commado.sty and filesdo.sty

—

Immediately Extend a One-Argument Macro to Comma-Separated Lists and Combinations of Filename Bases and Extensions*

Uwe Lück†

November 16, 2015

Abstract

commado.sty provides

`\DoWithCSL{`⟨*cmd*⟩`}{`⟨*list*⟩`}`

in order to apply an existing one-parameter macro ⟨*cmd*⟩ to each item in a list ⟨*list*⟩ in which items are separated by commas. filesdo.sty provides

`\DoWithBasesExts{`⟨*cmd*⟩`}{`⟨*bases*⟩`}{`⟨*exts*⟩`}`

in order to run ⟨*cmd*⟩`{`⟨*base*⟩`.`⟨*ext*⟩`}` for some (at most) one-parameter macro ⟨*cmd*⟩, each base filename ⟨*base*⟩ in the comma-separated list ⟨*bases*⟩ and each filename extension ⟨*ext*⟩ in the comma-separated list ⟨*exts*⟩. As opposed to LaTeX's internal `\@for`, no assignments are involved (unless ⟨*cmd*⟩ uses assignments—"expandability" in "TeX's gullet").

Both packages are "generic," i.e., should work with Plain TeX, LaTeX or even other formats, relying on the plainpkg package for some minimal LaTeX-like behaviour.

Related packages: loops and others mentioned in the dowith package documentation.

Keywords: macro programming, programming structures, loops, lists

*This document describes v0.11 of commado.sty as of 2012/11/30 and version v0.1 as of 2012/11/27 of filesdo.sty..

†`http://contact-ednotes.sty.de.vu`

Contents

1 Installing and Calling **2**

2 Syntax and Relation to the dowith Package **3**

3 Example for filesdo.sty **4**

4 The File commado.sty **5**
4.1 Package File Header (Legalese and plainpkg) 5
4.2 Auxiliaries 5
4.3 Processing a Comma-Separated List 5
4.4 Leaving the Package File 6
4.5 VERSION HISTORY 6

5 The File filesdo.sty **7**
5.1 Package File Header (Legalese and plainpkg) 7
5.2 Documentation 7
5.3 commado Required 7
5.4 Category Code 7
5.5 User Commands 7
5.6 Leaving the Package File 8
5.7 VERSION HISTORY 8

1 Installing and Calling

The files commado.sty and filesdo.sty are provided ready, installation just requires putting them somewhere where TEX finds them (which may need updating the filename data base).[1] However, installation of the package plainpkg[2] and the package stacklet.sty (catcodes[3] bundle) is required additionally.

As to calling (loading): commado.sty and filesdo.sty are "plainpkg packages" in the sense of the plainpkg documentation that you may consult for details. So roughly,

- load it by `\usepackage{`⟨*pkg*⟩`}` if you can,
- otherwise by `\RequirePackage{`⟨*pkg*⟩`}` (perhaps from within another "plainpkg package"),
- or by `\input␣`⟨*pkg*⟩`.sty`
- or even by `\input{`⟨*pkg*⟩`.sty}` ...

—where ⟨*pkg*⟩ is '`commado`' or '`filesdo`'.

[1] http://www.tex.ac.uk/FAQ-inst-wlcf.html
[2] http://ctan.org/pkg/plainpkg
[3] http://ctan.org/pkg/catcodes

2 Syntax and Relation to the dowith Package

In ⟨*list*⟩ with `\DoWithCSL{⟨cmd⟩}{⟨list⟩}`, blank spaces before entries or after commas as well as preceding the closing brace are ignored. So

```
\DoWithCSL{⟨cmd⟩}{␣cfg,␣sty,␣tex␣}
```

works like

```
\DoWithAllOf{⟨cmd⟩}{{cfg}{sty}{tex}}
```

from dowith.sty.[4] With `\DoWithCSL` (at present), an item cannot be empty or consist of blank spaces only. Empty or blank items can be handled by `\DoWithAllOf`.

As to **relevance**, the previous remark addresses those who want to understand what I am doing, such as me. The two commands provide a rather obvious "link" between the commado and the dowith bundle. But neither of them is a command that "you shouldn't miss." commado.sty is mainly a programming tool (and I isolate it as an object of study).

Choosing between dowith and commado. What I **really use** in the case of comma-separated lists is **filesdo.sty** as described in the next section. It is great to keep certain files or file sections small (**readability**), actually in describing package bundles, on CTAN as well as with a complex book at which I am working. However, the comma feature is a little more complex than the dowith way, and I like to avoid unnecessarily complex things. The real advantage of filesdo, as I feel it, is the readability of the code with combinations of file *basenames* and *extensions*, similar to *brace expansion* in the Bash shell. The braces make obvious which are the basenames and which are the extensions, and the commas structure each of these two lists.

In applications of **dowith**, saving **tokens** and **expansion** steps is more important to me (a kind of sports). *Tokens* actually are saved with dowith when ⟨*list*⟩ consists of single tokens rather than "**brace groups**," and the former is the application of dowith in my langcode[5] package (in the dowith discussion, I compare this with the xspace[6] package). On the other hand, a ⟨*list*⟩ of brace groups even needs more tokens than a comma-separated list. The number of tokens or expansion steps is relevant when a list is stored as a macro or in a token register. It is less relevant when a list is processed once only at the moment the input file is read and then is not stored any longer.

There is a situation where I prefer lists of *brace groups* to comma-separated lists although the former need more tokens and their readablility is worse: I use something like

```
\autrefs{{apples}{oranges}{pears}}                    (1)
```

to generate a list of internal links in a HTML file. `\autrefs` this way rests on `\DoSeparateWith` from domore.sty. `\DoSeparateWith` is based on `\DoWith`. Sometimes I use the former one directly as an *author*. The alternative

```
\autrefs{apples,oranges,pears}                        (2)
```

would need less tokens (which is absolutely irrelevant in this case because the line is in TEX's memory for a moment only) and may be easier to read. However, often I want to change the order of the items. When I try this by cut&paste in a comma-separated list, I always wonder whether I should cut/copy the right-hand comma of a list item or the left-hand comma; and at the next moment, I have forgotten whether I cut the right-hand comma or the other one. With the dowith approach, I just cut&paste a brace group. A function key opens an empty brace group in my favorite editor.

[4] http://ctan.org/pkg/dowith
[5] http://ctan.org/pkg/langcode
[6] http://ctan.org/pkg/xspace

3 Example for filesdo.sty

In the file `srcfiles.tex` for the nicetext[7] bundle, there is a line

```
\DoWithBasesExts\ReadFileInfos{fifinddo,niceverb}{sty,tex}
```

This works like

```
\ReadFileInfos{fifinddo.sty}
\ReadFileInfos{niceverb.sty}
\ReadFileInfos{fifinddo.tex}
\ReadFileInfos{niceverb.tex}
```

or actually (a special feature of readprov's[8] `\ReadFileInfos` is that its argument may be a comma-separated list already)

```
\ReadFileInfos{fifinddo.sty,niceverb.sty,
               fifinddo.tex,niceverb.tex}
```

I ponder providing a shorthand `\ReadBaseExtInfos` for

```
\DoWithBasesExts\ReadFileInfos
```

and reimplementing `\ReadFileInfos` using `\DoWithCLS` in myfilist.sty (2012-11-27).

[7] http://ctan.org/pkg/nicetext
[8] http://ctan.org/pkg/readprov

4 The File commado.sty

4.1 Package File Header (Legalese and plainpkg)

```
1                                                    \input plainpkg
2   % \NeedsTeXFormat{LaTeX2e}[1994/12/01]
3   \ProvidesPackage{commado}[2012/11/30 v0.11 iterate on CSL (UL)]
4   %%
5   %% Copyright (C) 2012 Uwe Lueck,
6   %% http://www.contact-ednotes.sty.de.vu
7   %% -- author-maintained in the sense of LPPL below --
8   %%
9   %% This file can be redistributed and/or modified under
10  %% the terms of the LaTeX Project Public License; either
11  %% version 1.3c of the License, or any later version.
12  %% The latest version of this license is in
13  %%     http://www.latex-project.org/lppl.txt
14  %% We did our best to help you, but there is NO WARRANTY.
15  %%
16  %% Please report bugs, problems, and suggestions via
17  %%
18  %%   http://www.contact-ednotes.sty.de.vu
19  %%
20  \PushCatMakeLetterAt
```

4.2 Auxiliaries

```
21  \ifltx \else    %% unless provided by LaTeX already
22      \long\def\@firstoftwo#1#2{#1}
23      \long\def\@secondoftwo#1#2{#1}
24  \fi
```

... 4 less than

```
25  % \long\def\@firstsecondofthree#1#2#3{#1#2}
26  % \long\def\@firstthirdofthree#1#2#3{#1#3}
```

4.3 Processing a Comma-Separated List

Most of the following code aims at removing the final space in the comma-separated list. A variant of parsing as in fifinddo.sty (nicetext[9] bundle) and bitelist.sty[10] package is employed (while I am about to use different approaches there, one may see here how, inspired by `\@ifblank` in url.sty).[11] The purpose of the following `\edef` of `\DoWithCSL{`⟨*cmd*⟩`}{`⟨*list*⟩`}` is to get a space token after `\@firstoftwo` in the parameter text.

[9] http://ctan.org/pkg/nicetext
[10] http://ctan.org/pkg/bitelist
[11] http://ctan.org/pkg/url

```
\let\CD@final@comma\relax
\edef\DoWithCSL#1#2{%
    \CD@final@comma#2\CD@final@comma    %% 2nd \ 2012/11/30
    \noexpand\@firstoftwo
%     \noexpand\@firstsecondofthree
    \space\CD@final@comma
    \noexpand\@secondoftwo
%     \noexpand\@firstthirdofthree
    \noexpand\end{#1}{#2}}
\def\CD@final@comma#1 \CD@final@comma#2#3\end#4#5{%
%     \expandafter\@secondfirstoftwo\expandafter{%
%         #2{#1}#5}{\do@with@csl#4}%
    #2{\do@with@csl{#4}#1}{\do@with@csl{#4}#5}%
```

. . . 15 vs. 13:

```
%       #2{\do@with@csl{#4}}{#1}{#5}%
            ,\StopDoing,}
\catcode`\Q=3               %% not in #1
\def\do@with@csl#1#2#3,{%
```

`#1` is ⟨*cmd*⟩. `#2` takes the first token from (remaining) ⟨*list*⟩ that is not a space token. Trying to enter a blank list item would result in using the *next comma as a list item!*—The following is an alternative to the analogue in domore.sty:

```
    \unless@stop@doing#2#3\StopDoing
        #1{#2#3}\do@with@csl{#1}\StopDoing Q}
\def\unless@stop@doing#1\StopDoing#2\StopDoing#3Q{#2}
```

. . . somewhat replaces `\@secondfirstoftwo`—but will the latter be dropped? `\unless@stop@doing` is specific for `\StopDoing`—but can be used with `\DoWith` too. TODO

```
\catcode`\Q=11
```

4.4 Leaving the Package File

```
\PopLetterCatAt
\endinput
```

4.5 VERSION HISTORY

```
v0.1   2012/11/24f.  started
       2012/11/26    code ready
       2012/11/27    documented
v0.11  2012/11/30    code typo corrected, removing final space,
                     doc. \urlfoot's
```

5 The File filesdo.sty

5.1 Package File Header (Legalese and plainpkg)

```
1                                                    \input plainpkg
2   % \NeedsTeXFormat{LaTeX2e}[1994/12/01]
3   \ProvidesPackage{filesdo}[2012/11/27 v0.1 iterate on files (UL)]
4   %%
5   %% Copyright (C) 2012 Uwe Lueck,
6   %% http://www.contact-ednotes.sty.de.vu
7   %% -- author-maintained in the sense of LPPL below --
8   %%
9   %% This file can be redistributed and/or modified under
10  %% the terms of the LaTeX Project Public License; either
11  %% version 1.3c of the License, or any later version.
12  %% The latest version of this license is in
13  %%     http://www.latex-project.org/lppl.txt
14  %% We did our best to help you, but there is NO WARRANTY.
15  %%
16  %% Please report bugs, problems, and suggestions via
17  %%
18  %%   http://www.contact-ednotes.sty.de.vu
19  %%
```

5.2 Documentation

For documentation in PDF format, see `commado.pdf`.

5.3 commado Required

filesdo is based on commado:

```
20  \RequirePackage{commado}
```

5.4 Category Code

Use `@` as part of "command names" (plainpkg, stacklet):

```
21  \PushCatMakeLetterAt
```

5.5 User Commands

`\DoWithExtBases{`⟨*cmd*⟩`}{`⟨*ext*⟩`}{`⟨*basenames*⟩`}` runs ⟨*cmd*⟩`{`⟨*base*⟩`.`⟨*ext*⟩`}` for all items ⟨*base*⟩ in ⟨*basenames*⟩ and a single filename extension ⟨*ext*⟩:

```
22  \def\DoWithExtBases#1#2{\DoWithCSL{\do@with@ext@base{#1}{#2}}}
23  \def\do@with@ext@base#1#2#3{#1{#3.#2}}
```

`\DoWithBasesExts{`⟨*cmd*⟩`}{`⟨*basenames*⟩`}{`⟨*exts*⟩`}` runs ⟨*cmd*⟩`{`⟨*base*⟩`.`⟨*ext*⟩`}` for all items ⟨*base*⟩ in ⟨*basenames*⟩ and all items ⟨*ext*⟩ in ⟨*exts*⟩:

```
\def\DoWithBasesExts#1#2{\DoWithCSL{\distrib@basenames@do{#1}{#2}}}
```

`\distrib@basenames@do` exchanges arguments in order to reduce the task to `\DoWithCSL` and `\DoWithCSL`:

```
\def\distrib@basenames@do#1#2#3{%
    \DoWithCSL{\DoWithExtBases{#1}{#3}}{#2}}
```

5.6 Leaving the Package File

```
\PopLetterCatAt
\endinput
```

5.7 VERSION HISTORY

```
v0.1  2012/11/24f.  started
      2012/11/26    code ready
      2012/11/27    documented,
                    \DoWithBaseExts -> \DoWithBasesExts

```