The `qwertz` SGML Document Types

(Version 1.2 Reference Manual)

Tom Gordon

The `qwertz` Project
Institute for Applied Information Technology (F3)

German National Research Center
for Computer Science (GMD)

October 29, 1992

Contents

List of Tables

Chapter 1

Why Not Just Use LaTeX?

The `qwertz` document types are a set of Standard Generalized Markup Language (SGML) document type definitions (DTDs) for articles, reports, books, letters, notes, slides (or overhead transparencies), bibliographies, and manual pages. Except for manual pages, the document types have been heavily influenced by the LaTeX document types of the same names [4], so LaTeX users should feel right at home. Indeed, there is a translator, called `format`[1], which converts `qwertz` documents into LaTeX and other formats. Anyone familiar with LaTeXshould have no difficulty understanding the LaTeX versions of documents generated by `format`; with minor exceptions, only standard LaTeX commands and environments are used.

So why not just use LaTeX directly, you might ask? Although LaTeXis indeed somewhat of a *de facto* standard in some scientific circles, it is tied to a particular formatting system, namely TeX. SGML is an ISO standard language for defining the logical structure of documents. There are a number of tools available for editing, parsing and translating SGML documents. SGML structure editors, for example, take advantage of the logical structure of the document being edited to offer functions similar to so-called *outline processors*, and often use on-screen formatting, similar to WYSIWYG text editors, to eliminate the necessity of typing and viewing the "tags" marking the various regions of text (such as paragraphs, footnotes, and citations). Thus, users need not give up the benefits of an intuitive, easy-to-use editor to acquire the powerful formatting capabilities of systems such as TeX or troff. On the other hand, SGML documents can also be written using any text editor, such as emacs, which many authors may prefer.

Also, the use of SGML provides an amount of independence from particular formatters. SGML parsers and translators provide the tools necessary to ease the job of translating documents marked-up in SGML into any formatting language. It is a relatively painless job to translate an SGML document into TeX, MS-Word, or troff, for example, but can be quite difficult to translate documents formatted using one of these systems into one of the others.[2]

Finally, SGML files can be used for purposes other than printing documents. In `qwertz`, for example, we use SGML for *literate programming* [3], extracting and piecing together the source code of our programs directly from their documentation. But there are other uses. For example, SGML has been used in implementations of hypertext and hypermedia.

[1] `format` is described in the "Unix Commands" chapter of this manual.

[2] In fact, `format` now is able to generate nroff/troff versions of `qwertz` documents, in addition to LaTeX.

Chapter 2

A Short Tour

Suppose you would like to write a letter to a colleague. Let's go through the steps of writing, previewing and then printing the letter. You may want to actually type along at your computer. Don't worry at this point if you don't quite understand the details of SGML syntax.

There are just two prerequisites to understanding this tour: you need to know how to use a text editor such as `textedit`, `emacs` or `vi`, and you need some basic knowledge about working with a Unix command shell, such as `ksh` or `csh`. [1]

Now, just start your favorite editor and type after me:

```
<!doctype qwertz system>
<letter>

<from> Dr. Alexander Hertzberg
<address>
Institute for Insanely Great Software &nl
Research Center for Stuff Not Yet Discovered &nl
Former Capital City, Germany

<to> Prof. Wanda Wonderful
<address>
Dept. of Eccentrics &nl
Universal University &nl
Sandy Ego, California, U.S.A.

<opening> Dear Wanda,

Have you heard about that insanely great SGML
to LaTeX translator developed by those guys at the
GMD?  They've done it again!  It does everything
and more.  I'll never go back to using a (phooey!)
WYSIWYG editor like <tt/brand name deleted/
again.

<closing> Yours truly,

</letter>
```

[1] It is also assumed that your `PATH` environment variable has been set so as to include the directory containing the programs described in Chapter 5.

To preview the document, there are several possibilities, depending on what you have available. First of all, save your file as `phooey.sgml` and shift the focus of your attention to some Unix shell. Then, if you have a PostScript previewer, such as `ghostview`[2], try:

```
format phooey | qtex > phooey.ps; ghostview phooey.ps
```

If you are using Open Look, you may have the `pageview` PostScript previewer:

```
format phooey | qtex | pageview
```

In either case, if you (and I) haven't made any typing errors, your letter should soon appear on your screen.

If you do not not have a PostScript previewer, you may also be able to preview the document using a `dvi` previewer, such as `xdvi`:

```
format phooey | qtex -d > phooey.dvi;  xdvi phooey
```

However, `xdvi` is not effective if PostScript fonts have been used, as is the case at our site, where letters are formatted so as to appear to have been printed on our institute's letterhead stationery.

To view the letter on an ordinary ASCII terminal, try this:

```
format -T nroff | nroff -ms | more
```

Now, to print the letter, formatted using LaTeX, just type:

```
format phooey | qtex | lpr
```

Another common type of document is a journal article. Appendix A contains the SGML source of a short "article". It is only a few pages long, but contains examples of most of the elements necessary, including sections and subsections, headers, footnotes, various kinds of emphasis, lists, mathematical formulas, tables, figures and a bibliography.

Perhaps this short tutorial, together with the extended example in Appendix A, are sufficient for using this reference manual to write `qwertz` documents. But you well may want to read one or more of [1, ?, ?], each of which contains an introduction to SGML.

[2] Ghostview is part of the Gnu system from the Free Software Foundation.

Chapter 3

The qwertz Document Type Definition

All of the `qwertz` document "styles", except bibliographies, are defined in a single SGML document type definition (DTD), called `qwertz`. It is essentially a SGML reconstruction of Lamport's LaTeX [4]. We have not attempted to include every feature of LaTeX in this DTD, but have included the features we use regularly. Others may of course find that something they deem important is missing. We welcome suggestions for improvements or extensions.

We will be making use of several *parameter entities* in this DTD:

```
<!entity % emph
        " em | it | bf | sf | sl | tt " >

<!entity % xref
        " label | ref | pageref | cite | ncite " >

<!entity % inline
        " (#pcdata | f | x | %emph; | sq | %xref)* " >

<!entity % list
        " list | itemize | enum | descrip " >

<!entity % par
        "  %list; | comment | lq " >

<!entity % mathpar " dm | eq " >

<!entity % thrm
        " def | prop | lemma | coroll | proof | theorem " >

<!entity % litprog " code | verb " >

<!entity % sectpar
        " %par; | figure | tabular | table | %mathpar; |
          %thrm; | %litprog; ">
```

These are just macros used in the definitions of various elements, to avoid retyping and to ease maintenance. The `emph` parameter lists the various kinds of emphasis. The `inline` parameter is for the elements which may be used anywhere within the document. The `list` parameter is for various kinds of lists. `par` lists several basic kinds of elements at the level of paragraphs. The `mathpar` parameter includes the elements for *displayed* mathematical formulas. The `thrm` parameter is for the set of elements used to represent such things as definitions, theorems and proofs. The `litprog` parameter is for literate programming elements. Finally, the `sectpar` parameter lists the elements which may occur at the level of paragraphs within sections (or chapters). Notice that this parameter uses other parameters.

Several kinds of documents may be written using LaTeX: articles, reports, books, letters and slide (or transparency) presentations. The `qwertz` DTD includes two others as well: `notes`, for documents such as notes to yourself which do not require a title, sections, footnotes and the like; and `manpage`, for Unix manual pages.

```
<!element qwertz o o
        (sect | chapt | article | report |
         book | letter | telefax | slides | notes | manpage ) >
```

Notice that sections (`sect`) and chapters (`chapt`) may also be processed separately, before being put together into an article, report or book.

LaTeX also includes BibTeX, a program for creating bibliographies whose entries can be easily cited in LaTeX documents. The `qwertz` document type for this purpose is described in Chapter 5.

3.1 General Purpose Entities and Elements

This section describes the SGML entities and elements available in all `qwertz` documents.

```
<!entity % general system -- general purpose characters -- >
%general;
```

3.1.1 Characters Entities

Most characters are created just by typing the character wanted on the keyboard. This simple method does not suffice when the character wanted isn't in the character set available, or at least not associated with a key on the keyboard, or when the character currently has special meaning to SGML or, perhaps, TeX. In this section, a fairly large number of general purpose character entities will be presented. Symbols and characters which may be used only in mathematical formulas will be discussed separately, in section 3.1.9.

When may it be necessary to use of an entity reference to produce some character? There are three cases to watch out for:

SGML Concrete Syntax Delimiters.

Although the SGML standard allows alternative concrete syntaxes to be defined, we use the so-called *reference concrete syntax* in the `qwertz` document types. In this reference syntax, `<` is the *start tag open* character, and `</` is the *end tag open* delimiter. The other SGML delimiter authors should be aware of is &, the *entity reference open* delimiter of the reference syntax.

The appropriate entity to use to generate these characters depends on the context. Normally, use `lt` to represent `<` and `amp` to get &, when these appear in strings which might otherwise be interpreted as starting tags or entity references. However, within the `code` or `verb` elements for literate programming, described in section 3.1.8, use the `ero` entity to represent & and the `etago` entity for the sequence `</`.

```
<!entity   lt    sdata "{\lt}" >
<!entity   amp   sdata "\&" >
<!entity   ero   sdata "&" >
<!entity   etago sdata "</" >
```

SGML Short Reference Delimiters.

In SGML document types *short reference maps* may be defined which allow single characters to be interpreted as arbitrarily complex sequences of characters, including SGML tags and entity references. Thus, to know precisely when a certain character will be interpreted literally or as a short reference (i.e. macro) for something else, one has to know which map is in effect in the context of the current element. Just about all punctuation characters which are not used as delimiters in the concrete syntax can be used as short reference delimiters:

```
" # % ' ( ) * + , - : ; = @ [ ] ^ _ { | } ~
```

For each of these characters, there is an SGML entity which may be used to generate the ASCII character in the printed document, listed in table 3.1. *Usually, it will not be necessary to use these entities; the character can simply be typed and will be interpreted literally.* However, if the results are not as expected, check to see if there is a map in effect at that point in the document in which the character has been redefined. As maps are associated with elements, the section in this manual describing an element will also direct you to a description of the applicable map, if there is one.

As it turns out, one important use of character maps is to generate exactly the character typed in the printed document. That is, the map is used to hide the special meaning of the character to the underlying formatter (e.g. TEX), replacing the character with the formatting instructions for generating the character. This has been the main use of maps in our `qwertz` document type definitions.

```
<!entity   dquot sdata "{\tt "}" >
<!entity   num  sdata "\#"    >
<!entity   percnt sdata "\%" >
<!entity   quot sdata "{\tt '}"  >
<!entity   lpar sdata "("   >
<!entity   rpar sdata ")"   >
<!entity   ast  sdata "\mch{\ast}"    >
<!entity   plus sdata "+"   >
<!entity   comma sdata "," >
<!entity   hyphen sdata "-"  >
<!entity   colon sdata ":" >
<!entity   semi  sdata ";"  >
<!entity   equals sdata "=" >
<!entity   commat sdata "@" >
<!entity   lsqb  sdata "["  >
<!entity   rsqb  sdata "]"  >
<!entity   circ sdata "\verb+^+"   >
<!entity   lowbar sdata "\_" >
<!entity   lcub  sdata "\{"     >
<!entity   verbar sdata "{\verbar}" >
<!entity   rcub  sdata "\}"    >
<!entity   tilde sdata "\verb+~+"   >
```

TEX Special Characters.

Ideally, it should be possible to hide the conventions of the underlying formatting system completely. In fact, SGML parsers which implement the full ISO standard have a feature which makes this possible. However, the SGML parser we are using does not include this feature: the only characters which can serve as short references are the characters allowed for this purpose by the reference concrete syntax. Unfortunately, this reference syntax does not allow &, $ and \ to be used as short references, which are all special TEX characters. Thus, the entities for these three characters (`amp, dollar` and `bsol`) must usually be used to produce them. (The $ and \ characters may be used directly within the `verb` and `code` elements, discussed below in section 3.1.8. Also, within these elements use the `ero` entity to represent & in strings which might otherwise be interpreted as entity references.)

```
<!entity   bsol sdata "\verb+\+" >
<!entity   dollar sdata "\$"     >
```

3.1.2 Spacing, Dashes and Ellipsis

The meaning of the ordinary space character is context sensitive. Sometimes there is a space *within* a single word. Such spaces can be typed using the *nonbreakable space* (`nbsp`) entity to avoid breaking the word at that point at the end of line. There are also contexts where one wants a certain amount of space to appear, without it being regarded by the formatter as being space which may be shrunk in order to clean-up the arrangement of words or characters on the line. There are three entities for this purpose: `emsp` denotes the amount of horizontal space required for the character "M". An `ensp` is just half as wide as an `emsp`, and a *thin space* (`thinsp`) is 1/6 of an `emsp`. Notice that these are relative amounts, depending on the font being used.

There are also three different kinds of dashes: `hyphen`, which was already mentioned above, is to be used for intra-word dashes, as in the word "intra-word".[1] `ndash` is to be used for number ranges, such as "23–56", and `mdash` is an alternative delimiter for parenthetical comments — certainly you've seen them used this way — perhaps to avoid too frequent use of commas or parentheses.

```
<!entity   nbsp sdata  "~" >
<!entity   emsp sdata "\hspace{1em}" >
<!entity   ensp sdata "\hspace{.5em}" >
<!entity   thinsp sdata "\hspace{.167em}" >
<!entity   mdash sdata "{---}" >
<!entity   ndash sdata "{--}" >
<!entity   hellip sdata "{\ldots}" >
```

3.1.3 Foreign Languages

There are a large set of entities for other Western European languages. Altogether, there are entities for almost all of the foreign language characters in ISO 8859, the Latin 1 character set for Western European languages.[2] Conveniently, these entities are all available in the usual Adobe PostScript fonts, as well as in TEX. Thus, all of the entities defined here can be printed in TEX, on PostScript printers, or displayed on any Latin 1 device. Depending on the computer and editor, it may also be possible to type these Latin 1 characters directly, instead of having to use these entities. A simple filter could translate Latin 1 files into ASCII files, replacing non-ASCII characters by entity references. The entity names chosen here for these characters conform to the SGML standard.

[1] However, the `hyphen` entity was not actually necessary here, as the - character was not being used in this context as a short reference.

[2] Only the four Icelandic characters are missing.

```
<!entity aacute  sdata '\'{a}' >
<!entity Aacute  sdata '\'{A}' >
<!entity acirc   sdata '\^{a}'  >
<!entity Acirc   sdata '\^{A}'  >
<!entity agrave  sdata '\`{a}' >
<!entity Agrave  sdata '\`{A}' >
<!entity aring   sdata '{\aa}'  >
<!entity atilde  sdata '\~{a}' >
<!entity Atilde  sdata '\~{A}' >
<!entity auml    sdata '\"{a}'   >
<!entity Auml    sdata '\"{A}'   >
<!entity aelig   sdata '{\ae}'  >
<!entity AElig   sdata '{\AE}'  >
<!entity ccedil  sdata '\c{c}' >
<!entity Ccedil  sdata '\c{C}' >
<!entity eacute  sdata '\'{e}' >
<!entity Eacute  sdata '\'{E}' >
<!entity ecirc   sdata '\^{e}'  >
<!entity egrave  sdata '\`{e}' >
<!entity Egrave  sdata '\`{E}' >
<!entity euml    sdata '\"{e}'   >
<!entity Euml    sdata '\"{E}'   >
<!entity iacute  sdata '\'{\i}' >
<!entity Iacute  sdata '\'{I}' >
<!entity icirc   sdata '\^{\i}'  >
<!entity Icirc   sdata '\^{I}'  >
<!entity igrave  sdata '\`{\i}' >
<!entity Igrave  sdata '\`{I}' >
<!entity iuml    sdata '\"{\i}'   >
<!entity Iuml    sdata '\"{I}'   >
<!entity ntilde  sdata '\~{n}' >
<!entity Ntilde  sdata '\~{N}' >
<!entity oacute  sdata '\'{o}' >
<!entity Oacute  sdata '\'{O}' >
<!entity ocirc   sdata '\^{o}'  >
<!entity Ocirc   sdata '\^{O}'  >
<!entity ograve  sdata '\`{o}' >
<!entity Ograve  sdata '\`{O}' >
<!entity oslash  sdata '{\o}' >
<!entity Oslash  sdata '{\O}' >
<!entity otilde  sdata '\^{o}' >
<!entity ouml    sdata '\"{o}'   >
<!entity Ouml    sdata '\"{O}'   >
<!entity szlig   sdata '{\ss}'  >
<!entity uacute  sdata '\'{u}' >
<!entity Uacute  sdata '\'{U}' >
<!entity ucirc   sdata '\^{u}'  >
<!entity ugrave  sdata '\`{u}' >
<!entity Ugrave  sdata '\`{U}' >
<!entity uuml    sdata '\"{u}'   >
<!entity Uuml    sdata '\"{U}'   >
<!entity yacute  sdata '\'{y}' >
```

```
<!entity Yacute  sdata '\'{Y}' >
<!entity yuml    sdata '\"{y}'   >
```

The qwertz document types were developed in a German research center, so we have included entities for the German characters with shorter names than the entity names used in the SGML standard. Notice that these are just synonyms for the standard entities, which are also included.

```
<!entity   Ae  '&Auml;' >
<!entity   ae  '&auml;' >
<!entity   Oe  '&Ouml;' >
<!entity   oe  '&ouml;' >
<!entity   Ue  '&Uuml;' >
<!entity   ue  '&uuml;' >
<!entity   sz  '&szlig;' >
```

3.1.4 Other Symbols

Finally, there are entities for a few miscellaneous symbols, such as §, ¶, ©, ¬, ÷, ±, ×, and μ. All of these entities name symbols in the Latin 1 character set. They may be used anywhere within a document. (In particular, the mathematical symbols shown here need not be within one of the formula elements described below, in section 3.1.9.) The entity names for these, and all the other character entities discussed above, are listed in table 3.1. *A document which does not include mathematical formulas or graphics and which uses only the character entities defined in this chapter can be displayed or printed using a single Latin 1 font.*

```
<!entity   gt     sdata "{\gt}" >
<!entity   sect   sdata "{\S}">
<!entity   para   sdata "{\P}">
<!entity   copy   sdata "{\copyright}">
<!entity   iexcl  sdata "{!`}" >
<!entity   iquest sdata "{?`}" >
<!entity   cent   sdata "\hbox{\rm\rlap/c}" >
<!entity   pound  sdata "{\pounds}" >
<!entity   not    sdata "\mch{\neg}" >
<!entity   divide sdata "\mch{\div}" >
<!entity   plusmn sdata "\mch{\pm}" >
<!entity   times  sdata "\mch{\times}"  >
<!entity   mu     sdata "\mch{\mu}" >
```

3.1.5 Sentences, Paragraphs, Emphasis and Quotations

Sentences need not be marked up with tags. There is no sentence element as such. Rather, these are marked implicitly using the usual conventions for beginning and ending sentences.

Paragraphs are delimited with the p tag. Both the starting tag and ending tag are optional.

```
<!element p o o ( %inline | %sectpar )+ >
<!entity ptag '<p>' >
<!entity psplit '</p><p>' >

<!shortref pmap
        "&#RS;B" null
        "&#RS;B&#RE;" psplit
```

AElig	Æ	Aacute	Á	Acirc	Â	Ae	Ä
Agrave	À	Atilde	Ã	Auml	Ä	Ccedil	Ç
Eacute	É	Egrave	È	Euml	Ë	Iacute	Í
Icirc	Î	Igrave	Ì	Iuml	Ï	Ntilde	Ñ
Oacute	Ó	Ocirc	Ô	Oe	Ö	Ograve	Ò
Oslash	Ø	Ouml	Ö	Uacute	Ú	Ue	Ü
Ugrave	Ù	Uuml	Ü	Yacute	Ý	aacute	á
acirc	â	ae	ä	aelig	æ	agrave	à
amp	&	aring	å	ast	∗	atilde	ã
auml	ä	bsol	\	ccedil	ç	cent	¢
circ	ˆ	colon	:	comma	,	commat	@
copy	©	divide	÷	dollar	$	dquot	"
eacute	é	ecirc	ê	egrave	è	emsp	
ensp		equals	=	euml	ë	gt	>
hellip	...	hyphen	-	iacute	í	icirc	î
iexcl	¡	igrave	ì	iquest	¿	iuml	ï
lcub	{	lowbar	_	lpar	(	lsqb	[
lt	<	mdash	—	mu	μ	nbsp	
ndash	–	not	¬	ntilde	ñ	num	#
oacute	ó	ocirc	ô	oe	ö	ograve	ò
oslash	ø	otilde	ô	ouml	ö	para	¶
percnt	%	plus	+	plusmn	±	pound	£
quot	’	rcub	}	rpar	)	rsqb	]
sect	§	semi	;	sz	ß	szlig	ß
thinsp		tilde	˜	times	×	uacute	ú
ucirc	û	ue	ü	ugrave	ù	uuml	ü
verbar	|	yacute	ý	yuml	ÿ		

Table 3.1: General Purpose Characters

```
        "&#RS;&#RE;" psplit
        '"' qtag
        "[" ftag
        "~" nbsp
        "_" lowbar
        "#" num
        "%" percnt
        "^" circ
        "{" lcub
        "}" rcub
        "|" verbar >

<!usemap pmap p>
```

Sentences or phrases within paragraphs can be emphasized in a number of ways. The `em` tag is used to choose the default form of emphasis, which is usually *italic* type, but depends on the style of the background text. If the background text is formatted in italics type, as it usually is in definitions, for example, than emphasized text will be formatted using a plain, roman typeface. However, various forms of emphasis can be explicitly chosen. These include: **bold face** (`bf`), *italics* (`it`), sans serif (`sf`), *slanted* (`sl`), and `typewriter` (`tt`) styles.

```
<!element em - - (%inline)>
<!element bf - - (%inline)>
<!element it - - (%inline)>
<!element sf - - (%inline)>
<!element sl - - (%inline)>
<!element tt - - (%inline)>
```

The `tt` element simulates a “typewriter”. That is, with a couple of exceptions, characters are printed exactly as they appear on the display. This is useful for including small segments of computer code within paragraphs. See the section on literate programming for more information, 3.1.8.

Sentences within paragraphs can be quoted using the *short quote*, (`sq`) tag, as in `<sq>The rain in Spain falls mainly on the plain.</>`, but this is usually not necessary. In most contexts where one will want to use quotations, there is a map allowing the `"` symbol to be used as a short reference for both the starting and ending `sq` tags. So one can just type: `"The rain in Spain falls mainly on the plain."`

Quotations extending over a number of paragraphs are marked using the *long quote* (`lq`) element. Long quotes are formatted in LaTeX by indenting the left and right margins. For example, [4, pp. xiii]:

> The LaTeX document preparation system is a special version of Donald Knuth's TeX program. TeX is a sophisticated program designed to produce high-quality typesetting, especially for mathematical text. ...
>
> LaTeX represents a balance between functionality and ease of use. Since I implemented most of it myself, there was also a continual compromise between what I wanted to do and what I could do in a reasonable amount of time. ...

```
<!element sq - - (%inline)>

<!entity   ftag     '<f>'    -- formula begin -- >
<!entity   qendtag  '</sq>'>
```

```
<!shortref sqmap
      "&#RS;B" null
      '"' qendtag
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar >

<!usemap   sqmap    sq >

<!element lq - - (p*)>
```

3.1.6 Lists

Four types of lists are supported, which differ according to the type of label used to mark each item in the list. Use `itemize` to create a list in which each item is marked with some symbol such as a dash or bullet. The `enum` tag is used to create an enumeration, i.e. a list in which each item is labelled with a number (or letter) indicating its rank or position in the list. The `list` type of list does not label the items at all. Finally, use `descrip` to create a list in which each item is labelled by some tag of your own choice. Lists of various types can nested. For example:

```
<itemize>
<item>
A level one item.
<item> Here's level two:
        <enum>
        <item> A level two item.
        <item> Here's level three:
                <enum>
                <item> A level three item.
                <item>Here's level four:
                      <descrip>
                      <tag/Red./  Is the color of my true love's hair.
                      <tag/Blue./  Is a property of some movies.
                      <tag/Yellow./  Characterizes some forms of journalism.
                      </descrip>
                <item>A last level three item
                </enum>
        <item>A last level two item
        </enum>
<item>A last level one item.
</itemize>
```

This is formatted by LaTeX as:

- A level one item.
- Here's level two:

  1. A level two item.
  2. Here's level three:
     (a) A level three item.
     (b) Here's level four:
        Red.
          Is the color of my true love's hair.
        Blue.
          Is a property of some movies.
        Yellow.
          Characterizes some forms of journalism.
     (c) A last level three item
  3. A last level two item

- A last level one item.

```
<!element itemize - - (item+)>
<!element list - - (item+)>
<!element enum - - (item+)>
<!element descrip - - ((tag, (%inline; | %sectpar;)*, p*)+) >
<!element item o o ((%inline; | %sectpar;)*, p*) >
<!element tag - o (%inline)>
<!usemap global (list,itemize,enum,descrip)>
```

For reasons having to do with our translation into LaTeX, line feeds within `tag` elements are translated into spaces, using the following short reference map:

```
<!entity space " ">
<!entity null "">
<!shortref bodymap
     "&#RS;B&#RE;" ptag
     "&#RS;&#RE;" ptag
      '"' qtag
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap bodymap tag>
```

3.1.7 Figures and Tables

Figures and tables are floating elements; they may appear at a different location in the printed version of the document than in the input file. There is a location (`loc`) attribute, which can be used to influence the location chosen by the formatter. The value of the `loc` attribute is a string of up to four letters, where each letter declares a location at which the figure or table may appear, as follows:

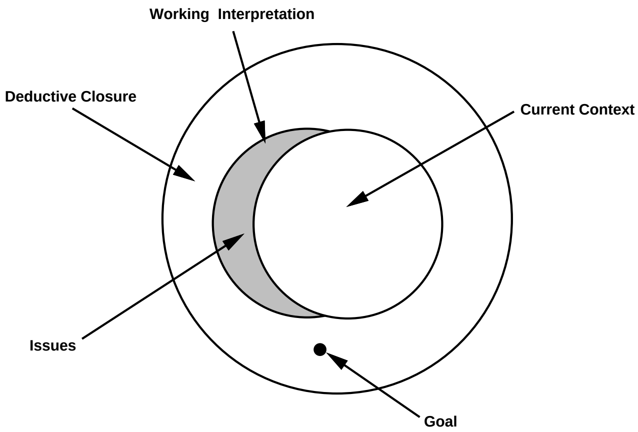

Figure 3.1: An `idraw` Drawing

`h.`

At the same relative location as it appears in the SGML input file (i.e. *here*).

`t.`

At the *top* of a page.

`b.`

At the *bottom* of a page.

`p.`

On a separate *page* containing only figures and tables.

The default value of the `loc` attribute is `tbp`.

A `figure` is a graphic combined with an optional caption. Two types of figures are currently supported. The first, and easiest, is to use the `eps` tag to include an Encapsulated PostScript file in the document. Encapsulated PostScript files are centered horizontally on the page. The size of the graphic is its "natural" size; i.e. the size it would have if printed directly on a PostScript printer. You need only know the name of the file containing the graphic.

Encapsulated PostScript graphics can be created using a variety of different editors. If you are using Unix with an X11-based graphical user-interface, you may want to try `idraw`, which stores its documents directly as Encapsulated PostScript files. Other interesting X11-based drawing program are `xfig` and `tgif`.

For example, to include the graphic contained in an Encapsulated PostScript file named `issues.ps`, you would type:

```
<figure>
        <eps file="issues">
        <caption>An <tt>idraw</> Drawing </>
</figure>
```

Which would then appear as in figure 3.1.

Notice that the ".ps" extension is *not* to be included in the file attribute of the `eps` element, but that the actual file must include the ".ps" extension.

The second possibility is to use the *placeholder* (`ph`) tag to leave space in which to later paste the graphic, in the old, reliable manner. For example, to leave 10 cm space for some graphic, type:

```
<figure>
        <ph vspace="10cm">
</figure>
```

Be sure not to leave a space between the number and the unit of measurement used, which may be `cm`, `mm` or `in`.

```
<!element figure - - ((eps | ph ), caption?)>
<!attlist figure
        loc cdata "tbp">

<!element eps - o empty  >
<!attlist eps
        file cdata #required>
<!element ph - o empty >
<!attlist ph
        vspace cdata #required>

<!element caption - o (%inline)>

<!shortref oneline
     "&#RS;&#RE;" null
     "&#RS;B&#RE;" null
      '"' qtag
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap oneline caption>
```

Next, there is a `tabular` element. Using LaTeX, tabulars must be small enough to fit on a single page. The current `tabular` element has been kept quite simple. It certainly does not (yet) offer all the flexibility of LaTeX. However, it may well be that it is sufficient for most users. More complex tables can, depending on your choice of formatters, be created using LaTeX or Unix's `tbl` program, with the `x` element, or with any program capable of generating Encapsulated PostScript, which can then be included using an `eps` element.

A `tabular` consists of a number of rows, separated by the `rowsep` element, each of which consists of a number of columns separated by the `colsep` element.

The format of the tabular is controlled by the *column alignment* (`ca`) attribute. For each column in the tabular there is a letter in the `ca` attribute: 1) `c` for centered; 2) `l` for flush left; or 3) `r` for flush right. In addition, | can be used to insert vertical lines running the complete height of the table. This will be made clear in the example which is coming shortly.

First, however, let me describe the short reference map defined for tabulars. Rather than typing `<colsep>` and `<rowsep>` explicitly, one can just type | to separate columns, and @ to separate rows. Also, within tabulars, [can be used to start a mathematical formula, and " starts short quotes as usual. (The other short references just hide any special meaning the character may have to TeX.)

```
<!entity % tabrow "(%inline, (colsep, %inline)*)" >
<!element tabular - -
        (%tabrow, (rowsep, hline?, %tabrow)*, caption?) >

<!attlist tabular
        ca cdata #required>

<!element rowsep - o empty>
<!element colsep - o empty>
<!element hline  - o empty>

<!entity rowsep "<rowsep>">
<!entity colsep "<colsep>">

<!shortref tabmap
     "&#RE;" null
     "&#RS;&#RE;" null
     "&#RS;B&#RE;" null
     "&#RS;B" null
      "B&#RE;" null
      "BB"  null
      "&#SPACE;" null
      "&#TAB;" null
      "@" rowsep
      "|" colsep
      "[" ftag
      '"' qtag
      "_" thinsp
      "~" nbsp
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub >

<!usemap  tabmap tabular>
```

The **hline** element can be use to draw a horizontal line along the length of the table, to separate rows.

A **table** element consists of a **tabular** followed by an optional **caption**. Unlikes tabulars, A **table** is a floating "body", like a figure. It may be moved to another (near) location within the formatted document. A **tabular**, however, appears at the same place in the formatted document as in the SGML source file.

```
<!element table   - - (tabular, caption?) >
<!attlist table
        loc cdata "tbp">
```

Here is how table 3.1 was typed:

```
<table>
<tabular ca="ll|ll">
ae   |       &ae   | Ae   |    &Ae       @
```

```
oe   |       &oe   | Oe   |    &Oe       @
ue   |       &ue   | Ue   |    &Ue       @
sz   |       &sz   | amp  |    &amp      @
bsol |       &bsol | circ |    &circ     @
     .
     .
     .
Dagger |       &Dagger | sect  |  &sect     @
para   |       &para   | copy  |  &copy     @
mdash  |       &mdash  | tilde |  &tilde
</tabular>
<caption><label id="GPC">
        General Purpose Characters
</caption>
</table>
```

3.1.8 Literate Programming

The original motivation behind the development of these document types was to create an environment for literate programming in an arbitrary programming language similar to Donald Knuth's WEB system for literate programming in Pascal [3]. The basic idea is to include the source code of a program inside of its documentation, instead of the other way around: including comments within the source code.

The features offered here to support literate programming, or merely the documentation of existing programs, have been kept to a minimum. Snippets of code can be mentioned within sentences using the `tt` tag. These are formatted using a `typewriter` font suitable for program code, but the spacing and indentation of the code is not retained. Within `tt` elements, the only characters which may not be literally interpreted are $, \, &, and `</`. For the $ and \ symbols, always use the `dollar` and `bsol` entities. For the & and `<` symbols, use the `amp` and `lt` entities if the string in which they occur could be mistaken for an entity reference, an element start tag or an element end tag.

To include larger segments of code, retaining its line breaks, tabulation and spacing, use the `code` tag or the `verb` tag. Within these tags just about all characters are interpreted literally. The exceptions are:

1. As SGML entities may be used within `verb` and `code` elements, use the `ero` entity to represent the &symbol in strings which might otherwise be mistaken for entity references. (Notice that the `amp` entity is not used to represent &in this context.)

2. As there must be some way of ending such elements, use the `etago` entity to represent `</` in strings which might otherwise be interpreted as end tags. (Do not use the `lt` entity for this purpose here.) Start tags can be typed literally in this context, without using entities.

3. Unfortunately TeX peeks through a bit here as well; The string `\end{verbatim}` may not occur within `code` or `verb` elements. Presumably this will not often be a problem.

For example, to include the "hello world" C program in a document, just type:

```
<code>
main ()
{
        /* This is the famous hello world program */

        printf("hello world\n");
```

```
}
</code>
```

When formatted, spaces and line breaks are preserved:

```
main ()
{
        /* This is the famous hello world program */

        printf("hello world\n");
}
```

Notice that no entities where required in this code. With few exceptions, it should be possible to just wrap `verb` or `code` tags around existing pieces of code without change.

The idea of literate programming is that the documentation *is* the program, so there must be some way of extracting the source code from the SGML document. Just how to do this is described in chapter 5, below.

The user must have a means of indicating which pieces of code are to be included in the source code, and in which order. Our solution to this problem is very simple: *Only* `code` *elements are to be extracted, and they are extracted in the same order as they appear in the document.* That is, `verb` elements are *not* extracted, and may be used, e.g., for examples or draft versions of the code included for explanatory or tutorial purposes.

`code` and `verb` elements may be formatted differently. Using our translation into LaTeX, for example, `code` elements are distinguished by being bracketed by lines the width of the page.

```
<!element code - - rcdata>
<!element verb - - rcdata>

<!shortref ttmap
        "&#RS;B" null
        '#'     num
        '%'     percnt
        '~'     tilde
        '_'     lowbar
        '^'     circ
        '{'     lcub
        '}'     rcub
        '|'     verbar >

<!usemap ttmap  tt>
```

3.1.9 Mathematical Formulas

The `qwertz` document types include elements for describing mathematical formulas completely within SGML, similar to the system described in [5]. To start, there are a fairly large number of entities for mathematical symbols. (The set of entities chosen are for the symbols available in both TeX and in the PostScript Symbol font.) Although this may be a minor irritation for seasoned TeX users, we have decided to follow the naming conventions for mathematical symbols adopted in the SGML Standard [6]. The complete set of mathematical symbols currently defined, including the Greek alphabet are listed in tables 3.2 and 3.3, in alphabetical order.

```
<!entity % math system -- math symbols -- >
%math;
```

Prime	$''$	aleph	$\aleph$	and	$\wedge$	ang	$\angle$
ap	$\approx$	arr	$\downarrow$	bottom	$\perp$	bull	$\bullet$
cap	$\cap$	cir	$\circ$	clubs	$\clubsuit$	congr	$\cong$
cup	$\cup$	diams	$\diamondsuit$	divide	$\div$	dot	$\cdot$
empty	$\emptyset$	equiv	$\equiv$	exist	$\exists$	forall	$\forall$
ge	$\geq$	hArr	$\Leftrightarrow$	harr	$\leftrightarrow$	hearts	$\heartsuit$
image	$\Im$	infin	∞	isin	$\in$	lArr	$\Leftarrow$
lang	$\langle$	larr	$\leftarrow$	le	$\leq$	mid	$\mid$
minus	$-$	nabla	∇	ne	$\neq$	nequiv	$\not\equiv$
not	$\neg$	notin	$\notin$	nsub	$\not\subset$	nsube	$\not\subseteq$
nsup	$\not\supset$	nsupe	$\not\supseteq$	nvDash	$\not\models$	nvdash	$\not\vdash$
oplus	$\oplus$	or	$\vee$	otimes	$\otimes$	part	∂
plusmn	$\pm$	prime	$'$	prop	$\propto$	rArr	$\Rightarrow$
rang	$\rangle$	rarr	$\rightarrow$	real	$\Re$	setmn	$\setminus$
spades	$\spadesuit$	square	$\square$	sub	$\subset$	sube	$\subseteq$
sup	$\supset$	supe	$\supseteq$	times	$\times$	uArr	$\Uparrow$
uarr	$\uparrow$	vDash	$\models$	vdash	$\vdash$		

Table 3.2: Math Symbols

alpha	α	beta	β	gamma	γ
Gamma	Γ	delta	δ	Delta	Δ
epsi	ε	zeta	ζ	eta	η
thetas	θ	Theta	Θ	iota	ι
kappa	κ	lambda	λ	mu	μ
nu	ν	xi	ξ	Xi	Ξ
pi	π	Pi	Π	rho	ρ
sigma	σ	sigmav	ς	Sigma	Σ
tau	τ	upsi	υ	Upsi	Υ
phis	ϕ	Phi	Φ	chi	χ
psi	ψ	Psi	Ψ	omega	ω
Omega	Ω				

Table 3.3: Greek Letters

TEX symbols not in table 2 may nonetheless be generated, by defining an entity using the `mc` element. For example, to print the $\leadsto$ symbol, you could first define an entity, perhaps using the name adopted for this symbol in the SGML standard:

```
<!entity rarrw "<mc/<x/\leadsto//">
```

Of course, this approach is TEX dependent. But this dependency is clearly noted at the beginning of the document, and it would be an easy matter to replace the TEX command for such entities with the appropriate commands for some other formatter.

The `mc` tag used in this entity definition is for *math characters*. The entity could have been defined using only the `x` tag described in section 3.3, but it is "safer" to use the `mc` tag when defining entities which are only to be used within formulas, as the SGML parser will complain if they are used elsewhere. If `x` were used instead, such errors would first be caught by TEX.

```
<!element  mc  - - cdata >
```

There are a number of parameters for formulas. These will most likely be of little interest to most users, but are stated here for the sake of completeness.

```
<!entity % sppos     "tu" >
<!entity % fcs       "%sppos;|phr" >
<!entity % fcstxt    "#pcdata|mc|%fcs;" >
<!entity % fscs      "rf|v|fi" >
<!entity % limits    "pr|in|sum" >
<!entity % fbu       "fr|lim|ar|root" >
<!entity % fph       "unl|ovl|sup|inf" >
<!entity % fbutxt    "(%fbu;) | (%limits;) |
                      (%fcstxt;)|(%fscs;)|(%fph;)" >
<!entity % fphtxt    "p|#pcdata" >
```

There are three elements for representing formulas: `f`, for ordinary short formulas appearing "in-line"; `dm` for *displayed formulas* to be centered on a line (or lines) by themselves; and `eq` for displayed formulas which are to be numbered sequentially throughout the document (i.e. so-called "equations").

```
<!element  f         - - ((%fbutxt;)*) -(footnote) >

<!entity   fendtag  '</f>'   -- formula end -- >

<!shortref fmap
      "&#RS;B" null
      "_" thinsp
      "~" nbsp
      "]" fendtag
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap   fmap     f >
```

```
<!element  dm       - - ((%fbutxt;)*) -(footnote)>
<!element  eq       - - ((%fbutxt;)*) -(footnote)>

<!shortref dmmap
     "&#RE;" space
      "_" thinsp
      "~" nbsp
      "]" fendtag
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap dmmap (dm,eq)>
```

Usually it is not necessary to type the starting and ending tags of the `f` element explicitly: [and] are short reference delimiters, allowing one to simply type, for example, `[&alpha &rarr &beta]`, instead of `<f>&alpha &rarr &beta</f>` to represent $\alpha \rightarrow \beta$.[3]

The only characters of interest in `fmap` are _ ~ and]. _ is a short reference for `thinsp`, which adds a little extra horiztonal space. ~ means `nbsp`, which in turn denotes a non-breaking space. TEX will not start a new line at a `nbsp`. Finally,] is used to end the formula. The other characters in this map just protect us from any special meaning TEXgives them.

The `dmmap` is much the same as the `fmap`. There are just two differences: 1)] is not a short reference for the `f` closing tag (and instead has its literal meaning), and 2) carriage returns and new lines are replaced by spaces, for reasons having to do with the way TEX formats formulas. Use the `tu` element, defined a bit later, to force line breaks in formulas.

Of course, formulas consist of more than just a string of math symbols. There are elements for representing fractions (`fr`), products (`pr`), integrals (`in`), sums (`sum`), roots (`root`) and arrays (`ar`). Each of these will be described next.

A fraction consists of a numerator (`nu`) and a denominator (`de`). For example, 12/37 can be written as:

```
[<fr><nu>12<de>37</fr>]
```

Of course, this is rather lengthy. For simple fractions such as this, you may prefer to just type `[12/37]`, which is formatted by LATEX in the same way.[4]

```
<!element  fr       - - (nu,de) >
<!element  nu       o o ((%fbutxt;)*) >
<!element  de       o o ((%fbutxt;)*) >
```

Products, integrals and sums all have similiar structure, consisting of a *lower limit* (`ll`), an *upper limit* (`ul`) and an optional *operand* (`opd`).

```
<!element  ll       o o ((%fbutxt;)*) >
<!element  ul       o o ((%fbutxt;)*) >
<!element  opd      - o ((%fbutxt;)*) >
```

[3] TEX users will appreciate that this notation is no more verbous than TEX.

[4] On the other hand, if you are a SGML purist, you may prefer not to do this, as it makes assumptions about the formatting system being used.

```
<!element  pr       - - (ll,ul,opd?) >
<!element  in       - - (ll,ul,opd?) >
<!element  sum      - - (ll,ul,opd?) >
```

So, for example,

$$\sum_{i=1}^{n} x_i = \int_0^1 f$$

was typed as:

```
<dm>
        <sum><ll>i=1<ul>n<opd>x<inf>i</></sum> =
        <in><ll>0<ul>1<opd>f</in>
</dm>
```

This example also shows how to represent subscripts, using the `inf` tag. There is also a `sup` tag for superscripts.

For operators with upper and lower limits other than products, sums or integrals, use the `lim` element.

```
<!element  lim      - - (op,ll,ul,opd?) >
<!element  op       o o (%fcstxt;|rf|%fph;) -(tu) >
```

For example,

$$\bigcup_{i=0}^{n}\{\alpha_i \rightarrow \beta\}$$

was typed as

```
<!entity bigcup "<mc>\bigcup</>">
...
<dm>
        <lim>&bigcup<ll>i=0<ul>n</>
             <opd>{&alpha<inf>i</> &rarr &beta}</>
        </lim>
</dm>
```

Notice that it isn't necessary to type the `op` tag here.

Roots can be represented using the, what else, `root` element. By default, `root` produces square roots. The `n` attribute of `root` can be used for other roots. For example, type `[<root n=3/x+y/]` to get $\sqrt[3]{x+y}$.

```
<!element  root     - - ((%fbutxt;)*) >
<!attlist  root
        n cdata "">
```

Arrays, or matrices, consist of a sequence of rows, each of which contains a sequence of columns. Every row in the array must contain the same number of columns. Rows are *separated* by the `arr` tag; columns by the `arc` tag. The array itself is delimited by the `ar` tag.

```
<!element col o o ((%fbutxt;)*) >
<!element row o o (col, (arc, col)*) >

<!element  ar       - - (row, (arr, row)*) >
<!attlist  ar
    ca     cdata    #required >
<!element  arr      - o empty >
<!element  arc      - o empty >
```

This is a place where an SGML short reference map has proven useful:

```
<!entity   arr "<arr>" >
<!entity   arc "<arc>" >

<!shortref arrmap
     "&#RE;" space
      "@" arr
      "|" arc
      "_" thinsp
      "~" nbsp
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub >

<!usemap   arrmap   ar >
```

Columns can be separated using the | character; rows with the @ character.

For example, this matrix

$$\begin{array}{clcr} a+b+c & uv & x-y & 27 \\ a+b & u+v & z & 134 \\ a & 3u+vw & xyz & 2,978 \end{array}$$

was typed as:

```
<ar ca=clcr>
a+b+c  |  uv     |  x-y  |  27     @
a+b    |  u+v    |  z    |  134    @
a      |  3u+vw  |  xyz  |  2,978
</ar>
```

The *column alignment* of an array must be specified using the `ca` attribute, as shown in the example. For each column in the array, there is a letter in the `ca` attribute. There are three alternatives: 1) `c` for centered; 2) `l` for flush left; and 3) `r` for flush right.

There remain a few miscellaneous math elements to describe. `sup` and `inf`, for superscripts and subscripts, were mentioned above. `unl` and `ovl` can be used to *underline* or *overline* formulas. `rf` is used for identifiers, such as function names (e.g. `cos` or `sin`) within formulas. Similarly, `phr` is used to delimit phrases of ordinary text within formulas. (Both of these are necessary, as strings of characters within formulas denote sequences of variables, not words.) The `v` tag can be used to denote a *vector*, as in $\vec{x}$. Calligraphic characters, such as $\mathcal{L}$, can be denoted using the `fi` tag. Finally, line breaks can be inserted into formulas using the `tu` element.

```
<!element  sup      - - ((%fbutxt;)*) -(tu) >
<!element  inf      - - ((%fbutxt;)*) -(tu) >
<!element  unl - - ((%fbutxt;)*) >
<!element  ovl - - ((%fbutxt;)*) >
<!element  rf  - o (#pcdata) >
<!element  phr - o ((%fphtxt;)*) >
<!element  v   - o ((%fcstxt;)*)
        -(tu|%limits;|%fbu;|%fph;) >
<!element  fi  - o (#pcdata) >
<!element  tu  - o empty >

<!usemap global (rf,phr)>
```

3.1.10 Definitions, Lemmas and Theorems

There are a number of elements useful for representing *definitions* (`def`), *propositions* (`prop`), *lemmas* (`lemma`), *corollaries* (`coroll`), *proofs* (`proof`), and *theorems* (`theorem`).

```
<!element def - - (thtag?, p+) >
<!element prop - - (thtag?, p+) >
<!element lemma - - (thtag?, p+) >
<!element coroll - - (thtag?, p+) >
<!element proof - - (p+) >
<!element theorem - - (thtag?, p+) >
<!element thtag - - (%inline)>

<!usemap global (def,prop,lemma,coroll,proof,theorem)>
<!usemap bodymap thtag>
```

With the exception of `proof`, these all have the same structure: an optional `thtag` followed by some paragraph level elements. Here is an example:

Theorem 1 (Alexander's Theorem) *Let $\mathcal{G}$ be a set of nontrivially achievable subgoals and $<$ an order on $\mathcal{G}$. $<$ is abstractly indicative if and only if it is a linearization of $<^*_{\mathcal{G}}$.*

This was typed as:

```
<theorem><thtag>Alexander's Theorem</>

 Let [<fi/G/] be a set of nontrivially achievable
subgoals and &lt an order on [<fi/G/].  &lt
is abstractly indicative if and only if it is a
linearization of
[<lim>&lt <ll> <fi/G/ <ul> &ast </lim>].

</theorem>
```

3.1.11 The `global` Short Reference Map

The `global` short reference map, which is the default map in effect within `qwertz` documents, allows the " symbol to be used to start a *short quote* (`sq`) and [to start a *formula* (`f`). Also, ˜ is used for non-breaking spaces. The rest of the short references just serve to hide any special meaning TEX gives these characters, allowing them to be directly typed without having to use entity references.

```
<!entity   qtag     '<sq>' >

<!shortref global
      "&#RS;B" null  -- delete leading blanks --
      '"' qtag
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap global qwertz>
```

3.2 Cross References

Places within a document can be marked using the `label` element. Labels have an `id` attribute for naming the label. The SGML parser will check that these identifiers are unique within the document, and that they are referenced. That is, the parser will complain if there is no reference to a label. For this reason, labels should probably be created on demand, rather than in anticipation of the need for a reference to the element.

There are two kinds of references: `ref` for references to the number of some element, such as a section, figure or theorem, and `pageref`, for references to the number of the page on which the text around the label occurs when the document is printed. Both types of references have an `id` attribute for stating the identifier of the label being referenced. The number of the element or page will be printed at the place of the `ref` or `pageref`.

```
<!element label - o empty>
<!attlist label id cdata #required>

<!element ref - o empty>
<!attlist ref
        id cdata #required>

<!element pageref - o empty>
<!attlist pageref
        id cdata #required>
```

For example, a reference to the section on miscellaneous elements of this manual, section 3.3, would be typed as:

```
... section <ref id=misc>, would be ...
```

The label itself was typed as:

```
<sect><heading><label id="misc">
        Miscellaneous Elements</>
```

3.3 Miscellaneous Elements

There are just a couple general purpose elements remaining to be discussed, which don't seem to have found a suitable home yet elsewhere in this manual.

Editorial comments and reminders to oneself can be marked with the `comment` tag. These comments will be printed using a different type style than the body of the text. In the `qwertz` mapping into TEX, they are printed using the *slanted type style*.

If you do not want the comment to be printed, use the standard SGML notation for comments instead: `<!-- ... -->`.

Finally, there is an "escape" element, allowing you to include raw formatting code at any place in your document, the `x` element. This code will be passed on to the formatter, such as TEX, inline, at the point it appears in your document. Of course, this "feature" should be used judiciously, as it limits the formatter independence of the document.

```
<!element comment - - (%inline)>
<!element x - - ((#pcdata | mc)*) >
<!usemap   #empty   x >
```

Notice that math character (`mc`) elements may appear within `x` elements. This allows you to use SGML entity references for math characters, to help avoid having to rememember both the SGML and the formatter's names for these symbols. Other entities may also be used, so long as they expand to character data.

3.4 Articles, Reports and Books

Articles, reports and books are structurally very similar. They may be formatted differently, of course, but this is of little importance during the writing phase of primary interest to authors. Seen abstractly, each type of document consists of a *title page*, for such information as the title of the document, the names of the authors and so on, followed perhaps by an *abstract*, and then by a sequence of *chapters* or *sections*. There may be *citations*, which are references to documents listed at the end, in a *bibliography*. Perhaps there are one or more *appendices*. Finally, these documents may also contain *footnotes*.

Let us first precisely describe the overall structure of these document types, before moving on to describe their various components. The article element is defined as:

```
<!element article - -
        (titlepag, header?, abstract?,
         toc?, lof?, lot?, p*, sect*,
         (appendix, sect+)?, biblio?) +(footnote)>

<!attlist article
        opts cdata "null">
```

The *options* attribute (`opts`) of `article` provides a place to state *formatting* options, which are passed on to LATEX. The particular options available depends on the installation of LATEX being used, but the following should always be available:

`11pt, 12pt.`

Set the "normal" font size to eleven, or twelve, point, instead of the default 10 point size.

`twoside.`

Formats the document for printing on both sides of a page.

`twocolumn.`

Formats the document with two columns per page, as is common in the proceedings of scientific conferences, for example.

`titlepage.`

Causes the title page and abstract to be printed on a separate page.

Other options which may be supported include:

`dina4.`

Formats the document for printing on **DIN A4** size paper. (As this is the size paper used at our installation, this option is included automatically during the translation.)

`german.`

Causes the TEX hyphenation algorithm to "think German", and sections, bibliographies and such to be labelled using the appropriate German terms.

`times, bookman, palatino` ...

Causes the "main" font to be the selected PostScript font, instead of the standard TEX font, Computer Modern, and maps all other type faces to some suitable PostScript font or type style.

For example, the starting tag for some article might be:

```
<article opts="bookman,11pt">
```

Reports are just like articles, except that they consist of a sequence of chapters (`chapt`), instead of sections (`sect`):

```
<!element report - -
        (titlepag, header?, abstract?, toc?, lof?, lot?, p*,
         chapt*, (appendix, chapt+)?, biblio?) +(footnote)>

<!attlist report
        opts cdata "null">
```

Books are similar to reports, except that they may not include an abstract:

```
<!element book  - -
        (titlepag, header?, toc?, lof?, lot?, p*, chapt*,
         (appendix, chapt+)?, biblio?) +(footnote) >

<!attlist book
        opts cdata "null">
```

The options attribute (`opt`) for `report` and `book` elements is the same as that for articles, just described, except the `titlepage` option, which is applicable only for articles.

The rest of this chapter describes the common elements of articles, reports and books, starting with title pages.

3.4.1 Title Pages

A title page (**titlepag**) consists of a title, a number of authors (**author**) and an optional date (**date**). The title may refer to a footnote and may also include a **subtitle**. If the date element is omitted, today's date will be printed by default. To avoid having a date printed, include an empty **date** element.

```
<!element titlepag o o (title, author, date?)>
<!element title - o (%inline, subtitle?) +(newline)>
<!element subtitle - o (%inline)>
<!usemap oneline titlepag>
```

The **author** element includes the **name** and, optionally, institution (**inst**) of the author. If there are multiple authors, these are separated with the **and** tag. Also, acknowledgements can be expressed using the **thanks** element. These are formatted by LaTeX as footnotes on the title page.The **author** element includes the **name** and, optionally, institution (**inst**) of the author. If there are multiple authors, these are separated with the **and** tag. Also, acknowledgements can be expressed using the **thanks** element. These are formatted by LaTeX as footnotes on the title page.

```
<!element author - o (name, thanks?, inst?,
                        (and, name, thanks?, inst?)*)>
<!element name o o (%inline) +(newline)>
<!element and - o empty>
<!element thanks - o (%inline)>
<!element inst - o (%inline) +(newline)>
<!element date - o (#pcdata)>
<!usemap global thanks>
```

Within the **titlepag**, the **title**, **subtitle**, **author** and **inst** elements can be broken into multiple lines using the **newline** element or, if you prefer, the **nl** entity.

```
<!element newline - o empty >
<!entity nl "<newline>">
```

The title page of this manual was typed as:

```
<title>The <tt/qwertz/ SGML Document Types
<subtitle>(Version 1.1 Reference Manual)
<author>Tom Gordon
<inst> Institute for Applied Information Technology (F3) &nl&nl
        German National Research Center  &nl
           for Computer Science (GMD)
```

Notice the **titlepag** tags are optional. The simplest title page would include a title and author:

```
<title> A Very Short Title Page
<author> Snoopy
```

3.4.2 Abstracts

Articles and reports, but not books, may have an abstract, which consists of one or more paragraphs, including the various kinds of lists, mathematical formulas and elements for literate programming:

```
<!element abstract - - (p+)>
```

3.4.3 Table of Contents

There are three elements for stating whether or not a table of contents, list of figures or list of tables should be included in the document. These tables and lists are generated by LaTeX. Therefore the contents of these elements is empty. They are only used to specify that the list or table should be included.

```
<!element toc - o empty>
<!element lof - o empty>
<!element lot - o empty>
```

3.4.4 Headers

A `header` element specifies what should be printed at the top of each page. It consists of a left heading (`lhead`) and a right heading (`rhead`). Both elements are required, if a heading is used at all, but either may be left empty, so that the effect of having only a left or right heading can be achieved easily enough.

```
<!element header - - (lhead, rhead) >
<!element lhead - o (%inline)>
<!element rhead - o (%inline)>
```

As we will see, an initial header can be given after the title page. Afterwards, a new header can be given for each new chapter or section. The header printed on a page is the one which is in effect at the end of the current page. So that the header will be that of the last section starting on the page.

3.4.5 Sectioning

The naming scheme we have adopted for sections is a bit different than that of LaTeX, because the names of SGML identifiers may be at most only eight characters long. But we think the scheme we have chosen has its advantages. In books and reports, the top-level sectional unit is the *chapter* (`chapt`). In articles, it is the *section* (`sect`). The lower sectional units are `sect1`, `sect2`, `sect3`, and `sect4`, in that order.

Each section (or chapter) consists of a `heading`, followed by an optional `header`, a number of paragraphs (including such things as graphics), and then sections of the next lower level.

```
<!entity % sect "heading, header?, p* " >
<!element heading o o (%inline)>
<!element chapt - o (%sect, sect*) +(footnote)>
<!element sect  - o (%sect, sect1*) +(footnote)>
<!element sect1 - o (%sect, sect2*)>
<!element sect2 - o (%sect, sect3*)>
<!element sect3 - o (%sect, sect4*)>
<!element sect4 - o (%sect)>
<!usemap bodymap (chapt,sect,sect1,sect2,sect3,sect4)>
```

Don't confuse the headers with headings. The `heading` is just the text printed at the point where the section begins, naming the section. The `header` changes the text printed at the top of each page.

If there are cross references to the section, put the `label` in the heading. For example, you could type:

```
<sect><heading><label id=mysect>My First Section</>
```

If a label isn't required, you can leave the `heading` tag implicit:

```
<sect>My First Section
```

The `appendix` element marks the begin of a sequence of appendices. These are chapters or sections, depending on whether the document is an article, report or book, and differ from ordinary chapters or sections only in the way the are numbered, and of course their placement at the end of the document.

```
<!element appendix - o empty >
```

3.4.6 Footnotes

The tag for footnotes is, simply enough, `footnote`.[5]

```
<!element footnote - - (%inline)>
<!usemap global footnote>
```

Footnotes can appear anywhere within a section (or chapter). The `usemap` declaration is required to cancel the `lines` map used in title pages.

3.4.7 Citation

Literature references can be made using the `cite` and `ncite` elements. The only difference between them is that the `ncite` allows a short *note* to be included in the reference, for such things as page numbers.

```
<!element cite - o empty>
<!attlist cite
        id cdata #required>

<!element ncite - o empty>
<!attlist ncite
        id cdata #required
        note cdata #required>
```

For example, one might type

```
<ncite id="Bryan88" note="pg.68">
```

to refer to page 88 of Martin Bryan's book on SGML. This would appear, using LaTeX, as [2, pg. 68] in the printed document.

The `id` attribute of a `cite` or `ncite` is a reference to an identifier of a BibTeX bibliography file. There is a `qwertz` SGML document type for creating such bibliographies, described below.

The bibliography itself, or list of references, is generated by including a `biblio` element near the end of the document, before the appendix.

```
<!element biblio - o empty>
<!attlist biblio
        style cdata "qwertz"
        files cdata "">
```

[5] To be sure the marker for the footnote is formatted propertly, be sure not to leave a space between the character after which the footnote marker is to appear and the beginning of the footnote element itself.

The `files` attribute of `biblio` is a list of the names of the bibliographies used, separated by commas. The names should not include any file suffixes, such as ".bib" or ".sgml". For example, to cite publications on artificial intelligence and cognitive science, where the bibliograhies are maintained in two files, `ai.sgml` and `cogsci.sgml`, you would type:

```
<biblio files="ai,cogsci">
```

The `style` attribute determines how the bibliography is formatted. Five styles are supported:

`plain`
: Entries are sorted alphabetically and labeled with numbers.

`unsrt`
: The same as `plain` except the entries are ordered as they appear in the document, rather than alphabetically.

`alpha`
: The same as `plain`, except that labels are made from the author's name and the year of publication.

`abbrv`
: The same as `plain` except that first names, month names, and journal names are abbreviated.

`qwertz`
: The same as `plain` except that all words of the entry are capitalized exactly as they appear in the source file of the bibliography. The `plain` style applies capitalization rules which are inappropriate, e.g., for German titles.

3.5 Slides

The `slides` element is for making a series of slides or, more commonly, overhead transparencies. Although you may often prefer to use some other program for preparing presentations, this approach has its advantages when you want to include parts of an existing article or book on your transparencies. You can just "cut and paste" the SGML source from an article onto a slide. You may also prefer this approach if your presentation includes mathematical formulas, to be able to take advantage of TEX's excellent mathematics typesetting.

```
<!element slides - - (slide*) >

<!attlist slides
        opts cdata "null">
```

Each slide consists of an optional title, followed by one or more `slpar` elements:

```
<!element slide - o (title?, p+) >
```

Notice that not every element available in an article or book is also available here. In particular, there are no sectioning elements, cross references, footnotes or a bibliography.[6]

The `title` element will be centered on the line. You can break up the title into multiple lines with `newline` elements. The various type style elements, such as `em` and `bf`, can also be used here; indeed anywhere on a slide.

[6] Our translation into TEX does not use SliTEX, so as to allow slides to include tables and figures.

3.6 Letters and Electronic Messages

The `letter` element is for making letters and e-mail messages. Just how a letter is formatted may depend on whether it is a business or personal letter. If it is a business letter, it may be printed to appear as if the company's letterhead stationery had been used.

The structure of a letter can be quite complex, but most the elements to be described here are optional. Using an example from [4], a simple letter would be typed like this:

```
<letter>
<from>
        R. (Ma) Dillo
        <address> 1234 Ave.~of the Armadillos &nl
                  Gnu York, G.Y. 56789
<to>
        Dr.~G. Nathaniel Picking
        <address> Acme Exterminators &nl
                  33 Swat Street &nl
                  Hometown, Illinois 62301

<cc> Jimmy Carter &nl
     Richard M. Nixon

<opening> Dear Nat,

 I'm afraid that the armadillo problem is still
with us.  I did everything ...

 ... and I hope we can get rid of the nasty beasts
this time.

<closing> Best regards,

</letter>
```

The `from` and `to` elements are for the sender's and receiver's names and addresses, respectively. The address may be either a street address, using `address`, or an electronic mail address, using `email`, or both. You may also include a telephone number, using the `phone` element. (If you are using your company's letterhead stationery, it may be that you should type only your extension, rather than your complete telephone number.) Finally, a telefax number can be provided, using the `fax` element.

Notice that in the `closing` you must type a comma yourself, if you want one. Also, do not type your name again after the closing; the `name` of the sender will be printed after the closing as expected.

There are several optional elements which may be of interest:

`subject`

For the purpose or, well, subject of the letter. If you would like this subject line to appear as "re: ...", for example, you must type the "re: " yourself, as part of the subject.

`sref, rref, rdate`

These are tags for the *sender's reference*, *receiver's reference* and *receiver's date* where you can include whatever code is used by your, or the recipient's, company or institution to uniquely identify letters. For example, if this letter is a response to some other letter, you may use the **rref** and **rdate** elements to identify the original letter. There is no **sdate** tag, as the date this letter is printed will be included in the letter at some appropriate place by the formatter.

cc

This used to be an acronym for "carbon copies", which were to be sent to persons other than the principal recipient of the letter. The cc tag can be used to list these other recipients, even though the copies they receive today are perhaps printed by a laser printer on recycled paper. As in the above example, you can separate the names of these recipients with **newline** elements (using the **nl** entity if you prefer).

encl

Use this tag to list *enclosures*. These can also be separated with **newline** elements, or simply with commas, if you prefer.

ps

A postscript, not to be confused with PostScript, can be included with this tag. Any kind of element which can appear in the body of the letter (i.e. **sectpar** elements) can also be used here.

To summarize, here are the relevant SGML declarations:

```
<!entity  % addr "(address?, email?, phone?, fax?)" >

<!element letter - -
        (from, %addr, to, %addr, cc?, subject?, sref?, rref?,
         rdate?, opening, p+, closing, encl?, ps?)>

<!attlist letter
        opts cdata "null">

<!element from          - o (#pcdata) >
<!element to            - o (#pcdata) >

<!usemap oneline (from,to)>

<!element address       - o (#pcdata) +(newline) >
<!element email         - o (#pcdata) >
<!element phone         - o (#pcdata) >
<!element fax           - o (#pcdata) >

<!element subject       - o (%inline;) >
<!element sref          - o (#pcdata) >
<!element rref          - o (#pcdata) >
<!element rdate         - o (#pcdata) >

<!element opening       - o (%inline;) >
<!usemap bodymap opening>

<!element closing - o (%inline;) >
<!element cc - o (%inline;) +(newline) >
<!element encl - o (%inline;) +(newline) >

<!element ps - o (p+) >
```

3.7 Telefax Messages

The structure of a telefax message is the same as for letters and e-mail messages, except that the `fax` number of the recipient is, of course, required, rather than optional.

```
<!element telefax - -
        (from, %addr, to, address, email?,
         phone?, fax, cc?, subject?,
         sref?, rref?, rdate?,
         opening, p+, closing, ps?)>

<!attlist telefax
        opts cdata "null"
        length cdata "2">
```

3.8 Notes

The `notes` element is a new top-level document "style", like articles, books and letters. It is useful for miscellaneous purposes, such as jotting down notes to oneself, where the complex structure of the other styles is unnecessary. Notes here simply a sequence of section paragraphs (i.e. paragraphs, lists, comments, long quotations, figures, tables, displayed mathematical formulas, and program code). An optional title is also available. The contents of a notes document can be copied and pasted into a section or chapter of a book or article.

```
<!element notes - - (title?, p+) >
<!attlist notes
        opts cdata "null" >
```

3.9 Manual Pages

The `manpage` element is for Unix manual pages. Here we see again an advantage of SGML. Using this element, the very same manual page can be viewed on just about every terminal, using `nroff`, or be included as a section of an article, report or book to be formatted by TEX.

```
<!element manpage - - (sect1*)
        -(sect2 | f | %mathpar | figure | tabular |
          table | %xref | %thrm )>


<!attlist manpage
        opts cdata "null"
        title cdata ""
        sectnum cdata "1" >
```

A manpage consists of a sequence of sections. There are two SGML attributes, for the command name and manual section number, respectively. Each section of the manual page is delimited by a `sect1` element. *Notice that these sections may not contain further subsections.* Sections are represented as `sect1` elements, rather than `sect`, to allow the manual page to be easily cut and pasted into a `sect` section of an article, report or book. (Of course, if the manual page is to be used a chapter of a book, then these sections of the manual page will need to be replaced with `sect` elements.)

Notice that Many elements, such as tables, figures and mathematical formulas, cannot be used within manual pages, because of limitations of ASCII terminals, or the Unix `man` macro package for `nroff`.

There is a short reference map in effect within the scope of the `manpage`. With the exception of [, which is not used here to start formulas, this map *has the same effect* as the `global` map.

```
<!shortref manpage
      "&#RS;B" null
      '"' qtag
      "[" ftag
      "~" nbsp
      "_" lowbar
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar>

<!usemap manpage  manpage >
```

3.9.1 Manual Page Conventions

For detailed information about the conventions for Unix manual pages, see your Unix documentation. But here is a brief summary. The typical manual page has the following sections, in this order:

NAME.
The name, or list of names, by which the command or function is called, followed by a dash and then a one-line summary of its purpose.

SYNOPSIS.
For the syntax of the command and its arguments. (The Sun documentation suggests that literals be formatted using boldface type, and that variables be formatted using italics type. Use the `tt` and `em` elements, respectively, here for this purpose.)

DESCRIPTION.
An overview of the command or function's purpose, effects and use.

OPTIONS.
A list and description of all command-line options.

FILES.
A list of files associated with the command which may be of interest to users.

SEE ALSO.
A comma-separated list of related Unix commands, and references to other relevant publications.

DIAGNOSTICS.
A list and explanation of any diagnositic messages the command may write to the standard error output file.

BUGS.
A description of any known bugs, problems, or limitations.

Some of you may be asking yourselves why manpage wasn't designed so that each of these conventional sections of a manual page is represented by its own SGML element. That certainly would have been possible, but on the other hand the approach taken has the advantage that users can simply cut and paste sections between manual pages and article, reports and books. Of course it would have been easy to write a filter to convert between these formats, but it was felt that the benefits of a special manpage format would be too small to warrant even this limited effort. After all, unless one is using an SGML structure editor, users must refer to the SGML document type definition to know what is expected in the manual page. It is just as easy to check this documentation to see what sections conventionally appear in manual pages. There is also a file which can be used as a template or form for writing manual pages. See the Unix Commands chapter for details.

The only reason there is a manpage document type, instead of just another translation of, say, the article document type into nroff is that the man macros used for the Unix documentation are not powerful enough to format all of the features available in our latex document type. Having this separate manpage document type provides a means of checking whether the manual page can be formatted by nroff using these man macros. Again, as this document type is designed to be a subset of the latex document type, the sections of a manual page can also be included within instances of the latex document type.

3.9.2 Manual Page Example

Here is how the manual page for the cd command could have been typed using this document type definition:

```
<manpage title="CD">

<sect1> NAME

cd &mdash change working directory

<sect1> SYNOPSIS

 cd [ <em>directory</> ]

<sect1> DESCRIPTION

 <em>directory</> becomes the new working directory.  The process
must have execute (search) permission in <em>directory</>.  If cd is
used without arguments, it returns you to your login directory.

...

<sect1> SEE ALSO

 csh(1), pwd(1), sh(1)
</manpage>
```

This is the end of the qwertz document type definition.

```
<!-- end of qwertz dtd -->
```

Chapter 4

Bibliographies

This chapter describes an SGML document type definition for bibliographies. The logical structure of these bibliographies is the same as in BibTEX. There is a Unix tool, described in the "Unix Commands" chapter, for translating bibliographies in this SGML format into BibTEX's `bib` format or, if you prefer, into Unix's `refer` format.

Why use this SGML format instead of the `bib` format directly? There may be a few reasons. If you are using SGML to write your articles, then it is probably easier to use this SGML document type for your bibliographies. The syntax of BibTEX's `bib` files is, not surprisingly, completely different. Also, here too SGML offers the possibility of easily translating these bibliographies into some other format, such as Unix's `refer` format. Finally, some may prefer to maintain their bibliographies in a semi-WYSIWYG way, using an SGML structure editor.

This chapter is divided into an overview followed by a reference section. The overview contains a simplified description of the structure of bibliographies; it is correct as far as it goes, but certain details are not discussed. The reference section contains the complete SGML Document Type Definition for bibliographies and should be consulted for further information or clarification.

4.1 Overview

First, here is an example of a (very short) bibliography in the format of the `biblio` document type.

```
<!doctype biblio public "-//GMD//DTD biblio//EN">
<biblio>

<inproc id="gordon:etal:92:acmml">
<author> T.F. Gordon <and> J. Hertzberg <and> A. Horz
<title> Experiences with ML for Building an AI Planning Toolbox
<btitle> Proc. ACM Workshop on ML and its Applications
<year> 1992
<note> To appear
<annote> The <tt/qwertz/ toolbox is a system of ML modules for
implementing AI planners.  We discuss why ML was chosen to implement
the system, provide a few guidelines for using the language, based on
our experience, and make some suggestions for language extensions and
programming environments.
<keywrds> qwertz AC

<report id="guesgen:hertzberg:92:AP608">
<author> H.W. G&ue;sgen <and> J. Hertzberg (eds.)
```

```
<title> On Space and Time
<inst> GMD
<type> Arbeitspapier
<no> 608
<year> 1992
<annote> The report is a collection of papers about spatial and
temporal reasoning, stemming from members of GMD's AI division. Some
of them are in German, and some of them have been published previously.
<keywrds> qwertz AC

<report id="guesgen:hertzberg:91:tasso31">
<author> H.W. G&ue;sgen <and> J. Hertzberg
<title> A Constraint Based Approach to Spatiotemporal Reasoning
<inst> BMFT-Verbundprojekt TASSO
<type> TASSO-Report
<no> 31
<year> 1991
<annote> In this paper, we introduce a form of spatiotemporal
reasoning that uses homogeneous representations of time and the three
dimensions of space.  The basis of our approach is Allen's temporal
logic on the one hand and general constraint satisfaction algorithms
on the other, where we introduce a new view on constraint reasoning to
cope with the affordances of spatiotemporal reasoning as introduced
here. As a realization for constraint reasoning, we suggest a
massively parallel implementation in form of Boltzmann machines.
<keywrds> qwertz TASSO AC

<incoll id="hertzberg:92:racProbleme">
<author> J. Hertzberg
<title> &Ue;ber Verzweigungen, Kulissen und das Yaler Schie&sz;en --
        Probleme beim reasoning about change
<editor> H.W. G&ue;sgen <and> J. Hertzberg
<btitle> On Space and Time
<publish> Arbeitspapiere der GMD No. 608
<year> 1992
<pp> 143-158
<annote> In diesem Aufsatz stelle ich die klassischen Probleme
zusammen, die auftreten, wenn man &ue;ber die Effekte von Ereignissen logisch
schlie&sz;t, und ich beschreibe, wie sie zusammenh&ae;ngen. Im einzelnen
behandle ich die folgenden Probleme: qualification problem, prediction
problem, persistence problem, frame problem und ramification
problem. Zus&ae;tzlich beschreibe ich kurz das Yale shooting
problem, das allerdings auf einer anderen Ebene liegt als die
vorgenannten.
<keywrds> qwertz TASSO AC

<report id="rutten:91">
<author> E. Rutten
<title> A Temporal Non-linear Planner: TRIPTIC
<inst> GMD
<type> Arbeitspapier
<no> 582
<year> 1991
```

```
<annote> The problem addressed in this report is that of the coupling
of a non-linear planning algorithm with a time map manager.  We give a
description of a solution integrating a planner on top of a time map
manager, determining a particular share of the tasks between the two
levels.
<keywrds> qwertz TASSO

</biblio>
```

Supposing this bibliography is in a file named "mybib.sgml", you can check that it is syntactically correct by following this simple procedure:

1. Make sure the `/home/qwertz/format/bin` directory is included in your Unix `PATH` variable.
2. execute this command:

   ```
   $ biblio -c mybib.sgml
   ```

That's it! The `biblio` command will write easy to understand error messages to your terminal, if there are any.

Once you've corrected your errors, the bibliography can be converted into BibTeX, if you'd like, with this command:

```
$ biblio mybib.sgml > mybib.bib
```

The first entry of our sample bibliograhy starts with the `<inproc id="gordon:etal:92:acmml">` *tag*. It is used to state the type of the entry, here an article in the proceedings of a conference, and give it a unique identifier.[1] There are many other types of entries, in addition to articles, such as books, reports and Ph.D. theses. A complete list will follow shortly.

This entry has seven *fields*: author, title, journal, volume, number, pages, and year. (The names of some of these fields are abbreviated in the entry.) The fields required for an entry depends on the type of the entry. In the case of articles, the `author`, `title`, `journal` and `year` fields must be included.[2] The `vol`, `no`, and `pp` fields are optional.

Within each field, just about every ASCII character can be used. There are only a few exceptions to be aware of. The most important in practice is &, the ampersand sign. Use the `amp` entity to obtain this character. Foreign language — that is, non-English — characters must be entered using SGML entities. For example, German characters can be entered as follows: ä (`ae`), Ä(`Ae`), ö (`Oe`), Ö (`Oe`), ü (`ue`), Ü (`Ue`), and ß (`sz`). There are also entities for several general purpose symbols, such as £, ¶ and §. See the *General Purpose Characters* table in Chapter 4 for a complete list. The table also include entity names for several ASCII symbols. Use these entities if some ASCII is not being printed as expected. Mathematical formulas cannot be used within bibliographies.

Multiple authors or editors are separated by `and` elements, as in this example:

```
<book id="Smith88">
<author>  Joan M. Smith <and> Robert Stutely
<title> SGML: the user's guide to ISO 8879
<publish> Ellis Horwood
<year> 1988
```

[1] Conveniently, the SGML parser will complain if the identifier is not unique.
[2] The order of fields is no longer fixed; they may appear in any order, so long as each required field is present.

Altogether, there are almost 30 different fields available. Which of these are required or optional depends on the entry type. A book, for example, requires different fields than a Ph.D. thesis. An alphabetical list describing all of the fields follows. After this list, the role these fields play in each of the entry types will be explained. First, here are the fields:

`address`

The address of the publisher. For large publishers, just provide a city. For smaller publishers, type the full address.

`annote`

An annotation. However, it is not included in the list of references, using any of the standard styles.

`author`

The name of the author or authors. Each author's name can be typed in full without using tags to separate the first and last names. However, sometimes BibTEX needs help parsing names correctly, in which case you can use the `first` and `last` tags to unambiguously delimit the parts of the name. If there are mulitple authors, use the `and` tag to separate them, as in the previous example.

`btitle`

The book title, when the entry is for a part of the book, such as a chapter. If the entry is for the book itself, its title is given using the `title` element.

`chap`

The *number* of the chapter in a book.

`edition`

The edition of the publication, such as "second".

`editor`

The editor (or editors) of the publication. See the description for the `author` field above for information about how to enter names.

`howpub`

To describe how the document was published, if not by a publisher or some institution.

`id`

The library identification number of the book, for *your* library. This will not be included in entry as it is printed in the list of references, at least not when using one of the standard styles.

`inst`

The *institution* which published the item, such as a university or research institute, if not a commercial publisher.

`journal`

The name of the journal or magazine in which an article was published.

`key`

Used for alphabetizing and creating a label when there is no `author` or `editor`.

`keywrds`

A list of *key words* for categorizing the item. These words are not included in the list of references when using one of the standard styles.

`loc`

The current *location* of the document. Useful for keeping track of who has borrowed your books! Not printed in the list of references when using one of the standard styles.

`month`

The month the work was published or, if unpublished, written.

`note`

Additional information which may be useful to the reader. Shorter than an annotation, using `annote`.

`no`

The *number* of a journal, magazine or technical report. An issue of a journal is usually identified by both its *volume* (using the `vol` element described below) and number.

`organ`

The *organization* sponsoring a conference. Used in the entries for proceedings.

`pp`

One or more page numbers or range of page numbers, such as

```
<pp> 34,58,62
```

or

```
<pp> 56-108
```

Notice that it is not necessary here to use the `ndash` entity between the page numbers. An ordinary hyphen will be interpreted here by BibTEX as a dash for number ranges.

`private`

For reminders or comments to yourself, that you do not want to appear in an annoted bibliography (using `annote`).

`publish`

The name of the publisher.

`school`

The name of the university where a thesis or disseration was written.

`series`

The name of a series or set of books.

`title`

The item's title.

`type`

The type of a technical report; for example "Research Note".

`vol`

The volume of a journal or multivolume book.

`year`

The year the entry was published, or if it was never published, written. This should consist only of numbers, such as "1984".

Including articles and books, there are thirteen types of bibliographic entries. Each type is described below, including a list of required and optional fields. The order in which the fields are listed is no longer significant. A question mark behind the field name means the field is optional. Alternative fields are connected by a | symbol, as in `(author | editor)`. Finally, every entry, regardless of its type, can include any of these optional fields: `note?, annote?, keywrds?, loc?, id?, private?`.

`article`

For an article from a book, magazine, or journal. Use `inproc` for article in conference proceedings, and `incoll` for a chapter or part of a book with its own title. The fields of an article are: `author, title, journal, vol?, no?, pp?, month?, year`

`book`

For published books, with a known publisher. If the book has yet to be publised, or the publisher is unknown, use an `unpub` or `booklet` element. The fields of a book entry are: `(author | editor), title, edition?, publish, series?, vol?, address?, month?, year`

`booklet`

For documents without a named publisher or sponsoring institution. If the `author` is not known, a `key` is required so that the entry can be ordered alphabetically in the list of references. Its fields are: `(author | key), title, howpub?, address?, month?, year?`

`inbook`

A part of a book, either a chapter or some range of pages. The fields are: `(author | editor), title, ((chap, pp?) | pp), edition?, publish, series?, vol?, address?, month?, year`

`incoll`

A part of a book with its own title. Its fields are: `author, title, editor?, btitle, chap?, edition?, publish, address?, month?, year, pp?`

`inproc`

An article in the proceedings of a conference. An `inproc` entry consists of: `author, title, editor?, btitle, organ?, publish?, address?, month?, year, pp?`

`manual`

For user manuals and other technical documentation. Its fields are: `(author | key), title, edition?, address?, month?, year?, organ?`

`masters`

For a master thesis. The fields are: `author, title, school, address?, month?, year`

`misc`

For documents which don't seem to be one of the other entry types: `(author | key), title?, month?, year?, howpub?`

`phd`

For a Ph.D. thesis. The fields are the same as for `masters` entries: `author, title, school, address?, month?, year`

`proc`

The proceedings of a conference: `(editor | key), title, organ?, publish?, address?, month?, year`

`report`

For technical reports published by a university or research institute. These are usually numbered within a series. Its fields are: `author, title?, inst, address?, type?, no?, month?, year`

`unpub`

Unpublished documents: `author, title, month?, year`

4.2 Reference Section

This reference section is annotated SGML source code of the `biblio` document type definition.

4.2.1 Entities and Short Reference Maps

As this document type is completely independent of the `qwertz` DTD described in chapter 4, we must redefine here the entities and short reference maps required. These are almost exactly the same as those described in chapter 4, so you may want to just skip this section. The only difference you should be aware of, is that mathematical formulas cannot be used within bibliographies. Thus, the entities for mathematical symbols defined in chapter 4 cannot be used here. On the other hand, as the [symbol is not need as a short reference for starting formulas, it can be typed literally.

```
<!entity % general system -- general purpose characters -- >
%general;

<!entity   Ae  '&Auml;' >
<!entity   ae  '&auml;' >
<!entity   Oe  '&Ouml;' >
<!entity   oe  '&ouml;' >
<!entity   Ue  '&Uuml;' >
<!entity   ue  '&uuml;' >
<!entity   sz  '&szlig;' >

<!entity   qtag     '<sq>'   -- short quote begin -- >
<!entity   qendtag  '</sq>'>

<!shortref global
      '"' qtag
      "_" emsp
      "~" nbsp
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
```

```
      "}" rcub
      "|" verbar>

<!usemap global biblio >

<!shortref sqmap
      '"' qendtag
      "_" emsp
      "~" nbsp
      "#" num
      "%" percnt
      "^" circ
      "{" lcub
      "}" rcub
      "|" verbar >

<!usemap   sqmap    sq >
```

4.2.2 Entry Types

There are thirteen different types of entries:

```
<!entity % etype
   "( article | book | booklet | inbook | incoll |
      inproc  | manual | masters | misc | phd |
      proc | report | unpub )" >

<!attlist %etype  id cdata #required >
```

The `id` attribute of each entry type is required. It is the identifier used in a `cite` or `ncite` element (cf. chapter 4, section 4.7) to refer to the entry.

A bibliography file consists of one or more entries.

```
<!element biblio - - ((%etype)+) +(x)>
```

Notice that `x` elements, for inline formatting code, can be included anywhere within the bibliography.

A following *parameter entities*, will be used in the definitions of the entry types below.

```
<!entity % info
        "note? & annote? & keywrds? & loc? & id? & private?" >

<!entity % emph " em | it | bf | sf | sl | tt " >

<!entity % inline " (#pcdata | %emph; | sq )* " >
```

A description of each entry type is in the overview section, above. The fields used in these entries are defined in the following section.

```
<!element article - o
        (author & title & journal & vol? & no? &
         pp? & month? & year & %info )>

<!element book  - o
        ((author | editor) &  title &  edition? &
         publish & series? & vol? & address? &
         month? &  year & %info )>

<!element booklet - o
        ((author | key) & title & howpub? &
         address? & month? & year? & %info )>

<!element inbook - o
        ((author|editor) & title & ((chap, pp?) | pp) &
         edition? & publish & series? & vol? & address? &
         month? & year & %info )>

<!element incoll - o
        (author & title & editor? & btitle & chap? &
        edition? & publish & address? &  month? &
        year & pp? & %info )>

<!element inproc - o
        (author & title & editor? & btitle & organ? &
         publish? & address? & month? & year & pp? & %info )>

<!element manual - o
        ((author | key) & title & edition? & address? &
         month? & year? & organ? & %info )>

<!element masters - o
        (author & title & school & address? & month? &
         year & %info )>

<!element misc   - o
        ((author | key) & title? & month? & year? &
         howpub? & %info )>

<!element phd    - o
        (author & title & school & address? & month? &
         year & %info )>

<!element proc  - o
        ((editor | key) & title &  organ? & publish? &
         address? & month? & year & %info )>

<!element report - o
        (author & title & inst & address? & type? & no? &
         month? & year & %info )>

<!element unpub  - o
        (author & title & month? & year & %info)>
```

4.2.3 Fields

Here we describe the purpose of the elements used in the entry types above. First of all, `x`, `sq` and various sorts of emphasis, such as `tt` and `em`, can be used in most fields of an entry. (Recall, however, that the `global` and `sqmap` short reference maps are designed so that you need not type the `sq` starting and ending tags. Just use the " symbol as usual.)

```
<!element x  - - rcdata   >
<!element sq - - (%inline)>
<!element em - - (%inline)>
<!element bf - - (%inline)>
<!element it - - (%inline)>
<!element sf - - (%inline)>
<!element sl - - (%inline)>
<!element tt - - (%inline)>
```

The fields available are described in the overview, above.

```
<!element address       - o (%inline) >
<!element annote        - o (%inline) >
<!element author        - o (names) >
<!element names         o o
        ((%inline | (first, last)),(and, names)?) >
<!element last          - o (%inline)>
<!element first         - o (%inline)>
<!element and           - o empty >
<!element btitle        - o (%inline) >
<!element chap          - o (%inline) >
<!element edition       - o (%inline) >
<!element editor        - o (names) >
<!element howpub        - o (%inline) >
<!element id            - o (#pcdata) >
<!element inst          - o (%inline)  >
<!element journal       - o (%inline) >
<!element key           - o (#pcdata) >
<!element keywrds       - o (#pcdata) >
<!element loc           - o (%inline) >
<!element month         - o (#pcdata) >
<!element note          - o (%inline) >
<!element no            - o (#pcdata) >
<!element organ         - o (%inline) >
<!element pp            - o (#pcdata) >
<!element private       - o (%inline) >
<!element publish       - o (%inline) >
<!element series        - o (%inline) >
<!element school        - o (%inline) >
<!element title         - o (%inline) >
<!element type          - o (%inline) >
<!element vol           - o (#pcdata) >
<!element year          - o (#pcdata) >
```

Here is the end of the `biblio` document type definition.

```
<!-- end of biblio dtd -->
```

Chapter 5

Unix Commands

The `qwertz` document processing system consists of a suite of four Unix shell scripts:

`format`

: translates `qwertz` documents into various formats, including LaTeX and various versions of `roff`. `format` can also be used to simply check that a document is a correct instance of the `qwertz` DTD.

`qtex`

: is a front-end to LaTeX and BibTeX which may be used as a filter in Unix pipes. By default, it translates LaTeX documents into PostScript, which can then be piped to `lpr` for printing. Alternatively, a `dvi` file can be generated.

`qroff`

: is a front-end to `groff`, analogous to `qtex`. By default, it translates `groff` documents using the `ms` macro package into a formatted `ascii` file suitable for electronic messages. It can also be used to generate PostScript, to obtain an alternative layout style to that produced by LaTeX.

`extract`

: extracts source code from `qwertz` documents. It is useful for "literate programming".

`biblio`

: translates a bibliography written using our `biblio` document type definition into your choice of BibTeX's `bib` format or Unix's `refer` format.

`fax`

: sends a message, in PostScript format, to a telefax machine.

The next few sections are the manual pages for each of these commands.

5.1 Formatting, Previewing and Printing

5.1.1 NAME format — translate SGML into LaTeX or nroff/troff

5.1.2 SYNOPSIS

`format [-c] [-t int] [-T format] file[.sgml]`

5.1.3 DESCRIPTION

`format` translates an SGML file of the `qwertz` document type into LaTeX and various forms of `nroff/troff`. The ".sgml" file name extension is optional. It first checks that the file is a syntactically correct instance of the `qwertz` document type. The default action is to write LaTeX to the standard output. To print a `qwertz` file named `foo.sgml`, using LaTeX, just type

```
format foo | qtex | lpr
```

`qtex` is a front-end to LaTeX designed to be used as a filter in this way. See `qtex(1)` for more information.

If you need more control over printing than is available using `lpr` alone, you can first create a `dvi` file, and then print the file using a `dvi` to PostScript converter, such as `dvips`. For example, to print pages 3 to 10 of `foo`, you could type

```
format foo | qtex -d > foo.dvi
dvips -p 3 -n 8 foo | lpr
```

If there is a PostScript previewer available, such as `ghostview`, from the GNU distribution, you can preview the document by typing

```
format foo | qtex > foo.ps; ghostview foo.ps
```

Or, you may also preview the document using a `dvi` previewer such as `xdvi` by typing

```
format foo | qtex -d > foo.dvi;  xdvi foo
```

You can also use `nroff` to view the document on an ASCII terminal:

```
format -T nroff foo | nroff -ms | more
```

Finally, `nroff` may also be suitable for sending the document to someone by electronic mail:

```
format -T nroff foo | nroff -ms | mail thomas@gmdzi.gmd.de
```

5.1.4 OPTIONS

In the following, it is assumed that the file to be formatted is named `foo.sgml`, or simply `foo`.

`-help`

If this is the only option, a short message listing all the options available, and their purpose, is printed to the standard out.

`-c`

Just *check* the syntax of the document, to see that it is a correct instance of the `qwertz` document type.

`-T [latex | nroff | grops | man]`

Use `-T` to select the format in which to translate the `qwertz` document. By default, LaTeX is used. `nroff` is appropriate for ASCII devices. `grops` is a `troff` format for `groff`, a reconstruction of `troff` from the Free Software Foundation. `grops` uses `groff's` names for non-ASCII characters and symbols. To use `groff` to generate an ASCII version of the document, use the `nroff` format, as in:

```
format -T nroff foo | groff -ms -T ascii | more
```

Both the `grops` and `nroff` formats use the `ms` macro package. Also, documents containing tables and formulas will need to be preprocesed by `tbl(1)` and `eqn(1)`, respectively.

The `man` format generates `nroff` using the `man` macro package for Unix manual pages. Manual pages may also be formatted using any of the other formats.

By default, `format` looks in the `$FORMAT/rep` directory for the files controlling the translation of `qwertz` documents into various formats. If you wish to customize the appearance of your documents, or support additional formats, you may create your own mappings. Just set the `REPDIR` shell variable to the directory containing your formats. For each format, `f`, there should be a subdirectory named `f` in the `REPDIR` directory. Within this subdirectory, there must be three files, named: `mapping`, `general` and `math`. The `mapping` file is a simple table from SGML start and end tags into instructions or commands for the goal formatting system. (The table must be in the format first used by the Amsterdam SGML Parser.) See `$FORMAT/rep/qwertz/latex/mapping` and `$FORMAT/rep/qwertz/grops/mapping` for examples. The `general` and `math` files are SGML files containing the formatter specific definitions of general purpose characters and mathematical symbols, respectively. Again, see the examples in `$FORMAT/rep/qwertz/latex`.

`-t` *n*

`format` always pipes the input document through `expand` to replace tabs by spaces. By default, each *tab* is replaced by eight spaces. You can change this by using this option.

5.1.5 SEE ALSO

`qtex(1), nroff(1), tbl(1), eqn(1), groff(1), dvips(1),`
`expand(1), sed(1), lpr(1), ghostview(1), xdvi(1)`

5.1.6 DIAGNOSTICS

SGML syntax errors are reported to standard out. Each error message mentions the line number containing the error.

5.2 Processing LaTeX files

5.2.1 NAME qtex — translate a LaTeX file into PostScript

5.2.2 SYNOPSIS

```
qtex [-dsxgbvc] file[.tex] | lpr
cat file.tex | qtex [-dsxgbvc] | lpr
```

5.2.3 DESCRIPTION

`qtex` is a front-end to LaTeX, BibTeX and a `dvi` to PostScript driver, which makes them considerably easier, and more Unix-like, to use. Unlike LaTeX, it can also process LaTeX files read from the standard input file, or in a Unix pipe. By default, `qtex` writes PostScript to the standard output file, but there is a command line switch, described below, for generating the usual TeX`dvi` file instead. The ".tex" file extension is optional. If LaTeX finds an error in your input file, a message will be printed to the standard error file and execution is terminated, with a non-zero exit status. There will be a ".log" file in the same directory as the input file describing the errors.

5.2.4 OPTIONS

`-help`

If this is the only option, a short message listing all the options available, and their purpose, is printed to the standard out.

`-d`

Create a TeX device-independent file, i.e. a `dvi` file, instead of PostScript. This is piped to the standard output file. To save the `dvi` file, write, for example:

```
qtex -d foo > foo.dvi
```

`-s`

Save all temporary files, such as `foo.tex`, `foo.dvi`, and `foo.aux`. PostScript is generated and written to standard out as usual.

`-x`

Use this option if the input file contains *cross references*. This option causes the LaTeX file to be processed by TeX twice, to resolve the references.

`-g`

Use this option if the document is written in *German*. It causes TeX to use another algorithm for hyphenation. Remember to *also* use the `german` option in the options list of the document itself.

`-b`

Use this option if the file includes a list of references or *bibliography*. The option causes LaTeX, then BibTeX and then LaTeX (twice) again to be run. The bibliography referenced in the document is expected to be a BibTeX `bib` file in the same directory as the document. If your bibliography is being maintained in the `biblio` SGML format, use the `biblio` command to first translate it into `bib` format. The `-b` option implies the `-x` option, for cross-references, described above.

`-v`

Verbose mode. If your document is particularly long, you may want to use this mode to be kept posted about just what `qtex` is doing at the moment. The messages are written to standard error, so that they are not mixed with the PostScript being piped to the printer!

`-c`

To process a single chapter or section not containing the necessary LaTeX header (i.e. the `\documentstyle` and `\begin{document}` commands), use this option. They will be added to the file before it is processed by TeX This option is necessary when formatting a LaTeX chapter or section generated from SGML using the `format` command.

5.2.5 SEE ALSO

`tex(1), format(1)`

5.3 Generating Formatted `ASCII` Files

5.3.1 NAME qroff — translate a `groff ms` file into ASCII

5.3.2 SYNOPSIS

```
format -T nroff file | qroff [-b <sgml bibliography>]
format -T grops file | qroff -p [-b <sgml bibliography>] | lpr
```

5.3.3 DESCRIPTION

`qroff` is a greatly simplified front end to `groff` and `grefer`. Its primary purpose is to generate formatted `ASCII` versions of `qwertz` SGML documents suitable, e.g., for electronic messages. However, it may also be used as an alternative to LaTeX for printing the document. Some may prefer `groff`'s layout style. (Unfortunately, `groff` does not support cross-references, so if you use these, you will need to LaTeX .)

`qroff` only reads from the standard input; it is designed to be used in a pipe, as in the examples in the synopsis, above.

5.3.4 OPTIONS

`-help`

If this is the only option, a short message listing all the options available, and their purpose, is printed to the standard out.

`-p`

Create a PostScript file, instead of `ASCII`.

`-b <sgml bibliography>`

If the document contains a list of references, you will need to use this option to provide the name of the file containing the bibliography, which must be in the format defined in the `biblo` document type definition. See also `biblio (1)`.

5.3.5 SEE ALSO

`grefer(1), gtbl(1), geqn(1), groff(1), biblio(1)`

5.4 Extracting Source Code

5.4.1 NAME extract — filter the source code out of a "literate program"

5.4.2 SYNOPSIS

```
extract file[.sgml]
```

5.4.3 DESCRIPTION

Extracts the source code out of a literate program written using the `qwertz` SGML document type. The syntax of the document is checked while the source code is being extracted. If syntax errors are found, error messages are written to standard error and the program may terminate before all code has been extracted. The code being extracted is written to standard out, but this will usually be redirected to a file with some appropriate file name extension, depending on the programming language being used. The ".sgml" filename extension of the input file is optional.

For example, if the input file is named `foo.sgml` and contains the source code of a C program, you could extract and compile the program by typing:

```
extract foo > foo.c
cc foo
```

In this example, as standard error has not been redirected, error messages will appear on the terminal.

5.4.4 DIAGNOSTICS

SGML syntax errors are written to standard error. If the input file contains such errors, `extract` will terminate with exit status 1 before all the code has been extracted from the file.

5.5 Converting Bibliographies

5.5.1 NAME biblio — convert SGML bibliographies

5.5.2 SYNOPSIS

```
biblio [-T (bibtex | refer | ps | ascii)] file[.sgml]
```

5.5.3 DESCRIPTION

`biblio` translates a bibliograhy written using the `biblio` SGML document type into BibTeX's `bib` format, Unix's `refer` format, PostScript, and an ASCII representation suitable electronic mail. By default, `biblio` generates `bib` files. Use the `-T` option to select another format. The ".sgml" extenstion of the input file is optional. `biblio` writes to the standard output. To create a `bib` file from a `foo.sgml` file, type:

```
biblio foo > foo.bib
```

To print a formatted version of the bibliography on a PostScript printer, type:

```
biblio -T ps foo | lpr
```

To send a formatted version of the bibliography to someone via email, type, for example:

```
biblio -T ascii foo | mail joe@somewhere
```

5.5.4 SEE ALSO

`bibtex (1)`, `refer (1)`, `mail (1)`, `lpr (1)`, `lookbib (1)`

5.6 Sending Fax Messages

5.6.1 NAME fax — send a Telefax message

5.6.2 SYNOPSIS

```
format file | qtex | fax [-a]
```

5.6.3 DESCRIPTION

`fax` is simple front-end to the `lpr` command, customized for sending Telefax messages. It is designed to be used within a Unix pipe, as shown in the synopsis.

5.6.4 OPTIONS

`-help`

If this is the only option, a short message listing all the options available, and their purpose, is printed to the standard out.

`-a`

Send an acknowledgement, by electronic mail, if the fax message was received without difficulty.

5.6.5 BUGS

The telefax number is written to a file named ".fax=num", in the current directory, by `qtex`. This file is read and then deleted by this `fax` program. Should there coincidentally be a file with this name which is not write protected, it will be removed.

5.6.6 SEE ALSO

`format(1), qtex(1)`

Appendix A

A Sample Article

```
<!doctype qwertz system
[
<!entity TeX "TeX" >
<!entity LaTeX "LaTeX">
<!entity et "&etago;">
<!entity lhead "The <tt/qwertz/ Document Types">
]>

<report opts="times,11pt">

<title>The <tt/qwertz/ SGML Document Types
<subtitle>(Version 1.1 Reference Manual)
<author>Tom Gordon
<inst>The <tt>qwertz</> Project &nl
Institute for Applied Information Technology (F3) &nl&nl
German National Research Center &nl
for Computer Science (GMD)

<chapt>Why Not Just Use &LaTeX;?

 The <tt/qwertz/ document types are a set of Standard
Generalized Markup Language (SGML) document type definitions (DTDs)
for articles, reports, books, letters, notes, slides (or overhead
transparencies), bibliographies, and manual pages.  Except for manual
pages, the document types have been heavily influenced by the
&LaTeX document types of the same names <cite id="Lamport86">, so
&LaTeX users should feel right at home.  Indeed, we presently
translate most <tt/qwertz/ documents into &LaTeX for
printing and the &LaTeX produced is quite readable by anyone
familiar with &LaTeX;.

&hellip;

<chapt>The <tt/qwertz/ Document Type Definition
<header><lhead>&lhead<rhead>The <tt/qwertz/ DTD</header>

 &hellip
```

```
We will be making use of several <em/parameter entities/ in this
DTD:

<code>
<!entity % emph
        " em | it | bf | sf | sl | tt " >

<!entity % inline
        " f | x | %emph; | sq | label | ref |
          pageref | cite | ncite " >
</code>

&hellip

<sect> General Purpose Entities and Elements
<header><lhead>&lhead<rhead>General Purpose Entities</header>

&hellip

When may it be necessary to use of an entity reference to produce some
character?  There are three cases to watch out for:

<descrip>

<tag>SGML Concrete Syntax Delimiters.

Although the SGML standard allows alternative concrete syntaxes to be
defined, we use the so-called <em/reference concrete syntax/ in the
<tt/qwertz/ document types.  &hellip

<tag>SGML Short Reference Delimiters.

In SGML document types <em/short reference maps/ may be defined which
allow single characters to be interpreted as arbitrarily complex
sequences of characters, including SGML tags and entity references.
&hellip

<verb>
" # % ' ( ) * + , - : ; = @ [ ] ^ _ { | } ~
</verb>

For each of these characters, there is an SGML entity which may be
used to generate the ASCII character in the printed document, listed
in table <ref id="GPC">. <em/Usually, it will not be necessary to use these
entities; the character can simply be typed and will be interpreted
literally./ However, &hellip;

<tag>&TeX Special Characters.

Ideally, it should be possible to hide the conventions of the
underlying formatting system completely.  In fact, SGML parsers which
implement the full ISO standard have a feature which makes this
possible. &hellip
```

```
</descrip>

<sect1>Spacing, Dashes and Ellipsis
<header><lhead>&lhead<rhead>Spacing, Dashes and Ellipsis</header>

&hellip

There are also three different kinds of dashes: <tt/hyphen/ which was
already mentioned above, is to be used for intra-word dashes, as in
the word "intra-word".<footnote>However, the <tt/hyphen/ entity was
not actually necessary here, as the - character was not being used in
this context as a short reference.</footnote>

&hellip

<sect1> Foreign Characters
<header><lhead>&lhead<rhead>Foreign Characters</header>

There are a large set of entities for other Western European
languages.  Altogether, there are entities for almost all of the
foreign language characters in ISO 8859, the Latin 1 character set for
Western European languages.<footnote>Only the four
Icelandic characters are missing.</> &hellip

<table>
<tabular ca="ll|ll|ll|ll">
AElig   |       &AElig  |
Aacute  |       &Aacute |
Acirc   |       &Acirc  |
Ae      |       &Ae     @
Ntilde  |       &Ntilde |
Oacute  |       &Oacute |
Ocirc   |       &Ocirc  |
Oe      |       &Oe     @
Ue      |       &Ue     |
Ugrave  |       &Ugrave |
Uuml    |       &Uuml   |
Yacute  |       &Yacute @
aacute  |       &aacute |
acirc   |       &acirc  |
ae      |       &ae     |
aelig   |       &aelig  @
oe      |       &oe     |
ograve  |       &ograve |
oslash  |       &oslash |
otilde  |       &otilde @
sz      |       &sz     |
szlig   |       &szlig  |
thinsp  |               |
tilde   |       &tilde  @
times   |       &times  |
uacute  |       &uacute |
```

```
ucirc   |       &ucirc  |
ue      |       &ue     @
</tabular>
<caption><label id="GPC">(Some) General Purpose Characters</caption>
</table>

 &hellip

<sect1>Sentences, Paragraphs, Footnotes and Emphasis
<header>
<lhead>&lhead
<rhead>Sentences &hellip Emphasis
</header>

&hellip

Sentences or phrases within paragraphs can be emphasized in a number
of ways.  The <tt/em/ tag is used to choose the default form of
emphasis, which is usually <em/italic/ type, but depends on the style
of the background text.  If the background text is formatted in
italics type, as it usually is in definitions, for example, than
emphasized text will be formatted using a plain, roman typeface.
However, various forms of emphasis can be explicitly chosen.  These
include: <bf/bold face/ (<tt/bf/), <it/italics/ (<tt/it/), <sf/sans
serif/ (<tt/sf/), <sl/slanted/ (<tt/sl/), and <tt/typewriter/>
(<tt/tt/) styles.

&hellip

Long quotes are formatted in &LaTeX by indenting the left and
right margins.  For example, <ncite id="Lamport86" note="pp. xiii">:

<lq>

The &LaTeX document preparation system is a special version of
Donald Knuth's &TeX program.  &TeX is a sophisticated program
designed to produce high-quality typesetting, especially for
mathematical text.  &hellip

&LaTeX represents a balance between functionality and ease of use.
Since I implemented most of it myself, there was also a continual
compromise between what I wanted to do and what I could do in a
reasonable amount of time.  &hellip
</lq>

<sect1>Lists
<header><lhead>&lhead<rhead>Lists</header>

Three types of lists are supported, which differ according to the
type of label used to mark each item in the list.  Use <tt/itemize/
to create a list in which each item is marked with some symbol such as
a dash or bullet. The <tt/enum. tag is used to create an
enumeration, i.e. a list in which each item is labelled with a number
```

```
(or letter) indicating its rank or position in the list. Finally, use
<tt/descrip/ to create a list in which each item is labelled by some
tag of your own choice. Lists of various types can nested.  For
example:

&hellip

<itemize>
<item> A level one item.
<item> Here's level two:
        <enum>
        <item> A level two item.
        <item> Here's level three:
                <enum>
                <item> A level three item.
                <item>Here's level four:
                      <descrip>
                      <tag/Red./  Is the color of my true love's hair.
                      <tag/Blue./  Is a property of some movies.
                      <tag/Yellow./  Characterizes some forms of journalism.
                      </descrip>
                <item>A last level three item
                </enum>
        <item>A last level two item
        </enum>
<item>A last level one item.
</itemize>

&hellip

<sect1>Figures and Tables

Encapsulated PostScript graphics can be created using a variety of
different editors.  If you are using Unix with an X11-based graphical
user-interface, you may want to try <tt/idraw/, which stores its
documents directly as Encapsulated PostScript files. Another
interesting X11-based drawing program is <tt/tgif/.

&hellip

Which would then appear as in figure <ref id="issues">.

<figure>
        <eps file="issues">
        <caption><label id="issues">An <tt/idraw/ Drawing </>
</figure>

<sect1>Literate Programming


The original motivation behind the development of these document types
was to create an environment for literate programming in an arbitrary
programming language similar to Donald Knuth's WEB system for literate
```

```
programming in Pascal <cite id="Knuth84">.  &hellip

When formatted, spaces and line breaks are preserved:

<verb>
main ()
{
        /* This is the famous hello world program */

        printf("hello world\n");
}
</verb>

<sect1>Mathematical Formulas

The <tt/qwertz/ document types include elements for describing
mathematical formulas completely within SGML, similar to the system
described in <cite id="daphne89">.  &hellip

So, for example,

<dm>
        <sum><ll>i=1<ul>n<opd>x<inf>i</></sum> =
        <in><ll>0<ul>1<opd>f</in>
</dm>
was typed as:
<verb>
<dm>
        <sum><ll>i=1<ul>n<opd>x<inf>i&et>&et;sum> =
        <in><ll>0<ul>1<opd>f&et;in>
&et;dm>
</verb>

&hellip

<biblio files="lit">
</report>
```

Bibliography

[1] David Barron. Why use SGML? *Electronic Publishing*, 2(1):3–24, April 1980.

[2] Martin Bryan. *SGML, An Author's Guide to the Standard Generalized Markup Language*. Addison-Wesley, 1988.

[3] Donald E. Knuth. Literate Programming. *The Computer Journal*, 27(2):97–111, 1984.

[4] Leslie Lamport. LaTeX*, A Document Preparation System*. Addison-Wesley, 1986.

[5] A. Scheller, C. Smith, C. Fuhrhop, and E. Wilde. DAPHNE: Document Application Processing in a Heterogeneous Network Environment. Technical report, Deutsches Forschungsnetz (DFN), April 1989.

[6] Joan M. Smith and Robert Stutely. *SGML: the user's guide to ISO 8879*. Ellis Horwood, 1988.