Hints & Tricks

Glisterings

Peter Wilson

> All that glisters is not gold —
> Often have you heard that told.
>
> *Merchant of Venice*, Act II scene 7
> William Shakespeare

The aim of this column is to provide odd hints or small pieces of code that might help in solving a problem or two.

Corrections, suggestions, and contributions will always be welcome.

An issue that has cropped up recently on the `comp.text.tex` (`ctt`) newsgroup is what can be done when two packages clash by defining the same macro.

> And we are here as on a darkling plain
> Swept with confused alarms of struggle and flight,
> Where ignorant armies clash by night.
>
> *Dover Beach*
> Alfred, Lord Tennyson

1 Package/package clashes

A very simple method of undefining a macro, perhaps `\amacro`, is to let it be undefined, as:

```
\let\amacro\undefined
```

Of course, `\undefined` must never be defined. You might feel safer if instead you used, say

```
\let\amacro\uNdEFiNed
```

or some other unlikely name.

If two packages are being used, say packA and packB, which both create `\amacro` then, provided the second has used `\newcommand` and not the TEX `\def` macro which will silently replace any prior definition, it will complain that `\amacro` is already defined. If the definitions in packA and packB are identical then the following resolves the problem.

```
\usepackage{packA}
\let\amacro\undefined
\usepackage{packB}
```

Life being what it is, the definitions are usually different. In this case both definitions can be used but the name of the first definition has to be altered.

```
\usepackage{packA}
\let\Aamacro\amacro
\let\amacro\undefined
\usepackage{packB}
```

Following this, you use `\Aamacro` when you want packA's version and `\amacro` for the packB version.

Of course, life gets even more awkward if packA uses `\amacro` as part of another macro that you might use, in which case you have to hope that the author of at least one of the packages will change it to eliminate the clash.

2 Class/package clashes

A slightly different version of the same problem is when there is some clash between the code in a class and the code in a package. I came across this when I was developing the memoir class [3] which incorporates code from many[1] packages. In some cases I

[1] Mostly written by me.

needed to make sure that a particular package was not used with the class. I came up with this macro that fooled LaTeX into thinking that a package had been loaded, even though it hadn't been. The argument to the macro is the package name.

```
\newcommand*{\@memfakeusepackage}[1]{%
  \@namelet{ver@#1.sty}\@empty}
\newcommand*{\@namelet}[1]{%
  \expandafter\let\csname #1\endcsname}
```

(The code must be put where @ is treated as a letter.)

The LaTeX kernel has two useful macros for composing and using macro names which do not necessarily consist only of letters, namely: `\@namedef{`⟨*text*⟩`}{`⟨*def*⟩`}`, and `\@nameuse{`⟨*text*⟩`}`. The first of these lets you define a macro called \⟨*text*⟩ and the second lets you call a macro called \⟨*text*⟩ As an example, the result of the next piece of code is shown afterwards; note that you can't directly call a macro whose name includes analphabetic characters.

```
\makeatletter
\newcommand*{\ru}{are you}
\@namedef{ru4me}#1{#1, are you for me?}
‘\ru4me{Fred}’ he asked. \\
‘\@nameuse{ru4me}{Fred}’ he asked.
\makeatother
```

‘are you4meFred’ he asked.
‘Fred, are you for me?’ he asked.

In the same vein the macro `\@namelet{`⟨*text*⟩`}` defined above is for `\let`ing. Thus, calling

```
\@memfakeusepackage}{pack}
```

effectively expands to

```
\let\ver@pack.sty\@empty
```

which appears to be the magic incantation to make LaTeX believe it has already used the pack package.

The memoir class includes code very similar, but not identical, to the array, dcolumn, delarray and tabularx packages and I used `\@memfakeusepackage` to make sure these were not loaded again.

The memoir class also includes code corresponding to Heiko Oberdiek's ifpdf package [1] but I did not do anything to prevent loading the package. This resulted in a thread on `ctt` where the poster was using

```
\documentclass{memoir}
\usepackage{ps4pdf}
```

only to be told that `\ifpdf` was already defined. It turns out that the ps4pdf package uses the ifpdf package which defines `\ifpdf` which was also defined in memoir.

Heiko Oberdiek [2] gave the simple ‘let to undefined’ solution and the following more complex one:

```
\documentclass{memoir}
     %% memoir defines \ifpdf
\makeatletter
     %% save memoir's \ifpdf
\let\saved@ifpdf\ifpdf
     %% then undefine it
\let\ifpdf\@undefined
     %% use ifpdf package (defines \ifpdf)
\usepackage{ifpdf}
     %% is \ifpdf undefined?
\@ifundefined{ifpdf}{%
     %% yes, used the saved memoir version
  \let\ifpdf\saved@ifpdf
}{%
     %% no, check for matching definitions
  \ifx\ifpdf\saved@pdf
  \else
     %% mismatch, write error message
    \latex@error{Different meaning
              of \@backslash ifpdf}\@ehc
  \fi
}
\makeatother
     %% use ps4pdf which uses \ifpdf
\usepackage{ps4pdf}
```

This scheme can be applied to similar situations. Note that it produces an error if the second and first definitions are different, which could very well be useful.

References

[1] Heiko Oberdiek. The ifpdf package, July 2001. Available on CTAN in `latex/macros/contrib/oberdiek`.

[2] Heiko Oberdiek. Re: memoir, ps4pdf and `\ifpdf`. Post to `comp.text.tex` newsgroup, 3 September 2004.

[3] Peter Wilson. The memoir class for configurable typesetting, 2004. Available on CTAN in `latex/macros/contrib/memoir`.

◇ Peter Wilson
18912 8th Ave. SW
Normandy Park, WA 98166
USA
herries.press@earthlink.net