blogexec.sty

—

Overcoming blog.sty's Pure Expansion*

Uwe Lück†

December 20, 2012

Abstract

blog.sty before v0.7 generated HTML by pure macro expansion and could use LaTeX macros (redefined) only to a very limited extent. On adding blogexec.sty, some macros (configurable) are "intercepted" before expansion in a blog run for "running" some code, basically `\begin`, `\end`, and a new general `\EXECUTE`. A table environment with active characters inside only is provided—perhaps "nicer than LaTeX."

Contents

*This document describes version v0.21 of blogexec.sty as of 2012/12/20.

†`http://contact-ednotes.sty.de.vu`

1 Features and Usage

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

blogexec.sty may be loaded by

```
\RequirePackage{blogexec}
```

in a driver file for blog.sty. **Alternatively**, the following commands in a blog driver file (in a certain way even in a *source* file) load blogexec.sty and then are carried out according to their definitions in blogexec:

`\BlogInterceptExecute` intercepts `\EXECUTE` only.

`\BlogInterceptEnvironments` intercepts `\EXECUTE`, `\begin`, and `\end` only; the latter two then work much as with LATEX. They expand to HTML code as with blog; `\begin{`⟨*env*⟩`}` additionally executes commands according to an (optional)

`\MakeBlogBeginRun{`⟨*env*⟩`}`⟨*arguments*⟩`{`⟨*begin-code*⟩`}`.

`\BlogInterceptExtra` intercepts all the commands in certain lists (using the dowith package), including `\EXECUTE`, `\begin`, `\end`. E.g.,

`\MakeBlogOneArgInterception{`⟨*cmd*⟩`}{`⟨*run*⟩`}{`⟨*write*⟩`}`

adds ⟨*cmd*⟩ to such a list and tells that ⟨*run*⟩ should be carried out and that ⟨*cmd*⟩⟨*one-argument*⟩ should be replaced by ⟨*write*⟩ in a line containing ⟨*cmd*⟩ (not hidden in braces, and there better should not be much more in the line).

`\BlogInterceptHash` does *not* choose an **"interception level"** as the previous commands do, but may be necessary for allowing parameters in macro definitions to be run in the course of an interception. It is automatically ("implicitly") envoked by the star forms of the above commands, i.e., by either of

- `\BlogInterceptExecute*`
- `\BlogInterceptEnvironments*`
- `\BlogInterceptExtra*`

The reader may find additional details in the following sections near the code implementing the commands.

blogexec.sty also modifies blog.sty's (v0.7) `{stdallrulestable}` environment as follows:

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

`|` The vertical stroke becomes an active character that closes a table cell and opens another one (being an alias for blog.sty's v0.7 `\endcell`)—just as `&` does it with TEX/LATEX.

`&` The ampersand becomes an active character that—**differently to TEX/LATEX**—as an alias for blog.sty's `\figurespace` produces the Unicode figure space for alignment of figures.

Outside the `{stdallrules}` environment, both characters have their "usual" meaning, i.e., `&` may be used for accessing HTML entities (as blog.sty allows it). `\cr` (and `\endline`, provided by blog.sty v0.7) ends a table row and starts a new one. `\\` is not touched—a **difference to LATEX** and may still be used for breaking a line within a table cell.

2 Package File Header (Legalize)

```
1  \NeedsTeXFormat{LaTeX2e}[1994/12/01] %% \newcommand* etc.
2  \ProvidesPackage{blogexec}[2012/12/20 v0.21
3                             assignments with blog.sty (UL)]
4  %% copyright (C) 2011 Uwe Lueck,
5  %% http://www.contact-ednotes.sty.de.vu
6  %% -- author-maintained in the sense of LPPL below.
7  %%
8  %% This file can be redistributed and/or modified under
9  %% the terms of the LaTeX Project Public License; either
10 %% version 1.3c of the License, or any later version.
11 %% The latest version of this license is in
12 %%     http://www.latex-project.org/lppl.txt
13 %% We did our best to help you, but there is NO WARRANTY.
14 %%
15 %% Please report bugs, problems, and suggestions via
16 %%
17 %%   http://www.contact-ednotes.sty.de.vu
```

3 Requirements

The dowith package is needed for managing and running lists of macros to be intercepted:

```
18 \RequirePackage{dowith}
```

Admittedly, `\do` and `\@elt` lists (as discussed in `dowith.pdf`) would be faster than the dowith method, which might be relevant here (TODO: how much?). I may abandon dowith later, I just cannot afford removing it now (2011/11/05, TODO).

4 Processing Source Files

With `\BlogInterceptExtra`, blog.sty deals with *empty* input lines just like

`\BlogCopyFile[`⟨*changes*⟩`]{`⟨*src-file*⟩`}`

does; *otherwise* the content of `\fdInputLine` is copied to `\fdOutputCode`. Before the latter is writen to the output file ⟨*output*⟩ (as determined by a recent `\ResultFile{`⟨*output*⟩`}`), `\BlogInterceptions` is run, its purpose is to extract assignment and other "execution" commands and to turn `\fdOutputCode` into an expandable macro. We use `\def` because blog.sty may have provided a preliminary definition earlier:

```
\def\blog@icl@xtra{%
    \let\BlogProcessLine\BlogAllowIntercepting
    \let\BlogInterceptions\AllBlogInterceptions}
\def\BlogInterceptExtra{\@ifstar@intercept@hash\blog@icl@xtra}
\def\@ifstar@intercept@hash#1{\@ifstar{#1\BlogInterceptHash}#1}
```

And this is the default setting (TODO!?):

```
\BlogInterceptExtra
```

Below, there are commands for restricted (faster—TODO: relevant? or less complex, to reduce danger) interception functionality. (Maybe the file should be restructured.) `\AllBlogInterceptions` first is nothing:

```
\InitializeListMacro\AllBlogInterceptions
```

—and should become more below.

`\BlogAllowIntercepting` stores the difference to blog.sty:

```
\newcommand*{\BlogAllowIntercepting}{%
    \let\fdOutputCode\fdInputLine
    \BlogInterceptions
```

When, after removing the intercepted command, the line is empty, it is *not* written into output:

```
    \ifx\fdOutputCode\@empty \else
        \WriteResult{%
            \ProcessExpandedWith\fdOutputCode\BlogOutputJob}%
```

. . . enabling "ligatures" with blog.sty v0.7.

```
    \fi}
```

. . . TODO: in fifinddo with something like `\fdInterceptions`?

Especially for storing file-specific macro definitions with `\EXECUTE` (below), a parameter character (usually hash mark) is needed. fifinddo.sty (so far—2011/11/20) does not include it with `\BasicNormalCatCodes`, and blog.sty does not include it with `\BlogCodes` —the following `\BlogInterceptHash` does. Moreover, `\MakeHashParameter` enables such definitions when placed in a source file within the argument of a separate(!) `\EXECUTE`.

```
\providecommand*{\MakeHashParameter}{\catcode`\#6 }
\def\BlogInterceptHash{%
    \ToListMacroAdd\BlogCodes\MakeHashParameter
    \MakeHashParameter}
```

TODO: default? 0-arg interception?

5 Intercepting Single-Parameter Commands

5.1 The General Method

Macros to be intercepted that have a single argument will be collected in `\blogOneArgInterceptions`:

```
\InitializeListMacro\blogOneArgInterceptions
\ToListMacroAdd\AllBlogInterceptions{%
    \DoWithAllIn\blogTryOneArgCmd
                \blogOneArgInterceptions}
```

Here `\blogTryOneArgCmd{`⟨*cmd*⟩`}` creates a "sandbox" for parsing in a similar way as fifinddo does it, searching for ⟨*cmd*⟩. The method there was made thinking of reading files with "plain text" category codes, not aware of blog.sty. Maybe this was a mistake, and I will reconsider it. There I also introduce a separate sandbox macro for each search pattern, thinking of different types of sandboxes. This is not done/needed here (strangely, TODO).—The sandbox starts with the parsing macro. The latter's name derives from ⟨*cmd*⟩ by prefixing something to its name. `\StripEsc` is a little helper for removing the backslash from a macro name.

```
\providecommand*{\StripEsc}{\expandafter\@gobble\string}
```

Name spaces:

```
\newcommand*{\blog@x}{\StripEsc\blogx}
\newcommand*{\blogTryOneArgCmd}[1]{%
    \csname \blog@x:\StripEsc#1\expandafter\endcsname
        \fdOutputCode \@gobble#1\@empty\@nil}
```

Here, `\@empty` is the dummy argument for ⟨*cmd*⟩—this is what must be modified for ⟨*cmd*⟩ with more than one parameter. At present (2011/11/05), that tail starting with `\@gobble` may stay at the end of `\fdOutputCode` for each interception per `\fdInputLine`, until it expands to nothing in the `\write`.

`\MakeBlogOneArgInterception{`⟨*cmd*⟩`}{`⟨*run*⟩`}{`⟨*write*⟩`}` says that when ⟨*cmd*⟩ is found in `\fdOutputCode`, ⟨*run*⟩ should be executed, and ⟨*cmd*⟩⟨*arg*⟩ should be replaced by ⟨*write*⟩ in `\fdOutputCode` where ⟨*arg*⟩ is the argument for ⟨*cmd*⟩ found in `\fdOutputCode`. Let ⟨*arc*⟩ be ⟨*arg*⟩ without delimiting braces if ⟨*arg*⟩ is `{`⟨*arc*⟩`}` (otherwise ⟨*arc*⟩ is the same as ⟨*arg*⟩). Then use `#2` for referring to ⟨*arc*⟩ inside ⟨*run*⟩ and ⟨*write*⟩. (Sorry, I cannot afford replacing `#2` by a more natural placeholder right now.)

```
\begingroup
  \catcode`\|\z@ |MakeOther|\%                    %% \z@ 2011/11/22
  |@ifdefinable|MakeBlogOneArgInterception{%
    |gdef|MakeBlogOneArgInterception#1#2#3{%
```

First we add ⟨*cmd*⟩ to `\blogOneArgInterceptions`, unless it is already there:

```
      |TestListMacroForToken|blogOneArgInterceptions#1%
      |ifin@
        |PackageWarning{blogexec}{Redeclaring |string#1.}%
      |else
        |ToListMacroAdd|blogOneArgInterceptions#1%
      |fi
```

Now the parsing macro is defined, together with the actions depending on the result:

```
      |@namedef{|blog@x:|StripEsc#1}##1#1##2##3|@nil{%
```

`#3` will be empty if and only if ⟨*cmd*⟩ does *not* occur in `\fdOutputCode`. A backslash made "other" will not occur in `\fdOutputCode`, therefore the following `\ifx` becomes true if and only if `#3` is empty, i.e., ⟨*cmd*⟩ does *not* occur in `\fdOutputCode`:

```
          |ifx\##3\%
```

In this case we just do nothing.

```
          |else
```

Otherwise, we apply ⟨*run*⟩ and ⟨*write*⟩:

```
              #2%
              |def|fdOutputCode{##1#3##3}%
          |fi}%
    }%
  }%
|endgroup
```

5.2 \EXECUTE

`\EXECUTE{`⟨*run*⟩`}` runs ⟨*run*⟩ and is removed from the output line:

```
\MakeBlogOneArgInterception\EXECUTE{#2}{}
```

You can **store settings** ⟨*set*⟩ for processing a source file in this file by `\EXECUTE{`⟨*set*⟩`}` (e.g., shorthand macros only useful in this single file). You even can switch off the interception functionality after running the other settings ⟨*set*⟩ by `\EXECUTE{`⟨*set*⟩`\BlogCopyLines}`.

`\EXECUTE{`⟨*run*⟩`}` may be a great relief thinking of pure expansion with blog.sty. You may be happy enough with it and *restrict* the interception functionality to `\EXECUTE` by `\BlogInterceptExecute`. Its definition may be a redefinition of the preliminary macro in blog.sty. (TODO: option for stopping here, avoid dowith.)

```
\def\blog@icl@exec{%
    \let\BlogProcessLine\BlogAllowIntercepting
    \def\BlogInterceptions{\blogTryOneArgCmd\EXECUTE}}
\def\BlogInterceptExecute{\@ifstar@intercept@hash\blog@icl@exec}
```

5.3 \begin and \end

At present (2011/11/06), only \begin{⟨*env*⟩} will run settings. Macros \⟨*env*⟩ and \end⟨*env*⟩ will expand in the .html as with blog.sty alone, not touched here. Settings to be *run* must be stored in a macro \blogx.b:⟨*env*⟩. If this has not been done, only \relax (from \csname) will be "run."

```
\MakeBlogOneArgInterception\begin{%
```

Indeed, we have a "modified selection" from LaTeX's original \begin:

```
    \@ifundefined{#2}%
      {\def\@tempa{\@latex@error{Environment #2 undefined}\@eha}}%
      {\def\@tempa{\def\@currenvir{#2}%
%        \edef\@currenvline{\on@line}%              %% not in source
         \csname \blog@x.b:#2\endcsname}}%  %% \StripEsc->: 2012/08/28
    \begingroup \@tempa}{%
    \csname #2\endcsname}
```

\MakeBlogBeginRun{⟨*env*⟩}⟨*args*⟩{⟨*begin-code*⟩}

resembles

\newenvironment*{⟨*env*⟩}⟨*args*⟩{⟨*begin-code*⟩}{⟨*end-code*⟩}

except that it does not have {⟨*end-code*⟩}:

```
\newcommand*{\MakeBlogBeginRun}{\@makeblogbeginrun\newcommand}
```

v0.2 allows redefinition by

\ChangeBlogBeginRun{⟨*env*⟩}⟨*args*⟩{⟨*begin-code*⟩}

```
\newcommand*{\@makeblogbeginrun}[2]{%
    \expandafter #1\expandafter *%
        \csname \blog@x.b:#2\endcsname}   %%  \StripEsc->: 2012/08/28
\newcommand*{\ChangeBlogBeginRun}{\@makeblogbeginrun\renewcommand}
```

Moreover, v0.2 allows copying that action by

\CopyBlogBeginRunTo{⟨*env*⟩}{⟨*enw*⟩}

```
\newcommand*{\CopyBlogBeginRunTo}[2]{%
    \withcsname \let \blog@x.b:#2\expandafter\endcsname
                    \csname \blog@x.b:#1\endcsname}
```

\end{⟨*env*⟩}:

```
\MakeBlogOneArgInterception\end{\@checkend{#2}\endgroup}{\end{#2}}
%    \expandafter\show\csname blogx:end\endcsname
```

`\BlogInterceptEnvironments`

restricts interception functionality to `\EXECUTE`, `\begin`, and `\end`:

```
\def\blog@icl@envs{%
    \BlogInterceptExecute
    \ToListMacroAdd\BlogInterceptions{%
        \blogTryOneArgCmd\begin\blogTryOneArgCmd\end}}
\def\BlogInterceptEnvironments{\@ifstar@intercept@hash\blog@icl@envs}
```

TODO: 1. imitate LaTeX's toggling with `\emph` (redefine it in italic environments) 2. code indenting (cf. inputtrc)

5.4 Skipping Source Code

The `{noblog}` environment "suppresses" TeX source code in the sense that it does not produce HTML code—while blog.sty's `{commentlines}` produces an HTML comment.

```
\newenvironment*{noblog}{}{}    %% 2012/03/04 from ...
\MakeBlogBeginRun{noblog}{%
    \BlogInterceptEnvironments  %% 2012/06/22
    \let\WriteResult\@gobble}
```

5.5 A Comfortable Table Environment

As an application of `\MakeBlogBeginRun` for blog.sty's `{stdallrulestable}`, we provide '`|`' as an active character invoking blog.sty's `\endcell` (move to next cell) and an active character '`&`' for `\figurespace`, i.e., a Unicode symbol for aligning figures. Indeed, we are *not* going back to LaTeX and Plain TeX by using `&` for moving to the next cell, I consider the present choice more intuitive.

```
\MakeBlogBeginRun{stdallrulestable}{%
    \MakeActiveDef\|{\endcell}\MakeActiveDef\&{\figurespace}}
```

I hope nobody will confuse `&` and `8`. A little drawback may be that you now can't use `&` for inserting HTML entities. However, recall that these settings are restricted to the `{stdrulestable}` environment, and that you can use `\MakeBlogBeginRun{stdallrulestable}` again for your own choice of shorthands. (TODO: `\MakeActiveLet`)

6 Intercepting Two-Parameter Macros

Here especially I have a macro `\labelsection{`⟨*label*⟩`}{`⟨*title*⟩`}` *in mind* (TODO). It could be handled by the one-argument approach by storing the first argument and inserting another macro that reads the second argument. Therefore I am not sure ... (2011/11/04)

7 Leaving and HISTORY

```
\endinput

      VERSION HISTORY

v0.1    2011/11/04  started; arrived at \EXECUTE
        2011/11/05  rm. \blogx@dummy, corrected loop,
                    \BlogInterceptExtra, \BlogInterceptExecute
        2011/11/06  \BlogAllowIntercepting, emptiness test
                    with "other" backslash, \begin/\end
        2011/11/07  debugging (\catcode... in \@ifdefinable);
                    warning on reusing interceptor,
                    \BlogInterceptEnvironments;
                    doc.: raise interception level in \EXECUTE
        2011/11/08  \BlogInterceptHash (understanding needed hours)
        2011/11/10  `v0.1' in \Provides..., doc. fix,
                    removing experimental code, doc. all 1-arg
                    interceptions in one section
        2011/11/20  \BlogInterceptHash improved
        2011/11/20  doc. `%' doubled
        2011/11/21  \BlogOutputJob
        2011/11/22  TODO + \z@ for \MakeBlogOne...
        2011/12/15  rm. TODO
v0.2    2012/08/28  \begin/\end revised (\StripEsc wrong)
        2012/08/29  \ChangeBlogBeginRun, \CopyBlogBeginRun,
                    \blog@x
v0.21   2012/12/20  {noblog}

```