provided for writing an internal declaration subset. The contents of an internal declaration subset constructed this way may be any internal declaration subset material. However, some care is required for entering characters that are special. To ease the handling of special characters four metacommands have been provided for use inside the internal declaration subset:

```
\attlist{...}
\element{...}
\entity{...}
\notation{...}
```

For example, a user who wishes to be able in source to use "`&quo;`" to reference the ASCII quotation mark when writing consciously in basic mode for TEI.2 would begin the source file with:

```
\documenttype{TEI.2}[
\entity{quo "&#x22;"}
]
```

3.2 \newcommand

From a user's viewpoint this provides emulation of LaTeX's *newcommand*. But it is a simple facility providing macro substitution with arguments forward in a source file from the point of its occurrence. It differs from the *newcommand* facility in LaTeX in that it does not add the name of a newcommand to any namespace. Instead, as the syntactic translator encounters each newcommand definition, it performs the corresponding expansions and then forgets the name.[7]

The general construction of a newcommand definition is

`\newcommand{`*name*`}[`*nargs*`][`*first*`]{`*value*`}`

where *name* is the name of the newcommand, *nargs* optionally specifies the number of its arguments, *first* is an optionally-provided default value for the first argument, and *value* denotes the value string.

In the value string the character `'#'` is used to reference an argument by the numeric value of its position. Thus, `"#1"` refers to the first argument that is provided at an invocation of the newcommand, `"#2"` the second, etc., and there is no limit on the number of arguments. It is not required that *nargs* be supplied in order to define and use a newcommand taking arguments. However, for the sake of automatic error checking the use of *nargs* is strongly recommended when the definition of a newcommand taking arguments is entered.

Example. In writing HTML one might use

```
\newcommand{\href}[2][http://www.w3.org/]{\a[href="#1"]{#2}}
```

for brevity in writing many HTML anchors. With this definition the invocation

```
\href{http://www.myweb.mydomain/me.html}{my web page}
```

gives rise to the HTML markup

```
<a href="http://www.myweb.mydomain/me.html">my web page</a>
```

while the invocation "`\href{Web Central}`" gives rise to

```
<a href="http://www.w3.org/">Web Central</a> .
```

Rules.

1. The name of a newcommand may not be referenced in its value string.[8]
2. A newcommand may not be invoked before it is defined unless the invocation occurs in the value string of another newcommand definition in which case the definition of the latter may be first and **must** be first if the invocation of the former in the definition of the latter involves argument substitutions.

[7]This means that it is a relatively slow form of processing, but the author believes that it is a good match for the intuitive expectations of most LaTeX users.

[8]In a future release an alternative metacommand called *frontcommand* may be provided which could be used, for example, if one wishes to have a macro name of some kind match the name of an actual SGML element.

Failure to observe these rules may cause the syntactic translator to enter an infinite recursive loop. If a user suspects this may have happened, then the invocation of the syntactic translator should be interrupted (section 7.4.3).

3.3 \begin{} ... \end{}

These provide emulation of LaTeX environment notation without actually providing anything that is not otherwise available. Markup which resembles that for a LaTeX environment simply resolves to an SGML element. This usage may be convenient for SGML elements of large scope such as, for example, the *body* of an HTML document.

With advanced mode the special form

```
\begin{document} . . . \end{document}
```

may be used to emulate the corresponding feature of LaTeX for a document type, such as the didactic *article* document type, that in the large consists of a preamble and a body.

3.4 \macro and \Macro

Use of these is discouraged in the absence of a need. One situation that presents a need is name “fronting”: see the discussion below in section 3.5. Please note that in most situations one may use *newcommand* without an argument for simple macro substitutions.

macro and *Macro* do very much the same thing except for the order of expansions. Every *macro* is expanded forward as encountered before any newcommand definition is expanded. Every *Macro* is expanded forward after every newcommand has been expanded.

There are four primary differences between *macro* and *Macro*, on the one hand, and *newcommand*, on the other hand.

1. Neither *macro* nor *Macro* can be used to define a macro that takes arguments.
2. The name of a newcommand must consist of word characters, but there is no restriction on the characters, apart from unbalanced brace characters (`‘{’` and `‘}’`), that may appear in the name field of a *macro* or *Macro* metacommand.
3. If the name of a *macro* or *Macro* does not begin with the command sequence introducer, i.e., the character `‘\’`, then an invocation of that metacommand is given by every forward match of its name. The use of such names is strongly discouraged because document segments can then become opaque much too easily.
4. A newcommand invocation, absent the use of a semi-colon for termination, is only effective at the whole word level — with *word* here denoting a maximal string of successive word characters — whereas *macro* and *Macro* invocations are effective regardless of word boundaries.

Elaboration on the last point: If `x` is the name of a newcommand, then invocations are only considered by the syntactic translator on the string “`\x`” when it is followed by a non-word character. For example, if the locale is US-ASCII, then the word characters are the 52 upper and lower case letters and the ten numerals. Therefore with *newcommand*, as with its namesake in regular LaTeX, the use of "x" as a name will not intercept the occasions of `"\x"` as an initial substring of `"\xy"`. With either *macro* or *Macro* such interception does take place. One may block it at the point of an invocation with the markup `"\x;"`, and when this is done, the terminating semi-colon is removed.

Human authors using either *macro* or *Macro* may find unanticipated interactions between the three forms of macro substitution.

Unbalanced brace characters, i.e., the characters `'{'` and `'}'`, may not be used in the name field or the value field of any form of macro metacommand.

3.5 Macro-Level Fronting of SGML Element Names

The word *fronting* as used here describes the practice of modifying the meaning of an SGML element name by using one or more of the macro facilities to generate usage of the same name as an element combined with other markup using the syntax that would otherwise correspond to basic use of the element.

Suppose, for example, one wants all paragraphs in HTML (marked with `\p` in GELLMU source) to be placed in a (style) class called *custom*.

Recommended procedure: Create a new unique name and then use *macro* to front it.

```
\newcommand{\cp}[1][]{\p[class="custom"]{#1}}
\macro{\p}{\cp}
```

Each invocation of "`\p{...}`" will first be replaced by "`\cp{...}`" because all *macro* definitions are expanded before any newcommand definition is expanded. Subsequent expansion of the newcommand will yield

```
\p[class="custom"]{...} .
```

This will not intercept the alternate, otherwise nearly equivalent, markup given with `\begin{p}` ... `\end{p}` since *newcommand* is based on simple macro substitution and does not operate at the level of namespaces.

4 Advanced GELLMU

The idea with advanced GELLMU is that for SGML document types sharing structural characteristics with LaTeX one might wish to have the syntactic translator provide LaTeX-like markup syntax beyond the level used with basic GELLMU and that these additional layers of syntax should be configurable. The only substantial realization of this program so far is the case of the GELLMU didactic document type called *article*. The specifics of that realization are discussed in the following section.

4.1 Illustrations.

One might want to be able to use blank lines, as in LaTeX, for introducing new paragraphs in a document type that provides paragraphs.

In some article-level document types each sectional unit has a unit header providing markup for various, often optional, unit descriptors. It is convenient to be able to use LaTeX-like multiple argument/option syntax (section 4.2) for these.

If the document type provides authoring-level mathematical markup beyond inclusions of the World Wide Web Consortium's Mathematical Markup Language[9] (MathML) under its XML namespace[10] regime, then one might want to be able to use the `'$'` character to toggle in and out of inline math, and if the document provides for math displays, then one might want to use, as in LaTeX, the strings `"\["` and `"\]"` as delimiters for unstructured mathematical displays, and markup such as

```
\begin{equation} . . . \end{equation}
```

for a single equation, and markup such as

```
\begin{eqnarray} . . . \end{eqnarray}
```

for a list of equations.

It is important to emphasize, however, that by overall system design the syntactic translator operates without substantial knowledge of vocabulary. While it is true that if `'$'` is to provide a toggle for an inline element containing math, say, *tmath*, then the syntactic translator needs to have that association made, but the association is provided as the value of a configuration variable in the syntactic translator that can be changed between documents so that the syntactic translator may be used with many document types.

One way to make such configuration convenient is to use an array of *Elisp* functions that are fronts with various variable configuration packages for the basic function *gellmu-trans* in the syntactic translator.

The general outline for advanced GELLMU with arbitrary document types is not fully developed in the present release. Instead the project has concentrated on the realization of these ideas for the project's didactic *article* document type, which is the subject of the next section.

As the syntactic translator stands now, basic mode is characterized in the syntactic translator by the true setting for its Boolean variable *gellmu-straight-sgml*, while the configuration used by default for the didactic document type (which *could* be handled in basic mode with more verbose source markup) has that variable set false and also the variable *gellmu-regular-sgml* set false.

The term *regular* GELLMU refers to use of the syntactic translator with the default configuration for the didactic document type. It involves nearly maximal emulation of LaTeX-like markup; it implies both *advanced* mode and the didactic document type (section 5) *article*.

4.2 Multiple Argument/Option Syntax

An essential point in the present design is that the whole system is built from components, each

[9]URI: http://www.w3.org/Math/
[10]URI: http://www.w3.org/TR/REC-xml-names

of which has its own function[11]. Consistent with this design the syntactic translator operates with knowledge of syntax but little or no knowledge of language.

Multiple argument/option syntax has been built into advanced mode as part of the overall idea of providing, where sensible, LaTeX-like features in a precise user markup interface for writing in document types under SGML and XML.

What are the rules for converting the multiple argument/option syntax in source markup into SGML? Direct conversion by the syntactic translator of this type of usage into XML is not available because such conversion requires some language knowledge and the program does not operate with knowledge of language at that level[12]. One obtains an XML version of a document in the didactic production system by using a translator with minimal knowledge of the command vocabulary to create the XML version from an SGML version that is the immediate output of the syntactic translator.

In multiple argument/option syntax, which is much like that of LaTeX, arguments and options follow command names. Arguments are delimited by braces, i.e., '`{`' and '`}`' and options by square brackets, i.e., '`[`' and '`]`'. There must be no white space between the arguments and options nor between the command name and the first member of an argument/option sequence.

Each command with a multiple argument/option sequence is translated to an open tag whose name is the name of the command. Each argument is translated to an *ag0* element and each option to an *op0* element. (Both *ag0* and *op0* lie in GELLMU's reserved name space.) There are two exceptional cases.

1. The first argument or option is an option inside which the very first character is a colon, i.e., '`:`'. This is the method provided in advanced mode for the direct entry of an SGML attribute sequence.[13] The entire contents of the option string, apart from the leading '`:`', which is discarded, are understood to be a sequence of SGML attributes for the SGML element whose name is the name of the command. There is no syntax check of the attribute contents by the syntactic translator. Such an *attribute option* is not treated as an *op0* element. In particular, an attribute option is correctly followed immediately by a semi-colon, i.e., the character '`;`', if and only if the corresponding SGML element is a defined-empty element under the SGML document type. Since SGML attributes correspond to very little of classical LaTeX[14], attribute options are seldom used[15] in the didactic production system. One such use is for the GELLMU equivalent of latex's

[11] In the prototype production system based on the didactic *article* document type the output from each stage is available for examination and, where necessary, intervention. However, such use of intervention is intended only for temporary expedient use while a GELLMU system is being designed or enhanced. As with LaTeX, enhancement is an ongoing process.

[12] In handling GELLMU source markup one could provide a more elaborate processor that can be configured to know for each such command the list of names for its positional arguments and options. It was decided that this goes somewhat beyond syntactic handling but that the question of whether a list of arguments and options corresponds to sole content might be regarded as a syntactic matter.

[13] Its use is optionally permitted in basic mode as well.

[14] Indeed, LaTeX usage allows markup in options, but (element level) markup is not permitted in SGML attributes. Note, however, that in the didactic document type the SGML content model for an option is more restrictive than that for an argument. Also in the didactic document type some options, such as that for *anch*, which is described later in this section, are practically required.

[15] To say *seldom* is not to say *not*. Two important instances in the didactic production system are the *series* attribute for the *label* command, which stands in, to the extent possible, for the notion of *counter* in LaTeX, and the *type* attribute for the *series* command, which provides emulation of counter conversion from, say, number values to letter values.

*equation** and *eqnarray** environments, which is marked up this way:

```
\begin{equation}[:nonum="true"]
e = mc^2
\end{equation}
```

to produce:

$$e \;=\; mc^2$$

2. The first argument is the only argument and there are no options apart from a possible attribute option. This case, which is extremely common, is exceptional relative to argument/option handling since the sole argument simply becomes element content without an *ag0* wrapper.

When a command has a multiple argument/option sequence, the question arises whether the *ag0* and *op0* elements that arise from the arguments and options are the only content of the element corresponding to the command. The syntax does not provide a way to determine this. On the other hand, the SGML document type definition does provide information that indicates whether other content is possible. It is beyond the scope of the design of the syntactic translator for the syntactic translator to read a document type definition. The syntactic translator does, however, have a configurable list variable *gellmu-autoclose-list* that contains the names of elements for which no content beyond the elements arising from arguments and options is possible. While it is not necessary that every such command be entered in this list, when such a command not in the list is not explicitly followed by an element closing command, it is possible in some instances for an SGML parser to infer incorrectly the location of end of the element. Thus, the didactic production system provides a command *anch* for making anchors. The document type definition provides for one option, a reference, and one argument, the anchored text.[16] Because the syntactic translator does not consider the document type definition, if one enters the markup

```
\anch[href="http://www.w3.org/"]{W3C} HQ ,
```

unless the name *anch* is in the list[17] *gellmu-autoclose-list*, an SGML parser will not have reason to close the *anch* until it sees the space following the anchored text "W3C", and so that space will be considered insignificant white space with the result that there will be no space between the anchored text and the following "HQ".

4.3 Limitation in Regard to XML

A final general comment about advanced mode is that the features it can provide beyond basic mode when one is writing consciously for an XML document type are somewhat limited. For example, blank lines cannot easily be made adequate for paragraph markup with the XML form of the didactic *article* document type. Although it is not a specific limitation for future editions of the syntactic translator, the vision is that use of advanced mode will be specifically for a somewhat rich SGML version of a document type.

[16] In the XML version of *article* the option becomes the element *anchref* and the anchored text becomes the element *anchv*. One may use these names directly in GELLMU source, but option/argument notation is more familiar and more succinct.

[17] There was no list of this type in early pre-release versions of the syntactic translator.

5 The Didactic Document Type

The didactic document type is the document type underlying what is called regular GELLMU. It is the heart of the idea of GELLMU as a bridge for authors from LaTeX to the world of XML. More specifically, the bridge is from the world of a LaTeX *article* to a document type in the world of XML, also called *article*, that has a structure and a vocabulary similar to those of the LaTeX document class.[18] The techniques used in the didactic production system are extensible and can be carried over to other types of documents than articles. It is important to note that there are many features in regular LaTeX which have no analogue so far in the development of this project. One might hope to get an idea of the extent of coverage by reviewing the examples in the project archive (appendix A).

When an author prepares a document as a LaTeX *article*, the document is being marked up as data for a specific typesetting program: Donald Knuth's program TeX running with the main LaTeX facilities loaded.

When an author prepares a document as GELLMU source, the syntactic translator provides a LaTeX-like markup interface, but its output is not data for a specific typesetting program. Rather it is data for a broad class of processors. This means that multiple output formats can be obtained from a single source without the need for human intervention because XML provides a framework that makes it relatively easy to create reliable programs for translation from an XML document type to other formats. It offers, moreover, the possibility of translation to future formats free of any need for human intervention once translators from the original document type to such formats are written. The small price one pays for this advantage in moving from LaTeX markup to GELLMU markup is that the author must learn a few new things.[19]

There are two formal constructions of the didactic document type. The name of the document type is *article*. The first construction is an SGML version of *article* that provides features convenient for authors that are not available under XML. The second is an XML version. For most non-technical purposes the two constructions should be regarded as equivalent.

The SGML construction of an *article* is derived from GELLMU source markup for a document by using the syntactic translator. The didactic document type is accompanied by a translator implemented under the Perl language framework *sgmlspl* by David Megginson (see the release notes in appendix B for more information) for converting the SGML version of an *article* to the XML version.

The description in this section of the manual deals primarily with source level markup for the didactic document type and with how it is handled [20] by the time the XML version of an article is generated. Secondarily there is comment on how it is rendered in the chief output formats

[18]The LaTeX concept of document class has only a loose correspondence with the SGML concept of document type.

[19]It is not inconceivable that at some future point conscious writing for some XML document types using LaTeX-like markup might be subsumed in the LaTeX project, but in saying this, the author is neither predicting it nor assessing the merits of the idea. He has no affiliation with the LaTeX project other than as a user.

[20]Strict discussion of an SGML document type would not allow use of the word *handled*. In this instance a coordinated pair of document types is being described, one SGML and the other an XML translation. For most purposes the SGML document type is the richer of the two. However, because of its use of a handful of generic elements (in its reserved namespace consisting of names that contain `0` (zero) as first numeric character) for modeling certain convenience features of LaTeX, it is possible for a correct translation of a valid SGML article to yield an XML version that fails validation because the content models of the generic elements are necessarily loose.

of the didactic production system, which are PDF, classic HTML, and XHTML+MathML.

A quick glance at the flowchart (section 7.5.2) shows that the first XML stage — author-level XML — may be viewed as a second entry point to didactic production system processing. Some day this could become a reasonable formatting route for translations from things like *Texinfo*, *DocBook*, and, even perhaps, classical LaTeX itself via a processor such as *tex4ht*.

5.1 Suggestions and Caveats

Although this is the manual for a software release, it is not a book. A document of book size would be required for a full description of the didactic production system.

Much of the markup vocabulary is copied from LaTeX. There are some instances where there is some deviation from LaTeX usage, and many of those instances are mentioned here.

Definitive information about the didactic document type may be derived by consulting the document type definition. Because the didactic production system is conceived as a base for future development there are sketches in the document type definition that are not covered by the didactic processors. For example, although there is sketched code for the analogues of LaTeX's *paragraph* and *subparagraph* commands, which are sectional in nature, that is found in the translation from SGML to XML, there is no sketched code for these elements in the two formatters.

Another way to obtain information about the didactic production system is by studying examples including this manual and the examples in the project archive (appendix A).

5.2 Markup Fundamentals

There are several kinds of commands:

5.2.1 Explicitly named commands.

Apart from macro level metacommands an explicitly named command begins with **a maximal string introduced by the character '`\`' followed by word characters, including the numerals '`0`', '`1`', . . . , '`9`'**. The notion of *word character* depends on one's locale, a concept that is formalized in GNU *Emacs*. In the ASCII character set the word characters are the 52 upper and lower case letters and the 10 numerals. **The first numeral, if any, must not be '`0`'** since such names are reserved for use by the syntactic translator. Command names are case sensitive.

An explicitly named command is a *container*, corresponding to an SGML *element*, if its name is immediately followed, without intervening white space, by the character '`{`'. In that case the delimited zone of containment normally ends with the subsequent balancing character '`}`'. (LaTeX-like multiple argument/option (section 4.2) chains deserve more discussion; for now it will suffice to point out that the use of the `\anch` command in this document for making "anchors" is an example, and, of course, LaTeX's `\frac` command is another example. For the present discussion these commands are considered to be containers.)

An explicitly named command corresponds to an SGML *defined-empty element* if its name is immediately followed, without intervening white space, by the character '`;`'.

An explicitly named command corresponds to an SGML element closing tag if its name is immediately followed, without intervening white space, by the character `':'`.

The name of an explicitly named command is terminated by a non-word character. There is a small, possibly acceptable, level of syntactic ambiguity unless the name terminator is one of `'{'`, `';'`, `':'`, or `'['`.

In basic mode if the name terminator is `'['`, then that character introduces a list of SGML attribute specifications, each of the form *name*=`"`*value*`"`, and the list must be terminated by the character `']'`. Then if the following character is `'{'`, the named command is a container that ends with the balancing `'}'`. Otherwise the following character may be `';'` if the named command is a defined-empty element and must be so in that case for direct editing of an XML document type.

In advanced mode if the name terminator is `'['`, then that character introduces a LaTeX-like command option — part of the emulation of LaTeX's multiple argument/option syntax (section 4.2) — unless it is immediately followed, without intervening white space, by the character `':'`, in which case the bracketed content is a list of SGML attribute specifications. (The initial `':'`, which may be used optionally in basic mode, is discarded.)

In any other case there is some syntactical ambiguity. The syntactic translator will produce a corresponding SGML open tag unless the logical variable *gellmu-xml-strict* has been set.[21] If the usage is consistent with the structure of the document type, an SGML parser will in many cases be able to handle the result correctly. The result of this type of syntactic ambiguity in source markup is not tolerated if one is editing directly for an XML document type. The terminator can be a blank space, but, if so, the blank space is likely to become invisible after SGML parsing much in the way that in LaTeX the markup

```
\LaTeX document
```

will be collapsed into the single word form "LaTeXdocument" when typeset.[22]

5.2.2 Certain single characters.

The characters `'\'`, `'{'`, `'}'`, `'^'`, `'_'`, `'$'`, `'%'`, and `'~'` have command meanings that are similar to their meanings in LaTeX. The characters `';'` and `':'` are ordinary characters that have special meaning at the end of a command name. The character `'#'` is also a special character used, as with LaTeX, in *newcommand* templates. In source for the didactic *article* document type any non-alphanumeric ASCII character may be escaped (referenced for itself) with a named command, e.g. `'~'` may be referenced for itself as `\tld;`. This is **necessary** in order to provide delayed evaluation for ultimate translation to one of many possible ultimate formats.

The following language meanings apply to both basic and advanced markup:

1. "\": **Command introducer.**
   Escape in basic mode: `\\` . This escape is incorrect in advanced mode since this notation

[21] The variable *gellmu-xml-strict* is by default unset in advanced mode.

[22] The correct LaTeX markup is "`\LaTeX{} document`". In the didactic production system the name of LaTeX is "latex", which is a defined-empty element, that for the SGML version of *article* may be marked up safely either as "`\latex;`" or as "`\latex{}`".

has a different meaning — forced linebreak — in LaTeX itself. For the didactic *article* document type the escape is `\bsl;` . For other document types one may resort to an entity reference if adverse to providing a corresponding defined-empty element or if one lacks control of the document type.

2. "`{`": **Command argument opener.**
   Escape: `\{` or `\lbr;` .

3. "`}`": **Command argument closer.**
   Escape: `\}` or `\rbr;` .

4. "`[`": **Command option opener.**
   Escape: `\lsb;` (usually not necessary[23]).

5. "`]`": **Command option closer.**
   Escape: `\rsb;` .

6. "`;`": **Command terminator for defined-empty tags.**
   Escape: `\scl;` . Usually an ordinary character. Its use as a command terminator is invisible and may be omitted optionally in some contexts. This syntax is analogous to the use of '`;`' as an entity reference terminator in SGML.

7. "`:`": **Command terminator for close tags.**
   Escape: `\cln;` . Otherwise an ordinary character. Its use as a command terminator is invisible.

8. "`%`": **Comment introducer, in force to end of line.**
   Escape: `\%` or `\pct;` .

9. "`#`": **Argument marker in *newcommand* definitions.**
   Escape: `\#` or `\hsh;` . In the definition of a newcommand "`#1`" indicates the first invocation argument, "`#2`" the second invocation argument, etc. (There is no limit on the number of arguments.)

The following language meanings apply to advanced markup with allusion to the didactic *article* document type.

1. "`~`": **Non-breaking interword space.**
   Escape: `\tld;`.[24]
   Equivalent: `\nbs;`, cf. `&nbsp;` in HTML.

2. "`^`": **Superscript command.**
   Escape: `\crt;` .
   Equivalent: `\sup` or, in math, `\msup` .

3. "`_`": **Subscript command.**
   Escape: `\_` or `\und;` .
   Equivalent: `\sub` or, in math, `\msub` .

[23]In math '`[`' sometimes needs to be escaped to prevent confusion between its markup use and its ordinary use in an instance such as the markup for $\mathbb{Z}[t]$. The syntactic translator would need to know vocabulary — at least the argument/option pattern for *mathbb* — in order to elude the syntactic ambiguity.

[24]"`\~`" is an example of a markup string that is defined in LaTeX (for an accent) that is not defined in the didactic document type. A LaTeX user may recover a prior markup habit of this type using *newcommand* possibly in combination with *macro*. For more information see the discussion of accents (section 5.8).

4. "`&`": **Dual use: tabular cells and entity introducer.**
   Escape: `\&` or `\amp;` .
   '`&`' introduces an entity reference if it is followed by anything other than white space. It is used, as in LaTeX, as a *tabular* cell delineator when it is followed by white space.

5. "`$`": **Toggle inline math mode.**
   Escape: `\$` or `\dol;` .
   Equivalent: "`\tmath{ . . . }`".
   Nearly equivalent: "`\( . . . \)`" or "`\math{ . . . }`".[25]

5.2.3 Certain strings

These are strings of plain text with markup significance in the didactic document type that are part of markup in LaTeX.

1. "`--`"
   Short dash as used with a range of numbers, e.g., 1–2.
   Equivalent: `\rdash;` .

2. "`---`"
   Long dash as used for punctuation, e.g., a dash — like this.
   Equivalent: `\pdash;` .

3. "`\ `"
   Blank interword space.
   Equivalent: `\spc;` .

4. "`\,`"
   Small horizontal space.
   Equivalent: `\hsp;` .

5. "`\\`"
   Forced line break.
   `"\\"` may be used at the end of a line of input for a forced line break. In a *tabular* environment (with the didactic *article* document type, as in LaTeX) it begins a new *tabular* row. Any other use is deprecated, and will result in translation to the defined-empty element *bsl* corresponding to the ASCII character '`\`' with a warning from the syntactic translator.
   Equivalents: `\brk;` for a line break outside of a *tabular* environment.[26]

6. Blank line.
   Begin new paragraph command.
   Equivalent: `\parb` . Nearly equivalent: `\par` .

[25]It is possible to merge the inline *math* and *tmath* zones at any level of processing beyond the syntactic translator. These are indeed the same in LaTeX, but the syntactic translator resists the temptation here to go beyond syntax and merge them. With the didactic *article* document type the formatting to LaTeX inserts the LaTeX markup `"\,"` for a small horizontal space before and after *math*, but not before and after *tmath*.

[26]The syntactic translator simply outputs the SGML defined-empty element *brk0*, which belongs to its reserved name space. The dual use of *brk0* involves some SGML chicanery that is resolved during translation to the XML version of the *article* document type, where *tabular* is converted to *table* and non-tabular use of *brk0* is converted to *brk*. See also the handling of *array*, which is different even though the source markup, as in LaTeX is similar.

7. "``"
   Left (double) quotation mark.
   Equivalent: `\ldq;` .

8. "''"
   Right (double) quotation mark.
   Equivalent: `\rdq;` .

9. "`\(`"
   Begin *math mode* command.
   Equivalent: `\begin{math}` .

10. "`\)`"
    End *math mode* command.
    Equivalent: `\end{math}` .

11. "`\[`"
    Begin *displaymath mode* command.
    Equivalent: `\begin{displaymath}` .

12. "`\]`"
    End *displaymath mode* command.
    Equivalent: `\end{displaymath}` .

13. "`.  `", "`?  `", "`!  `"
    End-of-sentence marks.
    Equivalent: `\eos;`, `\eoq;`, `\eoe;` .
    A period followed either by two blank spaces or by a newline is recorded as an end of sentence. There is similar provision for the question mark and the exclamation point. These tagged forms may be used explicitly for the corresponding punctuation inside math displays to distinguish punctuation from mathematical use of the punctuation symbols. Explicit markup for a comma is "`\cma;`".[27]

5.3 Large Structure

Overall an *article* consists of a *preamble* followed by a *body*. As noted previously (section 3.3) advanced mode provides a special form of usage

```
\begin{document} . . . \end{document}
```

with the paired metacommands *begin* and *end* that with an *article* delimit its *body*. This is enabled with the *gellmu-trans* call for the syntactic translator, and, consistent with regular LaTeX usage the *preamble* is present without explicit tagging.

The only required markup in a *preamble* is a *title*, and it is formally correct for its content to be empty. The SGML content model for a *body* is somewhat loose, but is usually understood to consist of *section*s and may contain ordinary paragraphs (*par*, which must be marked up explicitly, or *parb*, which is begun with a blank line). Although a *body* may be entirely empty

[27] Alternate forms "`\aos;`", "`\aoq;`", "`\aoe;`" of sentence ending punctuation are provided that may be used following inline mathematical markup at the end of a sentence. Similarly, "`\aoc;`" is an alternate form for a comma.

(likely not useful) or may consist only of *par* and *parb* elements, all inline text must be within one of these basic paragraph containers.

For example this formally correct textless document is handled without noise by the didactic production system:

```
\documenttype{article}
\title{}
\begin{document}\end{document}
```

while the following document with text outside of a paragraph fails initial validation in the didactic production system:

```
\documenttype{article}
\title{}
\begin{document}
x
\end{document}
```

The error may be corrected by placing a blank line before the character `'x'`.

The content model for *preamble* is tighter than that for *body*. This makes it possible to have a greater level of error-checking than would otherwise be possible. For example, a preamble must have exactly one *title* although it is not specified where in the preamble a title might be. There may be at most one of each of the elements *surtitle*, *subtitle*, and *date*. Although *newcommand*, which is a metacommand, does not affect the document type definition and may be used at different locations in the preamble, the small number of actual elements that may be multiply used in the preamble, such as *mathsym* (which is partly a metacommand) must be located together.

5.4 Sectioning

In classical LaTeX one writes simply

```
\section{Some title for a section}
```

to begin a new section. While this markup specifically delineates the section title, LaTeX understands that a section has begun. The *section* command has an option whose content is an alternate title for the table of contents, and the starred form of the command suppresses an otherwise automatic section number.

With SGML the classical approach is something along the following lines:

```
<section><sectiontitle>Some title for a section</sectiontitle>
<para> A paragraph.
<para> Another paragraph.
    . . .
</section>
```

Basic GELLMU markup corresponding to this would be:[28]

[28]Or one could use:

```
\Section{\sectiontitle{Some...}
```

```
\begin{section}\sectiontitle{Some title for a section}
\para A paragraph.
\para Another paragraph.
    . . .
\end{section}
```

If, moreover, the markup is to be well-formed XML markup, then each *para* tag would need explicit closure with `</para>`.

The main point here is that the open and close tags for a section in a classical SGML document type encompass the whole contents of a section, and separate markup is required for the section title.[29]

The didactic SGML document type under *advanced* GELLMU seeks to model classical LaTeX as closely as possible in order to provide a bridge for authors from LaTeX to SGML (and, indeed, XML). For this reason a command *section* similar to the classical LaTeX command that delineates a section title is provided in the didactic SGML document type. At the same time this document type has a whole section container *Section* (upper case *S*) that in the simplest case consists of a *shead* container for its title (or header) followed by any number of paragraphs. The XML form of the didactic document type supports only this latter tag, which means that the translation script that converts SGML to XML has an unambiguous way of performing the conversion from *section* to *Section* provided that the author's source leads to an SGML instance which is valid[30] under the didactic document type.

5.5 Labels, References, and Anchors

A *label* command may be placed anywhere that text may be placed in order to mark a location and associate a symbolic "key" with that location. A *ref* command may be placed anywhere that text may be placed to generate a visible allusion called its *reference value*, for example a section number, to the location associated with a label key. An *anch* command may be placed anywhere that text may be placed to provide a hypertext reference either by key to an internal location or by uniform resource identifier (URI) to an external resource.

At this point the discussion will become descriptive of of the entire didactic production system rather than simply of its document type.

5.5.1 Labels and Sequencing

The basic usage for *label* is

`\label[:series="` *series-name* `" serseq="` *number* `" refkey="` *ref-key* `"]{` *key* `}`

```
\para A para...  . . .}
```
instead of using the environment-like *begin*/*end* construction.

[29]In fact, classical SGML document types are often even more elaborate than this.

[30]Caveats:

No document type definition under SGML is actually a complete language definition. A document type definition describes a document markup structurally; in particular, it does not provide definition for legal "field" values.

While the GELLMU syntactic translator is now considered as "alpha" software, the document type and the accompanying translators, which constitute the didactic production system are developmental. These materials have some support for obsolete practice and also contain sketch sections that are not fully robust.

where use of the attributes is optional and, moreover, the required *key* may be empty, i.e., the markup `\label{}` is permitted. The *key* must be a case-sensitive string that is **monocase** unique.[31] The didactic production system will provide a unique automatically generated string value for *key* if none is provided by the author; this can be useful if a *series* name is specified as an attribute for the label.

An author should never reference a label key not provided by the author, but the empty element *popkey* is intended for processing evaluation as the last label *key*, automatic or not, preceding its location.

Sequencing[32] may be handled under the GELLMU didactic *article* document type using labels and references. Toward this end one makes use of three SGML attributes that are provided with the *label* tag:

series The value is the name of a sequenced family of labels. A label may belong to at most one family, but there may be multiple labels at the same location. There is no default value of series.

serseq The value is the sequence number of the label in its series if a series is defined. It is meaningless if no series is specified for the label.

refkey The name of a label key from which to spawn an automatically generated value for the attribute *serseq* of the current label.

Every label has a reference value that is normally accessed with the *ref* command. This results in the creation, when the XML version is generated, of an XML entity reference with name based on the reference's *key* argument, that matches a CDATA entity definition at the top of the XML document. The use of indirection provided by this entity technique means that it is immaterial whether the reference is forward or backward.

The *evalref* command gives access to the literal value of a reference without indirection, and places that value as a literal in the XML version of an article. Thus, *evalref* is the name of a tag only in the SGML document type and in the author-level XML document type but not in the elaborated XML document type. An *evalref* invocation will be successful only with a backward reference. This is extremely useful for managing non-default numbering of sectional units. For ordinary label references its use is undesirable even though it is possible for backward references.

The reference value of a label is determined as follows:

1. Basic reference values are positive integers. Upper and lower case alphabetic values and upper and lower case roman numeral values may be obtained by applying the *series* command (not to be confused with the *series* attribute for the *label* command) to a basic reference value with *type* attribute of *series* set to one of `'A'`, `'a'`, `'I'`, or `'i'`.

2. If the label is assigned to a label series and is given a *refkey* attribute, then the reference value of the label is the reference value of the label referenced by the key that is the value

[31]It is recommended that the characters in label key strings be restricted to lower case ASCII letters, the digits 0–9, and possibly the character `'-'` or the character `'.'` for maximal inter-operability with current and future formattings. For example, the `'_'` is problematical in this context.

[32]Although one *could* provide SGML modeling for LaTeX's counters, it would not be very much along the lines of main track SGML or XML document types.

of the *refkey*. (This mechanism is used to re-start the sequencing of the sectional unit id's at the end of this document.)

3. Else if the label is assigned to a label series and the author supplies an explicit **literal** numeric value for the *serseq* attribute, then the value of *serseq* is its reference value. (Markup — in particular, *evalref* — cannot be used in defining an attribute value.)
4. Else if the label is assigned to a label series, the reference value of the label is the next (positive integer) value for a label in that series. (This mechanism is used to control the sectional id of the last section of this document. It may also be used to run parallel interleaved sequences of sectional units, such as, for example, questions and answers, within a document.)
5. Else the reference value of the label is the sectional unit identifier, i.e., its *sunit* value rather than the logical value in its attribute *sid*, of the smallest sectional unit containing the label. These values may not be numeric. For example the string "A.3.2" might be a sectional unit identifier. The *series* command should not be used to express type conversion of a reference value that might resolve as a sectional unit identifier.

5.5.2 Anchors

The basic usage for *anch* is

`\anch[` *reference specs* `]{` *anchored text* `}`

where the option, which is not an attribute option but rather becomes the element *anchref* in XML (while the anchor's argument — its "presented content" — becomes *anchv*), is expected to contain white space separated strings of the form `name="value"`[33] with name restricted to one of the following:

fref

A footnote reference. Value is a string that becomes the content of an automatically created footnote to the text in *anchv*. This usage is deprecated; use \footnote instead.

href

A web reference as with *href* in HTML. That is, value is a URI. It *could be* of the form `"#labelkey"` where `"labelkey"` is the name of a label key that is preferably and more easily used with *iref*. (The `'#'` needs to be escaped, i.e., marked up as `"\#"`.) In a non hypertext formatting the URI may be presented as a note or footnote associated with the text in *anchv*.

Href

Same as *href* except that the author wishes to suppress any note or footnote presentation of the URI. This might be the case if, for example, the URI might be obviously deducible from the *anchv* content.

[33]The value strings may contain simple markup such as, for example, "`\tld;`" to provide robust multiple output processing of `'~'` whereas an attribute option may not contain markup.

iref

An internal reference; value must be a label key arising from *label* or *klabel*[34].

Note also that there is a command *urlanch* (probably should have been *urianch*), taking a single argument, used for URIs, which is intended to have the same effect as a newcommand with one argument for creating a web anchor with the URI as presented content.

5.5.3 Example Emulating a LaTeX Counter

Suppose that one wants to fashion an inline enumerated list at some point in a document. An *enumerate* environment is a list structure that, while it may occur in a paragraph, is not inline. The idea is to give each item a label and fashion the inline list item number by referencing that label. One writes a newcommand to ease this.

The name of a LaTeX counter is emulated with the name of the *series* attribute for one or more labels. Unless one wishes to be able to reference the items apart from the immediate reference, one need not provide a label key. The didactic production system will provide a default labelkey and the command "`\popkey`", which grabs the key of the last preceding label, may be used to furnish the key for the immediate reference. If one wants small Roman numerals, one wraps the "`\ref`" command in a "`\series`" command. Since *series* requires an actual number, one must use *evalref* instead of *ref*, which is permitted so long as the reference follows the label. Here's the markup:

```
\label[:series="foo" serseq="0"]{} % zero the counter named foo
\newcommand{\ti}{%
\label[:series="foo"]{}(\series[:type="i"]{\evalref{\popkey}})}
Hilbert's three most important contributions to algebraic geometry
are \ti~the basis theorem, \ti~the nullstellensatz, and \ti~the syzygy
theorem.
```

The rendering is this: Hilbert's three most important contributions to algebraic geometry are (i) the basis theorem, (ii) the nullstellensatz, and (iii) the syzygy theorem.

With the current didactic production system if one had used "`\ref`" instead of "`evalref`", the translator from author-level XML to elaborated XML, which resolves references, would have issued a warning in the scroll, but the build would have run to otherwise successful completion by ignoring the presence of the *series* command.

5.6 General Usage for Sectional Units

This describes the content model in the GELLMU didactic document type for the "whole" sectional units *Section*, *Subsection*, ... as fully tagged.

5.6.1 The Content Model

The content model for sectional units is:

[34] A *klabel* is a "visible key" label.

```
((sopt)?,(sprefix)?,(sunit)?,(shead),(%UnitContent)*)
```

where:

%UnitContent

refers to the subsections, loose paragraphs, and other general content that is allowed inside a section.

shead

is the required[35] section header or title.

sopt

is an optional title for use in the table of contents[36].

sprefix

is optional markup for text that is to precede the sectional unit sequence notation. For example, if the sectional unit sequence is "B" and *sprefix* is the markup string `"Appendix "` (ending with a blank space), then the visible indicator for the sectional unit, when consistent with the setting of *secnumdepth*, is "Appendix B" both at the beginning of the section and in the table of contents. Or if the sequence is "3" and the *sprefix* is "A", then the visible indicator is "A3".

sunit

is an optional setting for the sectional unit sequence. The GELLMU didactic document type has an attribute "sid" for the sectional units *Section*, *Subsection*, ... that is optional (and rare for author usage) in the SGML version but required in the XML version. The translator from SGML to XML computes it in the standard way. For example subsubsection 1 of subsection 3 in section 2 acquires the *sid* "2.3.1". It is expected that formatters will use this value for the visible sectional unit indicator, preceded, as previously described, by any *sprefix*, unless the user provides *sunit*. While *sunit* is intended to override the visible indicator, it is not provided to override the *sid* attribute itself which a formatter should see as describing logical structure.

5.6.2 The LaTeX-Like Form of General Usage

Corresponding LaTeX-like *argument/option* syntax can be used, as previously indicated, in GELLMU source with *section*, *subsection*, ... If, however, *argument/option* syntax is used, one must be mindful of the ordering of the options, and use empty option brackets, as necessary, to indicate the position in the sequence of an option with content. For example, a sole option is understood as *sopt*, the version of the sectional unit title to be used in the table of contents. To provide only *sprefix* with *argument/option* syntax one precedes the bracket sequence for *sprefix* with an empty pair `"[]"` of option brackets[37] for *sopt*.

[35]It may be left empty, but it must be present.

[36]The presence of *sopt* does not cause a table of contents to be produced automatically. For that one uses `"\tableofcontents"`. Moreover, the presence of *sopt* should have no effect upon a manually constructed `"\TableOfContents"`.

[37]The didactic production system offers a way to furnish a formally empty string, which is an empty element called *empty* in the document type for use in places such as the the table of contents option of a sectional unit, where it is not otherwise possible to distinguish after parsing whether deliberately empty content was specified by the author. That is, the markup `"\sopt{}"` furnishes an empty string which, in turn, signals "no *sopt*", while `"\sopt{\empty;}"` indicates that empty content was specified for *sopt*.

5.7 *verbatim*, *verblist*, and *manmac*

The ordinary notion of "verbatim" is complicated in the context of a document type that is intended for processing toward multiple output formats. There is no special provision for verbatim markup under *basic* GELLMU[38], but there are two layers, one simplistic and one sophisticated, in the didactic production system with each layer having provision for both inline and block-level verbatim markup.

The simplistic approach involves the provision of an inline element *verb* and a block-level element *verbatim* in the didactic document type. With these the author is responsible for entering special characters in ways that are safe for input source notation, safe for syntactic translator output, and safe for each output format that is envisioned.[39] In the simplistic layer the formatting program in the didactic production system for HTML output renders *verb* as an HTML *kbd* element and *verbatim* as an HTML *pre* element. In formatting for regular LaTeX output each of these SGML elements is rendered as its LaTeX namesake. Neither the *pre* element in HTML nor the (basic) *verbatim* command in LaTeX is context-sensitive, and typically are formatted with left margin justification.

The sophisticated layer in the syntactic translator for *verbatim* is enabled by setting the variable *gellmu-verbatim-clean* true. If this is the case, then a user should input verbatim material literally between `\begin{verbatim}` and `\end{verbatim}` markers occupying whole lines and the syntactic translator will convert each of the 33 non-alphanumeric printable ASCII characters therein to the corresponding empty element in the didactic document type, and render each line of the converted material as a list item bearing the item name *nln* in a list element *verblist*.[40]

Similar arrangements pertain to the sophisticated inline analogue of *verb*, which is enabled in the syntactic translator by setting the variable *gellmu-manmac-bar-name* to a non-empty string value that is to become the name of the element, such as *quostr*, to contain the content, which should be delimited in source by successive `'|'` characters. By default the user must enter a verbatim string in "safe" form, but may enter it, apart from any occurrence of the character `'|'`, literally if the variable *gellmu-manmac-literal* is true. The term "manmac" is derived from an old plain TeX package for writing documentation by that name in which the character `'|'` is used to delimit inline literal material for verbatim presentation. Moreover, consistent with usage observed in LaTeX documentation markup of the form

\name | *literal-text* |

is translated by the syntactic translator to an element named as with simple `"|...|"` having a setting for its attribute with name the value of the string variable *gellmu-manmac-attribute*, which for *quostr* in the didactic document type would appropriately be its attribute *inv*.

Variable settings to provide for the use of literal input both for *verbatim* as a metacommand front for the list element *verblist* and *manmac* configuration for the element *quostr*, as described, are default when the syntactic translator is begun with the non-default function *gellmu-latex-faq*. This function also sets the variable *gellmu-squophrase-name*, which otherwise defaults to the empty string, to the value `"squophrase"`. This causes the syntactic translator to attempt

[38]When editing for HTML one may, of course use HTML's *pre*, which stands for "pre-formatted text"

[39]With a sufficiently long list of output format candidates each of the 33 non-alphanumeric printable ASCII characters is unsafe. However, one might use an external character-to-string conversion program to prepare a large amount of verbatim material for inclusion inside the simplistic *verbatim* command in GELLMU source.

[40]To export this procedure for general *advanced* mode usage, all of the names used need to be made user-configurable.

to interpret balancing character pairs consisting of the character `` ` `` and the character `'` as delimiters for the element *squophrase*, an alternate form along with *quophrase*, for "quoted phrase", with odd instances of the character `'` set as the empty element *apos*, which represents an apostrophe.

5.8 Accents

Traditional LaTeX markup employs accent commands for modifying ASCII characters in order to produce non-ASCII characters. On the other hand characters from 16 "byte-planes" of Unicode are now available in HTML and XML as ordinary characters, with treatment as the atoms of ordinary strings of text not requiring any special attention or notice from the viewpoint of markup. It may happen that this will affect development in the LaTeX project, and it is, therefore, unclear what the long term role might be of LaTeX's accent commands.

Nonetheless, the didactic document type provides element names corresponding to the 14 classical LaTeX accents although it does not provide classical markup notations such as `"\'"`, `"\""`, `"\~"`, ... inasmuch as names are required in syntactic translator output.[41] The names for the accents may be seen by locating the word "accent" in the didactic production system file `gellmu.dtd` and reading the following 14 lines.

An author, particularly an author residing in an English language locale, may introduce, for example, the character é in several ways:

1. by using an algorithmic accent command — in this case `\acute{e}`: é.
2. by direct Unicode-based entity reference – in this case `&#xE9;`: é.
3. by simple direct entry of the Unicode point in a file having the UTF-8 text encoding.

Note that Unicode values are numbers represented in hexadecimal notation (base 16); the digits are 0–9 and A–F. Thus, E9 is 233, and one could also use the entity reference `&#233;`.

While for the long term, the largely equivalent[42] second or third methods are superior, there are caveats for their use in 2007:

- One's LaTeX back end must have a suitably robust way of handling Unicode, presumably via the *inputenc* package or possibly via *Omega/Lambda*.
- Even when fonts are available, some web browsers seem only to handle Unicode via entity references.

5.9 Tabular Environment Emulation

At this point (2006-7) the didactic document type is able to accomodate emulation of LaTeX's *tabular* environment with `"l"`, `"c"`, `"r"`, and `"p"` column cell specifiers with, where robust CSS support is available for HTML, `"|"` indicators for vertical cell rules and `"\hline"` commands

[41] Familiar short forms may be introduced by a user using the *macro* facility.

[42] For the third method one's GELLMU driver script must provide appropriately for the text encoding of the source file.

for horizontal rules between rows. While *multicol*, and *multirow* are not modeled, cells may contain other *tabular*s recursively.

A "p" column specifier optionally takes a decimal argument that represents the fraction of available width to be made available for the cells in that column. When is no fractional width specifier, the default fractional width is $\frac{1}{n+1}$ where n is the total of number of columns.

What may be viewed as a shortcoming of the *tabular* emulation is that in each *tabular* row the first cell must contain some markup in order for the document to be structurally correct. This arises from a rule associated with SGML document types that might be overcome if GELLMU source were processed by a monolithic program. Instead, however, it is an assemby of separate components (appendix 7.5.2) in which the first stage has knowledge of syntax but not of the markup vocabulary.

One thing to keep in mind with *tabular* is that in the current, though probably not future, design of HTML tables are not allowed in paragraph content. The didactic production system works hard to deal with this. In most web browsers there will be a line break before an HTML table and again after an HTML table. (One way to finesse that is to place a table in a cell in a sur-table, which is abuse of markup.) This author generally places a *tabular* inside a *display*, which is the GELLMU object corresponding to LaTeX's *center*.

The document type does not offer "floating" objects, and, therefore, the name *table* is not used as as in LaTeX.

The name *table* is used for emulation in regular GELLMU source of HTML tables. A *table*, like a *tabular*, takes a required argument consisting of column specifiers. In the didactic production system at the point where an article is spun to the elaborated XML version of *article*, there is no longer a distinction between *tabular* and *table*.

There is also *array*, which, as in LaTeX, is the in-math version of *tabular*, except that in GELLMU its emulation of LaTeX's array survives translation to the XML version of *article*. An *array* has only "l", "r", and "c" cells.

Example: The markup for the table of DTD's in Appendix B.2.2 is this:

```
\begin{display}
\begin{tabular}{l|cc}
~                         & Latin-1              & \abbr{UTF-8}           \\
\hline
First stage \abbr{SGML} &  \quostr{gellmu.dtd} &  \quostr{ugellmu.dtd} \\
Author Level \abbr{XML} &  ~                   & \quostr{axgellmu.dtd} \\
Elaborated \abbr{XML}   & \quostr{xgellmu.dtd} & \quostr{uxgellmu.dtd}
\end{tabular}
\end{display}
```

Because the one horizontal divider and the one vertical divider rely on CSS support in HTML, these dividers might not be seen in a web browser with weak CSS support.

5.10 Graphic Inclusions

There is basic support for graphics inclusions suitable for the HTML backend and the LaTeX backend with both DVI and PDF outputs. The arrangements are not unlike those required for the use of the *includegraphics* command provided by LaTeX's *graphicx* package. This means that, apart from the didactic production system's chain of processors, one needs to provide several different versions of a given graphic object in order to accommodate the needs of the three different output formats (HTML, PDF, and DVI). A reliable graphics format converter such as that provided by *ImageMagick* (`http://www.imagemagick.org/`) or the *netpbm* utilities (`http://netpbm.sourceforge.net/`) and a broad TEX support environment similar to that provided by TUG's *TeXLive*[43] are essential for work with included graphics.

For example, if one begins with an encapsulated PostScript image `glmy.eps` made, say, with METAPOST, then one might run the commands

```
epstopdf glmy.eps
convert glmy.pdf glmy.png
```

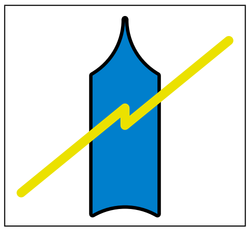
before making PDF with *pdflatex* and PNG (for inclusion in HTML). In this case the PDF version of the graphic should be regarded as best for use with *pdflatex*, and so one wants to have the PNG version out of view from *pdflatex* when building the GELLMU source `glman.glm` for this document. With the new *mmkg* drivers one may prepare a tiny file `glman.prx` that is a list of names of image files, one file per line, that should be out of view at the point in the pipeline when *pdflatex* is active. Thus, a line in the file `glman.prx` should contain the name `glmy.png`.

In `glman.glm` one has the code:

```
\begin{quotation}
   Sometimes a picture is worth a thousand words.
   \display[:frame="1.1"]{\includegraphics[:scale="0.2"]{glmy}}
\end{quotation}
```

which yields:

Sometimes a picture is worth a thousand words.



In this markup one sees, first of all, two uses of SGML attributes — *frame* and *scale*. Attribute options are opened with `"[:"` rather than simply with `"["`. Whereas options generally may contain markup, attribute options may not.

The meaning in this example of *scale* is that the **width** of the graphic in DVI and PDF outputs will be the result of multiplying the *scale* by LATEX's value of \textwidth.[44] The HTML formatter will link to the PNG without a width specification. Thus, the brower window width of the graphic will be the actual width of the PNG image unless, perhaps, it's too large for the browser's window. The meaning of *frame* is that for print outputs the graphic image will be surrounded by a LATEX *framebox* with diameter that is 1.1 times the diameter of the

[43]URI: http://www.tug.org/texlive/

[44]This meaning of *scale* differs from the meaning of *scale* with *includegraphics* under LATEX's *graphicx* package. New controls of this type may be introduced in a future version.

display's content. In the current HTML formatter no use is made of the numeric value of *frame* although a default frame is constructed via an HTML attribute giving values for the CSS properties *border* and *padding.* When image framing is desired, another approach is to incorporate framing in the graphic itself.

6 Mathematics in *article*

6.1 General

There was previous mention of the basic mathematical container element *tmath* in the discussion of single string (section 5.2.2) markup and the *math* and *displaymath* elements in the discussion of certain strings (section 5.2.3) of special markup significance.

The markup used inside these containers is very similar to that which is used in LaTeX although far from all of LaTeX's math markup, as extended by the *amsmath* package, has any analogue in the didactic production system.

There are a few things that deserve short mention:

\sum, \int, and \prod

are all similar to their LaTeX namesakes except that each requires explicit closure. For example,

```
\[ \sum_{0}^{\infty} \frac{x^k}{k!} \sum: \]
```

produces

$$\sum_{0}^{\infty} \frac{x^k}{k!} \quad .$$

\regch, \mbox, and \text

The didactic document type provides *regch*, for "regular character" that is a one character version of *mbox* provided for separating from general *mbox* matter the content to which a non-math algorithmic accent (section 5.8), may be applied. For example,

```
\newcommand{\Q}{\regch{\bold{Q}}}  % intended for use in math
\newcommand{\galQ}{\mbox{Gal}(\ovbar{\Q}/\Q)}  % only in math
$\galQ$\aos;
```

produces Gal($\bar{\mathbf{Q}}/\mathbf{Q}$).

Both *regch* and *mbox* should be used only for symbols. Note that "Gal" in the foregoing is a symbol. The command \text is provided for the use of text phrases – usually conjunctive in nature – inside math zones. For example, the markup

```
\[ \absval{x} = \lbalbr{\begin{array}{rl}
             x & \text{\ if\ } x \geq 0 \\
            -x & \text{\ if\ } x < 0
            \end{array}} \]
```

produces

$$|x| \;=\; \left\{ \begin{array}{rl} x & \text{ if } x \geq 0 \\ -x & \text{ if } x < 0 \end{array} \right.$$

In this example note that the command name *lbalbr* stands for “**l**eft **bal**anced **br**ace”. It corresponds to the use of `\left\{ ... \right.` in LaTeX[45].

\aos;, \aoc; \aoq;, and \aoe;

are the named forms of the LaTeX space adjustment commands `"\@."`, `"\@,"`, `"\@?"`, and `"\@!"`, which provide correct placement of inline punctuation immediately following inline mathematical markup. There is now also default provision in the didactic production system for the more LaTeX-like `"\@"` usage. Note that the use of `\@` is seldom necessary.

6.2 Assertions: Near Emulation of LaTeX’s Theorem-Like Environments

6.2.1 Examples

The container element *assertion* is used in the didactic document type for the creation of its analogue under SGML of theorem-like environments. As a first example of the near emulation of a LaTeX theorem-like environment, here is markup to give LaTeX-like meaning to “`\begin{theorem}`” and “`\end{theorem}`”.

```
\newcommand{\begin{thm}}[1][]{%
\begin{assertion}[#1][theorem]{Theorem}[\evalref{\popkey}]}
\newcommand{\end{thm}}{\end{assertion}}
```

For a first instance it is invoked without supplying the optional first argument, which is for a label key.

Theorem 1. *The continued fraction expansion of a real number is finite if and only if the real number is a rational number.*

This is just text to follow the statement of a theorem.

If we do the same thing again, the theorem number should go up by 1 since in the didactic production system this usage is equivalent to using the default counter associated with a label series name *theorem.*[46]

Theorem 2. *The continued fraction expansion of a rational number is eventually periodic if and only if the real number is an irrational number that is the root of some quadratic polynomial with rational coefficients.*

6.2.2 Usage for *assertion*

The general usage of *assertion* is the following:

```
\begin{assertion}[key][series ]{name}[id]
   . . .
```

[45]Note that the use of *lbalbr* in this instance is insufficiently semantic for translation to content MathML while it is meaningful for translation to presentation MathML. Adequate enhancement might be had by providing `mml="cases"` (using a name from *amsmath*) as an attribute for *lbalbr* with this example.

[46]There is also a default counter that is used when no label series name is present. That counter simply is the position of the underlying *assertion* in the list of all assertions.

```
\end{assertion}
```

The options *key* and *series* represent the same things as the corresponding *label* (section 5.5.1) options. The *id* option is for explicit customization of the visible identifier, e.g., theorem number, as illustrated with one of the examples later in this section. One may use the *id* option, which must follow the name argument, without using either of the other options. But in order to use the *series* option, a *key* option, which may simply be empty, must be present.

There is also a fully named way to proceed:

```
\begin{assertion}
\asstkey{. . .}\asstser{. . .}\asstname{. . .}\asstid{. . .}
  . . .
\end{assertion}
```

Here order is important, but any of the options may simply be omitted. For example, we may write

```
\newcommand{\begin{Thm}}[1]{%
\begin{assertion}\asstser{#1}\asstname{Theorem}%
\asstid{\sref;.\evalref{\popkey}}%
}
\newcommand{\end{Thm}}{\end{assertion}}
\begin{Thm}{XXseries}
If $[ n_1, n_2, \ldots ]$ is an infinite continued fraction, then
its sequence of convergents always has a limit.
\end{Thm}
```

to obtain:

Theorem 6.2.2.1. *If* $[n_1, n_2, \ldots]$ *is an infinite continued fraction, then its sequence of convergents always has a limit.*

Notice that the *asstid* argument is merging the reference value for (the new) series *XXseries* with the visible id of the current sectional unit.

6.3 Equations and Equation Arrays

The general usage for *equation* is the following:

```
\begin{equation}[key][series]
  . . .
\end{equation}
```

The options *key* and *series* represent the same things as the corresponding *label* (section 5.5.1) options. In order to use the *series* option, a *key* option, which may simply be empty, must be present. The use of "`equation*`" as a name for an equation display that is not numbered is not permitted, but one may instead use the *nonum* attribute as follows:

```
\begin{equation}[:nonum="true"] . . . \end{equation} .
```

General usage for an equation array (name *eqnarray*) is:

```
\begin{eqnarray}[key][series]
  .  .  .
\end{eqnarray}
```

where the content is an *eqnabody* consisting of *eqnrow*'s, each row may be terminated in LaTeX-like markup, as in LaTeX, with the string "`\\`" and consists of three parts, corresponding to elements *eqnleft*, *eqncenter*, and *eqnright*, that may be separated in LaTeX-like markup with the character '`&`'.

Support for numbering in eqnarrays is only minimally developed although there is suggestive sketching in the didactic document type that is not supported in the formatters. By default the equations in equation arrays are numbered consecutively throughout an article. This behavior can be altered by using the *series* attribute of an *eqnarray.* If that is done, then, as things have been sketched, numbering applies to whole arrays rather than to the equations within arrays. Numbering in an equation array may be suppressed by setting its attribute *nonum* to the string "`true`".

6.4 The \mathsym Metacommand

mathsym is a macro substitution metacommand that is available in the didactic production system for enabling an author to declare that a macro name represents a mathematical symbol. It is a formal way of recording statements commonly made by authors in introducing notation.

Unlike regular metacommands, which may appear at any point in GELLMU source, *mathsym* may only appear in the *preamble* of an *article*, or, equivalently with defaults in the syntactic translator, *mathsym* may only appear before the LaTeX-like `"\begin{document}"`.

Its usage is:

```
\mathsym{ symbol-name }{ symbol-rendering }[symbol-meta-info ] .
```

Here *symbol-name* is an alphanumeric string (case-sensitive) beginning with a letter. The second argument is the presentation rendering of the symbol in GELLMU markup. It is like the definition of a *newcommand* except that it may not involve arguments.[47] The optional third argument *symbol-meta-info* is an alpha-numeric string that might also include possibly a few other string characters such as '`/`', '`-`', '`,`', '`.`', '`*`', etc. Its exact structure depends on the typing system. (No typing system is part of the didactic production system.) For example, it might consist of (name, value) pairs for conveying meta-information about the symbol.

The syntactic translator replaces each invocation of a given *mathsym* with the specified rendering and writes for each *mathsym* definition a corresponding element in the SGML output whose content consists solely of the declared symbol name if there is no meta information but otherwise consists of the symbol name followed by a blank space and then whatever string of meta information is provided in the optional argument. Additionally, each invocation is wrapped in a rendering-inert *Sym* element whose *key* attribute reveals the name given to the symbol at the point of declaration (and by which the symbol is invoked). This makes it possible for a downstream authoring platform processor that has remembered the list of declared symbol names to match each invocation of a declared symbol with its associated meta information, if any, provided by the author in the symbol declaration.

[47]However, a declared math symbol may be invoked in a *newcommand* that takes arguments.

A related feature in the didactic GELLMU document type is the *mlg* tag for marking mathematical logical groups. This is somewhat akin to the *lgg* tag for TEX-like logical groups, traditionally created in TEX markup with braces that are not attached to a command.[48] As with *lgg* there is no obvious evidence of an *mlg* tag in a typeset rendering, but the presence of such a tag is intended as a signal to downstream mathematical parsers that the contents of the tag be given grouping priority as, say, with visible parentheses. Furthermore, the *mtype* and *mml* attributes of the *mlg* tag may be used to pass semantic information about the tag's contents to a processor.

7 The Didactic Production System

The didactic production system is the suite of processors and technical support files underlying what is called regular GELLMU.

7.1 Permission

The items of the didactic production system are copyrighted free software released under the GNU General Public License[49].

7.2 Materials

The release contains everything originating with the author that is currently used in "building" GELLMU documents.

It also contains a slightly modified version[50] of David Megginson's *Perl* module `"SGMLS.pm"` based on another slightly modified version that was furnished to me by Dave Holden in a very quick early 1999 response to my posted request for a modification that handles the labels provided (optionally) by *nsgmls* for SGML elements that are defined empty. A similar slight modification was also supplied a few days later by Vassilii Kachaturov and had been available at his web site.

Although the materials offered in this package aside from the syntactic translator pertain only to the didactic document type and the didactic production system, it should be understood that the larger design for GELLMU envisions other parties, on the one hand, building in various ways to extend the functionality of the didactic system, and, on the other hand, applying the methods of the didactic system to other document types and other formatting programs for those document types.

The basic items originating with the author, aside from the document type definition files (section B.2.2) are:

`gellmu.el` [51] the GELLMU syntactic translator, which makes SGML

[48]Such unattached braces in GELLMU markup lead to an *lg0* tag in the output of the syntactic translator that is translated to an *lgg* tag in the XML version of the didactic document type.
[49]URI: http://www.gnu.org/copyleft/
[50]URI: ../perllib/SGMLS.pm
[51]URI: ../gellmu.el

`xplaingart.pl` [52] converts SGML to author-level XML

`xmlgart.pl` [53] converts author-level XML to elaborated XML

`ltxgart.pl` [54] translates elaborated XML to LaTeX

`htmlgart.pl` [55] translates elaborated XML to classical HTML and translates specially prepared XML to XHTML+MathML

`mathcdata.pl` [56] first of two special preparations for translation toward XHTML+MathML

`mathprep.pl` [57] second of two special preparations for translation toward XHTML+MathML

`mval.pl` [58] check for certain types of MathML errors

Since some users will only be interested in the syntactic translator, additional description of these materials is found below in "Using the didactic production system" (section 7.5) and in the Release Notes (appendix B).

7.3 Other Required Software

To make use of the GELLMU syntactic translator a user must have or separately acquire *Emacs*[59].[60] ("Windows" users should look at the special FAQ[61].) *Emacs* is commonly found on GNU/Linux systems and on *ix systems. It may be found on other systems when provided by system managers.[62]

To make use of the didactic production system beyond the syntactic translator a user must have or acquire the following items of free cross-platform software

- an ESIS generating SGML parser such as found in the cross-platform package *SP*[63] by James Clark, which has stood the test of years, or the newer variant *OpenSP*[64], which is internationalized, from the *OpenJade* Team.
- *Perl* at CPAN[65].

[52]URI: ../xplaingart.pl
[53]URI: ../xmlgart.pl
[54]URI: ../ltxgart.pl
[55]URI: ../htmlgart.pl
[56]URI: ../mathcdata.pl
[57]URI: ../mathprep.pl
[58]URI: ../mval.pl
[59]URI: http://www.gnu.org/software/emacs/
[60]Version 20 or later should be adequate. Although the author began this project using version 19, he is no longer able to run tests with that version.
[61]URI: http://www.gnu.org/software/emacs/windows
[62]It is an embarrassment of the business world in the years since 1985 that many business computing installations do not provide general purpose cross-platform programming systems despite the widespread availability of excellent free robust systems such as *Emacs* (for Lisp), *gcc* (for C), and *Perl*. This new phenomenon apparently arises not so much from lack of organizational interest but from the fact that the responsibility for maintenance cannot be passed beyond the local system manager to a vendor.
[63]URI: http://www.jclark.com/sp/
[64]URI: http://openjade.sourceforge.net/
[65]URI: http://www.cpan.org/

- a complete TEX system, for which one may look to The TEX Users Group (TUG)[66] or The Comprehensive TEX Archive Network (CTAN[67]).
- *xmlwf* — a basic utility that is part of the release of James Clark's *expat*[68].

7.4 Using the syntactic translator

This explains how to use the syntactic translator, which is the Emacs Lisp program contained in the file `gellmu.el`[69].

7.4.1 Operation in Batch Mode

For linux and the other *ix a script like `bin/linux/g2s`[70] will be adequate if your working directory is the distribution directory[71].

Usage: `g2s` *stem-name* `[` *function-call* `]`

For example, if `"foo.glm"` is the name of the source file, then the first argument should be `"foo"`. The optional second argument *function-call* is the name of the function in the Emacs Lisp package `"gellmu.el"` that is to be used. The function call defaults to `"gellmu-trans"`, which is the correct name for LATEX-like usage under the didactic document type (section 5).

There are also parallel scripts `"g2h"` and `"g2x"`.

`"g2s"` will byte compile `"gellmu.el"` if `"gellmu.elc"` is not present.

`"g2h"` runs the function *gellmu-html* for the case where the GELLMU file has been written directly for HTML. The file `ghtml.glm` and the derived file `ghtml.html` are examples.

`"g2x"` runs the function *gellmu-xml* for the case where the GELLMU file has been written directly for XML.

The directory `"bin/win32"` contains parallel, though more complicated, batch files for use in a "DOS" command line under "Windows".

7.4.2 Interactive Operation

Open GNU *Emacs* interactively on the GELLMU source file. When finished editing the source, save it but keep *Emacs* open. Then do

```
M-x load-file gellmu.el
```

and

[66]URI: http://www.tug.org
[67]URI: http://www.ctan.org
[68]URI: http://expat.sourceforge.net/
[69]URI: ../gellmu.el
[70]URI: ../bin/linux/g2s
[71]None of the enclosed scripts either for linux or for win32 should be used without prior examination and verification for suitability.

```
M-x gellmu-trans  .
```

(It is better to have byte-compiled `"gellmu.el"` and if the byte-compiled version `"gellmu.elc"` is in your *Emacs load-path*, then

```
M-x load-library gellmu
```

is faster.)

The SGML output should come up in a second buffer. Save that buffer to save the output.

Make any corrections or changes in the GELLMU source buffer and re-run

```
M-x gellmu-trans  .
```

As with batch operation the functions *gellmu-html* and *gellmu-xml*, may be handled parallel to *gellmu-trans*.

There are a number of other functions besides these three for obtaining syntactic translation from GELLMU source to SGML. Each of these is, in fact, a front for calling *gellmu-trans* with various combinations of variable settings. There are a great many user configurable variables in the syntactic translator. Notable among these for *regular* GELLMU (section 4.1) are (1) *gellmu-parb-nogo* and (2) *gellmu-autoclose-list*. See the variable documentation text, available interactively when the GELLMU library is loaded in *Emacs* with the key combination qquostrC-h C-h v, for more information. For a list of the names of all user configurable variables see the variable documentation for *gellmu-public-vars*.

For example, setting *gellmu-verblist* true causes a sequence of lines beginning with the line `"\begin{verbatim}"` and ending with the line `"\end{verbatim}"` to be considered *verbatim* as in LaTeX, i.e., without requiring escaped forms of special characters, and then to be set as a simple *verblist*, which is in most circumstances superior to GELLMU's version of pre-historic *verbatim*.[72]

7.4.3 Interrupting the Syntactic Translator

Interruption of the GELLMU syntactic translator will be necessary in the event that the combined use of *newcommand*, *macro*, *Macro*, and *mathsym* (advanced mode only) leads to infinite recursive loops. Users should avoid the use of *macro* and *Macro* unless such use is absolutely necessary since these metacommands present greater looping risks.

Inasmuch as there are two ways to invoke the syntactic translator, there are two different procedures for interrupting it should that be necessary.

[72]The main reasons that this version is not the default with a call to *gellmu-trans* are:

1. It breaks the paradigm under which a GELLMU command name is the name of an SGML element.
2. It breaks backward compatibility with earlier versions of the syntactic translator, i.e., breaks older documents.
3. It is felt that the user invoking *verblist* this way should be aware of what is being done.

Note that direct invocation of *verblist* requires escaping special characters. Thus, using the function call *gellmu-verblist* converts the name *verbatim* from a command name to a meta-command name.

Batch mode invocation This is the case when GNU *Emacs* is launched in batch processing mode to run the syntactic translator. To interrupt the syntactic translator in this case one must interrupt the *Emacs* process. The author does not know of any case when *Emacs* does not respond to standard interrupts. For example, on *linux* systems pressing "Control-C" when the process was launched from a shell provides a standard interrupt. If the processed was launched in some other way, a normal `"kill"` addressed to the process should have the same effect.

Interactive invocation This is the case when the syntactic translator is launched from within the GNU *Emacs* editing interface. Use the standard *Emacs* function "quit", accessed with the key `"C-g"` (Control-G) to interrupt the syntactic translator.

7.5 Using the didactic production system

7.5.1 Staged Design

The items in the didactic production system are components for use with staged processing. The document type may be used with any SGML system. Of course, one may not use a parser that is limited to XML with the SGML version of the document type. Moreover, if one makes use of features in the SGML version such as the positional argument and option elements, then one might want to provide translation to the XML version of the document type.

No particular processing system is required for the XML version of the document type. For example, one might profitably write an XSLT[73] sheet for translation to some other format and then submit the document and the XSLT sheet to an XSLT engine such as *xt*, *xsltproc*, or *saxon*.

7.5.2 Default Staging

The release includes `bin/linux/lmkg`[74] and `bin/linux/mmkg`[75] as example driver scripts for running the following sequence. (The `bin/win32`[76] directory contains old driver scripts for the MS Windows command line that might someday be worth updating.) The behavior of these driver scripts depends on the way they are called though the specific of this are somewhat different for `lmkg` than for `mmkg`. The older `lmkg` scripts do not generate XHTML+MathML at this point they are provided primarily for backward compatibility.

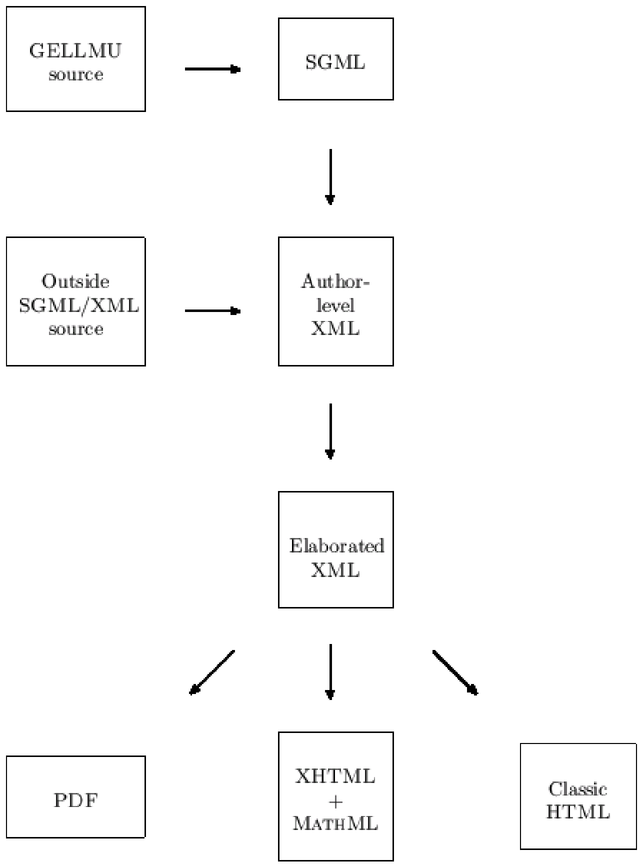
The `mmkg` scripts by default make XHTML+MathML but not if called by a name, e.g., via a symbolic link, without the substring `"mm"`. If an `mmkg` script is called with a name ending in the string `"froms"` or `"fromx"`, then it will take as starting point, respectively, an SGML, i.e., `".sgml"`, or author-level XML, i.e., `".xml"`, version of the document. Thus, for example, `mmkgfromx` might be used to operate on a document that is an author-level XML translation from a non-GELLMU source.

Caution. These scripts, like all shell scripts, should be examined for file system locations, system environmental variables, and other platform-specific and location-specific issues. The

[73]URI: http://www.w3.org/TR/xslt
[74]URI: ../bin/linux/lmkg
[75]URI: ../bin/linux/mmkg
[76]URI: ../bin/win32

user who introduces a script on a platform should understand the script. A user who does not understand a script should not attempt to introduce it on a local platform.

Flow in the didactic production system is portrayed in the following diagram:



These are the stages in the didactic production system pipeline:

1. Prepare GELLMU source using a text editor.
2. Process the source with the syntactic translator to obtain an SGML document under the didactic document type.
3. Use *nsgmls* to validate the SGML document and obtain an ESIS for it as output.
4. Submit the SGML ESIS as input[77] to the *Perl* program *sgmlspl* with the script `xplaingart.pl`[78] as file argument, obtaining an author-level XML document.
5. Use *nsgmls* to validate the author-level XML document and then submit its ESIS as input to to *sgmlspl* with the script `xmlgart.pl`[79], obtaining an elaborated XML document.

[77]Specifically, this mention of “input” refers to what is called “standard input” in a command line situation. There may be a challenge here on platforms that do not provide a command line.
[78]URI: ../xplaingart.pl
[79]URI: ../xmlgart.pl

This document, which is accompanied by several auxiliary files[80], has things such as sectional unit numbers and cross references fully resolved so that there will be consistency in these across the various output formats.

6. Use *nsgmls* to validate the elaborated XML document and submit its ESIS as input multiply to *sgmlspl*:

   (a) with the script `htmlgart.pl`[81] to obtain a classical HTML document that then will be validated if an HTML validation program is identified in the driver script.

   (b) with the script `ltxgart.pl`[82] as file argument, obtaining a LaTeX document. The LaTeX document is then built with *latex* to make a DVI file and with *pdflatex* to make a PDF file.

   (c) for a pipeline using successive runs of *sgmlspl* with 3 scripts, `mathcdata.pl`[83], `mathprep.pl`[84], and `htmlgart.pl`[85] (called in a special way) to make a XHTML+MathML file that is then checked for XML well-formedness using *xmlwf*, checked for certain kinds of MathML errors using *sgmlspl* with `mval.pl`[86], and validated if a suitable validation program is identified in the driver script.

7.5.3 Parsing with nsgmls

The program *nsgmls* is part of the SP[87] package, which includes extensive documentation. Those familiar with it will want to ignore these hints.

Since for both the SGML and the XML versions of the didactic document type SP requires non-default SGML declarations, it is recommended that the user employ SGML catalogs, one for SGML and another for XML. The file system location of a catalog is conveyed to *nsgmls* as the value of its command line argument immediately following the argument "-c".

Each catalog should contain an SGMLDECL directive that is the file system location of an SGML declaration. Aside from that a catalog may contain a number of three string lines of either of the following forms

PUBLIC *formal-public-identifier* *quoted-pathname*
SYSTEM *quoted-system-identifier* *quoted pathname*

where the quoted pathname, which may be relative to the location of the catalog, should for this context in each case be that of a DTD file.

It is recommended in each case that *nsgmls* be run with arguments "-l" (for propagating line number references) and "-oempty" (for flagging defined-empty elements). For processing the XML version of the didactic document type one should additionally use the argument "-wxml".

Additionally, a user may wish to make locally-specific arrangements for the handling of character sets.

[80]Formally, two of these auxiliary files are considered part of the elaborated XML document.
[81]URI: ../htmlgart.pl
[82]URI: ../ltxgart.pl
[83]URI: ../mathcdata.pl
[84]URI: ../mathprep.pl
[85]URI: ../htmlgart.pl
[86]URI: ../mval.pl
[87]URI: http://www.jclark.com/sp/

7.5.4 Processing with sgmlspl

The program *sgmlspl* is part of David Megginson's SGMLSPM package. Megginson's extensive documentation for it may be found in the (December 1995) release found at CPAN[88].[89]

One uses *sgmlspl* by calling the Perl program *sgmlspl* with an ESIS as input and a script as argument. Additional arguments become arguments for the script.

Although some operating systems provide a way for dealing with a Perl program, which is stored in a text file, as an executable object, in other cases one must explicitly call the Perl engine as a program with an ESIS as input, the system name of *sgmlspl* as first argument, and (the system name of) a script as second argument. In both cases one will want to arrange, perhaps with an environmental variable or perhaps with the "-I" argument to the Perl engine, for the directory containing `"SGMLS.pm"` and its supporting module `"SGMLS/Output.pm"` to be in its path array `@INC`.

7.5.5 Environmental Variables

There are a number of environmental variables that affect processing in the didactic production system. The names all begin with the string `"GELLMU_"`. Of course, the names are case-sensitive.

Many of these variables are set in the distributed driver scripts. When that is the case, the distributed driver scripts commonly check for a previous setting (which may, therefore, be easily placed in a fronting script that makes a setting and then just calls the distributed driver).

Setting environmentals can be difficult in a non-Unix-like operating system environment. This is one reason why the author generally recommends that Windows users install GELLMU under *Cygwin*[90].

`GELLMU_Dir`

The top of the directory tree where GELLMU is installed.

`GELLMU_StyleDir`

URI or directory location of CSS and XSLT style sheets that is used by the didactic production system in writing links in XML, HTML and XHTML+MathML files. The value usually has a different meaning under the eye of a web server than in a local file system. A relative URI or path is usually best. A value like `"../webstyle"` can often be made to work both ways.

`GELLMU_CSSName`

Name, relative if given relative syntax, to the value of `GELLMU_StyleDir`, of the CSS stylesheet written by the didactic production system in HTML and XHTML+MathML files.

[88]URI: http://www.cpan.org/

[89]At the time of this release there was discussion in the UseNet newsgroup `news:comp.text.sgml` about a proposed revision of SGMLSPM by another party. The code for `"SGMLS.pm"` included in this GELLMU release contains a very small modification of Megginson's 1995 release.

[90]URI: http://www.cygwin.com/

`GELLMU_XhtmlSuffix`

Suffix given to XHTML or XHTML+MathML files written by `htmlgart.pl`.

`GELLMU_NoUMSS`

Value 0 or 1: if 1, signals to `htmlgart.pl` that it should not link to W3C's UMSS[91] XSLT stylesheets.

`GELLMU_UTF8`

Value 0 or 1: signals to *sgmlspl* scripts that Perl's handling of the UTF-8 text encoding should be invoked. The meaning is subtly different between Perl versions 5.6 and 5.8.

`GELLMU_Encoding`

String value for text encoding that is set by the `xplaingart.pl` in writing author-level XML and by `xmlgart.pl` in writing elaborated XML. (HTML, XHTML, and XHTML+MathML are always written with the UTF-8 encoding.)

`GELLMU_LaTeXUTF8`

Value 0 or 1: signals to *latex* and *pdflatex* to expect the UTF-8 encoded text in their input.

`GELLMU_LaTeXStyle`

Pathname for LaTeX stylesheets that *latex* and *pdflatex* should use when such stylesheets are not properly positioned for TeX system KPSE-based location. (It's better to use a local or personal TDS tree.)

`GELLMU_PAPER`

String value for the paper used in printing; becomes an option for the *documentclass* command in the output LaTeX file.

`GELLMU_Memoir`

Value 0 or 1: if 1, use the *memoir*, rather than *article*, documentclass in the output LaTeX file.

`GELLMU_DefaultEmptyEqncenter`

Experimental. String value consisting of a small bit of LaTeX wrapped as a Perl string to use in tweeking the LaTeX-rendered appearance of a GELLMU *eqnarray* (which is rendered in LaTeX using either *align* or *aligned*, depending on numbering arrangments) in the case of an empty middle cell (*eqncenter*). The current default value used in `ltxgart.pl` is the string `" \qquad "`. Be mindful of how such a string can be entered as a literal string in Perl or as part of an on-the-fly environmental setting from a command line shell.

`GELLMU_XLink`

Value 0 or 1. How to handle links in MathML output when writing XHTML+MathML. Such links, which are currently illegal inside XHTML+MathML math zones, are confined to `\text{...}` areas in GELLMU. If the value is 1, use XLink; otherwise, switch into the HTML namespace and write an HTML anchor. (*Firefox* handles both, while more of the other browsers seem to choke on the namespace switch than choke on the necessarily cumbersome use of XLink.)

[91]URI: http://www.w3.org/Math/XSL/

`GELLMU_OriginLabel`

Name for an automatic label key, chiefly of occasional value for HTML and XHTML+MathML outputs, that, when this variable is present in the environment, places a link target, with id the value of this variable, at the top of the document.

Appendix A The GELLMU Archive

The GELLMU Archive is the web site `http://www.albany.edu/~hammond/gellmu/`.

It is the source for late breaking information about GELLMU. Among other things, it houses a largely uncommented archive of examples[92]. This is provided in the belief that the study of examples is one of the quickest ways to learn a markup language.

Of course, this document, which is furnished with the release, is also an example.

Another item, also an example, that is housed in the archive is The GELLMU FAQ[93].

Appendix B Release Notes

This version of the manual was prepared for release 0.8.5 in July 2007. The GELLMU materials (section 7.2) consist of:

1. The manual, which is this document.
2. The GELLMU syntactic translator, a *Emacs* Lisp program, which is the only item of software required for those who simply wish to use GELLMU markup for the conscious preparation of HTML documents or documents under some other classical SGML or XML document type for which the user is otherwise equipped.
3. The GELLMU didactic production system, which consists of an SGML document type called *article*, an XML document type also called *article*, and three separate collections of *Perl* functions for the well-known *Perl* SGML processing framework *sgmlspl* by David Megginson[94]. A very slightly modified version of Megginson's *Perl* library *SGMLSpm* that provides a method for detecting defined-empty SGML elements, as flagged in an SGML parse stream in ESIS format, is included as part of the didactic production system. Since it is by size 60% of the software content of the Megginson package on CPAN[95], the rest of the package, licensed under the GNU General Public License[96] is distributed with the didactic production system as well, though without its documentation. The distribution includes 7 scripts for use with *sgmlspl* in the didactic production system pipeline. For more on this see "Using the didactic production system" (section 7.5).

[92]URI: http://www.albany.edu/˜hammond/gellmu/examples
[93]URI: http://www.albany.edu/˜hammond/gellmu/examples/gfaq.html
[94]URI: http://www.megginson.com/
[95]URI: http://www.cpan.org/
[96]URI: http://www.gnu.org/copyleft/gpl.html

B.1 Comments on the Syntactic Translator

The GELLMU syntactic translator is more mature than the other components. The following comments pertain to it.

internationalization

Internationalization has a considerable and evolving level of support in *Emacs*. The concept is that an author resides in a locale. When the author enters a character from a locale, it gives rise in *Emacs* to a somewhat complicated multibyte entity that can have "properties". Particularly relevant variables in *Emacs* are: `coding-system-for-write` and `buffer-file-coding-system`. GELLMU provides the user variable `gellmu-sgml-default-coding`, which should be properly coordinated via driver script settings with one's SGML parser.

inclusions

It is not actually a limitation that a GELLMU source file cannot be included in another. The primary reason is that one should make use of the inclusion mechanism of SGML. For that one needs to define the included pieces as entities in the direct internal subset [97] of the source file and then reference each as an entity, e.g., `"&sect2;"` at the appropriate location in the source file where it is to be included. Because the inclusion happens at the SGML level there are two points to observe:

1. Macro information is local to each source file.
2. The situation is optimal for the location of validation errors provided that one's parser reports such errors by filename and line number since the syntactic translator provides line number alignment between source and generated SGML.

A second reason is that source inclusion would disturb line number alignment between source and SGML output. This is important for the interpretation of SGML validation error messages. Such validation is considered routine, and plays an important role in detecting an author's mistakes. Some author errors are diagnosed in the syntactic translator.

A third reason, which at the same time might be considered also a disadvantage, is that all of GELLMU's macro facilities are local to each source file. This adds both robustness and flexibility at the price of the inconvenience of physical inclusion of common macro definitions.

variable management

This refers to the management of user variables in the syntactic translator. These are *Elisp* variables. One who is familiar with *Elisp* should be able to provide values in batch

[97] The direct internal subset is the content of the optional argument of the *documenttype* metacommand that follows its required argument. It should be noted in the didactic production system that the direct internal subset cannot be propagated to the XML form of *article* because it is digested by any standard SGML parser and, hence, by any translation based on a standard parsing. Thus, any pieces are merged in the XML form of an article although the translator `xmlgart` might be modified to construct an internal declaration subset there and provide partitioning of the XML version among filesystem pieces based on document structure.

mode without making changes in the *Elisp* source.[98] Setting values interactively in the Emacs editing interface can be done easily using `"M-x set-variable"`.

With a future release it is likely that additionally a user resource file for custom variable settings without the need for writing *Elisp* code will be provided.

Bugs

No serious existing problem is known in the GELLMU syntactic translator at the time of this release. Of course, as stated in source code comments, there is no warranty of any kind. Please report bugs to the author: `hammond@math.albany.edu`.

- **Reserved element names.** The GELLMU syntactic translator reserves for its own internal use all SGML or XML element names in which the first numeric character in the range "0–9" is the character "0".
- **Limitation on braces in macros.** Unbalanced braces are not permitted in either the name or the value field of any form of macro metacommand.
- **First cell limitation.** In the LaTeX-like emulation of an *array* or *tabular* environment the first cell in each row must have something other than whitespace. Of course, sometimes no content is wanted, and then `\empty` (for nil markup, not to be confused with the mathematical `\emptyset`) is one way to handle it, but this author usually uses something that is mostly inconsequential like "`~`" or "`\,`". Another way to handle it is to invoke the names of the SGML elements, i.e., `\firstcell{}` for *tabular* or `\firstacell{}` for *array*.
- **Concept of advanced GELLMU immature.** Inasmuch as didactic article is the only document type for which the idea of advanced GELLMU has been implemented, the general concept of advanced GELLMU is not fully developed in the GELLMU syntactic translator. Basic GELLMU is characterized in the GELLMU syntactic translator by the evaluation of the Boolean variable *gellmu-straight-sgml* to "true". This automatically make the Boolean variable *gellmu-regular-sgml* true. Full LaTeX-like support for the didactic production system is realized only by both of these Booleans being false. Thus, advanced GELLMU will need to evolve in the space in between, probably after the introduction of further such Boolean variables, some public and some private. This technique will make it possible for the code to continue performing as now when the variable *gellmu-straight-sgml*, the flag for basic GELLMU, is true and also when both of these flags are false.
- **Reserved strings.** The strings "`<<`" and "`>>`" have been reserved as future notation for mathematical objects. Although it might seem at first glance that this type of short hand has no place apart from the fully LaTeX-like environment of the GELLMU syntactic translator in the context of the didactic production system, in which they have not yet been used, it is actually not so clear that one could not make sensible use of such notation in the context of "XHTML plus MathML" under advanced GELLMU along with other features such as blank lines for new paragraphs and many other mathematical shortcuts. It awaits the further development of advanced GELLMU, and reserving this notation is necessary to ensure backward compatibility.

[98] Please observe the rules of the LaTeX project regarding filenames as well as the license rules of the GNU General Public License if you wish to distribute a modified version of the *Elisp* source. Alternatively, the author is always interested in learning of suggestions for change.

Consequently, for example, entering "`<<a>>`" is problematical, because only the first "`<`" or "`>`" will be converted to something appropriate. In basic mode "`&lt;`" and "`&gt;`" are one-step ways of circumventing this when these entities are available, which is guaranteed for any form of HTML as well as for any form of XML. In the didactic production system one should use "`\ltc;`" and "`\gtc;`". For other cases the one-step circumvention is to use entity references to the numeric character codes, e.g., in ASCII "`&#3C;`" and "`&#3E;`", and for convenience these may be brought up as macros, perhaps "`\lt`" and "`\gt`".

B.2 Comments on the Didactic Production System

The didactic production system is to be understood as a potential base for development. As such it is not intended ever to offer everything that might be imagined. The following comments pertain to it.

B.2.1 Internationalization

Internationalization has been a concern of the project. It is possible, for example, to use the ISO-Latin-1 character 'é' in the name of an element. The didactic *article* document type offers, for example, an element *étale*, which is a style, parallel to *bold*. Use of the character 'é' as a raw word character data with the didactic production system is less robust than the more careful `"\acute{e}"`[99] construction, which is desirable for translation of *article* to formats that do not support latin1. For that matter, the exact extent of LaTeX's support of latin1 is a bit tricky, and the whole matter of internationalization is currently under review in the LaTeX project.[100]

B.2.2 Document Type Definitions

Currently the project has one SGML document type definition and two XML document type definitions. Files under the various document types are suffixed as follows:

First stage SGML	`.sgml`
Author Level XML	`.xml`
Elaborated XML	`.exml`

Additionally, in the three steps of processing to generate an XHTML+MathML file from an elaborated XML file there are two intermediate XML files generated, the first with suffix `".yml"`, which lives under the document type definition for an elaborated XML document, and the second with suffix `".zml"`, an XML file for which there is no extant formal document type definition.[101]

[99]The corresponding usage in LaTeX would be `"\'e"`; this could be brought into GELLMU source using \macro, but it must be resolved to a name in the output of the GELLMU syntactic translator where everything that is markup needs a name. Rather than using a general container *acute*, the document type could have provided a name for the specific character.

[100]Alternatively LaTeX source can be submitted to the program *lambda* which is the LaTeX format for the program *omega* that is now under development as a substitute for Knuth's TeX, the program, with internationalization as a stated goal.

[101]There is no formal document type definition for a `".zml"` file because such a file is endowed via XML attributes with information about tree structure for mathematical zones.

The author-level XML document type is formalized by the DTD ”axgellmu.dtd”, while the elaborated XML document type is formalized by a modification that is found in the DTD ”uxgellmu.dtd”. (The latter was the only XML document type used with the regular GELLMU production stream prior to October, 2006.)

The document type represented by ”uxgellmu.dtd” is now called the elaborated XML document type.

The author-level XML document type is suitable as a translation target from other markups. The elaborated XML document type should not be used as a translation target other than from the GELLMU author-level XML document type.

All document type definitions are available under the UTF-8 text encoding. The two older document type definitions will continue to exist for a while under the Latin-1 (ISO-8859-1) text encoding. The text encoding of a so-called DTD file (not quite the same thing as a document type definition) is significant in regard to the names of SGML/XML entities and elements rather than in regard to document instances which might be processed. The names of the DTD files are:

	Latin-1	UTF-8
First stage SGML	`gellmu.dtd`	`ugellmu.dtd`
Author Level XML		`axgellmu.dtd`
Elaborated XML	`xgellmu.dtd`	`uxgellmu.dtd`

B.2.3 Translation to XML

Presently the author-level XML files link to a CSS stylesheet that provides primitive rendering. One could go further in this direction, but the rendering of mathematics will be limited without more development in that direction of CSS.

B.2.4 Translation to HTML

Math in classic HTML The classic HTML output does not use graphic images for mathematical zones in the manner of programs like *latex2html*. Instead it uses pseudo-TEX notation for math. There are a number of reasons:

1. Well typeset mathematics is available in the modern form of HTML that is more precisely called XHTML+MathML.
2. Graphical images completely block accessibility in the sense of the World Wide Web Consortium’s Web Accesibility Initiative[102].
3. The present classical HTML output files may be deciphered in terminal window browsers.
4. The present classical HTML output files may be “dumped” to plain text using a program such as *lynx* or *w3m* for various sometimes useful purposes.

Style. HTML and XHTML+MathML made with the didactic production system now rely on CSS, even, for some things, level 2 CSS.

[102]URI: http://www.w3.org/WAI/

B.2.5 Translation to LaTeX

This translator writes LaTeX2E. A number of packages, including *graphicx*, *amsmath*, *amssymb*, *amsfonts*, *bm*, *url* (not *hyperref* for the standard track where the focus is on printed output), and *inputenc* for UTF-8 (which may be needed even if the GELLMU source or, otherwise, the author-level XML source is not UTF-8 encoded). Apart from current font availability issues, the author would have preferred to invoke the T1 font encoding.

Even though GELLMU source uses the names *equation* and *eqnarray*, in the LaTeX formatting *amsmath* constructions are used.

A small modification of this translator can be used to write Adobe's Portable Document Format (PDF) with pages sized for screens rather than for paper.

B.2.6 Future Plans

This is a very limited list.

A literate document type definition.

> Capable of spawning not only the 5 DTD's but type definitions under other mechanisms such as, for example, *RelaxNG*.

Mathematical Semantics

> Provision for optional semantic tightening sufficient for authors wishing to be able to export mathematical markup into computer algebra systems.