Macros

TUGboat Authors' Guide

Ron Whitney and Barbara Beeton

With this article we hope to fill a lacuna (some might say "void") whose existence we have been attributing to the usual factors: tight schedules, alternative priorities and warty TeX code. We now feel the macros in use for *TUGboat* have stabilized to the extent that documentation and suggestions for authors will remain fairly constant, and we hope this article can serve as a reasonable guide to preparation of manuscripts for *TUGboat*. Authors who have used the *TUGboat* macros before will note several changes (including more modern names for the style files). Suggestions and comments are quite welcome at the addresses listed below.

TUGboat was originally typeset with a package based only on `plain`. Later, as demand for style files follows wherever LaTeX-devotees wander, a *TUG*boat variant of the LaTeX `article` style was also created. The two macro sets yield much the same output, differing in certain ways for input. Below we make comment on various aspects of the *TUGboat* package, first for the `plain`-based macros, then for LaTeX. The macro sets share the file `tugboat.com`, and users of either style should read the section entitled "Common Abbreviations and Utilities". We conclude with some general suggestions to help make the lives of those on the receiving end of (any kind of) electronic copy a little easier.

The plain-based macros: tugboat.sty

The macros are contained in two files, `tugboat.sty` and `tugboat.com`.

General description of tags. We attempt wherever possible to tag the various elements of *TUGboat* articles in a "generic" way, modified in some respects by convenience. Authors and editors, of course, need tools to shape their articles to the form they desire, but we also wish to encourage a tagging style which is appropriate for electronic interchange. It seems unfair to expect much thought from authors concerning the markup of their information if we only provide a bag of widgets and do-hickies to hack and pound an article together. The tags whose use we encourage are the higher-level tags that mark the logical document structure. Below these are formatting macros that we recognize may be essential for certain applications. Both sorts of tags are described in the following article.

Generally, to "mark up" the data ⟨*foo*⟩, a tag `\xxx` will precede ⟨*foo*⟩ and `\endxxx` will follow (thus: `\xxx` ⟨*foo*⟩`\endxxx`). We use the `{...}` form to delimit arguments of lower-level formatting macros. Optional commands follow tags and are enclosed in `[...]`, à la LaTeX. Several options may be enclosed within one set of square brackets, or each option may be enclosed in its own set of brackets. These "options" are actually just TeX commands, and it is always possible to insert raw TeX code as an option. Such practice violates truly generic markup, but it is *helpful* and at least confines The Raw and Dirty to a smaller area.

Perhaps a little more detail is of use to some readers here. Upon encountering a tag, the general operational scheme of the macros is as follows:

```
⟨read tag⟩
\begingroup
⟨set defaults⟩
\the\every...
⟨read options⟩
⟨branch to appropriate action,
    using "argument" as necessary⟩
⟨cleanup⟩
\endgroup
```

The scheme shows that code inserted as an option is localized and that it may be used to override certain defaults and to guide branching. Things are not always simple, however. Sometimes parameters are set after a branch is taken (e.g. the macros might only call `\raggedright` after determining whether the mode is "`\inline`" or "`\display`"), and, despite localization, parameter setting might affect the current paragraph if a branch has yet to be taken. This is *not* to say the macros don't work, but rather that those authors who venture beyond the documented regions of the macros should do so with their eyes open.

For convenience, we also allow the `*` as a delimiter for the higher-level tags; thus we could use either

```
\title \TUB\ Authors' Guide \endtitle
```

or

```
\title * \TUB\ Authors' Guide *
```

to indicate the title of this paper. To typeset a * within text delimited by `*`, the `plain` control sequence `\ast` has been extended to give * in text and the usual $\ast$ in math.

This markup scheme may suffer at the hands of TeX's parsing mechanism when tagged data is nested. In these cases, one may group ({...}) embedded data so that TeX knows to proceed to the next \end... or *.

In the cases where we show extra spaces and carriage returns around arguments in this article, those (discretionary) spaces are accommodated in the macros. Thus, for example, when the argument to \title above is typeset, \ignorespaces and \unskip surround it and the extra spaces have no untoward effect. Spaces are also gobbled between options.

Outer form. At the outermost level, a source file will have the form (using the *...* delimiters):

```
\input tugboat.sty

⟨perhaps additional macros for article⟩

\title * ⟨title⟩ *
\author * ⟨author⟩ *
\address * ⟨address⟩ *
\netaddress * ⟨network address⟩ *

\article
  .
  .
  .
⟨body of article⟩
  .
  .
  .
\makesignature
\endarticle
```

Data preceding \article is saved and typeset when \article is encountered. Each author should have his/her own

```
\author ...
\address ...
\netaddress ...
```

block, and the macros will do their best to combine the information properly in the appropriate places. Explicit linebreaks can be achieved within any of these items via \\. Title and authors are, of course, set at the beginning of an article; the address information is listed separately in a "signature" near the end of an article, and is present for the convenience of those who might photocopy excerpts from an issue of *TUGboat*. \makesignature does the typesetting work. Generally authors are listed separately in the signature. In cases where authors and addresses are to be combined, one may use \signature{...} and \signaturemark with some or all of

```
\theauthor {⟨author number⟩}
\theaddress {⟨author number⟩}
\thenetaddress {⟨author number⟩}
```

to get the desired result. For example, for an article with

```
\author * Ray Goucher *
\address * \TUG *
\netaddress *TUG@Math.AMS.com*

\author * Karen Butler *
\address * \TUG *
\netaddress *TUG@Math.AMS.com*
```

we could say

```
\signature {
  \signaturemark
  \theauthor1 and \theauthor2\\
  \theaddress1\\
  \thenetaddress1}
```

to obtain the signature

◇ Ray Goucher and Karen Butler
TeX Users Group
TUG@Math.AMS.com

Use of at least \thenetaddress is recommended for this just so that the network address gets formatted properly. The optional command [\network{...}] will introduce the network address with a network name, so

```
\netaddress[\network{Internet}]
   * TUGboat@Math.AMS.com *
```

produces

Internet: TUGboat@Math.AMS.com

\endarticle marks the end of input and is defined as \vfil\end for most uses. We redefine it as \endinput to assemble streams of articles in *TUGboat*.

Section Heads. Heads of sections, subsections, etc. are introduced with \head, \subhead, etc., respectively. The underlying macros all use \head, so \endhead is the long-form ending for all these tags. For example, the first two heads of this article could have been keyed as

```
\head The \plain-based macros:
   {\tt tugboat.sty} \endhead
```

and

```
\subhead General description of
   tags \endhead
```

In *TUGboat* style, the paragraph following a first-level head is not indented. This is achieved by a look-ahead mechanism which gobbles \pars and calls \noindent. Actually all of the \...head tags gobble pars and spaces after their occurrence. This allows one to enter a blank line in the source file between head and text, such practice being a visual aid to your friendly *TUGboat* editors (if not to you). Be careful of that `\noindent` after a first-level head: you will be in horizontal mode after the `\head *...*`, so spaces which *appear* innocuous, may not be so.

Lists. Lists are everywhere, of course, and a simple list hierarchy can transform a one-dimensional typesetting problem into something nasty. All of which is to say, we are certainly not done with this area of tagging, but here are the available macros.

Not surprisingly, `\list` marks the beginning of a list. A list can be itemized, wherein each item is tagged with `\item`, or unitemized wherein items are delimited by `^^M` (the end of your input line). The itemized style is the default and `[\unitemized]` will get the other. Tags for the items default to the `\bullet` (= •), but can be changed by feeding an argument to `\tag{...}`. The `[\tag{...}]` option used with `\list` assigns the tag for each item of the entire list, while `[\tag{...}]` used with `\item` changes only the tag for that item. The obvious dynamical tags are available with options

`\numbered`	
`\romannumeraled`	
`\lettered`	(lowercase)
`\Lettered`	(uppercase)

Lists can be set in several columns by setting `\cols=...`. The columns are aligned on their top baselines and the user must break the columns with `\colsep`. Thus,

```
\list[\unitemized\numbered][\cols=2]
Fourscore
and seven
years ago
our fathers
\colsep
brought forth
on this
continent
\endlist
```

yields

```
1. Fourscore          5. brought forth
2. and seven          6. on this
3. years ago          7. continent
4. our fathers
```

`\everylist` is a token register which is scanned at the beginning of each list after the default parameters are set and before options are read. If you want all your lists numbered, for example, you might insert

```
\everylist{\numbered}
```

at the top of your file rather than giving an option to each list.

Implementation of sublists is under construction.

Verbatim Modes. There are several variations on this theme. In each case, text is printed in a typewriter font and (almost) all input characters produce the glyph in the font position of their character-code (i.e. you get what you type, no escaping it). In addition to the long form

```
\verbatim...\endverbatim
```

the `|` character can be used to enter and leave verbatim mode, acting as a toggle much as the `$` does with math. `|...|` produces inline verbatim text, while `||...||` displays its output. `\verbatim` itself defaults to display form, but `\verbatim[\inline]` and its contraction `\verbinline` (both terminated by `\endverbatim`) produce the inline form. `^^M` yields a space inline, and a new paragraph in display. Generally, for snippets of text we use the `|...|` form, and for longer items the

```
\verbatim
 .
 .
 .
\endverbatim
```

form (although `||...||` is a good way to display a single line of code).

In addition to formatting text between `\verbatim` and `\endverbatim`, `\verbatim` may read and write data from and to files. We find this variant useful in (*almost*) guaranteeing consonance between macros in use and their published listings.

```
\verbatim[\inputfromfile{foo.inp}]
 .
 .
 .
\endverbatim
```

will incorporate the contents of file `foo.inp` in the listing before the text between `\verbatim` and `\endverbatim`. The shortened form

```
\verbfile{foo.inp}\endverbatim
```

accomplishes the above in the case that the text is empty. While the code around the data, `foo.inp`, above looks excessively long, do remember the implementation uses the basic `\verbatim` macro, so options can also be read after the filename. For example,

```
\verbfile{foo.inp}[\numbered]
\endverbatim
```

would number the lines of the listing.

We often rearrange code supplied to us so that it fits in the narrow measure of *TUGboat*'s two-column format, and we sometimes make corrections to macro sets (you thought you were perfect!). Since errors can (and do — we aren't perfect either) creep in with these modifications, we use the above technique to maintain consistency between the listing published in *TUGboat* and the underlying macros used for examples.

To write out information, use

```
\verbatim[\outputtofile{foo.out}]
  .
  .
  .
\endverbatim
```

An added bonus here is that characters which get internalized as moribund "letters" or "others" in the process of listing them, can return revitalized for perhaps their real use when written out to another file and read in again. The example above involving Ray and Karen was coded as

```
... to get the desired result.  For
example, for an article with
\verbatim[\outputtofile{ray.vbm}]
\author * Ray Goucher *
  .
  .
  .
\endverbatim
we could say
\verbatim[\outputtofile{sig.vbm}]
\signature {
  \signaturemark
  \theauthor1 and \theauthor2\\
  \theaddress1\\
  \thenetaddress1}
\endverbatim
to obtain the signature
\begingroup
\authornumber=0
\input ray.vbm
\input sig.vbm
\makesignature
\endgroup
```

This is perhaps not the most edifying example, but you get the gist. (We localize the process of storing and retrieving these authors and addresses so as not to clobber our own.) We would encourage our authors to use these mechanisms for connecting verbatim text to external files for the sake of maintaining consistency between active code and its documentation.

`\verbatim` scans to `\endverbatim` (a 12-token sequence since the `\` is of type 'other' after `\verbatim` gets going). Only this sequence of characters will interrupt the scan. On the other hand, `|` and `||` scan to the next `|` and `||`, respectively. Needless to say, one should use forms of `\verbatim` to set text which contains `|` (and `|` or `||` to set text containing `\endverbatim` if you are writing an article like this one). Both the `|` and `\verbatim` tags scan ahead for the usual `[` to check for options. In those rare cases when the `[` is really supposed to be the first character of the verbatim text, use the option `[\lastoption]` to stop option parsing. For example, to show

```
[\lastoption]
```

we keyed

```
|[\lastoption][\lastoption]|
```

There are situations where one wants to typeset most things verbatim, but "escape" to format something exceptional. For example, the insertions of metacode given in the listings above require some access to the italic font. By giving the option `[\makeescape\!]` to `\verbatim`, the `!` is made an escape character in that block. Thus,

```
\verbatim[\makeescape\!]
  .
  .
  .
...!it...
  .
  .
  .
\endverbatim
```

really calls the italic font in the middle of the listing (one might also want to use `\makebgroup` and `\makeegroup` in the options to define characters to localize this call; see p. 384). Situations will dictate preferences for what character may be used as an escape (we use the `|`, `!`, and `/` in this article). There is also a means of changing the setup of every occurrence of verbatim mode. The contents of token register `\everyverbatim` is scanned after the defaults of verbatim mode have been set. In this article, for example, we have made `<` active and defined it in such a way that `<...>` typesets as metacode. Since `\verbatim` ordinarily changes `<` to type 'other' on startup, we key

```
\everyverbatim{\enablemetacode}
```

at the beginning of the file to have the proper adjustment made whenever verbatim is started.

When "escaping" within a verbatim block, one should be aware that spaces and carriages returns are *active* and hence not gobbled as usual. Using the `!` as the active character, one might key

```
\verbatim[\makeescape\!]
  .
  .
  .
!vskip .5!baselineskip
  .
  .
  .
\endverbatim
```

to get an extra half line of space in the middle of the listing. The space and carriage return on this line, however, cause problems. The space expands to `\ifvmode\indent\fi\space` and TeX will not like the `\indent` after `\vskip`. The `^^M` expands to `\leavevmode\endgraf`, and therefore puts an extra line into the listing. The solutions, in this case, are to drop the space and to use `!ignoreendline` (which just gobbles the `^^M`), but one should be aware, generally, that some thought may be required in these situations.

The option `[\numbered]` causes the lines of a verbatim listing to be numbered, while `[\ruled]` places rules around the whole thing:

1. ⟨*code*⟩
2. ⟨*more code*⟩
3. ⟨*yet more code*⟩
4. ...

The option `[\continuenumbers]` picks up the numbering where it last left off.

5. ⟨*more*⟩
6. ⟨*and more*⟩
7. ...

The code underlying `\verbatim` in display style implements each line as a paragraph and places math-display-size whitespace above and below the verbatim section. Page and column breaks *are* permitted within these listings. To prohibit breaks at specific points or globally, one must insert penalties or redefine `^^M` to insert `\nobreak` in the vertical list at the end of each "paragraph" (i.e. line). We should also note that the bottom of such a verbatim listing is implemented so that ensuing text may or may not start a new paragraph depending on whether an intervening blank line (or `\par`) is or is not present.

Figures and Page Layout. Figures are keyed as

`\figure`

⟨*vertical mode material*⟩

`\endfigure`

These are generally implemented as single-column floating top-insertions, but the options `[\mid]` and `[\bot]` can change specific items to be mid- or bottom-insertions, respectively. Here we recommend that the long-form terminator be used (*not* the `*...*` form). One can think of the information "passed" as being "long" in the sense of possibly containing paragraphs, this being a mnemonic device only. The primary reason for the recommendation is that one is (in some sense, maybe) more likely to encounter a rogue `*` in longer text than in shorter text and hence more likely to encounter a surprising result due to a macro stopping short at the wrong `*`.

Perhaps here is a natural place to mention also that these macros sometimes read their arguments and then act, and sometimes act on the fly, not actually storing an argument as a string of tokens at all. `\title`, for example, is in the former category, while `\figure` is in the latter. Reasons may vary for the choice in methods. Storing a string of tokens as an argument does not allow re-interpretation of the category codes of the underlying character string. Thus, storing the "argument" of `\figure` all at once might misinterpret some characters which should appear as verbatim text. For this reason we set figures as we go and just close off the box with `\endfigure`. On the other hand, using information in multiple situations (e.g. titles and running heads) requires storing the information as a token string, not as a typeset list.

When text delimited by `*...*` is read as an argument, the `*`s are dropped by the parsing process. When the text is handled on the fly, the first `*` is gobbled and the second is made active to perform whatever action is necessary at the close of the macro. When possible, we prefer to operate on the fly since nested tags are handled properly in that case and no memory is consumed to store arguments. Examination of `tugboat.sty` will show which case applies in a given situation, but this general knowledge may help when trying to debug those situations in which an unexpected `*` has disrupted things.

A primitive `\caption{...}` option is available to `\ulap` (i.e. lap upward) its argument into the figure space, but formatting of the caption is left to the user. For example, the code:

```
\figure[\top]
  [\caption{\centerline{Odd Fig.~1}}]
\vbox to 5pc{}
\endfigure
```

produces the figure at the top of this column or the next.

Figures spanning columns at the top and bottom of a page are currently supported only on the first page of an article, but we expect they will soon be allowed on any page (a general rewrite of

Odd Fig. 1

the output routine is in progress). `\twocolfigure` (terminated by `\endfigure`) starts up such a figure and currently *must* occur before any material has been typeset on the first page (i.e. *before* `\article`).

Macros `\onecol`, `\twocol`, and `\threecol` provide one-, two-, and three-column layouts, but these cannot currently be intermixed on a page. We hope to provide automatic column-balancing and convenient switching between one- and two-column format within a year. `\newpage` in each format is defined to fill and eject enough columns to get to the next page. `\newcol` is just `\par\vfill\eject`.

Command List Summary. Tags are listed in the order discussed. Options are listed under tags.

```
\title
\author
\address
\netaddress
   \network
\signature
\article
\makesignature
\endarticle
\head
\subhead
\subsubhead
\list
   \numbered
   \romannumeraled
   \lettered
   \Lettered
   \ruled
   \tag{...}
\item
   \tag{...}
\everylist
\verbatim
   \numbered
   \ruled
   \inputfromfile{...}
   \outputtofile{...}
\verbinline
\verbfile
\figure
   \mid
   \bot
   \caption{...}
\twocolfigure
```

and, of course, `|` and `||`.

The LaTeX macros: `ltugboat.sty`

`ltugboat.sty` is the primary macro file, `tugboat.com` a collection of items common to both LaTeX and `plain` input. Articles will have the form:

```
\documentstyle[ltugboat]{article}

⟨perhaps additional macros for article⟩

\title {⟨title⟩}
\author{⟨author⟩}
\address{⟨address⟩}
\netaddress{⟨netaddress⟩}

\begin{document}
\maketitle
 .
 .
 .
⟨body of article⟩
 .
 .
 .
\makesignature
\end{document}
```

This is the usual form for LaTeX documents, of course, except that now each author will have his/her own

```
\author{...}
\address{...}
\netaddress{...}
```

block. As with the `plain` style, the author and address macros will store their information for later display. See the discussion of `\address`, `\netaddress` and `\makesignature` on page 379 to understand more. Linebreaks within `\title`, `\author`, and `\...address` are specified with `\\`.

We refer the user to the LaTeX manual for description of section heads, verbatim mode, insertions, and movement between one- and two-column format. The style of printed output has, of course, been somewhat modified to fit *TUGboat* style. `ltugboat.sty` might be of some use to others wishing to modify the `article` option in this direction.

Common Abbreviations and Utilities

Definitions of a number of commonly used abbreviations such as `\MF` and `\BibTeX` are contained in `tugboat.com`. Please use these whenever possible rather than creating your own. We will add to the list as necessary.

Several other constructions that we have found useful for both plain- and LaTeX-style input have been incorporated in `tugboat.com`. Various `\*laps` (`\ulap`, `\dlap`, `\xlap`, `\ylap`, `\zlap`) and `\*smash`es provide means of setting type which "laps" into neighboring regions. `\dash` and `\Dash` are en- and em-dashes that break properly. `\slash` is a breakable slash. The macro

`\makestrut [`⟨*ascender dimen*⟩`;`
⟨*descender dimen*⟩`]`

allows *ad hoc* construction of struts.

`\makeatletter \catcode`s the `@` for internal control-sequences. There are also more general functions

```
\makeescape
\makebgroup
\makeegroup
\makeletter
\makeother
\makeactive
```

that change the category of a given character into the type mentioned at the end of the macro name. For example, `\makeactive\!` changes the category of the `!` to 13. We have given many other examples of these in this article. Readers may look at the end of `tugboat.com` after the `\endinput` statement to see further documentation on the contents of the file.

Issue Makeup. Constructing an entire issue of *TUGboat* requires use of a few features that authors may notice when articles are returned for proofing. `\xrefto` allows for symbolic cross-referencing, but is enabled only late in the production process. The distribution version of `tugboat.com` defines `\xrefto` so that "???" is typeset whenever it is called. Not to worry.

We also put notes into the file since there are many things to remember, and these appear as `\TBremark{...}`. Authors can look for such things, if they are interested.

General Coding Suggestions

Probably 90% of the code we receive is easily handled, and for this we are most appreciative. We do have suggestions of a general nature that authors should keep in mind as they create articles for transmission here or anywhere else.

Those who create code find it much easier to read and understand their own code than do others who read the "finished" product. In fact, some people seem to forget that the electronic file will be viewed (in fact, studied) in addition to the printed copy. Documentation and uniform habits of presentation always help. Blank lines are easier to digest by eye than `\par`s. Tables and display math can often be keyed in such a way that rows and columns are clear in the source file on a display screen as well as in print. Explanations or warnings of tricky code can be *very* helpful. Authors should place font and macro definitions in one location at the beginning of an article whenever possible.

Authors should anticipate that articles will undergo some transformation, and that positioning of some elements may change simply because articles are *run together* in *TUGboat*. Decisions on line-breaks, pagebreaks, figure and table placement are generally made after the text is deemed correct. We avoid inserting "hard" line- and page-breaks whenever possible, and will not do so, in any case, until the last minute. We also use floating insertions for figure and table placement when we first receive an article. It is easier for us to work with a clean file containing some bad breaks, overfull boxes or other unsightliness, than it is to handle a document containing *ad hoc* code dedicated to a beauteous (albeit narrowly specific) result.

When authors proof their articles in formats other than that of *TUGboat* (for example), they should expect that *TUGboat*'s changes in pagewidth and pagedepth may drastically alter text layout. Paragraphs are rebroken automatically when `\hsize` and `\vsize` change, but `\centerline` does not break, and we often see tables and math displays which are rigidly laid out. When possible, authors might use paragraphing techniques instead of calls to, say, `\centerline` (Beeton will be writing up her lectures on paragraphing techniques for a future issue of *TUGboat*), and they should try to code tables in such a way that widths of columns can be changed easily. Generally, authors should attempt to anticipate the work that might be necessary if requests for other reasonable formats of their texts are made. In the case of *TUGboat*, we make a strong effort to stuff macro listings and tables into the two-column format. Since these types of items are not generally susceptible to automatic line-breaking, we give thanks to stuffings made by authors ahead of time. In this context, we recommend the use of `\verbfile{...}` (see p. 380) to maintain consistency between documentation and reality.

Specifically in the domain of TeX macros, we find that many authors throw in unnecessary `%` characters to end code lines. Except in cases where the `^^M` means something other than end-of-line, linebreaks can reliably be placed after control-words and numerical assignments. We have seen TEX's buffer size exceeded when `%` was placed after *every* line.

A wider perspective in the matter of naming macros can prevent problems that occur when definitions are overwritten as articles are run together. The names of control sequences used in `plain`, LATEX, and AMS-TEX are documented and authors should avoid using them for other purposes. It is also wise to avoid commonly used names such as `\temp`, `\result`, `\1`, and `\mac` in presenting code that might be cribbed by other users. The frequently used technique of temporarily `\catcode`ing a character to be a letter (e.g. the `@`) provides a good method of hiding control sequences so that they will not be clobbered later. Readers are in need of small macros to do little tricks, and they often try suggestions brought forth in *TUGboat*. A little extra effort in making these macros consistent with the macros in wide distribution and in making them robust will be much appreciated.

Electronic Documentation and Submission Procedure

In addition to `tugboat.sty`, `ltugboat.sty`, and `tugboat.com`, a copy of this article, `tubguide.tex`, will be available at most TEX archives, including those at Clarkson and Aston.

Please address all electronic correspondence to the *TUGboat* maildrop:

`TUGboat@Math.AMS.com`

Mail to either of our personal addresses is liable to go unseen if vacation or illness intervenes. We also request that you supply an abstract of any expository article. This will be used as the basis for translation of abstracts to languages other than that in which the article is published.

◇ Ron Whitney
TEX Users Group
P. O. Box 9506
Providence, RI 02940-9506
`TUGboat@Math.AMS.com`

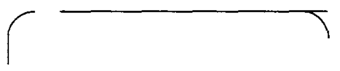
◇ Barbara Beeton
American Mathematical Society
P. O. Box 6248
Providence, RI 02940-6248
`TUGboat@Math.AMS.com`

Round boxes for plain TEX

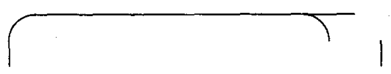
Garry Glendown

Doing presentation sheets, I stumbled over a small thing I had been missing for quite a while: boxes. Well, normal boxes are boring, so I thought about doing boxes with round corners.

To do that, I took a look at the circle fonts used for the LATEX pictures. They would work out fine. But, despite of all my TEX knowledge and the information from *The TEXbook*, it didn't work. Either the boxes would look like this:



or like this:



or some other, not very encouraging, way. After some hours (I think it was about 2 1/2 or so) I finally solved the problem as found in the listing below.

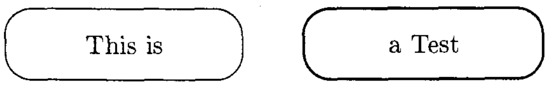
The problem is the strange (at least for normal usage) way the circle font has the width and reference point set. The width is exactly twice as big as the quarter circle, and the reference point of the right two quarters is far beyond the character. So, in order to get the right positioning of the characters, the boxes have to be much wider in the inside than they are on the outside.

Using `RBox`. To use the `RBox`-Macro, there are two simple forms: `\roundBox` and `\RoundBox`. Both get two parameters: the size of the box as a percent of `\hsize`, and the text. When calling `\roundBox`, you will get a box with a border .4pt thick; `\RoundBox` will result in one with a .8pt border.

If you type

```
\hbox{%
  \roundBox{.4}{This is}%
  \RoundBox{.4}{a Test}%
  }
```

it will result in:



In addition to these to 'interface'-macros, you may use the internal function called `\RBox`. The syntax is the following: