blog.sty

—–

Generating HTML Quickly with TEX*

Uwe Lück[†]

January 22, 2013

Abstract

blog.sty provides TEX macros for generating web pages, based on processing text files using the fifinddo package. Some LATEX commands are redefined to access their HTML equivalents, other new macro names "quote" the names of HTML elements. The package has evolved in several little steps each aiming at getting pretty-looking "hypertext" **notes** with little effort, where "little effort" also has meant avoiding studying documentation of similar packages already existing. [TODO: list them!] The package *"misuses"* TEX's macro language for generating HTML code and entirely *ignores* TEX's typesetting capabilities.—lnavicol.sty adds a more **professional** look (towards CMS?), and blogdot.sty uses blog.sty for HTML **beamer** presentations.

Contents

---

*This document describes version v0.81a of blog.sty as of 2013/01/21.

[†] `http://contact-ednotes.sty.de.vu`

1 Installing and Usage

The file blog.sty is provided ready, **installation** only requires putting it somewhere where TEX finds it (which may need updating the filename data base).[1]

User commands are described near their implementation below.

However, we must present an **outline** of the procedure for generating HTML files:

At least one **driver** file and one **source** file are needed.

The **driver** file's name is stored in `\jobname`. It loads blog.sty by

```
\RequirePackage{blog}
```

and uses file handling commands from blog.sty and fifinddo (cf. `mdoccheat.pdf` from the nicetext bundle).[2] It chooses **source** files and the name(s) for the resulting HTML file(s). It may also need to load local settings, such as `\uselangcode` with the langcode[3] package and settings for converting the editor's text encoding into the encoding that the head of the resulting HTML file advertises—or into HTML named entities (for me, `atari_ht.fdf` has done this).

The driver file could be run a terminal dialogue in order to choose source and target files and settings. So far, I rather have programmed a dialogue just for converting UTF-8 into an encoding that my Atari editor xEDIT can deal with. I do not present this now because it was conceptually mistaken, I must set up this conversion from scratch some time.

The **source** file(s) should contain user commands defined below to generate the necessary `<head>` section and the `<body>` tags.

2 Examples

2.1 Hello World!

This is the **source** code for a "Hello World" example, in `hellowor.tex`:

```
\ProvidesFile{hellowor.tex}[2012/11/30 hello world source]
\head
\title{Hello world!}
\body
Hello [[world]]!
\finish
```

---

[1] http://www.tex.ac.uk/cgi-bin/texfaq2html?label=inst-wlcf

[2] http://www.ctan.org/pkg/nicetext

[3] http://www.ctan.org/pkg/langcode

The HTML file `hellowor.htm` is generated from `hellowor.tex` by the following **driver** file `mkhellow.tex`:

```
\ProvidesFile{mkhellow.tex}[2012/11/30 blog demo]
\RequirePackage[ligs,mark]{blog}      %% general HTML generation
\BlogInterceptEnvironments*           %% ... using blogexec.sty
\UseBlogLigs                          %% smart markup
\RequirePackage{texlinks}             %% basic link shorthands
\RequirePackage{langcode}             %% \uselangcode...
\RequirePackage{catchdq}              %% " typographically
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% \input{jobname}                       %% call by "echo"
\newcommand{\htmljob}                 %% choose filename base
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
{hellowor}                            %% "Hello world!"
% {hallow} \uselangcode{de}             %% "Hallo Welt!"
% {markblog}                            %% easy syntax overview
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\ResultFile{\htmljob.htm}
\BlogProcessFinalFile[%\TextCodes      %% encoding settings
                       \catchdqs]     %% " typographically
                      {\htmljob.tex}
\stop                                 %% stop LaTeX run
```

2.2 A Style with a Navigation Column

A style of web pages looking more professional (while perhaps becoming outdated) has a small navigation column on the left, side by side with a column for the main content. Both columns are spanned by a header section above and a footer section below. The package lnavicol.sty provides commands `\PAGEHEAD`, `\PAGENAVI`, `\PAGEMAIN`, `\PAGEFOOT`, `\PAGEEND` (and some more) for structuring the source so that the code following `\PAGEHEAD` generates the header, the code following `\PAGENAVI` forms the content of the navigation column, etc. Its code is presented in Sec. 6. For real professionality, somebody must add some fine CSS, and the macros mentioned may need to be redefined to use the `@class` attribute. Also, I am not sure about the table macros in blog.sty, so much may change later.

With things like these, can blog.sty become a part of a "content management system" for TEX addicts? This idea rather is based on the *German* Wikipedia article.

As an example, I present parts of the source for my "home page"[4]. As the footer is the same on all pages of this style, it is added in the driver file

---

[4] `www.webdesign-bu.de/uwe_lueck/schreibt.html`

`makehtml.tex`. `schreibt.tex` is the source file for generating `schreibt.html`. You should find *this* `makehtml.tex`, a cut down version of `schreibt.tex`, and `writings.fdf` with my extra macros for these pages in a directory `blogdemo/writings`, hopefully useful as templates.

2.2.1 Driver File `makehtml.tex`

```
\def \GenDate {2012/08/02}          %%% {2012/06/07} {2011/11/01}
\ProvidesFile{makehtml.tex}
             [\GenDate\space TeX engine for "writings"]
%% reworked 2012/03/13:
\RequirePackage[autopars]{blog}[2011/11/20]    %% auto 2012/08/02
\BlogInterceptEnvironments*
\RequirePackage{texlinks,lnavicol}
\input{atari_ht.fdf}      %% 2012/06/07
\input{writings.fdf}
\NoBlogLigs               %% 2012/03/14 TODO remove HTML comments
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\input{jobname}
% \def \htmljob
% {_sitemap}
% {index}                         \BlogAutoPars
% {schreibt}  \uselangcode{de}    \BlogAutoPars %% mod. 2012/02/04
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% {about}                         \BlogAutoPars
% {contact}                                       % \tighttrue
% {kontakt}   \uselangcode{de}                    % \tighttrue
% {tutor}     \uselangcode{de}    \BlogAutoPars   \deeptrue
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% {writings}                      \BlogAutoPars   \deeptrue
% {repres}                        \BlogAutoPars   \deeptrue
% {critedl}                       \BlogAutoPars   \deeptrue
% {ednworks}                      \BlogAutoPars
% {public}                        \BlogAutoPars   \deeptrue
% {texproj}                       \BlogAutoPars   % \deeptrue
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\ResultFile{\htmljob\htmakeext}
\WriteResult\writdoctype                    %% TODO
% \BlogCopyFile[\TextCodes                  %% \BlogIntercept:
\BlogProcessFile[\TextCodes                 %% 2012/03/13
             \MakeActiveDef\"{\catchdq}%    %% TODO attributes!?
             ]{\htmljob.tex}
\WriteResult{\PAGEFOOT}
\WriteResult{\indentii\rainermaster}
\WriteResult{\indentii\\}
```

```
\WriteResult{\indentii\ueberseeport}        %% TODO BlogLigs!?
\WriteResult{\PAGEEND}
\ifdeep \WriteResult{\indenti\vspace{280}} \fi
\WriteResult{\finish}
\CloseResultFile
\stop
```

2.2.2 Source File `schreibt.tex`

```
\ProvidesFile{schreibt.tex}[2011/08/19 f. schreibt.html]
\head \charset{ISO-8859-1}
  \writrobots
  \writstylesheets
\title{\Uwe\ schreibt} \body \writtopofpage
\PAGEHEAD
  \headuseskiptitle{%
    \timecontimgref{writings}{0}{Zeit-Logo}{Russells Zeit}%
  }{10}{\Uwe\ \dqtd{schreibt}}
\PAGENAVI
  \fileitem{writings}{Intervallordnungen (Mathematik~etc.)}
  \fileitem{public}{Publikationen}
 \hrule
  \fileitem{critedltx}{Softwarepakete f\"ur kritische Editionen}
  \fileitem{texproj}{TeX-Projekte} %%% Makro-Projekte}
 \hrule
  \fileitem{tutor}{Mathe-Tutor}
  \indentii\item\href{texmap.htm}{Notizen}
 \hrule
  \deFIabout \deFIkontakt
\PAGEMAIN
\strong{Wissenschaft:}\enspace Diese Seiten entstanden zuerst
zur Präsentation zweier ETC.

\rightpar{\textit{Worms-Pfeddersheim, den 19.~August 2011,\\\Uwe}}
% \rightpar{\textit{München, den 31.~Juli 2011,\\\Uwe}}
  %% <- TODO VERSION
```

3 The File `blog.sty`

3.1 Preliminaries

3.1.1 Package File Header (Legalese)

```
\ProvidesPackage{blog}[2013/01/21 v0.81a simple fast HTML (UL)]
%% copyright (C) 2010 2011 2012 2013 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below.
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3c of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
%%   http://www.contact-ednotes.sty.de.vu
%%
```

3.1.2 `\newlet`

`\newlet`⟨*cmd*⟩⟨*cnd*⟩ is also useful in surrounding files:

```
\newcommand*{\newlet}[2]{\@ifdefinable#1{\let#1#2}}
```

3.2 Processing

3.2.1 Requirement

We are building on the fifinddo package (using `\protected@edef` for Sec. 3.2.7):

```
\RequirePackage{fifinddo}[2011/11/21]
```

3.2.2 Output File Names

`\htmakeext` is the extension of the generated file. Typically it should be `.html`, as set here, but my Atari emulator needs `.htm` (see `texblog.fdf`):

```
\newcommand*{\htmakeext}{.html}
```

3.2.3 General Insertions

`\CLBrk` is a *code line break* (also saving subsequent comment mark in macro definitions):

```
\newcommand*{\CLBrk}{^^J}
```

`\␣` is turned into an alias for `\space`, so it inserts a blank space. It even works at line ends, thanks to the choice of `\endlinechar` in Sec. 3.2.4.

```
\let\ \space
```

`\ProvidesFile{⟨file-name⟩.tex}[⟨file-info⟩]` is supported for use with the myfilist package to get a list of source file infos. In generating the HTML file, the file infos are transformed into an HTML comment. Actually it is `\BlogProvidesFile` (for the time being, 2011/02/22):

```
\@ifdefinable\BlogProvidesFile{%
    \def\BlogProvidesFile#1[#2]{%
        <!DOCTYPE html>\CLBrk               %% TODO more!? 2012/09/06
        \comment{ generated from\CLBrk\CLBrk
                  \ \ \ \ \ \ \ \ \ #1, #2,\CLBrk\CLBrk
                  \ \ \ \ \ with blog.sty,
                  \isotoday\ }}}
\edef\isotoday{%% texblog 2011/11/02, here 2011/11/20
   \the\year-\two@digits{\the\month}-\two@digits{\the\day}}
```

(TODO: customizable style.)—Due to the limitations of the approach reading the source file line by line, the "optional argument" `[⟨file-info⟩]` of `\ProvidesFile` must appear in the same line as the closing brace of its mandatory argument. The feature may require inserting

```
\let\ProvidesFile\BlogProvidesFile
```

somewhere, e.g., in `\BlogProcessFile`.

3.2.4 Category Codes etc.

For a while, line endings swallowed inter-word spaces, until I found the setting of `\endlinechar` (fifinddo's default is `-1`) in `\BlogCodes`:

```
\newcommand*{\BlogCodes}{%                              %% 2010/09/07
    \endlinechar`\ %
```

← Comment character to get space rather than `^^M`!—The tilde `~` is active as in Plain TeX too, it is so natural to use it for abbreviating HTML's `&nbsp;`!

```
%     \catcode`\~\active
    \MakeActiveDef\~{&nbsp;}%%          for \FDpseudoTilde 2012/01/07
```

'`'`' for HTML convenience (cf. Sec. 3.9.8):

```
    \MakeActiveLet\'\rq                         %% actcodes 2012/08/28
    \BasicNormalCatCodes}
% \MakeOther\< \MakeOther\>                         %% rm. 2011/11/20
```

3.2.5 The Processing Loop

`\BlogProcessFile[⟨changes⟩]{⟨source-file⟩}`

“copies” the TEX source file ⟨*source-file*⟩ into the file specified by `\ResultFile`.

```
\newcommand*{\BlogProcessFile}[2][]{%                   %% 2011/11/05
    \ProcessFileWith[\BlogCodes
                     \let\ProvidesFile\BlogProvidesFile %% 2011/02/24
                     \let\protect\@empty                %% 2011/03/24
                     \let\@typeset@protect\@empty       %% 2012/03/17
                     #1]{#2}{%
        \IfFDinputEmpty
            {\IfFDpreviousInputEmpty
                \relax
                {\WriteResult{\ifBlogAutoPars<p>\fi}}}%
            \BlogProcessLine                            %% 2011/11/05
    }%
}
```

fifinddo v0.5 allows the following

`\BlogProcessFinalFile[⟨changes⟩]{⟨source-file⟩}`

working just like `\BlogProcessFile` except that the final `\CloseResultFile` is issued automatically, no more need having it in the driver file.

```
\newcommand*{\BlogProcessFinalFile}{%
    \FinalInputFiletrue\BlogProcessFile}
```

TODO: optionally include `.css` code with `<style>`.

3.2.6 *Executing* Source File Code Optionally

For v0.7, `\BlogCopyFile` is renamed `\BlogProcessFile`; and in its code, `\CopyLine` is replaced by `\BlogProcessLine`. The purpose of this is supporting blogexec.sty that allows intercepting certain commands in the line. We provide initial versions of blogexec’s switching commands that allow invoking blogexec “on the fly”:

```
\newcommand*{\ProvideBlogExec}{\RequirePackage{blogexec}}
```

dowith.sty is used in the present package to reduce package code and documentation space:

```
\RequirePackage{dowith}
\setdo{\providecommand*#1{\ProvideBlogExec#1}}
\DoDoWithAllOf{\BlogInterceptExecute \BlogInterceptEnvironments
                    \BlogInterceptExtra   \BlogInterceptHash        }
```

`\BlogCopyLines` switches to the “copy only” (“compressing” empty lines) functionality of the original `\BlogCopyFile`:

```
\newcommand*{\BlogCopyLines}{%
%     \let\BlogProcessLine\CopyLine}
    \def\BlogProcessLine{%          %% 2011/11/21, corr. 2012/03/14:
        \WriteResult{\ProcessInputWith\BlogOutputJob}}}
```

← This is a preliminary support for "ligatures"—see Sec. 3.2.7. `\NoBlogLigs` sets the default to mere copying:

```
\newcommand*{\NoBlogLigs}{\def\BlogOutputJob{LEAVE}}
\NoBlogLigs
```

TODO more from `texblog.fdf` here, problems with `writings.fdf`, see its `makehtml.tex`

`\BlogCopyLines` will be the setting with pure blog.sty:

```
\BlogCopyLines
```

OK, let's not remove `\BlogCopyFile` altogether, rebirth:

```
\newcommand*{\BlogCopyFile}{\BlogCopyLines\BlogProcessFile}
```

3.2.7 "Ligatures", Package Options

With v0.7, we introduce a preliminary method to use the "ligatures" `--` and `---` with pure expansion. At this occasion, we also can support the notation `...` for `\dots`, as well as arrows (as in `mdoccorr.cfg`). Note that this is somewhat **dangerous**, especially the source must not contain "explicit" HTML comment, comments must use blog.sty's `\comment` or the `{commentlines}` environment. Therefore these "ligatures" must be activated explicitly by `\UseBlogLigs`:

```
\newcommand*{\UseBlogLigs}{\def\BlogOutputJob{BlogLIGs}}
```

In order to work inside braces, the source file better should be preprocessed in "plain text mode." (TODO: Use `\ifBlogLigs`, and in a group use `\ResultFile` for an intermediate `\htmljob.lig`. And TODO: Use `\let\BlogOutputJob`.) On the other hand, the present approach allows switching while processing with `\EXECUTE`! Also, intercepted commands could apply the replacements on their arguments—using `\ParseLigs{`⟨*arg*⟩`}`:

```
\newcommand*{\ParseLigs}[1]{\ProcessStringWith{#1}{BlogLIGs}}
```

(`\ProcessStringWith` is from fifinddo.)—The package blogligs.sty described in Sec. 4 does these things in a more powerful way. You can load it by calling blog.sty's package option `[ligs]` (v0.8):

```
\DeclareOption{ligs}{\AtEndOfPackage{\RequirePackage{blogligs}}}
```

The replacement chain follows (TODO move to `.cfg`). As opposed to the file `mdoccorr.cfg` for makedoc.sty, we are dealing with "normal TeX" code (regarding category codes, fifinddo.sty as of 2011/11/21 is needed for `\protect`). Moreover, space tokens after patterns are already there and need not be inserted after control sequences.

```
\FDpseudoTilde
\StartPrependingChain
\PrependExpandableAllReplacer{blog...}{...}{\protect\dots}
\PrependExpandableAllReplacer{blog--}{--}{\protect\endash}
\PrependExpandableAllReplacer{blog---}{---}{\protect\emdash}
```

← Cf. thin surrounding spaces with `\enpardash` (texblog, maybe *hair space* U+200A instead of thin space), difficult at code line beginnings or endings and when a paragraph starts with an emdash. I.e., perhaps better don't use it if you want to have such spaces.—'`---`' must be replaced before '`--`'!

```
\PrependExpandableAllReplacer{blog->}{->}{\protect\to}
\PrependExpandableAllReplacer{blog<-}{<-}{\protect\gets}
```

You also could set `\BlogOutputJob` to a later part of the chain, or more globally change the following:

```
\CopyFDconditionFromTo{blog<-}{BlogLIGs}
```

The package markblog.sty described in Sec. 5 extends this to some markup resembling wiki editing. This package may be loaded by blog.sty's package option `[mark]` (v0.8):

```
\DeclareOption{mark}{\AtEndOfPackage{\RequirePackage{markblog}}}
```

3.2.8 <p> from Empty Line, Package Option

As in TEX an empty line starts a new paragraph, we might "interpret" an empty source line as HTML tag `<p>` for starting a new paragraph. Empty source lines following some first empty source line immediately are ignored ("compression" of empty lines). However, this sometimes has unwanted effects (comment lines TODO), so it must be required explicitly by `\BlogAutoPars`, or by calling the package with option `[autopars]`. In the latter case, it can be turned off by `\noBlogAutoPars`

```
\newif\ifBlogAutoPars
\newcommand*{\BlogAutoPars}{\BlogAutoParstrue}
\newcommand*{\noBlogAutoPars}{\BlogAutoParsfalse}
```

`\BlogAutoPars` is issued by package option `[autopars]`:

```
\DeclareOption{autopars}{\BlogAutoPars}
\ProcessOptions
```

See Sec. 3.4 for other ways of breaking paragraphs.

3.3 General HTML Matters

The following stuff is required for any web page (or hardly evitable).

3.3.1 General Tagging

`\TagSurr{`⟨*elt-name*⟩`}{`⟨*attr*⟩`}{`⟨*content*⟩`}`

(I hoped this way code would be more readable than with `\TagSurround` …) and

`\SimpleTagSurr{`⟨*elt-name*⟩`}{`⟨*content*⟩`}`

are used to avoid repeating element names ⟨*elt-name*⟩ in definitions of TEX macros that refer to "entire" elements—as opposed to elements whose content often spans lines (as readable HTML code). We will handle the latter kind of elements using LATEX's idea of "environments." `\TagSurr` also inserts specifications of element **attributes**, [TODO: wiki.sty syntax would be so nice here] while `\SimpleTagSurr` is for elements used without specifying attributes. `\STS` is an abbreviation for `\SimpleTagSurr` that is useful as the `\SimpleTagSurr` function occurs so frequently:

```
\newcommand*{\SimpleTagSurr}[2]{<#1>#2</#1>}
\newlet\STS\SimpleTagSurr                             %% 2010/05/23
```

With the space in `\declareHTMLattrib` as of 2012/08/28, we remove the space between `#1` and `#2`. (Doing this by an option may be better TODO; any separate attribute definitions must take care of this.)

```
% \newcommand*{\TagSurr}[3]{<#1#2>#3</#1>}
```

… undone 2012/11/16, bad with "direct" use of `#2` (with attributes not declared):

```
\newcommand*{\TagSurr}[3]{<#1 #2>#3</#1>}
```

3.3.2 Attributes

Inspired by the common way to use `@` for referring to element attributes—i.e., `@`⟨*attr*⟩ refers to attribute ⟨*attr*⟩—in HTML/XML documentation, we often use

`\@`⟨*attr*⟩`{`⟨*value*⟩`}` to "abbreviate" ⟨*attr*⟩`="`⟨*value*⟩`"`

within the starting tag of an HTML element. This does not really make typing easier or improve readability, it rather saves TEX's memory by using a single token for referring to an attribute. This "abbreviation" is declared by `\declareHTMLattrib{`⟨*attr*⟩`}`, even with a check whether `\@`⟨*attr*⟩ has been defined before:

```
\newcommand*{\declareHTMLattrib}[1]{%
  \def\reserved@a{@#1}%
  \@ifundefined\reserved@a                   %% \res... 2012/09/06
               {\@namedef{@#1}##1{ #1="##1"}}%% space   2012/08/28
                \@notdefinable}
```

So after `\declareHTMLattrib{`⟨*attr*⟩`}`, `\@`⟨*attr*⟩ is a TeX macro expecting one parameter for the specification.

A few frequent attributes are declared this way here. `@class`, `@id`, `@style`, `@title`, `@lang`, and `@dir` are the ones named on *Wikipedia*:

```
\let\@class\relax     %% for tab/arr in latex.ltx
\let\@title\relax     %% for \title in latex.ltx, %% 2011/04/26
\DoWithAllOf\declareHTMLattrib{{class}{id}{style}{title}{lang}{dir}}
```

`@type` is quite frequent too:

```
\declareHTMLattrib{type}
```

`@href` is most important for that “hyper-text:”

```
\declareHTMLattrib{href}
```

. . . and `@name` (among other uses) is needed for hyper-text anchors:

```
\declareHTMLattrib{name}                        %% 2010/11/06
```

`@content` appears with `\MetaTag` below:

```
\declareHTMLattrib{content}
```

`@bgcolor` is used in tables as well as for the appearance of the entire page:

```
\declareHTMLattrib{bgcolor}
```

Of course, conflicts may occur, as the form `\@`⟨*ASCII-chars*⟩ of macro names is used for internal (La)TeX macros. Indeed, `\@width` that we want to have for the `@width` attribute already “abbreviates” TeX’s “keyword” (TeXbook p. 61) `width` in LaTeX (for specifying the width of a `\hrule` or `\vrule` from TeX; again just saving TeX tokens rather than for readibility).

```
\PackageWarning{blog}{Redefining \protect\@width}
\let\@width\relax
\declareHTMLattrib{width}
```

Same with `@height`:

```
\PackageWarning{blog}{Redefining \protect\@height}
\let\@height\relax
\declareHTMLattrib{height}              %% 2010/07/24
```

We can enumerate the specifications allowed for `@align`:

```
\newcommand*{\@align@c}{\@align{center}}
\newcommand*{\@align@l}{\@align{left}}
\newcommand*{\@align@r}{\@align{right}}
% \newcommand*{\@align}[1]{ align="#1"}
\declareHTMLattrib{align}               %% 2012/09/08
```

`@valign@t`:

```
% \newcommand*{\@valign@t}{v\@align{top}} %% 2011/04/24
\newcommand*{\@valign@t}{ valign="top"} %% 2012/09/08
```

Some other uses of `\declareHTMLattrib` essential for *tables:*

```
\declareHTMLattrib{border}              %% 2011/04/24
\declareHTMLattrib{cellpadding}         %% 2010/07/18
\declareHTMLattrib{cellspacing}         %% 2010/07/18
\declareHTMLattrib{colspan}             %% 2010/07/17
\declareHTMLattrib{frame}               %% 2010/07/24
```

Another problem with this namespace idea is that *either* this reference to attributes cannot be used in "author" source files for generating HTML—*or* `@` cannot be used for "private" (internal) macros.

3.3.3 Hash Mark

`#` is needed for numerical specifications in HTML, especially colours and Unicode symbols, while it plays a different (essential) role in our definitions of TEX macros here. We redefine LATEX's `\#` for a kind of "quoting" `#` (in macro definitions) in order to refer to their HTML meaning.

```
{ \MakeOther\# \gdef\#{#}                     %% \M... 2011/11/08
% \catcode`\&=12 \gdef\AmpMark{&}             %%   rm. 2011/11/08
}
```

. . . `\CompWordMark` etc.?

3.3.4 "Escaping" HTML Code for "Verbatim"

`\xmltagcode{⟨chars⟩}` yields '`<`⟨*chars*⟩`>`':

```
\newcommand*{\xmltagcode}[1]{\code{\lt#1\gt}}
```

`\xmleltcode{⟨name⟩}{⟨content⟩}` displays the code for an entire ⟨*name*⟩ element containing ⟨*content*⟩ without attributes:

```
\newcommand*{\xmleltcode}[2]{\code{\lt#1\gt#2\lt/#1\gt}}
```

`\xmleltcode{⟨name⟩}{⟨attrs⟩}{⟨content⟩}` displays the code for an entire ⟨*name*⟩ element *with* attribute text '⟨*attrs*⟩' containing ⟨*content*⟩:

```
\newcommand*{\xmleltattrcode}[3]{\code{\lt#1 #2\gt#3\lt/#1\gt}}
```

`\xmlentitycode{⟨name⟩}` yields the code '`&`⟨*name*⟩`;`' for an entity with name ⟨*name*⟩:

```
\newcommand*{\xmlentitycode}[1]{\code{\&#1;}}
```

3.3.5 Head

`\head` produces the first two tags that an HTML file must start:

```
\newcommand*{\head}{<html><head>}                 %% ^^J rm 2010/10/10
```

`\MetaTag{⟨inside⟩}` creates a `<meta>` tag:

```
\newcommand*{\MetaTag}[1]{\indenti<meta #1>}
```

`\charset{⟨code-page⟩}`

```
\newcommand*{\charset}[1]{%
  \MetaTag{ http-equiv="content-type"\@content{text/html; #1}}}
  %% <- space 2012/09/08
```

`\metanamecontent{⟨name⟩}{⟨content⟩}` obviously:

```
\newcommand*{\metanamecontent}[2]{%
    \MetaTag{\@name{#1}\@content{#2}}}
```

`\author{⟨name⟩}` and `\date{⟨date⟩}` set according metadata, somewhat opposing LaTeX (TODO!?):

```
\renewcommand*{\author}{\metanamecontent{author}}
\renewcommand*{\date}{\metanamecontent{date}}
```

The name of `\metadescription{⟨text⟩}` allows using `\begin{description}` (cf. secrefenv):

```
\newcommand*{\metadescription}{\metanamecontent{description}}
```

`\keywords{⟨text⟩}`:

```
\newcommand*{\keywords}{\metanamecontent{keywords}}
```

`\robots{⟨instructions⟩}`:

```
\newcommand*{\robots}{\metanamecontent{robots}}
    %% #2 juergenf: index, follow, noarchive
```

`\norobots` for privacy (cf. `noarchive.net/meta` and *Wikipedia*:

```
\newcommand*{\norobots}{\robots{noarchive,nofollow,noindex}}
```

`\metanamelangcontent{⟨name⟩}{⟨lang⟩}{⟨content⟩}`, in addition to the above, uses language code ⟨*lang*⟩:

```
\newcommand*{\metanamelangcontent}[3]{%
    \MetaTag{\@name{#1}\@lang{#2}\@content{#3}}}
```

So there can be language-dependent descriptions and keywords: `\langdescription{⟨text⟩}` and `\langkeywords{⟨⟩}`

```
\newcommand*{\langdescription}{\metanamelangcontent{description}}
\newcommand*{\langkeywords}   {\metanamelangcontent{keywords}}
```

`\stylesheet{⟨media⟩}{⟨css⟩}` uses ⟨*css*⟩`.css` for `media="`⟨*media*⟩`"`:

```
\newcommand*{\stylesheet}[2]{%
  \space\space                          %% 2010/09/10
  <link rel="stylesheet" media="#1"%
        \@type{text/css}%               %% \@type 2011/10/05
        \@href{#2.css}>}
```

Alternatively, style declarations may occur in the `<style>` element. It can be accessed by the `{style}` environment (cf. Sec. 3.7):

```
\newenvironment*{style}[1]
                {<style\@type{text/css} media="#1">}
                {</style>}
```

With `\title{⟨text⟩}`, ⟨*text*⟩ heads the browser window:

```
\renewcommand*{\title}{\space\space\SimpleTagSurr{title}}
```

3.3.6 Body

`\body` separates the `head` element from the `body` element of the page.

```
\newcommand*{\body}{</head><body>}
```

`\topofpage` generates an anchor `top-of-page`:

```
\newcommand*{\topofpage}{\hanc{top-of-page}{}}
```

`\finish` finishes the page, closing the `body` and `html` elements.

```
\newcommand*{\finish}{</body></html>}
```

3.3.7 Comments

`\comment{⟨comment⟩}` produces a one-line HTML comment. By contrast, there is an environment `{commentlines}{⟨comment⟩}` for multi-line comments. It is convenient for "commenting out" code (unless the latter contains other HTML comments ...) where ⟨*comment*⟩ is a *comment* for explaining what is commented out.

```
\newcommand*{\comment}[1]{<!--#1-->}
% \newcommand{\commentlines}[1]{\comment{^^J#1^^J}} %% 2010/05/07
%   %% <- TODO bzw. \endlinechar=‘\^^J 2010/05/09 back 2010/05/10
\newenvironment{commentlines}[1]                    %% 2010/05/17
  {<!--#1}
  {-->}
```

3.3.8 CSS

`\stylespan{⟨css-style⟩}{⟨text⟩}` applies the CSS styling ⟨*css-style*⟩ to ⟨*text*⟩:

```
\newcommand*{\stylespan}[1]{\TagSurr{span}{\@style{#1}}}
```

Not sure about `<div>` yet ... TODO

3.4 Paragraphs and Line Breaks

2010/04/28: `<br>` for manual line breaking can be generated either by `\newline` or by `\\`:

```
\renewcommand*{\newline}{<br>}
\let\\\newline
```

Automatical insertion of `<p>` tags for starting new paragraphs according to Sec. 3.2.8 has been difficult, especially comment lines so far insert unwanted paragraph breaks (TODO 2011/11/20). So here are some ways to use LaTeX/ Plain TeX commands—or . . . :

```
% \def\par{<p>} %% + empty lines !? 2010/04/26
```

← difficult with `\stop`; 2010/09/10: `\endgraf` produces `<p>`—TODO!?

```
\renewcommand*{\endgraf}{<p>}                  %% was </p> 2012/11/19
```

However, I rather have decided for inserting a literal ‘`<p>`’ using an editor (keyboard) shortcut.

`\rightpar{`⟨*text*⟩`}` places ⟨*text*⟩ flush right. I have used this for ‘Last revised . . . ’ and for placing navigation marks.

```
\newcommand*{\rightpar}{\TagSurr p\@align@r}           %% 2010/06/17
```

Often I use `\rightpar` with *italics*, now there is `\rightitpar{`⟨*text*⟩`}` for this purpose:

```
\newcommand*{\rightitpar}[1]{\rightpar{\textit{#1}}}
```

3.5 Physical Markup (Inline)

We “re-use” some LaTeX commands for specifying font attributes, rather than (re)defining macros `\i`, `\b`, `\tt`, . . .

`\textit{`⟨*text*⟩`}` just expands to `<i>`⟨*text*⟩`</i>`

```
\renewcommand*{\textit}{\SimpleTagSurr i}
```

etc. for `\textbf`, `\texttt` . . . :

```
\renewcommand*{\textbf}{\SimpleTagSurr b}
\renewcommand*{\texttt}{\SimpleTagSurr{tt}}            %% 2010/06/07
```

`\textsf{`⟨*text*⟩`}` chooses some sans-serif:

```
\renewcommand*{\textsf}{\stylespan{font-family:sans-serif}}
```

`\textup{`⟨*text*⟩`}` may undo surrounding slanting or . . . :

```
\renewcommand*\textup{\stylespan{font-style:normal}}
```

`\textcolor{⟨color⟩}{⟨text⟩}` is from LaTeX's color package that we won't load for generating HTML, so it is "new" here, it is just natural to use it for coloured text. `<font>` is deprecated, use `<span>` instead:

```
\newcommand*{\textcolor}[1]{\stylespan{color:#1}}
```

TeX/LaTeX's `\underbar{⟨text⟩}` is redirected to the `<u>` element:

```
\renewcommand*{\underbar}{\SimpleTagSurr u}
```

3.6 Logical Markup

`\heading{⟨level⟩}{⟨text⟩}` prints ⟨*text*⟩ with size dependent on ⟨*level*⟩. The latter may be one out of `1`, `2`, `3`, `4`, `5`, `6`.

```
\newcommand*{\heading}[1]{\SimpleTagSurr{h#1}}
```

. . . I might use `\section` etc. one day, I made `\heading` when I could not control the sizes of the section titles properly and decided first to experiment with the level numbers.

`\code{⟨text⟩}` marks ⟨*text*⟩ as "code," just accessing te `<code>` element, while standard LaTeX does not provide a `\code` command:

```
\newcommand*{\code}{\SimpleTagSurr{code}}   %% 2010/04/27
```

`\emph{⟨text⟩}` is LaTeX's command again, but somewhat abused, expanding to '`<em>`⟨*text*⟩`</em>`':

```
\renewcommand*{\emph}  {\SimpleTagSurr{em}}
```

. . . Note that LaTeX's `\emph` feature of switching to `up` when `\emph` appears in an italic context doesn't work here . . .

`\strong{⟨text⟩}` again just calls an HTML element. It may behave like `\textbf{`⟨*text*⟩`}`, or . . . I don't know . . .

```
\newcommand*{\strong}{\SimpleTagSurr{strong}}
```

`\var{⟨symbol(s)⟩}` accesses the `<var>` element:

```
\newcommand*{\var}{\SimpleTagSurr{var}}
```

For tagging acronyms, HTML offers the `<acronym>` element, and the TUGboat macros provide `\acro{⟨LETTERS⟩}`. I have used the latter for some time in my package documentations anyway. For v0.7, I add the latter here as an alias for `\acronym{⟨LETTERS⟩}` (supporting both naming policies mentioned in Sec. 3.7):

```
\newcommand*{\acronym}{\SimpleTagSurr{acronym}}
\newlet\acro\acronym
```

`\newacronym{⟨LETTERS⟩}` saves you from doubling the ⟨*LETTERS*⟩ when you want to create the shorthand macro \⟨*LETTERS*⟩:

```
\newcommand*{\newacronym}[1]{%
    \expandafter\newcommand\expandafter*\csname#1\endcsname{%
        \acronym{#1}}}
```

However, `<acronym>` is deprecated. You may use `\abbr{⟨LETTERS⟩}` and `\newabbr{⟨LETTERS⟩}` instead:

```
\newcommand*{\abbr}{\SimpleTagSurr{abbr}}              %% 2012/09/13
\newcommand*{\newabbr}[1]{%
    \expandafter\newcommand\expandafter*\csname#1\endcsname{%
        \abbr{#1}}}
```

3.7 Environments

We reduce LaTeX's `\begin` and `\end` to their most primitive core.

`\begin{⟨command⟩}` just executes the macro \\⟨*command*⟩, and

`\end{⟨command⟩}` just executes the macro `\end`⟨*command*⟩.

They don't constitute a group with local settings. Indeed, the present (2010/11/07) version of blog.sty does not allow any assignments while "copying" the TeX source into the `.htm`. There even is no check for proper nesting. `\begin` and `\end` just represent HTML elements (their starting/ending tags) that typically have "long" content. (We might "intercept" `\begin` and `\end` before copying for executing some assignments in a future version.)

```
\let\begin\@nameuse
\def\end#1{\csname end#1\endcsname}
```

. . . moving `{english}` to `xmlprint.cfg` 2010/05/22 . . .

As formerly with physical markup, we have *two* policies for **choosing macro names**: (i) using an *existing* HTML element name, (ii) using a LaTeX command name for accessing a somewhat similar HTML element having a *different* name. [2011/10/05: so what? TODO]

New 2011/10/05: With `\useHTMLelement{⟨ltx-env⟩}{⟨html-el⟩}`, you can access the `<`⟨*html-el*⟩`>` element by the ⟨*ltx-env*⟩ environment. The "starred" form is for "list" environments where I observed around 2011/10/01 that certain links (with Mozilla Firefox) need `</li>`:

```
\newcommand*{\useHTMLelement}{%
    \@ifstar{\@useHTMLelement[</li>]}{\@useHTMLelement}}
\newcommand*{\@useHTMLelement}[3][]{%
    \@namedef{#2}{<#3>}%
    \@namedef{end#2}{#1\CLBrk</#3>}}        %% \CLBrk 2012/04/03
```

Applications:

CARE: `{small}` is an environment here, it is not in LaTeX:

```
\useHTMLelement{small}{small}
```

`{center}`:

```
% \renewenvironment*{center}{<p align="center">}{</p>}
% \renewenvironment*{center}{<p \@align@c>}{</p>}
\useHTMLelement{center}{center}
```

The next definitions for `{enumerate}`, `{itemize}`, `{verbatim}` follow policy (ii):

```
\useHTMLelement*{enumerate}{ol}
\useHTMLelement*{itemize}  {ul}
```

`\begin{enumtype}{⟨type⟩}` starts an enumeration environment with enumeration type ⟨*type*⟩ which can be one out of `1`, `a`, `A`, `i`, `I` (somewhat resembling the functionality of the enumerate package):

```
\newenvironment{enumtype}[1]{<ol \@type{#1}}{</ol>}
```

With blog.sty, `{verbatim}` really doesn't work much like its original LaTeX variant. TeX macros inside still are expanded, and you must care yourself for wanted quoting:

```
\useHTMLelement{verbatim} {pre}
```

`{quote}`:

```
\useHTMLelement{quote}{blockquote}
```

For list `\item`s, I tried to get readable HTML code using `\indenti`. This fails with nested lists. The indent could be increased for nested lists if we supported assignments with `\begin` and `\end`. 2011/10/04 including `<`⟨*/li*⟩`>`, repairs more links in DANTE talk (missing again 2011/10/11!?):

```
\renewcommand*{\item}{%
    \indenti</li>\CLBrk                            %% 2011/10/11
    \indenti<li>}
```

LaTeX's `{description}` environment redefines the label format for the optional argument of `\item`. Again, *we* cannot do this here (we even cannot use optional arguments, at least not easily). Instead we define a different `\ditem{⟨term⟩}` having a *mandatory* argument (TODO star?).

```
\useHTMLelement{description}{dl}
\newcommand*{\ditem}[1]{\indenti<dt>\strong{#1}<dd>}
```

3.8 Links

3.8.1 Basic Link Macros

`\hanc{⟨name⟩}{⟨text⟩}` makes ⟨*text*⟩ an anchor with HTML label ⟨*name*⟩ like hyperref's `\hypertarget{⟨name⟩}{⟨text⟩}` (that we actually provide as well, towards printing from the same source):

```
\newcommand*{\hanc}[1]{\TagSurr a{\@name{#1}}}
\newlet\hypertarget\hanc
```

`\hancref{⟨name⟩}{⟨target⟩}{⟨text⟩}` makes ⟨*text*⟩ an anchor with HTML label ⟨*name*⟩ and at the same time a link to ⟨*target*⟩:

```
\newcommand*{\hancref}[2]{\TagSurr a{\@name{#1} \@href{#2}}}
```

`\href{⟨name⟩}{⟨text⟩}` makes ⟨*text*⟩ a link to ⟨*name*⟩ (as with hyperref):

```
\newcommand*{\href}[1]{\TagSurr a{\@href{#1}}}
```

3.8.2 Special cases of Basic Link Macros

`\autanc{⟨text⟩}` creates an anchor where ⟨*text*⟩ is the text and the internal label at the same time:

```
\newcommand*{\autanc}[1]{\hanc{#1}{#1}}              %% 2010/07/04
```

`\ancref{⟨name⟩}{⟨text⟩}` makes ⟨*text*⟩ a link to an anchor ⟨*name*⟩ on the same web page. This is especially useful for a "table of contents"—a list of links to sections of the page. It is just like hyperref's `\hyperlink{⟨name⟩}{⟨text⟩}`:

```
\newcommand*{\ancref}[1]{\href{\##1}}
\newlet\hyperlink\ancref
```

`\autref{⟨text⟩}` makes ⟨*text*⟩ a link to an anchor named ⟨*text*⟩ itself:

```
\newcommand*{\autref}[1]{\ancref{#1}{#1}}            %% 2010/07/04
```

3.8.3 Italic Variants

Some of the link macros get "emphasized" or "italic" variants. Originally I used "emphasized," later I decided to replace it by "italic," as I found that I had used italics for another reason than emphasizing. E.g., ⟨*text*⟩ may be 'bug,' and I am not referring to some bug, but to the Wikipedia article *Bug*. This has been inspired by some Wikipedia typography convention about referring to titles of books or movies. (The `em` → `it` replacement has not been completed yet.)

```
% \newcommand*{\emhref}[2]{\href{#1}{\emph{#2}}}
\newcommand*{\ithref}[2]{\href{#1}{\textit{#2}}}
\newcommand*{\itancref}[2]{\ancref{#1}{\textit{#2}}}%% 2010/05/30
\newcommand*{\emancref}[2]{\ancref{#1}{\emph{#2}}}
```

3.8.4 Built Macros for Links to Local Files

Originally, I wanted to refer to my web pages only, using

`\fileref{⟨filename-base⟩}`.

I have used extension `.htm` to avoid disturbing my Atari editor xEDIT or the the Atari emulator (Hatari). The extension I actually use is stored as macro `\htext` in a more local file (e.g., `.cfg`).—Later I realized that I may want to refer to local files other than web pages, and therefore I introduced a more general `\FileRef{⟨filename⟩}|`, overlooking that it was the same as `\href`.

```
% \newcommand*{\FileRef}[1]{\TagSurr a{\@href{#1}}}
\newcommand*{\htext}{.htm}                          %% 2011/10/05
\newcommand*{\fileref}[1]{\href{#1\htext}}
% \newcommand*{\emfileref}[2]{\fileref{#1}{\emph{#2}}}
\newcommand*{\itfileref}[2]{\fileref{#1}{\textit{#2}}}
```

`\fileancref{`⟨*file*⟩`}{`⟨*anchor*⟩`}{`⟨*text*⟩`}` links to anchor ⟨*anchor*⟩ on web page ⟨*file*⟩:

```
\newcommand*{\fileancref}[2]{%
  \TagSurr a{\@href{#1\htext\##2}}}
% \newcommand*{\emfileancref}[3]{\fileancref{#1}{#2}{\emph{#3}}}
```

← 2010/05/31 →

```
\newcommand*{\itfileancref}[3]{\fileancref{#1}{#2}{\textit{#3}}}
```

3.8.5 Built Macros for Links to Remote Files

blog.sty currently (even 2011/01/24) implements my style *not* to open a new browser window or tab for *local* files but to open a new one for *remote* files, i.e., when a file is addressed by a full URL. This may change (as with blogdot.sty, 2011/10/12, or more generally with local non-HTML files), so let us have a backbone `\hnewref{`⟨*prot*⟩`}{`⟨*host-path[#frag]*⟩`}{`⟨*text*⟩`}` that makes ⟨*text*⟩ a link to ⟨*prot*⟩⟨*host-path[*`#f`*rag]*⟩:

```
\newcommand*{\hnewref}[2]{%
    \TagSurr a{\@href{#1#2" target="_blank}}}
```

So

`\httpref{`⟨*host-path[#frag]*⟩`}{`⟨*text*⟩`}`

makes ⟨*text*⟩ a link to `http://`⟨*host-path[*`#f`*rag]*⟩:

```
\newcommand*{\httpref}{\hnewref{http://}}
```

With v0.4, macros based on `\httpref` are moved to texlinks.sty:

```
\RequirePackage[blog]{texlinks}[2011/02/10]
```

Former `\urlref` appears as `\urlhttpref` there …

```
\newlet\urlref\urlhttpref
```

… and `\ctanref` has changed its meaning there as of 2011/10/21. texlinks sometimes uses a “permanent alias” `\NormalHTTPref` of `\httpref`:

```
\newlet\NormalHTTPref\httpref
```

`\httpsref` is the analogue of `\httpref` for `https://`:

```
\newcommand*{\httpsref}{\hnewref{https://}}
```

3.9 Characters/Symbols

3.9.1 Basic Preliminaries

`&` is made `other` for using it to call HTML's "character entities."

```
\MakeOther\&
```

Again we have the two policies about choosing macro names and respectively two new definition commands. `\declareHTMLsymbol{⟨name⟩}` defines a macro \⟨*name*⟩ expanding to `&`⟨*name*⟩`;`. Checking for prior definedness hasn't been implemented yet. (TODO; but sometimes redefining . . .)

```
\newcommand*{\declareHTMLsymbol}[1]{\@namedef{#1}{&#1;}}
```

`\declareHTMLsymbols{⟨name⟩}{⟨list⟩}` essentially issues

`\declareHTMLsymbol{⟨attr⟩}\declareHTMLsymbols{⟨list⟩}`

while `\declareHTMLsymbols{}` essentially does nothing—great, this is an explanation by recursion!

```
\newcommand*{\declareHTMLsymbols}{\DoWithAllOf\declareHTMLsymbol}
```

`\renderHTMLsymbol{⟨macro⟩}{⟨name⟩}` *redefines* macro ⟨*macro*⟩ to expand to `&`⟨*name*⟩`;`:

```
\newcommand*{\renderHTMLsymbol} [2]{\renewcommand*{#1}{&#2;}}
```

Redefinitions of `\&` and `\%` (well, `\PercentChar` is fifinddo's version of LaTeX's `\@percentchar`):

```
\renderHTMLsymbol{\&}{amp}
\let\%\PercentChar
```

3.9.2 Diacritics

For the difference between diacritic and accent, see *Wikipedia*.

HTML entities `&eacute;` (é), `&ccedil` (ç), `&ocirc;` (ô) etc. can be accessed by TeX's accent commands `\'`, `\c`, `\^`, ``\` ``, `\"`:

```
% \declareHTMLsymbol{eacute}
% \declareHTMLsymbol{ocirc}
\renewcommand*{\'}[1]{&#1acute;}
\renewcommand*{\c}[1]{&#1cedil;}
\renewcommand*{\^}[1]{&#1circ;}
\renewcommand*{\`}[1]{&#1grave;}
\renewcommand*{\"}[1]{&#1uml;}
```

. . . former `\uml{⟨char⟩}` is obsolete, use `\"⟨char⟩` (or `\"`⟨*char*⟩) instead.

`\v{⟨char⟩}` just works with ⟨*char*⟩ = `s` and ⟨*char*⟩ = `S` for š and Š:

```
\renewcommand*{\v}[1]{#1caron;}
```

3.9.3 Ligatures and the Like

`\lig{⟨char1⟩⟨char2⟩}` forms a ligature from ⟨*char1*⟩ and ⟨*char2*⟩:

```
\newcommand*{\lig}[1]{&#1lig;}
```

With v0.81, we use this to reimplement `\ss` from Plain TEX and LATEX for the putative "s-z ligature", the German "sharp s" ("ß"):

```
% \renderHTMLsymbol{\ss}{szlig}
\renewcommand*{\ss}{\lig{sz}}
```

`\AE`, `\ae`, `\OE`, `\oe` ("Æ", "æ", "Œ", "œ") are reimplemented likewise:

```
\renewcommand*{\AE}{\lig{AE}}
\renewcommand*{\ae}{\lig{ae}}
\renewcommand*{\OE}{\lig{OE}}
\renewcommand*{\oe}{\lig{oe}}
```

3.9.4 Greek

```
\declareHTMLsymbols{{Alpha}{alpha}                      %% 2012/01/06
    {Beta}{beta}{Gamma}{gamma}{Delta}{delta}{Epsilon}{epsilon}
    {Zeta}{zeta}{Eta}{eta}{Theta}{theta}{Iota}{iota}{Kappa}{kappa}
    {Lambda}{lambda}{My}{my}{Ny}{ny}{Xi}{xi}{Omikron}{omikron}
    {Pi}{pi}{Rho}{rho}{Sigma}{sigma}{sigmaf}{Tau}{tau}
    {Upsilon}{upsilon}{Phi}{phi}{Chi}{chi}{Psi}{psi}
    {Omega}{omega}                    %% render -> declare 2011/02/26
    {thetasym}{upsih}{piv} }
```

3.9.5 Arrows

—somewhat completed 2012/07/25.

`\downarrow`, `\leftarrow`, `\leftrightarrow`, `\rightarrow`, `\uparrow`:

```
\renderHTMLsymbol {\downarrow}     {darr}   %% 2010/09/15
\renderHTMLsymbol {\leftarrow}     {larr}
\renderHTMLsymbol {\leftrightarrow}{harr}
\renderHTMLsymbol {\rightarrow}    {rarr}
\renderHTMLsymbol {\uparrow}       {uarr}   %% 2010/09/15
```

Aliases `\gets` and `\to` were implemented first as stand-alones, now are treated by `\let`:

```
\let \gets \leftarrow
\let \to   \rightarrow
```

`\Downarrow`, `\Leftarrow`, `\Leftrightarrow`, `\Rightarrow`, `\Uparrow` (i.e., double variants):

```
\renderHTMLsymbol {\Downarrow}     {dArr}
\renderHTMLsymbol {\Leftarrow}     {lArr}
\renderHTMLsymbol {\Leftrightarrow}{hArr}
\renderHTMLsymbol {\Rightarrow}    {rArr}
\renderHTMLsymbol {\Uparrow}       {uArr}
```

`\crarrow` accesses HTML's `crarr` entity (symbol for return key), named "downwards arrow with tip leftwards" in Unicode (U+21b2):

```
\newcommand*{\crarrow}{&crarr;}              %% 2012/09/13
```

3.9.6 Dashes

The ligatures `--` and `---` for en dash and em dash don't work in our expanding mode. Now, HTML's policy for choosing names often prefers shorter names than are recommended for (La)TeX, so here I adopt a *third* policy besides (i) and (ii) earlier; cf. LaTeX's `\textemdash` and `\textendash`.—`\newcommand` does not accept macros whose names start with `end`, so: `\endash`, `\emdash` ...

```
\def          \endash  {&ndash;}         %% \end... illegal
\newcommand*{\emdash} {&mdash;}
```

3.9.7 Spaces

"Math" (not only!) spaces `\,`, `\enspace`, `\quad`, `\qquad`:

```
\renderHTMLsymbol{\enspace}{ensp}
\renderHTMLsymbol{\quad}   {emsp}
\renewcommand*   {\qquad}  {\quad\quad}
```

2011/07/22: `&thinsp;` allows line breaks, so we introduce `\thinsp` to access `&thinsp;`, while `\thinspace` and `\,` use Unicode "Narrow No-Break Space" (`U+202F`, see *Wikipedia Space (punctuation)*; browser support?):

```
% \renderHTMLsymbol{\thinspace}{thinsp}
% \renderHTMLsymbol{\,}        {thinsp}
\declareHTMLsymbol{thinsp}
\renderHTMLsymbol{\thinspace}{\#8239}
\renderHTMLsymbol{\,}        {\#8239}
```

`\figurespace` (U+2007, cf. *Wikipedia*):

```
\newcommand*{\figurespace}{&\#8199;}
```

3.9.8 Quotes, Apostrophe

`\lq`, `\rq`

```
\renderHTMLsymbol{\lq}    {lsquo}
\renderHTMLsymbol{\rq}    {rsquo}
```

In order to use the right single quote for the HTML apostrophe, we must save other uses before. `\urlapostr` is the version of the right single quote for URLs of Wikipedia articles:

```
% \newcommand*{\screenqtd}[1]{‘#1’}                %% rm. 2011/11/08
\newcommand*{\urlapostr}   {’}                    %% 2010/09/10
```

The actual change of `'` is in `\BlogCodes` (Sec. 3.2.4).

`\bdquo` (bottom), `\ldquo`, `\rdquo`, `\sbquo` (single bottom):

```
\declareHTMLsymbol{bdquo}                           %% 2011/09/23
\declareHTMLsymbols{{ldquo}{rdquo}}
\declareHTMLsymbol{sbquo}                           %% 2010/07/01
\declareHTMLsymbols{{laquo}{raquo}}
```

Angled quotes `\laquo` and `\raquo` as well as their “single” versions `\lsaquo` and `\rsaquo`:

```
\declareHTMLsymbols{{laquo}{lsaquo}{raquo}{rsaquo}} %% 2012/10/25
```

As of 2012/09/17, `\asciidq` and `\asciidqtd{`⟨*no-dqs*⟩`}` (e.g., for attributes after `\catchdqs` or typesetting code) move to package catchdq.sty in the catcodes bundle.

`\quot` accesses the same symbol in HTML’s terms (e.g., for displaying code):

```
\declareHTMLsymbol{quot}                            %% 2012/01/21
```

`\endqtd{`⟨*text*⟩`}` quotes in the English style using double quote marks, `\enqtd{`⟨*text*⟩`}` uses single quote marks instead, `\dedqtd{`⟨*text*⟩`}` quotes in German style, `\quoted{`⟨*text*⟩`}` uses straight double quotation marks. Settings from the langcode package may need to be overridden. (A warning might be nice then TODO)

```
\def\endqtd#1{\ldquo#1\rdquo}
\def\enqtd #1{\lq#1\rq}                             %% 2010/09/08
\def\dedqtd#1{\bdquo#1\ldquo}
\def\deqtd #1{\sbquo#1\lq}                    %% corr. 2012/10/25
\newcommand*{\quoted}   [1]{\quot#1\quot}           %% 2012/01/21
```

`\squoted{`⟨*text*⟩`}` surrounds ⟨*text*⟩ with “straight” single quotation marks, useful for other kinds of quoting in computer code:

```
\newcommand*{\squoted}[1]{\urlapostr#1\urlapostr}   %% 2012/01/21
```

3.9.9 (Sub- and) Superscript Digits/Letters

As Plain TeX and LaTeX provides an alias `\sp` for `^`, I use `\spone`, `\sptwo`, `\spthree`, `\spa`, and `\spo` for superscript 1, 2, 3, ‘a’, and ‘o’:

```
\newcommand*{\spone}{&sup1;}
\newcommand*{\sptwo}{&sup2;}
\newcommand*{\spthree}{&sup3;}
\newcommand*{\spa}{&ordf;}
\newcommand*{\spo}{&ordm;}
```

For slanted fractions, I think of xfrac’s `\sfrac{`⟨*numerator*⟩`}{`⟨*denominator*⟩`}`. `\sfrac{1}{2}`, `\sfrac{1}{4}`, and `\sfrac{3}{4}` work so far:

```
\newcommand*{\sfrac}[2]{&frac#1#2;}
```

3.9.10 Math

Symbols (TeX math type "Ord")—`\aleph`:

```
\renderHTMLsymbol{\aleph}{alefsym}
```

I provide `\degrees` for the degree symbol. LaTeX already has `\deg` as an operator, therefore I do not want to use `\declareHTMLsymbol` here.

```
\newcommand*{\degrees}{&deg;}
```

We stick to TeX's `\emptyset`

```
\renderHTMLsymbol{\emptyset}{empty}                       %% 2011/04/14
```

`\exists` and `\forall`:

```
\renderHTMLsymbol{\exists}{exist}
\declareHTMLsymbol{forall}
```

`\prime` can be used for minutes, `\Prime` for seconds:

```
\renderHTMLsymbol{\prime}{prime} \declareHTMLsymbol{Prime}
```

Relations Because `<` and `>` are used for HTML's element notation, we provide aliases `\gt`, `\lt` for mathematical $<$ and $>$—and for reference to HTML (or just XML) code (see Sec. 3.3.4):

```
\declareHTMLsymbols{{gt}{lt}}
```

`\ge`, `\le`, and `\ne` for $\geq$, $\leq$, and $\neq$ resp.:

```
\declareHTMLsymbols{{ge}{le}{ne}}
```

We also provide their TeX aliases `\geq`, `\leq`, `\neq`:

```
\let\geq\ge      \let\leq\le      \let\neq\ne
```

Besides TeX's `\subset` and `\subseteq`, we provide short versions `\sub` and `\sube` inspired by HTML:

```
\declareHTMLsymbol{sub}                               %% 2011/04/04
\let\subset\sub                                       %% 2011/05/08
\declareHTMLsymbol{sube}                              %% 2011/03/29
\let\subseteq\sube                                    %% 2011/05/08
```

Delimiters Angle braces `\langle` and `\rangle`:

```
\renderHTMLsymbol{\langle}{lang}
\renderHTMLsymbol{\rangle}{rang}
```

The one-argument macro `\angled{⟨angled⟩}` allows better readable code (should be in a more general package):

```
\newcommand*{\angled}[1]{\langle#1\rangle}
```

Curly braces `\{` and `\}` . . . :

```
\begingroup
    \Delimiters\[\] \gdef\{[{] \gdef\}[}]
\endgroup
```

Binary Operations TeX's `\ast` corresponds to the "lower" version of the asterisk:

```
\renderHTMLsymbol{\ast}{lowast}                         %% 2011/03/29
```

`\pm` renders the plus-minus symbol:

```
\renderHTMLsymbol{\pm}{plusmn}
```

TeX and HTML agree on `\cap`, `\cup`, and `\times`: 2011/05/08 2011/04/04

```
\declareHTMLsymbols{{cap}{cup}{times}}              %% 2012/01/06
```

We need `\minus` since math mode switching is not supported by blog:

```
\declareHTMLsymbol{minus}                           %% 2011/03/31
```

We override HTML's '`&circ;`' to get TeX's `\circ` (i.e., $\circ$; but I cannot see it on my own pages!?):

```
\renderHTMLsymbol{\circ}{\#x2218}                   %% 2011/04/28
\renderHTMLsymbol{\cdot}{middot}                    %% 2011/05/07
```

`\sdot` generates `&sdot;`, a variant of of `&middot;` reserved for the dot product according to the German *Wikipedia*

```
\declareHTMLsymbol{sdot}                            %% 2011/05/08
```

Operators `\prod`, `sum`:

```
\renderHTMLsymbol{\prod}{product}
\declareHTMLsymbol{sum}
```

3.9.11 Currencies

`\cent`, `\currency`, `\euro`, `\pound`, `\yen`:

```
\declareHTMLsymbols{{cent}{currency}{euro}{pound}{yen}}
```

You get the $ symbol simply by `$`.

3.9.12 Other

The tilde `~` is used for its wonderful purpose, by analogy to TeX(TODO overridden by `\FDpseudoTilde`):

```
\renderHTMLsymbol{~}{nbsp}
```

But now we need a replacement `\tilde` for URLs involving home directories of institution members (should better be `\tildechar` or `\TildeChar`, cf. fifinddo):

```
{ \MakeOther\~ \gdef\tilde{~} \gdef\tildechar{~}}
```

Horizontal ellipsis: `\dots` …

```
\renderHTMLsymbol {\dots} {hellip}
```

Plain TeX's and LaTeX's `\-` becomes a soft hyphen:

```
\renderHTMLsymbol{\-}{shy}
```

`\copyright`:

```
\renderHTMLsymbol{\copyright}{copy}
```

`\bullet`

```
\renderHTMLsymbol{\bullet}{bull}
```

LaTeX's `\S` prints the section sign '§'. In HTML, the latter accessed by `&sect;`, we redirect `\S` to this:

```
\renderHTMLsymbol{\S}{sect}
```

`\dagger`, `\ddagger`:

```
\renderHTMLsymbol{\dagger}{dagger}
\renderHTMLsymbol{\ddagger}{Dagger}
```

`\P` renders the paragraph sign or pilcrow:

```
\renderHTMLsymbol{\P}{para}
```

Sometimes (due to certain local settings) the notations `&&`⟨*characters*⟩`;` or `&&&#`⟨*number*⟩`;` (for Unicode) may not be available. We provide

> `\htmlentity{`⟨*characters*⟩`}`

as well as

> `\unicodeentity{`⟨*decimal*⟩`}`

and

> `\unicodehexentity{`⟨*hexadecimal*⟩`}`

for such situations:

```
\newcommand*{\htmlentity}[1]{&#1;}
\newcommand*{\unicodeentity}[1]{&\##1;}
\newcommand*{\unicodehexentity}[1]{&\#x#1;}
```

3.10 TeX-related

Somebody actually using blog.sty must have a need to put down notes about TeX for her own private purposes at least—I expect.

3.10.1 Logos

“Program” names might be typeset in a special font, I once thought, and started tagging program names with `\prg`. It could be `\texttt` or `\textsf` like in documentations of LaTeX packages. However, sans-serif is of doubtable usefulness on web pages, and typewriter imitations usually look terrible on web pages. So I am waiting for a better idea and let `\prg` just remove the braces.

```
\newlet\prg\@firstofone
\newcommand*{\BibTeX}{\prg{BibTeX}} %% 2010/09/13
\renewcommand*{\TeX}{\prg{TeX}}
\renewcommand*{\LaTeX}{\prg{LaTeX}}
\newcommand*{\allTeX}{\prg{(La)TeX}}%% 2010/10/05
\newcommand*{\LuaTeX}{\prg{LuaTeX}}
\newcommand*{\pdfTeX}{\prg{pdfTeX}}
\newcommand*{\XeTeX}{\prg{XeTeX}}   %% 2010/10/09
\newcommand*{\TeXbook}{TeXbook}     %% 2010/09/13
```

3.10.2 Describing Macros

With v0.4, TeX-related *links* are moved to texlinks.sty.

`\texcs{\⟨tex-cmd-name⟩}` or `\texcs\`⟨*tex-cmd-name*⟩ (care for spacing yourself):

```
\newcommand*{\texcs}[1]{\code{\string#1}}           %% 2010/11/13
```

Good old `\cs{⟨tex-cmd-name⟩}` may be preferable:

```
\def\cs#1{\code{\BackslashChar#1}}                  %% 2011/03/06
```

`\metavar{⟨name⟩}`:

```
\newcommand*{\metavar}[1]{\angled{\meta{#1}}}
```

3.11 Tables

I am not so sure about this section …

3.11.1 Indenting

There are three levels of indenting:

`\indenti`, `\indentii`, and `\indentiii`.

The intention for these was to get readable HTML code. Not sure …

```
{\catcode`\ =12%% 2010/05/19
\gdef\indenti{  }\gdef\indentii{    }\gdef\indentiii{      }}
```

3.11.2 Starting/Ending Tables

`\startTable{⟨attributes⟩}` and `\endTable` have been made for appearing in different macros, such as in the two parts of a `\newenvironment`:

```
\newcommand*{\startTable}[1]{<table #1>}
\def\endTable{</table>}
```

`\@frame@box` among the `\startTable` ⟨*attributes*⟩ draws a frame around the table, `\@frame@groups` separates "groups" by rules:

```
\newcommand*{\@frame@box}{\@frame{box}}
\newcommand*{\@frame@groups}{\@frame{groups}}
```

`\begin{allrulestable}{⟨cell-padding⟩}{⟨width⟩}` starts a table environment with all possible rules and some code cosmetic. ⟨*width*⟩ may be empty ...

```
\newenvironment{allrulestable}[2]
  {\startTable{\@cellpadding{#1} \@width{#2}
               \@frame@box\ rules="all"}\CLBrk  %% \ 2011/10/12
   \ \tbody} %% <- tbody 2011/10/13, `\ ' 2011/11/09 ->
  {\ \endtbody\CLBrk\endTable}
```

`<tbody>...</tbody>` seemed to be better with `\HVspace` for blogdot.sty, so it gets an environment `{tbody}` (i.e., macros `\tbody` and `\endtbody`):

```
\useHTMLelement{tbody}{tbody}
```

3.11.3 Rows

I first thought it would be good for readability if some HTML comments explain nesting or briefly describe the content of some column, row, or cell. But this is troublesome when you want to comment out an entire table ...

`\begin{TableRow}{⟨comment⟩}{⟨attributes⟩}`
starts an environment producing an HTML comment ⟨*comment*⟩ and a table row with attributes ⟨*attributes*⟩, including code cosmetic.

```
\newenvironment*{TableRow}[2]{%% lesser indentation 2011/04/25
  \ \comment{ #1 }\CLBrk
  \indenti<tr #2>%
  }{%
  \indenti\endtr}                           %% \endtr 2011/11/08
```

`\begin{tablecoloredrow}{⟨comment⟩}{⟨background-color⟩}`
is a special case of `{TableRow}` where `@bgcolor` is the only attribute:

```
\newenvironment{tablecoloredrow}[2]
  {\TableRow{#1}{\@bgcolor{#2}}}
  {\endTableRow}
```

`\begin{tablecoloredboldrow}{⟨comment⟩}{⟨background-color⟩}`
is like `{tablecoloredrow}` except that content text is rendered in boldface (TODO horizontal centering?):

```
\newenvironment{tablecoloredboldrow}[2]         %% 2011/11/03/08
  {\TableRow{#1}{\@bgcolor{#2}
                 \@style{font-weight:bold}}}
  {\endTableRow}
```

`\begin{tablerow}{⟨comment⟩}` is a special case of `{TableRow}` where the only attribute yields "top" vertical alignment (TODO strange):

```
\newenvironment{tablerow}[1]{\TableRow{#1}{\@valign@t}}
                            {\endTableRow}
```

`\starttr` and `\endtr` delimit a row; these commands again have been made for appearing in different macros. There is *no* code indenting, probably for heavy table nesting where indenting was rather useless (? TODO only in texblog.fdf? there indents would have been useful).

```
\newcommand*{\starttr}{<tr>}
\def\endtr{</tr>}
```

3.11.4 Cells

`simplecell{⟨content⟩}` produces the most *simple* kind of an HTML table cell:

```
\newcommand*{\simplecell}{\SimpleTagSurr{td}}   %% 2010/07/18
```

`\TableCell{⟨attributes⟩}{⟨content⟩}` produces the most *general* kind of a cell, together with a code indent:

```
\newcommand*{\TableCell}[2]{\indentiii\startTd{#1}#2\endTd}
```

`\colorwidthcell{⟨color⟩}{⟨width⟩}{⟨content⟩}` uses just the `@bgcolor` and the `@width` attribute:

```
\newcommand*{\colorwidthcell}[2]{\TableCell{\@bgcolor{#1}\@width{#2}}}
```

`\tablewidthcell{⟨color⟩}{⟨width⟩}{⟨content⟩}` uses just the `@bgcolor` and the `@width` attribute:

```
\newcommand*{\tablewidthcell}[1]{\TableCell{\@width{#1}}}
```

`\tablecell{⟨content⟩}` is like `\simplecell{`⟨*content*⟩`}`, except that it has a code indent:

```
\newcommand*{\tablecell}{\TableCell{}}
```

`\tableCell{⟨content⟩}` is like `\tablecell{`⟨*content*⟩`}`, except that the content ⟨*content*⟩ is horizontically centered. The capital `C` in the name may be considered indicating "**c**entered":

```
\newcommand*{\tableCell}{\TableCell\@align@c}
```

Idea: use closing star for environment variants!?

`\begin{bigtablecell}{⟨comment⟩}` starts an *environment* yielding a table cell element without attributes, preceded by a HTML comment ⟨*comment*⟩ unless ⟨*comment*⟩ is empty. At least the HTML tags are indented:

```
\newenvironment{bigtablecell}[1]{\BigTableCell{#1}{}}
                                {\endBigTableCell}
%                {\ifx\\#1\\%             %% 2010/05/30
%                   \indentii\ \comment{#1}\CLBrk
%                 \fi
%                 \indentiii<td>}
%                {\indentii</td>}         %% !? 2010/05/23
```

\begin{BigTableCell}{⟨*comment*⟩}{⟨*attributes*⟩}
is like `\begin{bigtablecell{`⟨*comment*⟩`}}` except that it uses attributes ⟨*attributes*⟩:

```
\newenvironment{BigTableCell}[2]
    {\ifx\\#1\\\indentii\ \comment{#1}\CLBrk\fi
     \indentiii\startTd{#2}}
    {\indentii\endTd}           %% TODO indent? 2010/07/18
```

\startTd{⟨*attributes*⟩} and `\endTd` delimit a cell element and may appear in separate macros, e.g., in an environment definition. There is no code cosmetic. And finally there is `\StartTd` that yields less confusing code without attributes:

```
\newcommand*{\startTd}[1]{<td #1>}
\newcommand*{\StartTd}{<td>}                    %% 2011/11/09
\def\endTd{</td>}
```

`\emptycell` uses `<td />` instead of `<td></td>` for an empty cell:

```
\newcommand*{\emptycell}{<td />}                %% 2011/10/07
```

3.11.5 "Implicit" Attributes and a "TeX-like" Interface

After some more experience, much musing, and trying new tricks, I arrive at the following macros (v0.7). (i) When a page or a site has many tables that use the same attribute values, these should not be repeated for the single tables, rather the values should be invoked by shorthand macros, and the values should be determined at a single separate place. We will have `\stdcellpadding`, `\stdtableheadcolor` and `\stdtableheadstyle`. (ii) As with TeX, `\cr` should suffice to *close* a *cell* and a *row*, and then to *open* another *row* and its first *cell*. And there should be a single command to close a cell within a row and open a next one.

We use `\providecommand` so the user can determine the values in a file for blog where blogexec is loaded later. `\stdcellpadding` should correspond to the CSS settings, the value of `6` you find here is just what I used recently.

```
\providecommand*{\stdcellpadding}{6}
```

For `\stdtableheadcolor`, I provide a gray, `#EEEEEE`, that the German Wikipedia uses for articles about networking protocols (unfortunately, it doesn't have a CSS-3X11 color name):

```
\providecommand*{\stdtableheadcolor}{\#EEEEEE}
```

`\stdtableheadstyle` demands a boldface font. In general, it is used for the `@style` attribute:

```
\providecommand*{\stdtableheadstyle}{font-weight:bold}
```

`\begin{stdallrulestable}` starts an `{allrulestable}` environment with "standard" cell padding and empty width attribute, then opens a "standard" row element with a "standard" comment as well as a cell:

```
\newenvironment{stdallrulestable}{%
    \allrulestable{\stdcellpadding}{}\CLBrk
      \TableRow{standard all-rules table}%
               {\@bgcolor{\stdtableheadcolor}
                \@style{\stdtableheadstyle}}\CLBrk
        \indentii\StartTd
```

`\end{stdallrulestable}` will provide closing of a cell and a row, including a code cosmetic:

```
    }{\indenti\endTd\CLBrk\endTableRow\CLBrk
    \endallrulestable}
```

`\endcell` closes a cell and opens a new one. The idea behind this is that an active character will invoke it. The name is inspired by `\endgraf` and `\endline` from Plain TeX and LaTeX (`\newcommand` does not work with `\end...`):

```
\def\endcell{\endTd\StartTd}
```

Plain TeX's and LaTeX's `\cr` and `\endline` are redefined for closing and opening rows and cells, including code cosmetic:

```
\renewcommand*{\cr}{\indentii\endTd\CLBrk\indenti\endtr\CLBrk
                    \indenti\startTR\CLBrk\indentii\StartTd}
\let\endline\cr
```

`\startTR` is a hook defaulting to `\starttr`:

```
\newlet\startTR\starttr
```

3.11.6 Filling a Row with Dummy Cells

These macros were made, e.g., for imitating a program window with a title bar (spanning someting more complex below), perhaps also for a Gantt chart. `\FillRow{⟨span⟩}{⟨attributes⟩}` produces a cell without text, spanning ⟨*span*⟩ columns, with additional attributes ⟨*attributes*⟩.

```
\newcommand*{\FillRow}[2]{\indentiii\startTd{\@colspan{#1} #2}\endTd}
```

`\fillrow{⟨span⟩}` instead only uses the `@colspan` attribute:

```
\newcommand*{\fillrow}[1]{\FillRow{#1}{}}
```

`\fillrowcolor{⟨span⟩}{⟨color⟩}` just uses the `@colspan` and the `@bgcolor` attributes:

```
\newcommand*{\fillrowcolor}[2]{\FillRow{#1}{\@bgcolor{#2}}}
```

3.11.7 Skipping Tricks

`\HVspace{⟨text⟩}{⟨width⟩}{⟨height⟩}` may change, needed for blogdot.sty but also for `\vspace{⟨height⟩}` with texblog. It is now here so I will be careful when I want to change something. `<tbody>` improved the function of `\HVspace` constructions as link text with blogdot.sty.

```
\newcommand*{\HVspace}[3]{%
    \CLBrk
    \startTable{\@width{#2} \@height{#3}
                \@border{0}
                \@cellpadding{0} \@cellspacing{0}}%
      \tbody
       \CLBrk                                       %% 2011/10/14
        \tablerow{HVspace}%                         %% 2011/10/13
```

← inserting text at top for blogdot attempts—that finally did not help anything (2011/10/15) →

```
          \simplecell{#1}%
        \endtablerow                                %% 2011/10/13
       \CLBrk                                       %% 2011/10/14
      \endtbody
    \endTable
    \CLBrk}
```

`\hvspace{⟨width⟩}{⟨height⟩}` . . . :

```
\newcommand*{\hvspace}{\HVspace{}}
```

`\vspace{⟨height⟩}` . . . (TODO: `{0}`!?):

```
\renewcommand*{\vspace}[1]{\hvspace{}{#1}}
```

3.12 Misc

TEX's `\hrule` (rather deprecated in LATEX) is redefined to produce an HTML horizontal line:

```
\renewcommand*{\hrule}{<hr>}
```

For references, there were

```
% \catcode`\^=\active
% \def^#1{\SimpleTagSurr{sup}{#1}}
```

and

```
% \newcommand*{\src}[1]{\SimpleTagSurr{sup}{[#1]}}
```

as of 2010/05/01, inspired by the `<ref>` element of MediaWiki; moved to `xmlprint.tex` 2010/06/02.

3.13 Leaving and HISTORY

```
\endinput
        VERSION HISTORY
v0.1    2010/08/20  final version for DFG
v0.2    2010/11/08  final documentation version before
                    moving some functionality to ’fifinddo’
v0.3    2010/11/10  removed ^^J from \head
        2010/11/11  moving stuff to fifinddo.sty; \BlogCopyFile
        2010/11/12  date updated; broke too long code lines etc.;
                    \CatCode replaced (implemented in niceverb only);
                    \ifBlogAutoPars etc.
        2010/11/13  doc: \uml useful in ...; \texcs
        2010/11/14  doc: argument for {commentlines},
                         referring to environments with curly braces,
                         more on \ditem
        2010/11/15  TODO: usage, templates
        2010/11/16  note on {verbatim}
        2010/11/23  doc. corr. on \CtanPkgRef
        2010/11/27  "keyword"; \CopyLine without ‘fd’
        2010/12/03  \emhttpref -> \ithttpref
        2010/12/23  ‘%’ added to \texhaxpref
        2011/01/23  more in \Provides...
        2011/01/24  updated copyright; resolving ‘td’ ("today")
        JUST STORED as final version before texlinks.sty
v0.4    2011/01/24  moving links to texlinks.sty
v0.41   2011/02/07  \NormalHTTPref
        2011/02/10  refined call of ‘texlinks’
part of MOREHYPE RELEASE r0.3
v0.5    2011/02/22  \BlogProvidesFile
        2011/02/24  ... in \BlogCopyFile
        2011/02/25  ordering symbols
        2011/02/26  subsection Greek; note on \declareHTMLsymbol
        2011/03/04  diacritics
        2011/03/06  \cs
        2011/03/09  \var
        2011/03/16  \robots
        2011/03/19  doc. \fileancref arg.s corr.
        2011/03/29  \Sigma, ...
        2011/03/31  \minus
        2011/04/04  \times, \sub, \delta
        2011/04/11  Greek completed
        2011/04/14  \emptyset
        2011/04/22  \deqtd
        2011/04/24  doc.: folding, \stylesheet, ordered "tables";
                    @border, @align, @valign
        2011/04/25  lesser indentation with TableRow
        2011/04/26  \,, \thinspace, \@title; doc. \@name
        2011/04/28  [\circ] PROBLEM still
        2011/04/29  \rightitpar
        2011/05/07  \cdot
        2011/05/08  extended doc. on math symbols; \sdot;
                    \ast replaces \lowast; \subset, \subseteq;
                    \angled
        2011/05/09  \euro
        2011/05/11  |\geq| etc.; new section "logical markup"
        2011/05/12  corr. doc. \heading
        2011/05/14  right mark of \deqtd was rsquo instead of lsquo!
        2011/05/18  \S and note on \StoreOtherCharAs
        2011/06/27  \httpsref; doc: \acro
        2011/07/22  \thinspace vs. \thinsp; 'fifinddo''s
        2011/07/25  "todo" on \description
        2011/08/18f.removing \FileRef, 0.42-> 0.5
        2011/08/31  clarified use of \urlapostr
part of MOREHYPE RELEASE r0.4
v0.6    2011/09/08  doc. uses \HTML, \lq/\rq with &circ;,
                    doc. fix `mult-'; \degrees
        2011/09/21  \acronym
        2011/09/22  \metavar; TODO \glqq...
        2011/09/23  \bdquo
        2011/09/25  doc. `Characters/Symbols'; \figurespace
        2011/09/27  "universal" attributes completed, reworked doc.
        2011/09/30  end lists with </li>
        2011/10/01  \dagger, \ddagger
        2011/10/04  \item includes </li> [2011/10/11: ???]
        2011/10/05  {style}; doc. \acronym -> \acro, \pagebreak,
                    rm. \description; {center} accesses <center>,
                    \useHTMLenvironment replaces \declareHTMLelement
                    and \renderHTMLelement, message "generating"
        2011/10/07  \emptycell
        2011/10/10  doc.: page breaks, $$->\[/\]
part of MOREHYPE RELEASE r0.5
v0.61   2011/10/11  </li> in \item again, \Provides... v wrong
        2011/10/12  \hnewref, `\ ' in allrulestable
        2011/10/14  \CLBrk's
        2011/10/15  doc. note on \HVspace/blogdot
part of MOREHYPE RELEASE r0.51
v0.62   2011/10/16  \hyperlink, \hypertarget; doc. fixes there
        2011/10/20  \textcolor by <span>, \textsf
        2011/10/21  \ctanref now in texlinks.sty;
                    doc.: grammar with `that'
        2011/10/22  \BlogCopyFile message removed
part of MOREHYPE RELEASE r0.52
v0.7    2011/11/03  {tablecoloredboldrow}
        2011/11/05  \ContentAtt -> \@content,
                    \BlogCopyFile -> \BlogProcessFile (blogexec),
                    doc. different \pagebreak's
        2011/11/06  run \BlogCopyLines, doc. \[...\]
        2011/11/07  \ProvideBlogExec
        2011/11/08  \endtr in \endTableRow, using \MakeOther,
                    right quote change moves to \BlogCodes,
                    \BlogInterceptHash; rm. \AmpMark & doc. about it,
                    mod. on #; doc. for tables; start doc. "implicit"
                    table attributes and "TeX-like" interface
        2011/11/09  \tablecolorcell(?); cont. "implicit" etc.;
                    \StartTd
        2011/11/20  \isotoday, \BlogProcessFinalFile,
                    catcodes of ‘<’ ‘>’ untouched; restructured,
                    structured processing, misc -> ordinary
        2011/11/21  BlogLIGs
        2011/11/23  \xmltagcode, \xmlentitycode, \c;
                    doc: <p>, \secref, \pagebreak
        2011/11/24  doc: example results for diacritics
        2011/11/27  \ParseLigs; doc. rm. \pagebreak
        2011/12/12  \title uses \SimpleTagSurr
        2011/12/19  doc. fix {tablerow}
        2011/12/21  \asciidq, \asciidqtd
        2012/01/06  \acro; using dowith.sty (\declareHTMLsymbols);
                    doc.: cross-referring for naming policies
        2012/01/07  \MakeActiveDef\~ for \FDpseudoTilde
        2012/01/11  (C)
        2012/01/21  \quot, \quoted. \squoted
        2012/02/04  \newacronym
        2012/03/14  removed hidden and another comment with
                    \BlogCopyLines, fixed latter, TODO on \NoBlogLigs
        2012/03/17  tweaked \@typeset@protect for \EXECUTE
        2012/03/30  space in stdallrules... after @bgcolor
        2012/04/03  \CLBrk in \@useHTMLelement
        2012/04/09  \htmlentity, \unicodeentity
        2012/05/13  \ss; better comment on \uml;
                    #EEEEEE not "web-safe"
        2012/05/15  xEDIT folding in tables section
part of MOREHYPE RELEASE r0.6
v0.8    2012/06/07  \underbar
        2012/07/25  arrows completed [no: 2012/09/13];
                    doc. "police" -> "policy"
        2012/07/30  \spanstyle, applied; doc. \pagebreak
        2012/08/01  \textup
        2012/08/02  doc. corr. braces for \DeclareHTMLsymbols
        2012/08/06  sec. currencies
        2012/08/07  divided math section, using \declareHTMLsymbols,
                    various additional symbols
        2012/08/23  \startTR
        2012/08/28  \MakeActiveLet\’\rq with ‘actcodes.sty’,
                    attributes start with space
        2012/09/02  about -> around
        2012/09/06  Content-T -> content-t - bugfix?,
                    \BlogProvidesFile with DOCTYPE, some attribute
                    lists rely on space from \declareHTMLattrib,
                    there another \reserved@a;
                    "Head": \metanamecontent, \metanamelangcontent
        2012/09/07  "Head": \author, \date, \metadescription,
                    \keywords; lang variants
        2012/09/08  \TagSurr and \MetaTag without space,
                    \declareHTMLattrib{align}, \@valign@t adjusted;
                    \pagebreak[3]
        2012/09/13  \crarrow, "Fonts" -> "Physical markup" etc.,
                    \abbr, \newabbr
        2012/09/14  \xmleltcode, \xmleltattrcode; el-name -> elt-name
        2012/09/17  \asciidq + \asciidqtd move to ‘catchdq.sty’
        2012/10/03  \newlet;
                    doc.: label process -> catcodes, using \secref
        2012/10/05  moved \ast; \exists, \forall
        2012/10/24  quotes: completed, override ‘langcode.sty’
        2012/10/25  using \DeclareHTMLsymbols for quotes, corr. there,
                    \spone etc., \sfrac
        2012/10/28  spanstyle -> stylespan
        2012/11/16  \TagSurr and \MetaTag with space again
        2012/11/19  \endgraf -> <p>
        2012/11/29  ‘blogligs.sty’, ‘markblog.sty’ ([ligs], [mark])
part of MOREHYPE RELEASE r0.7
v0.81   2012/12/20  \-, {enumtype}
        2013/01/02  caron, "Ligatures ..." (&aelig; etc.)
        2013/01/04  updating copyright
part of MOREHYPE RELEASE r0.81
v0.81a  2013/01/21  \newlet in subsubsection

```

4 “Pervasive Ligatures” with **blogligs.sty**

This is the code and documentation of the package mentioned in Sec. 3.2.7, loadable by option `[ligs]`. See below for what is offered.

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01] %% \newcommand* etc.
\ProvidesPackage{blogligs}[2012/11/29 v0.2
                           pervasive blog ligatures (UL)]
%% copyright (C) 2012 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below.
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3c of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
```

```
%%   http://www.contact-ednotes.sty.de.vu
%%
```

4.1 blog Required

blogdot is an extension of blog, and must be loaded *later* (but what about options? TODO):

```
\RequirePackage{blog}
```

4.2 Task and Idea

`\UseBlogLigs` as offered by blog.sty does not work inside macro arguments. You can use `\ParseLigs{`⟨*text*⟩`}` at such locations to enable “ligatures” again. blogligs.sty saves you from this manual trick. Many macros have one “text” argument only, others additionally have “attribute” arguments. Most macros ⟨*elt-cmd*⟩`{`⟨*text*⟩`}` of the first kind are defined to expand to `\SimpleTagSurr{`⟨*elt*⟩`}{`⟨*text*⟩`}` or to `\TagSurr{`⟨*elt*⟩`}{`⟨*attrs*⟩`}{`⟨*text*⟩`}` for some HTML element ⟨*elt*⟩ and some attribute assignments ⟨*attrs*⟩. When a macro in addition to a “text” element has “attribute” parameters, `\TagSurr` is used as well.

```
% \let\blogtextcolor\textcolor
% \renewcommand*{\textcolor}[2]{\blogtextcolor{#1}{\ParseLigs{#2}}}
```

4.3 Quotation Marks

“Inline quote” macros ⟨*qtd*⟩`{`⟨*text*⟩`}` to surround ⟨*text*⟩ by quotation marks do not follow this rule. We are just dealing with English and German double quotes that I have mostly treated by `catchdq.sty`. `"`⟨*text*⟩`"` then (eventually) expands to either `\deqtd{`⟨*text*⟩`}` or `\endqtd{`⟨*text*⟩`}`, so we redefine these:

```
\let\blogdedqtd\dedqtd
\renewcommand*{\dedqtd}[1]{\blogdedqtd{\ParseLigs{#1}}}

\let\blogendqtd\endqtd
\renewcommand*{\endqtd}[1]{\blogendqtd{\ParseLigs{#1}}}
```

4.4 HTML Elements

When the above rule holds:

```
\let\BlogTagSurr\TagSurr
\renewcommand*{\TagSurr}[3]{%
    \BlogTagSurr{#1}{#2}{\ParseLigs{#3}}}
\let\BlogSimpleTagSurr\SimpleTagSurr
\renewcommand*{\SimpleTagSurr}[2]{%
    \BlogSimpleTagSurr{#1}{\ParseLigs{#2}}}
```

4.5 Avoiding “Ligatures” though

`\noligs{⟨text⟩}` saves ⟨*text*⟩ from “ligature” replacements (except in arguments of macros inside ⟨*text*⟩ where blogligs enables ligatures):

```
\newcommand*{\noligs}{}     \let\noligs\@firstofone     %% !!!
```

I have found it useful to disable replacements within `\code{⟨text⟩}`:

```
\renewcommand*{\code}[1]{\STS{code}{\noligs{#1}}}
```

TODO: kind of mistake, `\STS` has not been affected anyway so far, then defining `\code` as `\STS{code}` should suffice.

`\NoBlogLigs` has been meant to disable “ligatures” altogether again. I am not sure about everything . . .

```
\renewcommand*{\NoBlogLigs}{%
    \def\BlogOutputJob{LEAVE}%
%     \let\deqtd\blogdeqtd                       %% rm. 2012/06/03
    \let\TagSurr\BlogTagSurr
    \let\SimpleTagSurr\BlogSimpleTagSurr
    \FDnormalTilde
    \MakeActiveDef\~{&nbsp;}%                    %% TODO new blog cmd
}
```

TODO: `\UseBlogLigs` might be redefined likewise (in fact blogligs activates ligatures inside text arguments unconditionally at present, I keep this for now since I have used it this way with `texblog.fdf` over months, and changing it may be dangerous where I have used tricky workarounds to overcome the `texblog.fdf` mistake). But with

```
\BlogInteceptEnvironments
```

this is not needed when you use `\NoBlogLigs` for the contents of some LaTeX environment.

4.6 The End and HISTORY

```
\endinput
```

VERSION HISTORY

```
v0.1    2012/01/08ff. developed in ‘texblog.fdf’
v0.2    2012/11/29    own file


```

5 Wiki Markup by **markblog.sty**

5.1 Introduction

This is the code and documentation of the package mentioned in Sec. 3.2.7, loadable by option `[mark]`. See below for what is offered. You should also find a file ‘`markblog.htm`’ that sketches it. Moreover, ‘`texlinks.pdf`’ describes in detail to what extent Wikipedia’s “piped links” with ‘`[[`⟨*wikipedia-link*⟩`]]`’ is supported.

5.2 Similar Packages

wiki.sty from the nicetext[5] bundle has offered some Wikipedia-like markup as a front-end for ordinary typesetting with LaTeX (for DVI/PDF), implemented in a way very different from what is going on here, rather converting markup sequences *during* typesetting.

More similar to the present approach is the way how Wikipedia section titles in package documentation is implemented by makedoc from the nicetext bundle, based on **preprocessing** by fifinddo.

In general, John MacFarlane’s pandoc (cf. German Wikipedia) converts between wiki-like (simplified) markup and LaTeX markup. (It deals with rather fixed markup rules, while we here process markup sequences independently of an entire markup *language*.)

Another straightforward and well-documented way to *preprocess* source files for converting simplified markup into TeX markup is Paul Isambert’s interpreter. It relies on LuaTeX where Lua does the preprocessing.

5.3 Package File Header

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01] %% \newcommand* etc.
\ProvidesPackage{markblog}[2012/11/29 v0.2
                           wiki markup with blog.sty (UL)]
%% copyright (C) 2012 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below.
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3c of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
%%   http://www.contact-ednotes.sty.de.vu
```

---

[5] `http://www.ctan.org/pkg/nicetext`

```
%%
```

5.4 blog Required

blogdot is an extension of blog and must be loaded *later* (but what about options? TODO):

```
\RequirePackage{blog}
```

5.5 Replacement Rules

2012/01/06f.:

```
\FDpseudoTilde
```

`[[`⟨*wikipedia-link*⟩`]]`: a fifinddo job is defined that passes to the “ligature” job for arrows in blog.sty:

```
\MakeExpandableAllReplacer{blog[[}{[[}{\protect\catchdbrkt}{blog<-}
\def\catchdbrkt#1]]{\Wikiref{#1}}                 %% + t 2012/01/09
```

The stars are inspired by Markdown (thanks to Uwe Ziegenhagen October 2011), while I have own ideas about them.

```
\MakeExpandableAllReplacer{blog**}{**}
                                  {\protect\doublestar:}{blog[[}
\MakeExpandableAllReplacer{blog***}{***}
                                  {\protect\triplestar:}{blog**}
% \CopyFDconditionFromTo{blog***}{BlogLIGs}
```

Apostrophes:

```
\MakeActiveDef\'{\noexpand'}
\MakeExpandableAllReplacer{blog\string'\string'}{''}
                   {\protect\doubleapostr:}{blog***}
\MakeExpandableAllReplacer{blog\string'\string'\string'}{'''}
               {\protect\tripleapostr:}{blog\string'\string'}
\MakeOther\'
```

Replacing three apostrophes by ‘`\tripleapostr`’ becomes the first job called with ‘`\UseBlogLigs`’:

```
\CopyFDconditionFromTo{blog'''}{BlogLIGs}
```

5.6 Connecting to LaTeX commands

`\MakePairLaTeXcmd#1#2` replaces ‘`#1`⟨*text*⟩`#1`’ by ‘`#2{`⟨*text*⟩`}`’:

```
\newcommand*{\MakePairLaTeXcmd}[2]{%
    \@ifdefinable#1{\def#1:##1#1:{#2{##1}}}}
    %% ":" for "..." 2012/01/30
```

`**⟨text⟩**` is turned into ‘`\mystrong{`⟨*text*⟩`}`’, and `***⟨text⟩***` is turned into ‘`\myalert{`⟨*text*⟩`}`’. I have used two shades of red for them:

```
\MakePairLaTeXcmd\doublestar\mystrong
\MakePairLaTeXcmd\triplestar\myalert
```

As in editing Wikipedia, `''⟨text⟩''` renders ⟨*text*⟩ in italics (or *slanted*), and `'''⟨text⟩'''` renders ⟨*text*⟩ bold.

```
\MakePairLaTeXcmd\doubleapostr\textit
\MakePairLaTeXcmd\tripleapostr\textbf
```

5.7 The End and HISTORY

```
\endinput
```

VERSION HISTORY

```
v0.1    2012/01/06ff. developed in ‘texblog.fdf’
v0.2    2012/11/29    own file

```

6 Real Web Pages with lnavicol.sty

This is the code and documentation of the package mentioned in Sec. 2.2.

```
\ProvidesPackage{lnavicol}[2011/10/13
                           left navigation column with blog.sty]
%%
%% Copyright (C) 2011 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below --
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3c of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
%%   http://www.contact-ednotes.sty.de.vu
%%
```

6.1 blog.sty Required

—but what about options (TODO)?

```
\RequirePackage{blog}
```

6.2 Switches

There is a "standard" page width and a "tight one" (the latter for contact forms)—`\iftight`:

```
\newif\iftight
```

In order to move an anchor to the *top* of the screen when the anchor is near the page end, the page must get some extra length by adding empty space at its bottom—`\ifdeep`:

```
\newif\ifdeep
```

6.3 Page Style Settings (to be set locally)

```
% \newcommand*{\pagebgcolor}{\#f5f5f5}  %% CSS whitesmoke
% \newcommand*{\pagespacing}{\@cellpadding{4} \@cellspacing{7}}
% \newcommand*{\pagenavicolwidth}{125}
% \newcommand*{\pagemaincolwidth}{584}
% \newcommand*{\pagewholewidth}  {792}
```

6.4 Possible Additions to **blog.sty**

6.4.1 Tables

`\begin{spancolscell}{⟨number⟩}{⟨style⟩}` opens an environment that contains a row and a single cell that will span ⟨*number*⟩ table cells and have style ⟨*style*⟩:

```
\newenvironment{spancolscell}[2]{%
    \starttr\startTd{\@colspan{#1} #2 %
                     \@width{100\%}}% %% TODO works?
    }{\endTd\endtr}
```

The `{hiddencells}` einvironment contains cells that do not align with other cells in the surrounding table. The purpose is using cells for horizontal spacing.

```
\newenvironment{hiddencells}
    {\startTable{}\starttr}
    {\endtr\endTable}
```

`{pagehiddencells}` is like `{hiddencells}` except that the HTML code is indented:

```
\newenvironment{pagehiddencells}
    {\indentii\hiddencells}
    {\indentii\endhiddencells}
```

`\begin{FixedWidthCell}{⟨width⟩}{⟨style⟩}` opens the `{FixedWidthCell}` environment. The content will form a cell of width ⟨*width*⟩. ⟨*style*⟩ are additional formatting parameters:

```
\newenvironment{FixedWidthCell}[2]
    {\startTd{#2}\startTable{\@width{#1}}%
     \starttr\startTd{}}
    {\endTd\endtr\endTable\endTd}
```

`\tablehspace{`⟨*width*⟩`}` is a variant of LaTeX's `\hspace{`⟨*glue*⟩`}`. It may appear in a table row:

```
\newcommand*{\tablehspace}[1]{\startTd{\@width{#1} /}}
```

6.4.2 Graphics

The command names in this section are inspired by the names in the standard LaTeX graphics package. (They may need some re-organization TODO.)

`\simpleinclgrf{`⟨*file*⟩`}` embeds a graphic file ⟨*file*⟩ without the tricks of the remaining commands.

```
\newcommand*{\simpleinclgrf}[1]{\IncludeGrf{alt="" \@border{0}}%
                                           {#1}}
```

`\IncludeGrf{`⟨*style*⟩`}{`⟨*file*⟩`}` embeds a graphic file ⟨*file*⟩ with style settings ⟨*style*⟩:

```
\newcommand*{\IncludeGrf}[2]{<img #1 src="#2">}
```

`\includegraphic{`⟨*width*⟩`}{`⟨*height*⟩`}{`⟨*file*⟩`}{`⟨*border*⟩`}{`⟨*alt*⟩`}{`⟨*tooltip*⟩`}` . . . :

```
\newcommand*{\includegraphic}[6]{%
    \IncludeGrf{%
        \@width{#1} \@height{#2} %% data; presentation:
        \@border{#4}
        alt="#5" \@title{#6}}%
        {#3}}
```

`\insertgraphic{`⟨*wd*⟩`}{`⟨*ht*⟩`}{`⟨*f*⟩`}{`⟨*b*⟩`}{`⟨*align*⟩`}{`⟨*hsp*⟩`}{`⟨*vsp*⟩`}{`⟨*alt*⟩`}{`⟨*t*⟩`}` adds ⟨*hsp*⟩ for the `@hspace` and ⟨*vsp*⟩ for the `@vspace` attribute:

```
\newcommand*{\insertgraphic}[9]{%
    \IncludeGrf{%
        \@width{#1} \@height{#2} %% data; presentation:
        \@border{#4}
        align="#5" hspace="#6" vspace="#8"
        alt="#8" \@title{#9}}%
        {#3}}
```

`\includegraphic{`⟨*wd*⟩`}{`⟨*ht*⟩`}{`⟨*file*⟩`}{`⟨*anchor*⟩`}{`⟨*border*⟩`}{`⟨*alt*⟩`}{`⟨*tooltip*⟩`}` uses an image with `\includegraphic` parameters as a link to ⟨*anchor*⟩:

```
\newcommand*{\inclgrfref}[7]{%
    \fileref{#4}{\includegraphic{#1}{#2}{#3}%
                                {#5}{#6}{#7}}}
```

6.4.3 HTTP/Wikipedia tooltips

`\httptipref{⟨tip⟩}{⟨www⟩}{⟨text⟩}` works like `\httpref{⟨www⟩}{⟨text⟩}` except that ⟨*tip*⟩ appears as "tooltip":

```
\newcommand*{\httptipref}[2]{%
  \TagSurr a{\@title{#1}\@href{http://#2}\@target@blank}}
```

`\@target@blank` abbreviates the `@target` setting for opening the target in a new window or tab:

```
\newcommand*{\@target@blank}{target="_blank"}
```

`\wikitipref{⟨lc⟩}{⟨lem⟩}{⟨text⟩}` works like `\wikiref{⟨lc⟩}{⟨lem⟩}{⟨text⟩}` except that "Wikipedia" appears as "tooltip". `\wikideref` and `\wikienref` are redefined to use it:

```
\newcommand*{\wikitipref}[2]{%
    \httptipref{Wikipedia}{#1.wikipedia.org/wiki/#2}}
\renewcommand*{\wikideref}{\wikitipref{de}}
\renewcommand*{\wikienref}{\wikitipref{en}}
```

6.5 Page Structure

The body of the page is a table of three rows and two columns.

6.5.1 Page Head Row

`\PAGEHEAD` opens the head row and a single cell that will span the two columns of the second row.

```
\newcommand*{\PAGEHEAD}{%
  \startTable{%
    \@align@c\
    \@bgcolor{\pagebgcolor}%
    \@border{0}%%                       %% TODO local
    \pagespacing
    \iftight \else \@width\pagewholewidth \fi
  }\CLBrk
  %% omitting <tbody>
  \ \comment{ HEAD ROW }\CLBrk
  \indenti\spancolscell{2}{}%
}
% \newcommand*{\headgrf}  [1]{%                     %% rm. 2011/10/09
%     \indentiii\simplecell{\simpleinclgrf{#1}}}
% \newcommand*{\headgrfskiptitle}[3]{%
%   \pagehiddencells
%     \headgrf{#1}\CLBrk
%     \headskip{#2}\CLBrk
%     \headtitle1{#3}\CLBrk
%   \endpagehiddencells}
```

`\headuseskiptitle{⟨grf⟩}{⟨skip⟩}{⟨title⟩}` first places ⟨*grf*⟩, then skips horizontally by ⟨*skip*⟩, and then prints the page title as `<h1>`:

```
\newcommand*{\headuseskiptitle}[3]{%
  \pagehiddencells\CLBrk
    \indentiii\simplecell{#1}\CLBrk
    \headskip{#2}\CLBrk
    \headtitle1{#3}\CLBrk
  \endpagehiddencells}
```

`\headskip{⟨skip⟩}` is like `\tablehspace{`⟨*skip*⟩`}` except that the HTML code gets an indent.

```
\newcommand*{\headskip}    {\indentiii\tablehspace}
```

Similarly, `\headtitle{⟨digit⟩}{⟨text⟩}` is like `\heading`⟨*digit*⟩`{`⟨*text*⟩`}` apart from an indent and being put into a cell:

```
\newcommand*{\headtitle}[2]{\indentiii\simplecell{\heading#1{#2}}}
```

6.5.2 Navigation and Main Row

`\PAGENAVI` closes the head row and opens the "navigation" column, actually including an `{itemize}` environment. Accordingly, `writings.fdf` has a command `\fileitem`. But it seems that I have not been sure ...

```
\newcommand*{\PAGENAVI}{%
    \indenti\endspancolscell\CLBrk
    \indenti\starttr\CLBrk
    \ \comment{NAVIGATION COL}\CLBrk
    \indentii\FixedWidthCell\pagenavicolwidth
                             {\@class{paper}
```

← using `@class`=`paper` here is my brother's idea, not sure about it ...

```
                             \@valign@t}
    %% omitting '\@height{100\%}'
    \itemize}
```

`\PAGEMAINvar{⟨width⟩}` closes the navigation column and opens the "main content" column. The latter gets width ⟨*width*⟩:

```
\newcommand*{\PAGEMAINvar}[1]{%
    \indentii\enditemize\ \endFixedWidthCell\CLBrk
    \ \comment{ MAIN COL }\CLBrk
    \indentii\FixedWidthCell{#1}{}}
```

... The width may be specified as `\pagemaincolwidth`, then `\PAGEMAIN` works like `\PAGEMAINvar{\pagemaincolwidth}`:

```
\newcommand*{\PAGEMAIN}{\PAGEMAINvar\pagemaincolwidth}
```

6.5.3 Footer Row

`\PAGEFOOT` closes the "main content" column as well as the second row, and opens the footer row:

```
\newcommand*{\PAGEFOOT}{%
    \indentii\endFixedWidthCell\CLBrk
%     \indentii\tablehspace{96}\CLBrk %% vs. \pagemaincolwidth
  %% <- TODO margin right of foot
    \indenti\endtr\CLBrk
    \ \comment{ FOOT ROW / }\CLBrk
    \indenti\spancolscell{2}{\@class{paper} \@align@c}%
```

← again class "paper"!?

```
}
```

`\PAGEEND` closes the footer row and provides all the rest ... needed?

```
\newcommand*{\PAGEEND}{\indenti\endspancolscell\endTable}
```

6.6 The End and HISTORY

```
\endinput

HISTORY

2011/04/29   started (? \if...)
2011/09/01   to CTAN as ‘twocolpg.sty’
2011/09/02   renamed
2011/10/09f. documentation more serious
2011/10/13   ‘...:’ OK

```

7 Beamer Presentations with blogdot.sty

7.1 Overview

blogdot.sty extends blog.sty in order to construct "HTML slides." One "slide" is a 3×3 table such that

1. it **fills** the computer **screen**,
2. the center cell is the **"type area,"**
3. the "margin cell" below the center cell is a **link** to the **next** "slide,"
4. the lower right-hand cell is a **"restart"** link.

Six **size parameters** listed in Sec. 7.4 must be adjusted to the screen in `blogdot.cfg` (or in a file with project-specific definitions).

We deliver a file `blogdot.css` containing **CSS** font size declarations that have been used so far; you may find better ones or ones that work better with your screen size, or you may need to add style declarations for additional HTML elements.

Another parameter that the user may want to modify is the **“restart” anchor name** `\BlogDotRestart` (see Sec. 7.6). Its default value is `START` for the “slide” opened by the command `\titlescreenpage` that is defined in Sec. 7.5.

That slide is meant to be the “**title** slide” of the presentation. In order to **display** it, I recommend to make and use a **link** to `START` somewhere (such as with blog.sty’s `\ancref` command). The *content* of the title slide is *centered* horizontically, so certain commands mentioned *below* (centering on other slides) may be useful.

After `\titlescreenpage`, the next main **user commands** are

`\nextnormalscreenpage{`⟨*anchor-name*⟩`}` starts a slide whose content is aligned flush left,

`\nextcenterscreenpage{`⟨*anchor-name*⟩`}` starts a slide whose content is centered horizontally.

—cf. Sec. 7.7. Right after these commands, as well as right after `\titlescreen‘\-page’`, code is used to generate the content of the **type area** of the corresponding slide. Another `\next...` command closes that content and opens another slide. The presentation (the content of the very last slide) may be finished using `\screenbottom{`⟨*final*⟩`}` where ⟨*final*⟩ may be arbitrary, or `START` may be a fine choice for ⟨*final*⟩.

Finally, there are user commands for **centering** slide content horizontically (cf. Sec. 7.8):

`\cheading{`⟨*digit*⟩`}{`⟨*title*⟩`}` “printing” a heading centered horizontically—even on slides whose remaining content is aligned *flush left* (I have only used ⟨*digit*⟩=2 so far),

`\begin{textblock}{`⟨*width*⟩`}` “printing” the content of a `{textblock}` environment with maximum line width ⟨*width*⟩ flush left, while that “block” as a whole may be centered horizontically on the slide due to choosing `\nextcenterscreenpage`—especially for **list** environments with entry lines that are shorter than the type area width and thus would not look centered (below a centered heading from `\cheading`).

The so far single **example** of a presentation prepared using blogdot is `dantev45.htm` (fifinddo-info bundle), a sketch of applying fifinddo to package documentation and HTML generation. A “driver” file is needed for generating the HTML code for the presentation from a `.tex` source by analogy to generating any HTML file using blog.sty. For the latter purpose, I have named my driver files `makehtml.tex`. For `dantev45.htm`, I have called that file `makedot.tex`, the main difference to `makehtml.tex` is loading `blogdot.sty` in place of `blog.sty`.

This example also uses a file `dantev45.fdf` that defines some commands that may be more appropriate as user-level commands than the ones presented here (which may appear to be still too low-level-like):

`\teilpage{⟨number⟩}{⟨title⟩}` making a "cover slide" for announcing a new "part" of the presentation in German,

`\labelsection{⟨label⟩}{⟨title⟩}` starting a slide with heading ⟨*title*⟩ and with anchor ⟨*label*⟩ (that is displayed on clicking a *link* to ⟨*label*⟩)—using

`\nextnormalscreenpage{⟨label⟩}` and `\cheading2{⟨title⟩}`,

`\labelcentersection{⟨label⟩}{⟨title⟩}` like the previous command except that the slide content will be *centered* horizontally, using

`\nextcenterscreenpage{⟨title⟩}`.

Reasons to make HTML presentations may be: (i) As opposed to office software, this is a transparent light-weight approach. Considering *typesetting* slides with TeX, (ii) TeX's advanced typesetting abilities such as automatical page breaking are not very relevant for slides; (iii) a typesetting run needs a second or a few seconds, while generating HTML with blog.sty needs a fraction of a second; (iv) adjusting formatting parameters such as sizes and colours needed for slides is somewhat more straightforward with HTML than with TeX.

Limitations: First I was happy about how it worked on my netbook, but then I realized how difficult it is to present the "slides" "online." Screen sizes (centering) are one problem. (Without the "restart" idea, this might be much easier.) Another problem is that the "hidden links" don't work with Internet Explorer as they work with Firefox, Google Chrome, and Opera. And finally, in internet shops some HTML entities/symbols were not supported. In any case I (again) became aware of the fact that HTML is not as **"portable"** as PDF.

Some **workarounds** are described in Sec. 7.9. `\FillBlogDotTypeArea` has two effects: (i) providing an additional link to the *next* slide for MSIE, (ii) *widening* and centering the *type area* on larger screens than the one which the presentation originally was made for. An optional argument of `\TryBlogDotCFG` is offered for a `.cfg` file overriding the original settings for the presentation. Using it, I learnt that for "portability," some manual line breaks (`\\`, `<br>`) should be replaced by "ties" between the words *after* the intended line break (when the line break is too ugly in a wider type area). For keeping the original type area width on wider screens (for certain "slides", perhaps when line breaks really are wanted to be preserved), the `{textblock}` environment may be used. Better HTML and CSS expertise may eventually lead to better solutions.

The **name** 'blogdot' is a "pun" on the name of the powerdot package (which in turn refers to "PowerPoint").

7.2 File Header

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01] %% \newcommand* etc.
\ProvidesPackage{blogdot}[2013/01/22 v0.41b HTML presentations (UL)]
%% copyright (C) 2011 Uwe Lueck,
%% http://www.contact-ednotes.sty.de.vu
%% -- author-maintained in the sense of LPPL below.
%%
%% This file can be redistributed and/or modified under
%% the terms of the LaTeX Project Public License; either
%% version 1.3c of the License, or any later version.
%% The latest version of this license is in
%%     http://www.latex-project.org/lppl.txt
%% We did our best to help you, but there is NO WARRANTY.
%%
%% Please report bugs, problems, and suggestions via
%%
%%   http://www.contact-ednotes.sty.de.vu
%%
```

7.3 blog Required

blogdot is an extension of blog (but what about options? TODO):

```
\RequirePackage{blog}
```

7.4 Size Parameters

I assume that it is clear what the following six page dimension parameters

`\leftpagemargin`, `\rightpagemargin`, `\upperpagemargin`, `\lowerpagemargin`, `\typeareawidth`, `\typeareaheight`

mean. The choices are what I thought should work best on my 1024×600 screen (in fullscreen mode); but I had to optimize the left and right margins experimentally (with Mozilla Firefox 3.6.22 for Ubuntu canonical - 1.0). It seems to be best when the horizontal parameters together with what the brouswer adds (scroll bar, probably 32px with me) sum up to the screen width.

```
\newcommand*{\leftpagemargin}{176}
\newcommand*{\rightpagemargin}{\leftpagemargin}
```

So `\rightpagemargin` ultimately is the same as `\leftpagemargin` as long as you don't redefine it, and it suffices to `\renewcommand` `\leftpagemargin` in order to get a horizontically centered type area with user-defined margin widths.—Something analogous applies to `\upperpagemargin` and `\lowerpagemargin`:

```
\newcommand*{\upperpagemargin}{80}
\newcommand*{\lowerpagemargin}{\upperpagemargin}
```

A difference to the “horizontal” parameters is (I expect) that the position of the type area on the screen is affected by `\upperpagemargin` only, and you may choose `\lowerpagemargin` just large enough that the next slide won’t be visible on any computer screen you can think of.

```
\newcommand*{\typeareawidth}{640}
\newcommand*{\typeareaheight}{440}
```

Centering with respect to web page body may work better on different screens (2011/10/03), but it doesn’t work here (2011/10/04).

```
% \renewcommand*{\body}{%
%     </head>\CLBrk
%     <body \@bgcolor{\bodybgcolor} \@align@c>}
```

`\CommentBlogDotWholeWidth` procuces no HTML code …

```
\global\let\BlogDotWholeWidth\@empty
```

… unless calculated with `\SumBlogDotWidth`:

```
\newcommand*{\SumBlogDotWidth}{%
    \relax{%                        %% \relax 2011/10/22 magic ...
    \count@\typeareawidth
    \advance\count@ \leftpagemargin
    \advance\count@\rightpagemargin
    \typeout{ * blogdot slide width = \the\count@\space*}%
    \xdef\CommentBlogDotWholeWidth{%
        \comment{ slide width = \the\count@\ }}}}
```

7.5 (Backbone for) Starting a “Slide”

`\startscreenpage{⟨style⟩}{⟨anchor-name⟩}`

```
\newcommand*{\startscreenpage}[2]{%% 0 2011/09/25!?:
    \\\CLBrk                                %% 2012/11/19
```

← `\\` suddenly necessary, likewise in `texblog.fdf` with `\NextView` and `\nextruleview`. Due to recent `firefox`?

```
    \startTable{%
        \@cellpadding{0} \@cellspacing{0}%
        \maybe@blogdot@borders              %% 2011/10/12
        \maybe@blogdot@frame                %% 2011/10/14
    }%
    \CLBrk                                  %% 2011/10/03
    \starttr
```

First cell determines both height of upper page margin `\upperpagemargin` and width of left page margin `\leftpagemargin`:

```
      \startTd{\@width {\leftpagemargin }%
               \@height{\upperpagemargin}}%
%          \textcolor{\bodybgcolor}{XYZ}%
      \endTd
```

Using `\typeareawidth`:

```
%       \startTd{\@width{\typeareawidth}}\endTd
      \simplecell{%
        \CLBrk
        \hanc{#2}{\hvspace{\typeareawidth}%
                          {\upperpagemargin}}%
        \CLBrk
      }%
```

Final cell of first row determines right margin width:

```
      \startTd{\@width{\leftpagemargin}}\endTd
    \endtr
    \starttr
    \emptycell\startTd{\@height{\typeareaheight}#1}%
}
```

`\titlescreenpage` (`\STARTscreenpage` TODO?) opens the title page (I thought). To get it to your screen, (make and) click a link like

> `\ancref{START}{start␣presentation}` :

```
\newcommand*{\titlescreenpage}{%
    \startscreenpage{\@align@c}{START}}
```

7.6 Finishing a "Slide" and "Restart" (Backbone)

`\screenbottom{`⟨*next-anchor*⟩`}` finishes the current slide and links to the ⟨*next-anchor*⟩, the anchor of a slide opened by

> `\startscreenpage{`⟨*style*⟩`}{`⟨*next-anchor*⟩`}`.

More precisely, the margin below the type area is that link. The corner at its right is a link to the anchor to whose name `\BlogDotRestart` expands.

```
\newcommand*{\screenbottom}[1]{%
    \ifFillBlogDotTypeArea
      <p>\ancref{#1}{\BlogDotFillText}%    %% not </p> 2011/10/22
    \fi
    \endTd\emptycell
    \endtr
    \CLBrk
    \tablerow{bottom margin}%                       %% 2011/10/13
      \emptycell
      \CLBrk
      \startTd{\@align@c}%
        \ancref{#1}{\HVspace{\BlogDotBottomFill}%
```

← seems to be useless now (2011/10/15).

```
                            {\typeareawidth}%
                            {\lowerpagemargin}}%
      \endTd
      \CLBrk
      \simplecell{\ancref{\BlogDotRestart}%
                         {\hvspace{\rightpagemargin}%
                                  {\lowerpagemargin}}}%
    \endtablerow
    \CLBrk
    \endTable
}
```

The default for `\BlogDotRestart` is `START`—the title page. You can `\renewcommand` it so you get to a slide containing an overview of the presentation.

```
\newcommand*{\BlogDotRestart}{START}
```

7.7 Moving to Next "Slide" (User Level)

`\nextscreenpage{⟨style⟩}{⟨anchor-name⟩}` puts closing the previous slide and opening the next one—having anchor name ⟨*anchor-name*⟩—together. ⟨*style*⟩ is for style settings for the next page, made here for choosing between centering the page/slide content and aligning it flush left.

```
\newcommand*{\nextscreenpage}[2]{%
    \screenbottom{#2}\CLBrk
    \hrule           \CLBrk
    \startscreenpage{#1}{#2}}
```

`\nextcenterscreenpage{⟨anchor-name⟩}` chooses centering the slide content:

```
\newcommand*{\nextcenterscreenpage}{\nextscreenpage{\@align@c}}
```

`\nextnormalscreenpage{⟨anchor-name⟩}` chooses flush left on the type area determined by `\typeareawidth`:

```
\newcommand*{\nextnormalscreenpage}{\nextscreenpage{}}
```

7.8 Constructs for Type Area

If you want to get centered titles with `<h2>` etc., you should declare this in `.css` files. But you may consider this way too difficult, and you may prefer to declare this right in the HTML code. That's what I do! I use `\cheading{⟨digit⟩}{⟨text⟩}` for this purpose.

```
\newcommand*{\cheading}[1]{\CLBrk\TagSurr{h#1}{\@align@c}}
```

`\begin{textblock}{⟨width⟩}` opens a `{textblock}` environment. The latter will contain text that will be flush left in a narrower text area—of width ⟨*width*⟩—than the one determined by `\typeareawidth`. It may be used on ”centered” slides. It is made for lists whose entries are so short that the page would look unbalanced under a centered title with the list adjusted to the left of the entire type area. (Thinking of standard LaTeX, it is almost the `{minipage}` environment, however lacking the footnote feature, in that respect it is rather similar to `\parbox` which however is not an environment.)

```
\newenvironment*{textblock}[1]
    {\startTable{\@width{#1}}\starttr\startTd{}}
    {\endTd\endtr\endTable}
```

7.9 Debugging and `.cfgs`

`\ShowBlogDotBorders` shows borders of the page margins and may be undone by `\DontShowBlogDotBorders`:

```
\newcommand*{\ShowBlogDotBorders}{%
    \def\maybe@blogdot@borders{rules="all"}}
\newcommand*{\DontShowBlogDotBorders}{%
    \let\maybe@blogdot@borders\@empty}
\DontShowBlogDotBorders
```

`\ShowBlogDotFrame` shows borders of the page margins and may be undone by `\DontShowBlogDotFrame`:

```
\newcommand*{\ShowBlogDotFrame}{%
    \def\maybe@blogdot@frame{\@frame@box}}
\newcommand*{\DontShowBlogDotFrame}{%
    \let\maybe@blogdot@frame\@empty}
\DontShowBlogDotFrame
```

However, the rules seem to affect horizontal positions ...

`\BlogDotFillText` is a dirty trick ... seems to widen the type area and this way centers the text on wider screens than the one used originally. Of course, this can corrupt intended line breaks.

```
\newcommand*{\BlogDotFillText}{%            %% 2011/10/11
    \center
        \BlogDotFillTextColor{%             %% 2011/10/12
%                 X\\X                      %% insufficient
                 X X X X X X X X X X X X X X X X X X X X
                 X X X X X X X X X X X X X X X X X X X X
                 X X X X X X X X X X
                 X X X X X X X X X X
%                  X X X X X X X X X X X X X X X X X X X X
        }
    \endcenter
}
```

`\FillBlogDotTypeArea` fills `\BlogDotFillText` into the type area, also as a link to the next slide. This may widen the type area so that the text is centered on wider screens than the one the HTML page was made for. The link may serve as an alternative to the bottom margin link (which sometimes fails). `\FillBlogDotTypeArea` can be undone by `\DontFillBlogDotTypeArea`:

```
\newcommand*{\FillBlogDotTypeArea}{%
    \let\ifFillBlogDotTypeArea\iftrue
    \typeout{ * blogdot filling type area *}}       %% 2011/10/13
\newcommand*{\DontFillBlogDotTypeArea}{%
    \let\ifFillBlogDotTypeArea\iffalse}
\DontFillBlogDotTypeArea
```

`\FillBlogDotBottom` fills `\BlogDotFillText` into the center bottom cell. I tried it before `\FillBlogDotTypeArea` and I am not sure . . . It can be undone by `\DontFillBlogDotBottom`:

```
\newcommand*{\FillBlogDotBottom}{%
    \let\BlogDotBottomFill\BlogDotFillText}
```

. . . actually, it doesn't seem to make a difference! (2011/10/13)

```
\newcommand*{\DontFillBlogDotBottom}{\let\BlogDotBottomFill\@empty}
\DontFillBlogDotBottom
```

`\DontShowBlogDotFillText` makes `\BlogDotFillText` invisible, `\ShowBlogDotFillText` makes it visible. Until 2011/10/22, `\textcolor` (blog.sty) used the `<font>` element that is deprecated. I still use it here because it seems to suppress the `hover` CSS indication for the link. (I might offer a choice—TODO)

```
\newcommand*{\DontShowBlogDotFillText}{%
%     \def\BlogDotFillTextColor{\textcolor{\bodybgcolor}}}
    \def\BlogDotFillTextColor{%
        \TagSurr{font}{color="\bodybgcolor"}}}
\newcommand*{\ShowBlogDotFillText}{%
    \def\BlogDotFillTextColor{\textcolor{red}}}
\DontShowBlogDotFillText
```

As of 2013/01/22, texlinks.sty provides `\ctanfileref{`⟨*path*⟩`}{`⟨*file-name*⟩`}` that uses an online TEX archive randomly chosen or determined by the user. This is preferable for an online version of the presentation. In `dantev45.htm`, this is used for example files. When, on the other hand, internet access during the presentation is bad, such example files may instead be loaded from the "current directory." `\usecurrdirctan` modifies `\ctanfileref` for this purpose (i.e., it will ignore ⟨*path*⟩):

```
\newcommand*{\usecurrdirctan}{%
    \renewcommand*{\ctanfileref}[2]{%
        \hnewref{}{##2}{\filenamefmt{##2}}}}
```

(Using a local TDS tree would be funny, but I don't have good idea for this right now.)

`\TryBlogDotCFG` looks for `blogdot.cfg`,

`\TryBlogDotCFG[⟨file-name-base⟩]`

looks for ⟨*file-name-base*⟩`.cfg` (for recompiling a certain file):

```
\newcommand*{\TryBlogDotCFG}[1][blogdot]{%
    \InputIfFileExists{#1.cfg}{%
        \typeout{
            * Using local settings from \string‘#1.cfg\string’ *}%
    }{}%
}
\TryBlogDotCFG
```

7.10 The End and HISTORY

```
\endinput
```

VERSION HISTORY

```
v0.1    2011/09/21f.  started
        2011/09/25    spacing/padding off
        2011/09/27    \CLBrk
        2011/09/30    \BlogDotRestart
        used for DANTE meeting
v0.2    2011/10/03    four possibly independent page margin
                      parameters; \hvspace moves to texblog.fdf
        2011/10/04    renewed \body commented out
        2011/10/07    documentation
        2011/10/08    added some labels
        2011/10/10    v etc. in \ProvidesPackage
        part of morehype RELEASE r0.5
v0.3    2011/10/11    \HVspace, \BlogDotFillText
        2011/10/12    commands for \BlogDotFillText
        2011/10/13    more doc. on "debugging";
                      \ifFillBlogDotTypeArea, \tablerow, messages
        2011/10/14    \maybe@blogdot@frame
        2011/10/15    doc. note: \HVspace useless
        part of morehype RELEASE r0.51
v0.4    2011/10/21    \usecurrdirctan
        2011/10/22    FillText with <p> instead of </p>, its color
                      uses <font>; some more reworking of doc.
        part of morehype RELEASE r0.6
v0.41   2012/11/19    \startscreenpage with \\; doc. \
        2012/11/21    updating version infos, doc. \pagebreak
v0.41a  2013/01/04    rm. \pagebreak
        part of morehype RELEASE r0.81
v0.41b  2013/01/22    adjusted doc. on ‘texlinks’

```