The switch Package
Version 1.0a

Alceu Frigeri*

May 2025

Abstract

This package offers two commands aimed at implementing a switch/case alike command.

Contents

1 Introduction

There are many ways of implementing a switch case programming structure. Notably, one can use `\str_case:nn` from *expl3*, or go over a loop using `\pdfstrcmp`, or construct an if-then-else tower, etc.

This implements a solution, based on [1], which (besides being simple) has the advantage of being constant time: once the cases are set up, suffice a single (internal) if (`\ifcsname`) to select the correct code to be executed.

> ***Note:*** The implementation creates a `\csname` for each case, and it uses (at the end) the primitive `\ifcsname` to select the correct case.
> ***Note:*** The coding is done using *expl3*, just for the sake of readability, in the package comments one can find an implementation using just TEX primitives.

2 Commands

Two set of commands are created, one to be used in a *expl3* code régime, and another set to be used in a user document.

2.1 User Document ones

`\newswitch` `\newswitch` ⟨`switch`⟩ {⟨`default-code`⟩}

It will create a new switch ⟨`switch`⟩, which will expects a single argument. In case the argument doesn't corresponds to any defined case, ⟨`default-code`⟩ will be used. The resulting ⟨`switch`⟩ command is expandable, if ⟨`default-code`⟩ and ⟨`case-code`⟩ (added by `\addcase`) also are. This is just an alias for `\switch_new:Nn`

> ***Note:*** `#1` can be used in ⟨`default-code`⟩. An error is raised if ⟨`switch`⟩ is already defined.

*https://github.com/alceu-frigeri/switch

\addcase `\addcase ⟨switch⟩ {⟨case⟩}{⟨case-code⟩}`

It will add a ⟨case⟩ to a previously defined ⟨switch⟩ and associates ⟨case-code⟩ with it. ⟨case⟩ will be fully expanded at definition time. Once defined one can call `\switch {case}`, which will put said ⟨case-code⟩ in the input stream. This is just an alias for `\switch_addcase:Nnn`.

2.1.1 Example

First we create a switch, and associate a few (or more) cases. Note the possibility of using an auxiliary (fully expandable) macro/command when defining the cases.

```
\def\CaseAstring{case-A}
\newswitch \myCase  {I~ don't~ know:~ #1\par}
\addcase   \myCase  {\CaseAstring} {A~ was~ used\par}
\addcase   \myCase  {case-B}       {B~ was~ used\par}
```

To use the ⟨switch⟩, one just has to call it with ⟨case⟩ as an argument. Note the possibility of using an auxiliary macro/command (which has to be fully expandable) as a ⟨case⟩.

```
\def\somemacro{case-A}
\def\someothermacro{case-X}

If B, then \myCase{case-B}
If A, then \myCase{case-A}
If X, then \myCase{case-X}

if somemacro: \myCase{\somemacro}
if someothermacro: \myCase{\someothermacro}
```

If B, then B was used
If A, then A was used
If X, then I don't know: case-X
if somemacro: A was used
if someothermacro: I don't know: case-X

2.2 Expl3 ones

\switch_new:Nn `\switch_new:Nn ⟨switch⟩ {⟨default-code⟩}`

It will create a new switch ⟨switch⟩, which will expects a single, type n, argument. In case the argument doesn't corresponds to any defined case, ⟨default-code⟩ will be used. The resulting ⟨switch⟩ command is expandable, if ⟨default-code⟩ and ⟨case-code⟩ (added by `\switch_addcase:Nnn`) also are.

> ***Note:*** `#1` can be used in ⟨default-code⟩. An error is raised if ⟨switch⟩ is already defined.

\switch_addcase:Nnn `\switch_addcase:Nnn ⟨switch⟩ {⟨case⟩}{⟨case-code⟩}`

It will add a ⟨case⟩ to a previously defined ⟨switch⟩ and associates ⟨case-code⟩ with it. ⟨case⟩ will be fully expanded at definition time. Once defined one can call `\switch {case}`, which will put said ⟨case-code⟩ in the input stream.

\switch_if_exist:N*TF* ⋆ `\switch_if_exist:NTF ⟨switch⟩ {⟨if-true⟩}{⟨if-false⟩}`
\switch_if_case_exist:Nn*TF* ⋆ `\switch_if_case_exist:NnTF ⟨switch⟩ {⟨case⟩}{⟨if-true⟩}{⟨if-false⟩}`

new: 2025-05-13

Tests if the ⟨switch⟩, or ⟨case⟩, are defined or not. It doesn't test if they are really a ⟨switch⟩/⟨case⟩.

\switch_undefine:N `\switch_undefine:N ⟨switch⟩`
\switch_case_undefine:Nn `\switch_case_undefine:Nn ⟨switch⟩ {⟨case⟩}`

new: 2025-05-13

Undefine the ⟨switch⟩ and/or specific ⟨case⟩. Please note, when undefining a ⟨switch⟩, the `\csname` associated with the cases aren't undefined (if needed, they have to be undefined one by one).

2.2.1 Example

First we create a switch, and associate a few (or more) cases. Note the possibility of using an auxiliary (fully expandable) macro/command when defining the cases.

```
\ExplSyntaxOn
\def\CaseAstring{case-A}
\switch_new:Nn      \TextCase  {I~ don't~ know:~ #1\par}
\switch_addcase:Nnn \TextCase  {\CaseAstring} {A~ was~ used\par}
\switch_addcase:Nnn \TextCase  {case-B} {B~ was~ used\par}
\ExplSyntaxOff
```

To use the ⟨switch⟩, one just has to call it with ⟨case⟩ as an argument. Note the possibility of using an auxiliary macro/command (which has to be fully expandable) as a ⟨case⟩.

```
\def\somemacro{case-A}
\def\someothermacro{case-X}
If B, then \TextCase{case-B}
If A, then \TextCase{case-A}
If X, then \TextCase{case-X}
if somemacro: \TextCase{\somemacro}
if someothermacro: \TextCase{\someothermacro}
```

If B, then B was used
If A, then A was used
If X, then I don't know: case-X
if somemacro: A was used
if someothermacro: I don't know: case-X

3 Advanced Use

Since the resulting ⟨switch⟩ is fully expandable (if the provided ⟨case-code⟩s also are), one can design the ⟨case-code⟩s to absorb one or more parameter/tokens.

> ***Careful:*** make sure that all ⟨case-code⟩s absorb the same number of parameters, to avoid "leftovers" or tricky errors.

For instance, note the use of `\@gobble` to absorb an unused parameter, or how `\cmdY` is defined (with two parameters) then used with a "fixed one". The resulting command, `\TCase`, absorbs 2 tokens/parameters:

```
\NewDocumentCommand \cmdX{m}   {I got #1}
\NewDocumentCommand \cmdY{mm}  {Two: #1 and #2}
\NewDocumentCommand \Astring{} {case-A}

\makeatletter
\newswitch \TCase       {I~ don't~ know:~ #1 \@gobble}
\makeatother
\addcase \TCase  {\Astring} {\cmdY{A~ given}}
\addcase \TCase  {case-B} {B~ was~ used. \cmdX}
```

```
If B, then \TCase{case-B}{extra-B}\par
If A, then \TCase{case-A}{extra-A}\par
If X, then \TCase{case-X}{extra-X}\par
```

If B, then B was used. I got extra-B
If A, then Two: A given and extra-A
If X, then I don't know: case-X

Needless to say, the same applies under *expl3*.

```
\ExplSyntaxOn
\cs_new:Npn \__cmdX:n   #1   {I~ got~ #1}
\cs_new:Npn \__cmdY:nn  #1#2 {Two:~ #1~ and~ #2}
\tl_new:N   \l__case_tl
\tl_set:Nn  \l__case_tl {case-A}

\switch_new:Nn      \TxCase  {I~ don't~ know:~ #1 \use_none:n}
\switch_addcase:Nnn \TxCase  {\l__case_tl} {\__cmdY:nn{A~ given}}
\switch_addcase:Nnn \TxCase  {case-B} {B~ was~ used.~ \__cmdX:n}
\ExplSyntaxOff
```

```
If B, then \TxCase{case-B}{extra-B}\par
If A, then \TxCase{case-A}{extra-A}\par
If X, then \TxCase{case-X}{extra-X}\par
```

If B, then B was used. I got extra-B
If A, then Two: A given and extra-A
If X, then I don't know: case-X

References

[1] Paul Gaborit. *Stack Exchange answer about Implementing Switch Cases*. 2012. URL: `https://tex.stackexchange.com/questions/64131/implementing-switch-cases/343306#343306` (visited on 12/10/2016).