The labelcas package[*]

Ulrich Diez

August 14, 2006

Abstract

This LaTeX 2ε-package provides macros `\eachlabelcase` and `\lotlabelcase` as a means of forking depending on whether specific labels are defined in the current document.

Contents

[*]This document corresponds to labelcas v1.12, dated 2006/08/14.
Usage and distribution under LPPL-conditions. See *6 Legal Notes* for more details.

1 Introduction

The package's name labelcas is an eight-letter abbreviation for the phrases "label" and "case".

There are rare occasions where the author of a document would like to have detected whether specific labels are defined/in use within the document so that proper forking/referencing can take place. This package provides the macros `\eachlabelcase` and `\lotlabelcase` which might facilitate this task.

A mechanism for branching depending on whether referencing-labels exist, might be handy, e.g., when extracting a "snippet" from a large document: In case that within the snippet a label/document-part is referenced which is outside the snippet's scope, ugly '**??**' will intersperse the resulting output-file and warnings about undefined references will accumulate within the log-file.

By testing the label's existence, you can catch up the error and either change the way of referencing (e.g., refer to the snippet's bibliography instead) or completely suppress referencing for those cases. (By using David Carlisle's xr- or xr-hyper-package, you can make available labels of the large document to the snippet also. A label not defined in the snippet can be picked up from the large document. . .)

1.1 Space notation

When listing some piece of TEX-source-code, you may need to visibly distinguish word-separation from single space-characters. The symbol ␣ is chosen whenever it is important to give a visible impression of a space-character in a (possibly ASCII-encoded) TEX-input-file. $␣_x$ does not represent a character of an input-file but a token which occurs after tokenizing the input. The token's category-code is x, the character-number usually is 32, which is the ASCII-number of the space-character.

2 Package-loading

The package is to be loaded in the document-preamble by `\usepackage`.

`\usepackage{labelcas}` or
`\usepackage[DefineLabelcase]{labelcas}`.

The only package-option is `DefineLabelcase`. Its usage is described in section *4 Package option—Different spaces, different separators*.

3 The macros

3.1 Basic usage

`\eachlabelcase` The macro `\eachlabelcase` iterates on a comma-separated list of "argument-triplets", whereby each triplet specifies:
1. a label,
2. action if the label is defined,
3. action if the label is undefined.

During the iteration, an "action-queue" is gathered up from these specifications. After iterating, the "action-queue" will be executed. You can also specify a new macro-name within an optional argument. If you do so, the "action-queue" will not be executed but the macro will be defined to perform the actions specified in the queue:

```
\eachlabelcase[\macro]{ {⟨label 1⟩}{⟨action if label 1 defined⟩}{⟨action if label 1 undefined⟩},
                        {⟨label 2⟩}{⟨action if label 2 defined⟩}{⟨action if label 2 undefined⟩} ,
                                                   ...
                        {⟨label n⟩}{⟨action if label n defined⟩}{⟨action if label n undefined⟩}  }
```

Space-tokens which might surround the comma-separated triplets will be gobbled.

`\lotlabelcase` The macro `\lotlabelcase` iterates on a comma-separated list of label-names and tests for each name if the corresponding label is defined. Within the arguments you can specify actions for the cases:
1. all labels are defined,
2. none of the labels is defined,
3. some labels are defined/some are undefined,
4. the list does not contain any label.

Like in `\eachlabelcase`, you can also specify a new macro-name within an optional argument. If you do so, the action will not be executed but the macro will be defined to perform the action:

```
\lotlabelcase[\macro]{⟨label 1⟩,⟨label 2⟩,...,⟨label n⟩}
                     {⟨actions if all labels are defined⟩}
                     {⟨actions if all labels are undefined⟩}
                     {⟨actions if some labels are defined and some labels are undefined⟩}
                     {⟨actions if list is empty⟩}
```

Space-tokens which might surround the label-names will be gobbled. One level of braces will also be gobbled so that you can also test for labels the names of which start or end by a space or contain some comma.

3.1.1 Possible problems

- Testing for labels which are **not definable** according to the syntax-rules will lead to TeX-internal error-messages and deliver unexpected/unwanted results!
- "Label- and referencing management" in LaTeX 2ε is done by means of the aux-file, the content of which is gathered and corrected during several LaTeX-runs, and which does not yet exist in the first run. So, in the first run, all labels from the current document are undefined—when applying `\...labelcase` to labels of the current document, it will in any case take at least two LaTeX-runs until everything matches out correctly.

- It was mentioned that, in the macros `\eachlabelcase` and `\lotlabelcase`, space-tokens which surround the argument-triplets/label-names, will be gobbled. There are situations where the category-code of the input-character ␣ is changed—e.g., due to a preceding `\obeyspaces` or when using some package where the encoding of TeX-input-files is played around with. In such cases, the input-character ␣ does not get tokenized as space-token any more but as some $␣_{\neq 10}$-token, so that in such cases, triplets/labels in these macros may, in the input-file, not be surrounded by ␣-characters.
  If you want to have these $␣_{\neq 10}$-tokens gobbled anyway, you can easily achieve this by defining another set of these macros where the appropriate token, e.g., $␣_{13}$ (active-space) instead of $␣_{10}$ (space-token), is taken into account. How this is done, is described in section *4 Package option—Different spaces, different separators*.

- In the very unlikely case[1] that you wish `\lotlabelcase` (or variants thereof[2])to scan for the label `\@nil`, `\@nil` has to be put in braces and/or has to be surrounded by space-tokens. This is because the internal iterator-macros terminate on `\@nil`.

- Internally token-registers are used and temporary-macros get defined. So the macros `\eachlabelcase` and `\lotlabelcase` (and all variants[2]) are not "full-expandable". This means, `\edef` or `\write` or control-sequences the like which evaluate their arguments fully, cannot be applied to them.[3] Therefore they are declared robust.

- `\lotlabelcase` and `\eachlabelcase` can be nested. Inner instances will be gathered into the action-queues of outer instances.

- If the optional argument for defining a ⟨*macro*⟩ rather than having the action(s) executed immediately, is used, ⟨*macro*⟩ will only be defined within the group where the `\...labelcase`-command occurred.
  `\@ifdefinable` is involved into the assignment-process, so that an "already-defined"-error is forced whenever an existing macro is about to be overridden.
  If you need it global, you can achieve this—after having ⟨*macro*⟩ defined—by something like `\global\let\macro=\macro`.
  If you need a "long"-macro, you can achieve this—after having ⟨*macro*⟩ defined—by something like:
  `\expandafter\renewcommand\expandafter\macro\expandafter{\macro}`.
  But think about it. These macros don't take arguments!

- If you use the arguments of `\lotlabelcase`/`\eachlabelcase` for defining other referencing-labels, things can easily get very confusing...

[1]The case is very unlikely because it is a convention in LaTeX 2ε to leave `\@nil` undefined. If labels are defined in terms of macros, these macros are to expand to something that can be evaluated by a `\csname...\endcsname`-construct. If they are to expand to something, they must be defined...

[2]→*4 Package option—Different spaces, different separators*.

[3]In any case it cannot be ensured that all arguments supplied are "full-expandable"...

3.1.2 Examples

Within this document, only the labels `sec1`, `sec2`, `sec3`, `sec4`, `sec5` and `sec6` are defined.

```
\lotlabelcase{sec1, sec2 , {sec3} ,sec4}
             {All labels are defined.}
             {None of the labels is defined.}
             {Some labels are defined, some not.}
             {The list is empty.}
```

yields: All labels are defined.

```
\lotlabelcase{sec1, sec2 , UNDEFINED ,sec3}
             {All labels are defined.}
             {None of the labels is defined.}
             {Some labels are defined, some not.}
             {The list is empty.}
```

yields: Some labels are defined, some not.

```
\lotlabelcase{UNDEF1, UNDEF2 , {UNDEF3} ,UNDEF4}
             {All labels are defined.}
             {None of the labels is defined.}
             {Some labels are defined, some not.}
             {The list is empty.}
```

yields: None of the labels is defined.

```
\lotlabelcase{ ,,  ,}
             {All labels are defined.}
             {None of the labels is defined.}
             {Some labels are defined, some not.}
             {The list is empty.}
```

yields: The list is empty.

```
\lotlabelcase[\test]{sec1, sec2 , UNDEFINED ,sec3}
                    {All labels are defined.}
                    {None of the labels is defined.}
                    {Some labels are defined, some not.}
                    {The list is empty.}
```

defines: `\test`: `macro:->Some labels are defined, some not.`

```
\eachlabelcase{ {sec1}{sec1 defined/}{sec1 undefined/},
                {sec2}{sec2 defined/}{sec2 undefined/} ,
                {UNDEF}{UNDEF defined/}{UNDEF undefined/} ,
                {sec3}{sec3 defined.}{sec3 undefined.}  }
```

yields: sec1 defined/sec2 defined/UNDEF undefined/sec3 defined.

```
\eachlabelcase[\test]{ {sec1}{sec1 defined/}{sec1 undefined/},
                       {sec2}{sec2 defined/}{sec2 undefined/} ,
                       {UNDEF}{UNDEF defined/}{UNDEF undefined/} ,
                       {sec3}{sec3 defined.}{sec3 undefined.}  }
```

defines: `\test`:

```
    macro:->sec1 defined/sec2 defined/UNDEF undefined/sec3 defined.
```

3.2 Advanced usage (brace-matching, `\if`..., defining macros)

- Braces within the arguments/comma-separated items must be balanced.
- Within the "action-parts" of `\eachlabelcase`'s argument-triplets from which the action-queue is formed, balancing `\if`...`\else`...`\fi`-constructs is not required. But ensured must be, that in the resulting action-queue everything is balanced correctly in any case.

```
\eachlabelcase{  {sec1}     {\if aa}       {\if ab},
                 {sec2}  {a is a\else}   {a is b\else} ,
                 {sec3}{a is not a\fi.}{a is not b\fi.}   }
```

  is gathered to: `\if aaa is a\else a is not a\fi.`
  Executing the queue yields: a is a.

```
\eachlabelcase{  {sec1}     {\if aa}       {\if ab},
                 {UNDEF}  {a is a\else}  {a is b\else} ,
                 {sec3} {a is not a\fi.}{a is not b\fi.}   }
```

  is gathered to: `\if aaa is b\else a is not a\fi.`
  Executing the queue yields: a is b.

  When trying such obscure things, you must be aware that brace/group-nesting is independent from conditional-nesting! You might easily end up with a "forgotten-endgroup"-error or some "extra `\else`..."-error when placing such things into other `\if`...`\else`...`\fi`-constructs!

- If you wish to use the arguments/comma-separated items for defining macros, no extra #-level is needed as everything is accumulated within/processed by means of token-registers.

```
\eachlabelcase{ {sec1}{\def\testA#1#2#3}{\def\testB#1#2#3},
                {sec2}   {{#1,#2,#3}}      {{#3,#2,#1}}     }
```

  is gathered to: `\def\testA#1#2#3{#1,#2,#3}` .
  Executing the queue defines: `\testA: macro:#1#2#3->#1,#2,#3`
  `\testB: undefined` .

```
\eachlabelcase{ {sec1}{\def\testA#1#2#3}{\def\testB#1#2#3},
               {UNDEF}   {{#1,#2,#3}}      {{#3,#2,#1}}     }
```

  is gathered to: `\def\testA#1#2#3{#3,#2,#1}` .
  Executing the queue defines: `\testA: macro:#1#2#3->#3,#2,#1`
  `\testB: undefined` .

4 Package option—Different spaces, different separators

Above was said that space-tokens (␣$_{10}$-tokens) which surround the comma-list-arguments of `\eachlabelcase` and `\lotlabelcase` are gobbled.

There are circumstances where the category-code which gets assigned to the input-character ␣ during the tokenizing-process is changed, and thus the gobbling-mechanism is broken for these input-characters. E.g., due to a preceding `\obeyspaces` or when using some package where the encoding of TEX-input-files is played around with. This is because space-gobbling internally is implemented by means of macros with ␣$_{10}$-token-delimited arguments.

In normal circumstances, ␣-characters in the input-file which trail a control-word do not get tokenized when TEX "reads" an input. So it's kind of a problem to get space-tokens right behind the name of a control-word, e.g., as first items of the parameter-text when defining macros. A space within braces `{␣}` does get tokenized as it is not preceded by a control-word, but by a brace-character. So a solution to the problem is: Define a macro which takes an (en-braced) argument and use this macro for defining the desired control-word whereby the argument is placed right behind the name of the control-word which is about to be defined. (Henceforth the term *definer-macro* is applied in order to call special attention to the fact that defining other control-sequences is the only purpose of such a macro.) A ␣ as the definer-macro's argument gets tokenized while this argument is used as the first item of the desired control-word's parameter-text → the first item of the desired control-word's parameter-text will be a space-token.

`\DefineLabelcase` In case of the labelcas-package, the problem of getting space-tokens as delimiters right behind control-words, is also solved by implementing such a definer-macro. It is called `\DefineLabelcase` and used for defining both the user-level-macros `\eachlabelcase` and `\lotlabelcase` and the internal-macros `\lc@iterate`, `\lc@remtrailspace` and `\lc@remleadspace`. Usually it is discarded/destroyed when defining these macros has taken place. But you can specify the package-option `DefineLabelcase`. When you do so, `\DefineLabelcase` does not get destroyed, and you can use it for creating "new variants" of `\eachlabelcase` and `\lotlabelcase` plus internals while specifying proper space-tokens and separators. `\DefineLabelcase` takes four mandatory arguments:

`\DefineLabelcase{`⟨*space*⟩`}{`⟨*delimiter*⟩`}{`⟨*prefix*⟩`}{`⟨*global-indicator*⟩`}`

⟨*space*⟩ specifies the argument-surrounding token that is to be removed. Usually surrounding space-tokens shall be discarded. Usually: ␣$_{10}$ (space).

⟨*delimiter*⟩ specifies the delimiter/separator. Usually the argument-triplets or label-lists are comma-separated. Usually: `,`$_{12}$ (comma).

⟨*prefix*⟩ specifies the macro-name-prefix. You cannot assign the same name at the same time to different control-sequences. Therefore, when creating new variants of `\eachlabelcase` and `\lotlabelcase`, you have to specify a prefix which gets inserted at the beginning of the macro-name. E.g., specifying the prefix `FOO` leads to defining the macro-set:
`\FOOeachlabelcase`, `\FOOlotlabelcase` (user-macros) and
`\FOOlc@iterate`, `\FOOlc@remtrailspace`, `\FOOlc@remleadspace` (internal).
The original versions are just called `\eachlabelcase`, `\lotlabelcase`, `\lc@iterate`... (without a prefix in the macro-name). Usually: (empty).

⟨*global-indicator*⟩: In case that this argument contains only the token `\global`, defining the new macro-set takes place in terms of `\global`. Otherwise the scope is restricted to the current grouping-level. Usually: `\global`.

Don't try weird things like specifying the same token both for ⟨*space*⟩ and ⟨*delimiter*⟩, or leaving any of those empty, or specifying any of those to `\@nil` (, which is reserved for terminating the recursion)—unless you like error-messages! Please only specify tokens which may be used for separating parameters from each other within the parameter-text of a definition! Also please specify the ⟨*prefix*⟩ only in terms of letter-character-tokens! **There is no extra error-checking implemented on these things!**

```
\begingroup
\obeyspaces
\endlinechar=-1\relax%
\DefineLabelcase{␣}{/}{SPACEOBEYED}{local}%
\SPACEOBEYEDlotlabelcase[\test]{sec1/␣sec2␣␣␣/␣␣␣UNDEF␣␣␣␣␣/sec3}%
{All␣␣␣labels␣␣␣are␣␣␣defined.}%
{None␣␣␣of␣␣␣the␣␣␣labels␣␣␣is␣␣␣defined.}%
{Some␣␣␣labels␣␣␣are␣␣␣defined,␣␣␣some␣not.}%
{The␣␣␣list␣␣␣is␣␣␣empty.}%
\global\let\test\test%
\endgroup
```

defines: `\test: macro:->Some␣␣␣labels␣␣␣are␣␣␣defined,␣␣␣some␣not.`

```
\begingroup
\endlinechar=-1\relax
\DefineLabelcase{-}{/}{BAR}{local}%
\BARlotlabelcase[\test]{sec1/-sec2----/--%
                             ---/sec3}%
{All   labels   are   defined.}%
{None   of   the   labels   is   defined.}%
{Some   labels   are   defined,   some not.}%
{The   list   is   empty.}%
\global\let\test\test
\endgroup
```

defines: `\test: macro:->All labels are defined.`

```
\begingroup
\endlinechar=-1\relax
\DefineLabelcase{.}{/}{DOT}{local}%
\DOTeachlabelcase{.{sec1}..{sec1 defined/}{sec1 undefined/}/%
..................{sec2}...{sec2 defined/}...{sec2 undefined/}./..%
..................{UNDEF}{UNDEF defined/}...{UNDEF undefined/}./%
..................{sec3}{sec3 defined.}{sec3 undefined.}..}
\endgroup
```

yields: sec1 defined/sec2 defined/UNDEF undefined/sec3 defined.

5 Thanks, Acknowledgements

- Many thanks to all who encouraged me in making the attempt of getting things in this package less error-prone.
- Thanks to everybody who took the macro-writing challenges presented in the INFO-TEX-'Around the bend'-department which was initiated back in the early 90's by Michael Downes and regularly took place under his guidance. His summaries of the solutions are archived and online available at http://www.tug.org/tex-archive/info/aro-bend/. The information therein helps a great deal in understanding TEX in general and in learning about basic problem-solving-strategies—e.g., the removal of leading- and trailing spaces from an (almost) arbitrary token-sequence (exercise.015/answer.015).
- Thanks to everybody who provides valuable information at the TEX-newsgroups and mailing-lists. I received great help especially at comp.text.tex, where my—often trivial—questions were answered patiently again and again.
- Thanks to the LATEX-package authors, not only for providing means of achieving special typesetting-goals, but also for hereby delivering informative programming-examples. labelcas actually was inspired by David Carlisle's xr- and xr-hyper-packages which make available the labels of other LATEX-documents to the current one.

6 Legal Notes

[4]The latest version of this license is in http://www.latex-project.org/lppl.txt and version 1.3 or later is part of all distributions of LATEX version 2003/12/01 or later.

7 Implementation

7.1 A note about removing leading and trailing spaces

The matter of removing trailing spaces from an (almost) arbitrary token-sequence is elaborated in detail by Michael Downes, 'Around the Bend #15, answers', a summary of internet-discussion which took place under his guidance primarily at the INFO-TEX list, but also at comp.text.tex (usenet) and via private e-mail; December 1993. Online archived at http://www.tug.org/tex-archive/info/aro-bend/answer.015.

One basic approach suggested therein is using TEX's scanning of delimited parameters in order to detect and discard the ending space of an argument:

> ... scan for a pair of tokens: a space-token and some well-chosen bizarre token that can't possibly occur in the scanned text. If you put the bizarre token at the end of the text, and if the text has a trailing space, then TEX's delimiter matching will match at that point and not before, because the earlier occurrences of space don't have the requisite other member of the pair.
>
> Next consider the possibility that the trailing space is absent: TEX will keep on scanning ahead for the pair ⟨*space*⟩⟨*bizarre*⟩ until either it finds them or it decides to give up and signal a 'Runaway argument?' error. So you must add a stop pair to catch the runaway argument possibility: a second instance of the bizarre token, preceded by a space. If TEX doesn't find a match at the first bizarre token, it will at the second one.

(Look up the macros `\KV@@sp@def`, `\KV@@sp@b`, `\KV@@sp@c` and `\KV@@sp@d` in David Carlisle's keyval-package for an interesting variation on this approach.)

When scanning for parameters `##1`⟨*space*⟩⟨*bizarre*⟩`##2`⟨*B1*⟩ the sequence: ⟨*stuff where to remove trail-space*⟩⟨*bizarre*⟩⟨*space*⟩⟨*bizarre*⟩⟨*B1*⟩
, you can fork two cases:

1. Trailing-space:
   `##1`=⟨*stuff where to remove trail-space*⟩, but with removed space. (And possibly one removed brace-level!)
   `##2` = ⟨*space*⟩⟨*bizarre*⟩.
2. No trailing-space:
   `##1`=⟨*stuff where to remove trail-space*⟩⟨*bizarre*⟩.
   `##2` is empty.

So forking can be implemented depending on the emptiness of `##2`.

You can easily prevent the brace-removal in the first case, e.g., by adding (and later removing) something (e.g., a space-token) in front of the ⟨*stuff where to remove trail-space*⟩.

You can choose ⟨*B1*⟩=⟨*bizarre*⟩⟨*space*⟩.

'Around the Bend #15, answers' also presents a similar way for the removal of leading spaces from an (almost) arbitrary token-sequence:

> The latter method is perhaps most straightforwardly done as a mirror-image of the method for removing a trailing space: make the delimiter ⟨*bizarre*⟩⟨*space*⟩, and call the macro [...] by putting ⟨*bizarre*⟩ before the scanned text and a stop pair ⟨*bizarre*⟩⟨*space*⟩ after it, in case a leading space is not present

When scanning for parameters `##1`⟨*bizarre*⟩⟨*space*⟩`##2`⟨*B2*⟩ the sequence: ⟨*bizarre*⟩⟨*stuff where to remove lead-space*⟩⟨*bizarre*⟩⟨*space*⟩⟨*B2*⟩
, you can fork two cases:

1. Leading space:
   `##1`= is empty.
   `##2` = ⟨*stuff where to remove lead-space*⟩⟨*bizarre*⟩⟨*space*⟩ (but with a leading-space removed from ⟨*stuff where to remove lead-space*⟩).
2. No leading space:
   `##1`=⟨*bizarre*⟩⟨*stuff where to remove lead-space*⟩.
   `##2` is empty.

Thus forking can be implemented depending on the emptiness of either of the two arguments.

You can choose ⟨*B2*⟩=⟨*bizarre*⟩⟨*bizarre*⟩.

7.2 Flow of work

Both \⟨*prefix*⟩`eachlabelcase` and \⟨*prefix*⟩`lotlabelcase` iterate on (e.g., comma-) separated lists:

1. The list is passed as an argument to the user-macro.
2. The list is passed from the user-macro to \⟨*prefix*⟩`lc@iterate` whereby a leading ⟨*space*⟩ is added for brace-removal-protection.
3. \⟨*prefix*⟩`lc@iterate` recursively iterates on the list-items until the item ⟨*space*⟩`\@nil` occurs:
   a) The item will be passed to \⟨*prefix*⟩`lc@remtrailspace`. Here trailing ⟨*space*⟩ is removed recursively. If after removing trailing-space the result is empty, you can conclude that everything (incl the previously inserted “brace-removal-protection-⟨*space*⟩” was removed as either the item was empty or consisted of a sequence of ⟨*space*⟩. If the result does not imply an empty item, it will be passed to
   b) \⟨*prefix*⟩`lc@remleadspace` where leading ⟨*space*⟩ (also the previously inserted one) is removed recursively. After that \⟨*prefix*⟩`lc@remleadspace` passes the item to the macro
   c) `\@tempa` for further processing. `\@tempa` at this stage will be locally defined within the user-macro. `\@tempa` initiates the actual work which (hopefully!) results in adding the appropriate action-sequence to the queue which is represented by `\@temptokena`.
   d) Before processing the next item in the next iteration-round, a leading ⟨*space*⟩ for brace-removal-protection will be added in front of the remaining list by \⟨*prefix*⟩`lc@iterate`.
4. After iterating the list within the user-macro, the routine `\lc@macrodefiner` will check for the user-macro’s optional argument and, in case that it is present, modify the action-queue-register, so that, when “flushing” it, a macro will be produced instead of queue-execution.
5. The final step within the user-macro is “flushing” the action-queue-register.

7.3 Code

`\DefineLabelcase` `\DefineLabelcase` is used for providing parameters during the definition of the macros `\`⟨*prefix*⟩`eachlabelcase`, `\`⟨*prefix*⟩`lotlabelcase` (user),
`\`⟨*prefix*⟩`lc@iterate`, `\`⟨*prefix*⟩`lc@remtrailspace`,
`\`⟨*prefix*⟩`lc@remleadspace` (internal).
Parameters are: `#1`=⟨*space*⟩; `#2`=⟨*delimiter*⟩; `#3`=⟨*prefix*⟩; `#4`=⟨*global-indicator*⟩.

Defining of `\DefineLabelcase` takes place within a group, so that after closing the group it gets discarded. Package-options will also be evaluated within that group, right after defining `\DefineLabelcase`. By the option `DefineLabelcase`, `\DefineLabelcase` can be "globalized" before closing the group:

```
\begingroup
\DeclareOption{DefineLabelcase}%
             {\global\let\DefineLabelcase\DefineLabelcase}%
\newcommand\DefineLabelcase[4]{%
```

`\`⟨*prefix*⟩`lc@remtrailspace` It is assured that ⟨*delimiter*⟩ does not occur in the top-level of the ⟨*stuff where to remove trail-space*⟩, for ⟨*delimiter*⟩ is used in the list for separating the single items of ⟨*stuff where to remove trail-space*⟩ from each other. Therefore you can choose ⟨*bizarre*⟩=⟨*delimiter*⟩ and ⟨*B1*⟩=⟨*bizarre*⟩⟨*space*⟩=⟨*delimiter*⟩⟨*space*⟩:

```
  \expandafter\@ifdefinable\csname#3lc@remtrailspace\endcsname{%
    \expandafter\long
    \expandafter\def
    \csname#3lc@remtrailspace\endcsname##1#1#2##2#2#1{%
```

Above was said that forking can take place depending on emptiness of the second argument. The arguments come from the items of the comma-separated list—thus they might contain macro-definitions and/or unbalanced `\if...\else...\fi`-constructs. So put the second argument into a macro `\@tempa` by means of a token-register in order to prevent errors related to parameter-numbering:

```
      \begingroup
      \toks@{##2}%
      \edef\@tempa{\the\toks@}%
```

When forking takes place, the content of the arguments might—when placed into the corresponding `\if`- or `\else`-branches directly—erroneously match up those constructs. In order to prevent this, the action related to the different branches is handled by means of `\@firstoftwo` and `\@secondoftwo` which get expanded when "choosing the forking-route" is already accomplished:

```
      \expandafter\endgroup
      \ifx\@tempa\@empty
        \expandafter\@firstoftwo
      \else
        \expandafter\@secondoftwo
      \fi
```

The appropriate action in case of no more trailing ⟨*space*⟩ is checking if the item is not empty and if so, initiating the removal of leading ⟨*space*⟩. In this case `##1` is terminated by ⟨*bizarre*⟩. If the item is empty, the leading ⟨*space*⟩ inserted by the iterator for brace-protection is also removed so that `##1` equals ⟨*bizarre*⟩. If the item is not empty, start leading-⟨*space*⟩-removal, but add only ⟨*space*⟩⟨*B2*⟩ at the end instead of ⟨*bizarre*⟩⟨*space*⟩⟨*B2*⟩—above was said that ⟨*B2*⟩=⟨*bizarre*⟩⟨*bizarre*⟩=⟨*delimiter*⟩⟨*delimiter*⟩ in `\`⟨*prefix*⟩`lc@remleadspace`:

```
      {%
        {\toks@{##1}\edef\@tempa{\the\toks@}%
```

```
       \toks@{#2}\edef\@tempb{\the\toks@}%
       \expandafter}%
      \ifx\@tempa\@tempb
        \expandafter\@gobble
      \else
        \expandafter\@firstofone
      \fi
      {\csname#3lc@remleadspace\endcsname#2##1#1#2#2}%
    }%
```
The appropriate action in case of trailing ⟨*space*⟩ is checking and possibly removing more thereof:
```
    {\csname#3lc@remtrailspace\endcsname##1#2#1#2#2#1}%
  }%
}%
```
\⟨*prefix*⟩`lc@remleadspace` \⟨*prefix*⟩`lc@remleadspace` is similar to \⟨*prefix*⟩`lc@remtrailspace`, but with ⟨*B2*⟩=⟨*bizarre*⟩⟨*bizarre*⟩=⟨*delimiter*⟩⟨*delimiter*⟩:
```
\expandafter\@ifdefinable\csname#3lc@remleadspace\endcsname{%
  \expandafter\long
  \expandafter\def
  \csname#3lc@remleadspace\endcsname##1#2#1##2#2#2{%
```
Above was said that forking can take place e.g., depending on emptiness of the first argument. Arguments still come from the list-items, so let's use token-registers for the same reasons as in \⟨*prefix*⟩`lc@remtrailspace`:
```
    \begingroup
    \toks@{##1}%
    \edef\@tempa{\the\toks@}%
```
The single list-items might still contain macro-definitions, `\if`-forking and the like, therefore again choose the forking-route in terms of `\@firstoftwo` and `\@secondoftwo`:
```
    \expandafter\endgroup
    \ifx\@tempa\@empty
      \expandafter\@firstoftwo
    \else
      \expandafter\@secondoftwo
    \fi
```
The appropriate action in case of leading ⟨*space*⟩ is checking and possibly removing more thereof:
```
    {\csname#3lc@remleadspace\endcsname#2##2#2#2}%
```
In case of no more leading ⟨*space*⟩, the actual work, which is defined in user-macro's `\@tempa`, can be done:
```
    {\@tempa##1#2}%
  }%
}%
```
\⟨*prefix*⟩`lc@iterate` \⟨*prefix*⟩`lc@iterate` iterates on arguments which are delimited by ⟨*delimiter*⟩.
```
\expandafter\@ifdefinable\csname#3lc@iterate\endcsname{%
  \expandafter\long
  \expandafter\def
  \csname#3lc@iterate\endcsname##1#2{%
```

Make locally available the arguments as macros:
`\@tempa`=current argument
`\@tempb`=recursion-stop-item:

```
      \begingroup
      \toks@{##1}%
      \edef\@tempa{\the\toks@}%
      \toks@{#1\@nil}%
      \edef\@tempb{\the\toks@}%
```

End the group and test if the current argument equals the recursion-stop-item:

```
      \expandafter\endgroup\ifx\@tempa\@tempb
        \expandafter\@gobble
      \else
        \expandafter\@firstofone
      \fi
```

If not: Start trailing-space-removal…, then continue iterating the list and hereby add a preceding ⟨*space*⟩ to the next item for brace-protection during trailing-⟨*space*⟩-removal in the next run:

```
      {%
        \csname#3lc@remtrailspace\endcsname##1#2#1#2#2#1%
        \csname#3lc@iterate\endcsname#1%
      }%
    }%
  }%
```

\⟨*prefix*⟩`eachlabelcase`

\⟨*prefix*⟩`eachlabelcase`'s optional argument is the possibly-to-be-defined control-sequence. The mandatory-argument contains the argument-triplet-list.

```
  \expandafter\@ifdefinable\csname#3eachlabelcase\endcsname{%
    \expandafter\DeclareRobustCommand
    \csname#3eachlabelcase\endcsname[2][]{%
```

Locally define `\@tempa`—it is called by \⟨*prefix*⟩`lc@remleadspace` for working on a list-item when all surrounding ⟨*space*⟩ has been removed:

```
      {%
```

The stuff that results from ⟨*space*⟩-removing is surrounded by ⟨*delimiter*⟩. It cannot be processed at this place, as first the triplet needs to be split into its components by `\@tempb`:

```
        \long\def\@tempa#2####1#2{%
          \@tempb####1#2#1#2#2%
        }%
```

`\@tempb` is used for splitting the triplet and removing ⟨*space*⟩ between the triplet's components. In this process it redefines itself several times. In case that no label is defined the name thereof corresponds to the first component, add the third component to `\@temptokena`, otherwise add the second:

```
        \long\def\@tempb####1{%
          \begingroup
          \long\def\@tempb########1########2########3{%
            \expandafter\expandafter
            \expandafter\endgroup
            \expandafter\ifx
            \csname r@########1\endcsname\relax
              \expandafter\@firstoftwo
            \else
              \expandafter\@secondoftwo
          \fi
          {\@temptokena\expandafter{\the\@temptokena########3}}%
          {\@temptokena\expandafter{\the\@temptokena########2}}%
        }%
        \begingroup
        \toks@{}%
        \long\def\@tempb########1{%
          \long\def\@tempa#2################1#2{%
            \toks@\expandafter{\the\toks@{################1}}%
            \expandafter\endgroup\expandafter\@tempb\the\toks@
          }%
          \toks@\expandafter{\the\toks@{########1}}%
          \csname#3lc@remleadspace\endcsname#2%
        }%
        \toks@{{####1}}\csname#3lc@remleadspace\endcsname#2%
      }%
```

Let's clear the register where the action-queue is accumulated:

```
      \@temptokena{}%
```

Let's iterate on the list:

```
      \csname#3lc@iterate\endcsname#1##2#2\@nil#2%
```

In case that the optional argument is specified, the routine `\lc@macrodefiner` will modify the register to define a macro:

```
      \lc@macrodefiner{##1}%
```

Close the group and "flush" the register:

```
    \expandafter}\the\@temptokena
  }%
}%
```

\⟨*prefix*⟩`lotlabelcase`

\⟨*prefix*⟩`lotlabelcase`'s optional argument is the possibly-to-be-defined control-sequence. The five mandatory-arguments contain the label-list and the actions that shall take place in the cases: All of the labels are defined / none are defined / just some are defined / list is empty:

```
\expandafter\@ifdefinable\csname#3lotlabelcase\endcsname{%
  \expandafter\DeclareRobustCommand
  \csname#3lotlabelcase\endcsname[6][]{%
```

Locally define `\@tempa`—it is called by \⟨*prefix*⟩`lc@remleadspace` for working on a list-item when all surrounding ⟨*space*⟩ has been removed:

```
    {%
      \long\def\@tempa#2####1#2{%
```

The list item is a label. In case that it is undefined, have the helper-macro `\@tempb` defined/switched to `\relax`, otherwise do the same but use `\@tempc` instead:

```
        {\expandafter\expandafter\expandafter}\expandafter
        \ifx\csname r@####1\endcsname\relax
          \let\@tempb\relax
        \else
          \let\@tempc\relax
        \fi
      }%
```

Define `\@tempb` and `\@tempc` to empty. They may be "switched" to `\relax` when `\@tempa` is called during iteration.

```
      \def\@tempb{}%
      \def\@tempc{}%
```

Let's iterate on the list:

```
   \csname#3lc@iterate\endcsname#1##2#2\@nil#2%
```

Assign the register according to the label-defining-cases which are now represented by the definitions of `\@tempb` and `\@tempc` which are defined either `\relax` or `empty`:

```
   \ifx\@tempb\@empty
    \ifx\@tempc\@empty
     \@temptokena{##6}%
    \else
     \@temptokena{##3}%
    \fi
   \else
    \ifx\@tempc\@empty
     \@temptokena{##4}%
    \else
     \@temptokena{##5}%
    \fi
   \fi
```

In case that the optional argument is specified, the routine `\lc@macrodefiner` will modify the register to define a macro:

```
   \lc@macrodefiner{##1}%
```

Close the group and "flush" the register:

```
  \expandafter}\the\@temptokena
 }%
}%
```

If the ⟨*global-indicator*⟩-argument equals `\global`, the above definitions need to be made `\global`:

```
{\toks@{#4}\edef\@tempa{\the\toks@}\def\@tempb{\global}\expandafter}%
\ifx\@tempa\@tempb
 \expandafter\global\expandafter\let
  \csname#3lc@remtrailspace\expandafter\endcsname
  \csname#3lc@remtrailspace\endcsname
 \expandafter\global\expandafter\let
  \csname#3lc@remleadspace\expandafter\endcsname
  \csname#3lc@remleadspace\endcsname
 \expandafter\global\expandafter\let
  \csname#3lc@iterate\expandafter\endcsname
  \csname#3lc@iterate\endcsname
 \expandafter\global\expandafter\let
  \csname#3eachlabelcase\expandafter\endcsname
  \csname#3eachlabelcase\endcsname
 \expandafter\global\expandafter\let
  \csname#3lotlabelase\expandafter\endcsname
  \csname#3lotlabelcase\endcsname
\fi
```

Now the definition of `\DefineLabelcase` is complete:

```
}%
```

Remember that a group was started for performing `\DefineLabelcase`'s definition and that `\DefineLabelcase` will be gone when that group gets closed—unless some "globalizing" takes place before. So this is the time for checking if `\DefineLabelcase` shall be available to the user and in this case for making it global:

```
\ProcessOptions\relax
```

Now the group which was started for defining `\DefineLabelcase` can be closed—right after using it for defining the basic-usage-macros:

```
\expandafter\endgroup\DefineLabelcase{␣}{,}{}{\global}%
```

`\lc@macrodefiner` There is still the routine left which is applied by the user-macros for having the action-queue-register modified, so that when "flushing" it, a macro will be produced instead of queue-execution. `\lc@macrodefiner` takes as its argument the optional argument of a user-macro. In case that the argument is not empty, the action-queue-register is modified, so that "flushing" it yields the attempt of defining a macro from the argument which expands to the former content of the register:

```
\newcommand\lc@macrodefiner[1]{%
  {\def\@tempa{#1}\expandafter}%
  \ifx\@tempa\@empty
  \else
    \@temptokena\expandafter{%
                \expandafter\begingroup
                \expandafter\toks@
                \expandafter\expandafter
                \expandafter            {%
                \expandafter\expandafter
                \expandafter            \@temptokena
                \expandafter\expandafter
                \expandafter            {%
                \expandafter\the
                \expandafter\@temptokena
                \expandafter}%
                \expandafter}%
                \expandafter\@temptokena
                \expandafter{%
                \expandafter\@temptokena
                \expandafter{%
                \the\@temptokena}%
                \@ifdefinable#1{\edef#1{\the\@temptokena}}}%
                \expandafter\endgroup
                \the\expandafter\@temptokena
                \the\toks@
    }%
  \fi
}%
```

Change History

v1.0
- General: Initial public release.

v1.01
- \⟨*prefix*⟩`lc@remleadspace`: ⟨*B2*⟩=⟨*bizarre*⟩⟨*bizarre*⟩. 13
- \⟨*prefix*⟩`lc@remtrailspace`: ⟨*B1*⟩=⟨*bizarre*⟩⟨*space*⟩. 12
- General: Fixed documentation-inaccuracies.

v1.02
- General: Fixed documentation-inaccuracies.

v1.03
- \⟨*prefix*⟩`eachlabelcase`: Changed forking-mechanism to `\@firstoftwo`/`\@secondoftwo`. 14
- \⟨*prefix*⟩`lc@remleadspace`: Changed forking-mechanism to `\@firstoftwo`/`\@secondoftwo`. 13
- \⟨*prefix*⟩`lc@remtrailspace`: Changed forking-mechanism to `\@firstoftwo`/`\@secondoftwo`. 12

v1.04, v1.05
- General: Fixed documentation-inaccuracies.

v1.06
- \⟨*prefix*⟩`eachlabelcase`: `\@ifdefinable` instead of `\newcommand`. 14
- \⟨*prefix*⟩`lc@iterate`: `\@ifdefinable` instead of `\newcommand`. 13
- \⟨*prefix*⟩`lc@remleadspace`: `\@ifdefinable` instead of `\newcommand`. 13
- \⟨*prefix*⟩`lc@remtrailspace`: `\@ifdefinable` instead of `\newcommand`. 12
- \⟨*prefix*⟩`lotlabelcase`: `\@ifdefinable` instead of `\newcommand`. 15
- `\lc@macrodefiner`: `\@ifdefinable` instead of `\newcommand`. 17

v1.07
- \⟨*prefix*⟩`lc@iterate`: Define `\@tempa` in terms of `\long`. . . 13

v1.08
- \⟨*prefix*⟩`lc@iterate`: Changed forking-mechanism to `\@firstoftwo`/`\@secondoftwo`. 13
- General: DefineLabelcase-option declared within group for `hyperref`-compatibility.

v1.09
- \⟨*prefix*⟩`lc@iterate`: Changed forking-mechanism so that two temporary macros suffice. . . . 13
- General: Hyperlinks in documentation.

v1.10
- \⟨*prefix*⟩`lc@iterate`: Empty-argument-check removed. . . . 13
- \⟨*prefix*⟩`lc@remtrailspace`: Empty-argument-check added. 12

v1.11
- `\lc@macrodefiner`: Unnecessary `\expandafter` removed. 17

v1.12
- General: Fixed documentation-inaccuracies.

Index

Numbers written in italics refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.