The `varindex` package[*]

Martin Väth[†]
`vaeth@mathematik.uni-wuerzburg.de`

2001/05/09

Abstract

This LaTeX package provides a luxury front-end for the `\index` command. For example, it allows to generate multiple entries in almost any form by a single command. It is extremely customizable, and works with all versions of LaTeX and probably most other TeX formats, too.

Contents

[*]The package has version number 2.3, last revised 2001/05/06.

[†]This package bases on ideas of `indextwo.sty` which was written jointly with O. Karch `karch@informatik.uni-wuerzburg.de` and H.-C. Wirth `wirth@informatik.uni-wuerzburg.de`

1 Changes

v2.3 (*2001/05/06*) Introduced new macros `\varindexIndex`, `\varindexDoConvert`, `\varindexNoConvert`. The layout of the documentation was slightly improved. Moreover, a completely new section was added (on `\toolboxMakeDef`).

Use `\typeout` instead of `\message`, but provide a workaround for plain TeX. Avoid error with plain TeX and LaTeX2.09 (because `\newcount` is defined there as `\outer` which is rather annoying). Since `toolbox` works around this problem, finally also the local `\if`'s now actually *are* local.

v2.2 (*2001/05/01*) Introduced `=:^` flags into `\varindextwo`. Changed sorting of entries starting with "`-`" (see `\varindexOutSortDashBeg`).

Improved documentation, in particular of `\varindextwo` and `\varindexNewFormat`. Also new examples were added for both commands.

v2.1 (*2001/04/30*) Finally implemented placeholder replacement (`\varindexPreviousATexttrue` and friends) such that it works also in the expected way with commas.

Introduced separated command `\varindexCommaExpandtrue` which was in earlier versions automatically coupled with `\varindexCommaLeadstrue`. Changed the default of both values to 'true'.

Introduced `\varindexOutText...` and friends and enabled a hook into the format string processing via `\varindexNewFormat`.

Reordered documentation such that the more interesting customization parts are now mentioned earlier.

v1.21 (*2001/03/29*) Divided package into two: Many macros which are of independent interest have been put into the `toolbox` package. For the user visible is that the macros are now called e. g. `\toolboxMakeSplit` instead of `\varindexMakeSplit`. For backward compatibility, the old (now obsolete) names are still available, although not documented anymore.

v1.20 (*2001/01/13*) Changed `\varindextwo`: Added the `\varindextwoCommaTilde` functionality and the corresponding three flags `?!/`. Moreover, changed the default to `\varindextwoCommaTildetrue`. Note that due to this change the generated index entries may differ from those of earlier versions. To get a backward compatible output, you have to use `\varindextwoCommaTildefalse`.

Also use now the symbols `‘` and `’` instead of the (now obsolete) `P` and `p` flags. The reason for the latter is that `P` and `p` could not be used immediately after a command without a space.

v1.19 (*2000/12/07*) Eliminated a bug in the `P` flag.

v1.18 (*2000/11/30*) Eliminated a bug in `\SkipTricky`.

v1.17 (*2000/11/21*) `\varindexNr??String` is now defined as documented.

v1.16 (*2000/11/19*) Documentation rewritten for doc and docstrip utilities (required files are now `varindex.dtx` and `varindex.ins`).

v1.15 (*2000/10/10*) Fixed harmless bug which always caused a warning.

v1.14 (*2000/06/01*) First version released into public (as `varindex.sty`).

2 Introduction

If you have to write an index for a larger scientific text, you will probably sooner or later have the problem that you want to put certain texts into the index which consist of more than one word like *Theorem of Picard*.
There are several reasonable ways how one might do this: One might put the text `Theorem of Picard` in the index, or `Picard theorem`. An alternative way is `Picard, Theorem`, or `Picard's Theorem` or `Theorem!Picard` (here, `!` is the symbol used by `makeindex` to separate a subindex, i. e. the output will actually look like

Theorem

  Picard, 17

This is useful if one also has other entries starting with *Theorem*). Several other natural alternatives are also possible.
One may choose some or all of the above alternatives and then code them into TeX. However, if more than one of these alternatives is desired in the index, this requires several `\index` commands (which are similar but not identical). Moreover, if later another choice of the above alternatives is required, the index entries must be completely rewritten. This rewriting may be very cumbersome if one makes use of subindices for two reasons.
First, it may happen that one has to change even entries which are apparently not affected. For example, assume there were just two subindices for the index entry *Theorem*, namely `Theorem!Picard` and `Theorem!Riemann`. If the theorem of Picard is eliminated from the main text, one not only has to delete the entry `Theorem!Picard`: The single subindex `Theorem!Riemann` does not make much sense – one should replace it by `Theorem, Riemann`. One could write a program to do this task automatically, but my experience shows that it is better to do such things by hand, because there are some situations where exceptions are desirable.
However, things become really cumbersome, if one also has to sort the entries in a different way than they are written. For example, suppose that instead of *Theorem* in the above example, one has the text *ζ-function* which one wants to have sorted like *zeta-function*. In this case, the `\index` command must be changed from

```
\index{zeta-function@$\zeta$-function!Riemann}
```

into

```
\index{zeta-function, Riemann@$\zeta$-function, Riemann}
```

which is rather different!
Moreover, one may also want to have a placeholder. For example, instead of generating the entries

> Theorem
>
> > Liouville, 17
> >
> > Open Mapping, 17
> >
> > Picard, 17

One might like to have entries like

> Theorem
>
> > $\sim$ of Liouville, 17
> >
> > Open Mapping $\sim$, 17
> >
> > $\sim$ of Picard, 17

One might use the symbol `$\sim$` for the placeholder, but this needs of course its own sort entry in the `\index` command (in the above example, we sorted by *Liouville*, *Open Mapping*, and *Picard*, respectively, which is perhaps the most natural choice).
With the package `varindex` it is rather simple to generate any of the above `\index` entries, and it is easy to modify them, e. g. the above placeholders can be introduced easily and then activated or deactivated by a switch. Moreover, several `\index` entries like `Picard, Theorem of` and `Theorem of Picard` can be generated simultaneously with a *single* command (i. e. you have to write the phrase `Theorem of Picard` only once). In addition, it is also possible to produce not only corresponding `\index` commands but also to output the content into the running text.
Since my experience shows that in each new book there arise new special cases for the `\index` command, I was trying to provide a *highly customizable* solution which can be modified for all needs (I hope).

3 Installation

This package was tested with LaTeX2.09 and LaTeX 2_ε and should work with all other (future) versions of LaTeX and friends, too. It should even run with other (non-LaTeX) formats if an `\index` command (with LaTeX-Syntax) is provided.
To use “varindex”, you have to put the file `varindex.sty` in a path where TeX looks for its input files. You must also have the file `toolbox.sty` of the “toolbox” package in your path (toolbox v3.1 or newer is required). The TeX documents using varindex need the following modifications in their header:

- If you use LaTeX 2_ε, put in the preamble the command

  ```
  \usepackage{varindex}
  ```

- If you use LaTeX2.09, use `varindex` as a style option, e. g.

  ```
  \documentstyle[varindex]{article}
  ```

or

```
\documentstyle[varindex,12pt]{article}
```

- If you use some other (non-LaTeX) format, you will probably have to insert a line like

  ```
  \catcode`\@=11\relax\input varindex.sty\catcode`\@=12\relax
  ```

For TeX insiders: LaTeX-specific commands used in `varindex.sty` are only:

- `\makeatother`
- `\makeatletter`
- `\typeout`
- `\RequirePackage`
- `\newcommand` (used only in the form `\newcommand{`⟨*command*⟩`}{}` to ensure that ⟨*command*⟩ was not defined before)
- `\ProvidesPackage`

The above commands are used only if they are defined (otherwise, natural substitutes are used (`\newcommand` and `\ProvidesPackage` are then not used at all)).

4 Additional hint

Although this package provides a convenient way to write `\index` commands, this may not be sufficient: Also with this package, the `\index` commands may still be rather complex. For some books, it might be necessary to write the same `\index` entries several times (to get various page numbers). The first idea that one might have in this case is to define a list of the used `\index` commands near the beginning of the document e. g. as:

```
\newcommand{\Index}{\varindex(){\varindextwoScan}{\varindextwo}[]}
\newcommand{\Riemann}{\Index{Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\newcommand{\sigmaa}{\Index-={$\sigma$@sigma algebra}}
```

(the first line and the usage of `\Index` is explained in later sections; here, it is sufficient to know that `\Index` produces in the above form several `\index` entries). To produce the corresponding `\index` entries in the running text, one then just needs to use `\Riemann` resp. `\sigmaa`. However, this method has some disadvantages:

1. If one has many `\index` entries, it is easy to forget that e. g. `\sigmaa` is a command which should produce an index entry. Then `\sigmaa` in the main text might be rather confusing.
2. One has to take care of macros already provided by TeX, LaTeX or some packages. For example, it is not possible to use the name `\sigma` for the above purpose.

To avoid these problems, one may be very disciplinary and call the involved macros systematically e. g. `\IndexRiemann` `\Indexsigmaa` etc. However, this produces terrible long and unreadable macro names in the main text.
The `toolbox` package (version 3.1 or newer) provides a more convenient solution: You can use the command

```
\newcommand{\Index}{\varindex(){\varindextwoScan}{\varindextwo}[]}
\toolboxMakeDef{Idx}{#1}
```

and afterwards

```
\NewIdx{R-S}{\Index{Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\NewIdx{sigma-algebra}{\Index-={$\sigma$@sigma algebra}}
```

After these definitions, you can use the intuitive commands `\Idx{R-S}` and `\Idx{sigma-algebra}` to produce the corresponding index entries.
Note also the symbol "-" in the above names which is usually not allowed in TEX macros.
If several page numbers occur for one index entry, it may be convenient for the reader if the "main page number(s)" (e. g. the place(s) where the corresponding notion is defined) is written in a different style. Of course, this is supported by `\varindex` (as explained later). To use this feature in connection with `\toolboxMakeDef`, there are several possibilities. A straightforward way is to add in addition to the above commands also

```
\NewIdx{R-S*}{\Index[|textbf]%
  {Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\NewIdx{sigma-algebra*}{\Index[|textbf]%
  -={$\sigma$@sigma algebra}}
```

Then `\Idx{R-S*}` and `\Idx{sigma-algebra*}` produce index entries with boldface page numbers. Of course, you can automatize the task of generating macros which produce normal and boldface page numbers e. g. as follows:

```
\newcommand{\Index}{\varindex(){\varindextwoScan}{\varindextwo}[]}
\toolboxMakeDef{Idx}{#1}
\newcommand{\MakeIdx}[2]{%
  \NewIdx{#1}{\Index#2}%
  \NewIdx{#1*}{\Index[|textbf]#2}}

\MakeIdx{R-S}{{Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\MakeIdx{sigma-algebra}{-={$\sigma$@sigma algebra}}
```

This provides the commands `\Idx{R-S}` `\Idx{R-S*}` `\Idx{sigma-algebra}` and `\Idx{sigma-algebra*}` with the same functionality as explained before.
The above approach has the disadvantage that the names "...*" have automatically a fixed meaning. Alternatively, you can also generate different main commands. This is immediately supported by `\toolboxMakDef`:

```
\newcommand{\Index}{\varindex(){\varindextwoScan}{\varindextwo}[]}
\toolboxMakeDef{Idx}{\Index#1}
\toolboxMakeDef{IdxMain}{\Index[|textbf]#1}

\NewIdx{R-S}{{Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\NewIdxMain{R-S}{%
  {Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
```

Now you can use `\Idx{R-S}` and `\IdxMain{R-S}` to produce the normal respectively boldface page numbers. Of course, you can also automatize in this case the task of generating these commands:

```
\newcommand{\Index}{\varindex(){\varindextwoScan}{\varindextwo}[]}
\toolboxMakeDef{Idx}{\Index#1}
\toolboxMakeDef{IdxMain}{\Index[|textbf]#1}
\newcommand{\NewIdxBoth}[2]{%
  \NewIdx{#1}{#2}%
  \NewIdxMain{#1}{#2}}

\NewIdxBoth{R-S}{{Riemann Stieltjes integral}[1-23 2,1-~3 3!1-2~]}
\NewIdxBoth{sigma-algebra}{-={$\sigma$@sigma algebra}}
```

After the above commands, you can use `\Idx{R-S}`, `\IdxMain{R-S}`, `\Idx{sigma-algebra}` and `\IdxMain{sigma-algebra}` with their obvious meaning.

Summarizing: The last of the above solutions is the one which I recommend.

For further possibilities of the `\toolboxMakeDef` command (like making copies of entries, redefining entries, etc.), please read the original documentation of the `toolbox` package.

5 Examples

Probably you are very impatient and want to see some examples. So, here you are – but be warned that the examples are not explained in detail. You have to read the later sections to understand more precisely why the examples work the way they do.
Since the `\varindex` command is highly customizable, the following examples can only give you a rough impression of what you can actually do with it.

5.1 Typical example of usage (using the `varindextwo` macros)

Suppose the following customization of the `\varindex` command was defined:

```
\def\Index{\varindex(){\varindextwoScan}{\varindextwo}[\emph]}
```

Since we used the `varindextwo` macros here, by default *two* `\index` entries are generated with a single command. After the above definition, the command

```
\Index{ring with $1$@one}
```

will produces the `\index` entries for:

ring

  $\sim$ with 1, 17 (*sorted like* "with one")

1 (*sorted like* "one")

  ring with $\sim$, 17 (*sorted like* "ring with" (without "$\sim$"!))

One of the examples of the introduction can be generated with

```
\Index*{Theorem of@ Picard}
```

which produces the two `\index` entries for

Theorem
    ∼ of Picard, 17   (*sorted like* “Picard”)
Picard
    Theorem of ∼, 17   (*sorted like* “Theorem”)

In this example, the word `of` is ignored for the sorting, because we put the symbol `@`⟨*nothing*⟩ behind it which means that it should be sorted as the empty string (in cases like this, also the space in front of `Picard` is automatically ignored for the sort entry).
Since we used in the above example the form `\Index*` instead of `\Index`, additionally the text `\emph{Theorem of Picard}` is output into the running text (the string `\emph` stems from our customization of `\Index` defined in the beginning).

Completely different entries may be generated by your own rules. To this end, an optional parameter `[`⟨*format*⟩`]` can be appended. Internally, the main argument is splitted into words which are numbered $1, 2, \ldots$; in ⟨*format*⟩, you can refer to these words simply by using the corresponding numbers. An “empty” entry in ⟨*format*⟩ means all words in the original order (i. e. it is essentially equivalent to 12345 . . .). Some other symbols besides numbers are also allowed which are listed later.

```
\Index{Hausdorff␣measure␣of␣noncompactness}[4,23␣2!~34!1=␣]
```

generates *three* index entries:

noncompactness, measure of, 17
measure
    ∼ of noncompactness
        Hausdorff ≈, 17
Hausdorff measure of noncompactness, 17

The last of these entries occurs only, because the last character in the ⟨*format*⟩ argument `[4,23␣2!~34!1=␣]` is a space (the space following `=` tells `varindex` that another (empty) entry is desired).
If you want only slight modification(s) of the default, you need also just slight modification(s) of the command, e. g.

```
\Index*,{ring with $1$@one}
```

has the same effect as in our earlier example, just that the second entry reads

1, ring with ∼, 17   (*sorted like* “one, ring with”)

Moreover, since the ‘`*`’ occurred in the previous command, additionally

```
\emph{ring with $1$}
```

appears in the running text. Praxis shows that changes in the entries as above have to be made frequently in the 'fine tuning' of the index of a book. Note that with the original `\index`, the command would have to be changed completely for the above entries, because we have a "sort" entry for which a different rule applies if we use a subitem ("`!`") instead of a "`,␣`". (BTW: `\varindex` produces sort entries only if they are necessary).
Of course, you may combine the default with your own format:

```
\Index{internal␣integrable␣function}[23,1~␣+]
```

generates

internal function, integrable $\sim$, 17

internal

$\sim$ integrable function, 17

function

internal integrable $\sim$, 17

In this example, the first entry is generated by the symbols `23,1~` in the ⟨*format*⟩ argument, and the last two entries are those entries which would have been generated by the `varindextwo` macros if no ⟨*format*⟩ argument had been given – the magic symbol `+` instructs the `varindextwo` macros to not suppress these entries.

5.2 A simple example without the `varindextwo` macros

As known from `\index`, a `|see{...}` can be used to produce a reference to another index entry instead of a page number. Such a command may optionally be appended.

```
\varindex*{topology␣of␣a␣normed␣space}{5!4~!1=␣45,12}[|see{norm}]
```

space

normed $\sim$

topology $\approx$, *see also* norm

normed space, topology of, *see also* norm

(the precise appearance of "*see also* " depends on how the `\see` macro is defined in your style). The above command additionally inserts the tokens `{topology of a normed space}` into the running text (this would not happen if `\varindex` is used in place of `\varindex*`). Only in the first of the above entries, an '`@`' part will be added to the `*.idx` file: This entry is sorted as if "␣$\sim$" resp. "␣$\approx$" would not exist.

5.3 Another example without the `varindextwo` macros

The appearance of the cited page number and the appearance in the running text (only available with "`*`") can be customized easily.

```
\varindex[\emph][|textbf]%
    {$\sigma$@sigma!$(\sigma-)$ finite {measure space}}%
    {*1.23 23 23,1- 1-23}
```

generates the three index entries

> finite measure space, **42**
>
> finite measure space, σ-, **42** (*sorted like* "finite measure space, sigma-")
>
> σ-finite measure space, **42** (*sorted like* "sigma-finite measure space")

In all three entries the pages are printed using `\textbf`. Moreover, the tokens

```
\emph{($\sigma$-)finite measure space}
```

are put into the running text.

5.4 A simple example using a flag (without the `varindextwo` macros)

Note the token `1` in the following command:

```
\varindex1{$L$ and $M$@L and M}{}
```

This command generates the index entry

> L and M, 17 (*sorted like* "L and M")

With the command

```
\varindex1*{$L$ and $M$@L and M}{}
```

or

```
\varindex*1{$L$ and $M$@L and M}{}
```

additionally, `{$L$ and $M$}` is output into the running text.
Note that without the '`1`', the index generated were

> `$L$ and $M$ and M`, 17 (*sorted like* "`$L$ and L and M`")

5.5 An example for very primitive customized index commands (without the `varindextwo` macros)

If you defined your "default" shortcuts appropriately, you can still override some of your own defaults.

```
\def\myindexA{\varindex[\emph][|textbf]*}
\def\myindexB{\varindex(){}{}[\emph][|textbf]}

\myindexA{Foo1}{}
\myindexB{Foo2}[][]
\myindexB*{Foo3!{Foo 3}}
```

Generates the index entries `Foo1`, `Foo2`, and `Foo3`. Moreover, `\emph{Foo1}` and `\emph{Foo 3}` is put into the running text. The page number of `Foo 2` in the index is printed normal (i. e. `|textbf` is overridden), the others with `\textbf`. Note that by using braces, it was possible to include a space into the text `Foo 3`.

5.6 An example of a primitive customized index command (without the `varindextwo` macros)

TeX code can be inserted to customize the default formatting string.

```
\def\myindex{\varindex(\ifnum\varindexCount=2 1!2 2,1\fi){}{}[\emph]}
```

```
\myindex{Foo Bar}
```

generates the index entries

> Foo
>
>   Bar, 17
>
> Bar, Foo, 17

```
\myindex*{Foo Bar}
```

generates the same index entries as above and additionally outputs `\emph{Foo Bar}` into the running text.

```
\myindex*{other format}[21]
```

generates the index entry

> format other, 17

and outputs `\emph{other format}` into the running text.

```
\myindex[|textbf]*{BoBo}
```

generates the index entry

> BoBo, **42**

where the page number is printed with `\textbf`. Moreover, `\emph{BoBo}` is output into the running text.

5.7 A primitive varying customized index command (without the `varindextwo` macros)

```
\def\myindex{\varindex(\ifnum\varindexCount=2 1!2 2,1\fi)%
    (\ifnum\varindexCount=2*2,1 1!2 2,1\fi){}{}[\emph]}
```

With the same commands as in Section 5.6, the same index entries are generated, but the tokens `\emph{Far, Boo}`, `\emph{other format}`, and `\emph{Bobo}` are output into the running text.

5.8 Examples with the `varindextwo` macros

As mentioned earlier, if the `varindextwo` macros are used (and if the main arguments contains at least two entries), then two `\index` entries are generated by default. Roughly speaking, one `\index` entry is the argument in its given order and the other is the argument in reverse order (this is not precise, but explains the idea).
For the following examples, we use a similar definition for `\Index` as in Section 5.1, and define an `\iemph` macro which has '`*`' as a default and outputs with `\emph`. I recommend to put both sort of definitions into the preamble of all of your files which should use `varindex` (the names are of course subject to your personal taste).

```
\def\Index{\varindex(){\varindextwoScan}{\varindextwo}[]}
\def\iemph{\varindex(){\varindextwoScan\def\varindexStarPos{}}%
            {\varindextwo}[\emph]}
```

(note that the apparently simpler definition

```
\def\iemph{\varindex(){\varindextwoScan}{\varindextwo}[\emph]*}
```

is not as good as the above definition of `\iemph` as is explained below).
After the above customization, you may use the following commands.

```
\Index*{measure of $\gamma$-noncompactness@gamma-noncompactness!FOO}
```

generates the index entries

measure

$\sim$ of γ-noncompactness, 17 (*sorted like* "of gamma-noncompactness")

γ-noncompactness (*sorted like* gamma-noncompactness)

measure of $\sim$, 17

and outputs the tokens `{measure of FOO}` into the running text. Note the order of the words!

```
\Index*,?_'{flic of flac}
```

generates

flic of flac, 17

flac, flic, 17

and outputs `{flic of flac}`. Here, the two tokens `_` and `,` had the effect that a space resp. "`,␣`" was used instead of a subentry in the first resp. second `\index` entry; the token `'` caused the preposition `of` to disappear in the second entry, and the token `?` suppressed a trailing $\sim$ in the second entry.
A more realistic example is

```
\Index^={$\sigma$@sigma algebra}
```

which in both entries inserts a "-":

σ (*sorted like* “sigma”)

  $\sim$-algebra (*sorted like* “algebra” or “-algebra”, see below)

algebra

  σ-$\sim$, 17 (*sorted like* “sigma-”)

Here, the sorting “-algebra” is chosen when you used

```
\let\varindexOutSortDashBeg\varindexOutSortDash
```

as described later.
The flag `^` in conjunction with `-` has a special meaning:

```
\Index^->{$\sigma$@sigma algebra}
```

Then the first entry reads instead (the second is canceled because of `>`):

σ- (*sorted like* “sigma-”)

  $\sim$ algebra, 17 (*sorted like* “algebra”)

Similarly, the conjunction `^` and `.` has a special meaning:

```
\Index^.,:{ultra filter}
```

generates entries without a space following $\sim$ (and in view of the flag `,` no new subentry is used for the second entry but instead a “,”):

ultra

  $\sim$filter, 17 (*sorted like* filter)

filter, ultra$\sim$, 17 (*sorted like* filter, ultra)

For crossreferences, one may use something like

```
\iemph[|see{recursive recursion}],_{recursive recursion}
```

which generates

recursive recursion, *see also* recursive recursion

recursion, recursive $\sim$, *see also* recursive recursion

and outputs `\emph{recursive recursion}`. If we would have used instead the definition

```
\def\iemph{\varindex(){\varindextwoScan}{\varindextwo}[]*}
```

the above call were not possible, since the optional arguments `[|see...]` must occur in front of the flag ‘`*`’.

```
\Index-;*{Flip Flop}
```

generates

Flip-Flop, 17

FlopFlip, 17

and outputs `{Flip-Flop}` (note that the symbol `-` which is caused by the flag `-` occurs also in the output).
One may freely create other entries:

```
\Index>{Flippy Flop!GO}[*2-1-2 2!1-1 1-12 +]
```

generates

> Flop
>
> Flippy-Flippy, 17   *(no "∼"!)*
>
> Flippy-Flippy Flop, 17
>
> Flippy
>
> ∼ Flop, 17

and outputs `{GO-Flippy-GO}` (note that even without an explicit `*` flag, we were able to generate this output, just by using the `*` in ⟨*format*⟩).

```
\Index*_,{shape {of a} of star}
```

generates

> shape of a star, 17
>
> star, shape of ∼, 17   (*sorted like* star, shape of)

and outputs `{shape of a star}`. Note that for the first entry the proposition "`of a`" was chosen while for the second entry the alternative proposition `of` was chosen.
Also with the `varindextwo` macros the other flags keep their old meaning. For example, the flag `1` still means that the main argument is considered as a single entry.

```
\Index1*{$L$ feature@L feature}
\Index,{No $L$@L feature}
```

generate

> *L* feature, 17   (*sorted like* "`L feature`")
>
> No
>
> ∼ *L* feature, 17   (*sorted like* "`L feature`")
>
> feature, No *L* ∼, 17   (*sorted like* "`feature, No L`")

and outputs `{$L$ feature}`.

5.9 Example of a simple modification of the `varindextwo` macros

All flags and magic tokens in `varindextwo` can be customized to fit your own taste, just by modifying `\varindextwoScan`.

```
\def\myindextwoScan{%
    \varindextwoScan
    \varindexNewFlag ~\vxtSpaceA
```

```
     \varindexNewFlag 1\vxtSpaceB
     \varindexNewFlag !\varindexOneEntry
     \varindexMakeSplitExpand{/}{vxtSplitAtMagic}%
     \def\varindexStarPos{}%
}
\def\myIndex{\varindex(){\myindextwoScan}{\varindextwo}[]}
```

`\myIndex` behaves similar to the previous example with the following differences:

1. The flag `1` now has the previous meaning of `~`.
2. One may now use `~` and `_` equivalently.
3. The flag `!` now has the original meaning of the flag `1`.
4. Instead of `+` in the format string, the character `/` has to be used.

6 Main Description

`\varindex` Without additional customization, there are two possible calls for the `\varindex` command:

a) The first call has the form

   `\varindex[`⟨*textstyle*⟩`][`⟨*pagestyleA*⟩`]`⟨*flags*⟩`%`
   `{`⟨*main entries*⟩`}{`⟨*format*⟩`}[`⟨*pagestyleB*⟩`]`

   Here, `[`⟨*textstyle*⟩`]`, `[`⟨*pagestyleA*⟩`]`, `[`⟨*pagestyleB*⟩`]`, and ⟨*flags*⟩ are optional. (But if `[`⟨*textstyle*⟩`]` is omitted, also `[`⟨*pagestyleA*⟩`]` must be omitted).

   The meaning of the arguments is as follows:

   ⟨*textstyle*⟩ describes the textstyle used for the output into the running text (typically, textstyle is `\emph` or empty).

   ⟨*pagestyle*⟩ describes the pagestyle used for the output of the page number (you may also use a construct like `|see{...}` here). If `[`⟨*pagestyleB*⟩`]` is present, this is the value used. Otherwise, the value of `[`⟨*pagestyleA*⟩`]` is used (resp. nothing).

   ⟨*flags*⟩ can be one (or both) of the following tokens:

   `*` If this token appears in ⟨*flag*⟩, it has roughly the meaning "Output into running text". More precisely, if no `*` is occurs in ⟨*format*⟩ (see below), then ⟨*format*⟩ is automatically prepended by the tokens "`*␣`" resp. "`*`" (depending whether ⟨*format*⟩ starts with a space or not).

   `1` With this flag, the content of ⟨*main entries*⟩ is considered as a single entry (see below).

   ⟨*main entries*⟩ and

   ⟨*format*⟩ are explained later.

b) The alternative form to call `\varindex` is

```
\varindex(⟨formatA⟩)(⟨format*A⟩){⟨scan program⟩}%
    {⟨main program⟩}[⟨textstyle⟩][⟨pagestyleA⟩]⟨flags⟩%
    {⟨main entries⟩}[⟨formatB⟩][⟨pagestyleB⟩]
```

The arguments `[`⟨*textstyle*⟩`]`, `[`⟨*pagestyleA*⟩`]`, ⟨*flags*⟩, and `[`⟨*pagestyleB*⟩`]` have the same meaning as in a) (in particular, they are optional, but if `[`⟨*textstyle*⟩`]` is omitted, also `[`⟨*pagestyleA*⟩`]` must be omitted).

Also `(`⟨*format*A*⟩`)` and `(`⟨*formatA*⟩`)` are optional, but if `(`⟨*formatA*⟩`)` is omitted, also `(`⟨*format*A*⟩`)` must be omitted.

The "default" value for ⟨*format*⟩ is ⟨*formatA*⟩ resp. ⟨*format*A*⟩ (depending whether the flag `*` has been used or not). If `[`⟨*formatB*⟩`]` is given, then this is used as the format (i. e. the optional argument ⟨*formatB*⟩ can be used to override the default).

The other values are:

⟨*scan program*⟩ is executed (i. e. expanded) immediately when `\varindex` reads it. This parameter can be used to introduce *further* flags (other than `*` and `1`), see below.

⟨*main program*⟩ is executed *after* the ⟨*format*⟩ string(s) has been expanded.

It is recommended to define personal macros which contain all parameters up to the place where ⟨*flags*⟩ occurs (either with or without some flags, depending on the intention). See the examples section.

⟨*main entries*⟩ is the main argument which has the following form

⟨*entry1*⟩[␣⟨*entry2*⟩][␣⟨*entry3*⟩]...

where each of ⟨*entry1*⟩ ⟨*entry2*⟩ ⟨*entry3*⟩ ... in turn has the form

⟨*indextext*⟩`@`⟨*sorttext*⟩`!`⟨*outputtext*⟩

or

⟨*indextext*⟩`!`⟨*outputtext*⟩`@`⟨*sorttext*⟩

In each of these forms, `@`⟨*sorttext*⟩ and `!`⟨*outputtext*⟩ are optional. Here,

⟨*indextext*⟩ is the text which is put into the index for the entry

⟨*sorttext*⟩ is what is used as a sort criterion for the entry

⟨*outputtext*⟩ is what is output into the running text for the entry

Note that the symbol `@` has in a sense the opposite meaning as in the original `\index` command: *Before* that symbol, the desired text appears, and *after* the symbol the place where it has to be sorted. However, we chose the symbols `@` and `!`, because these two symbols are forbidden anyway (otherwise, `makeindex` will become confused).
As usual, multiple spaces are considered as *one* space and do not generate empty "ghost" entries. Moreover, a space after a TEX-command like `\LaTeX` is eliminated by TEX and thus also not considered as a separator. You may use braces`{...}` either around a whole entry or around ⟨*indextext*⟩ or ⟨*sorttext*⟩ or ⟨*outputtext*⟩ to

allow spaces which do not act as "separators" in the corresponding part. The braces around these parts will vanish. In particular, you may generate an empty entry with `{}` or `!` or `@` (bordered by spaces). If you want that braces occur in the *output*, you have to add an additional pair of braces. Be aware that you write sufficiently many braces, if you really should need them: For example, the command `\varindex{{{{A␣B}}}}{}` produces the index entry `{A␣B}`: The outermost pair of braces is eliminated immediately by TeX. The second pair is eliminated, because this is a pair around a complete entry in ⟨*main entries*⟩. Finally, another pair is eliminated, because it is a brace around ⟨*indextext*⟩. With the flag '`1`', ⟨*main entries*⟩ is considered as one single entry. Nevertheless, also with this flag, an additional pair of braces around ⟨*main entry*⟩ is eliminated.

The ⟨*format*⟩ has one of the following three forms:

> ⟨*IndexA*⟩[␣⟨*IndexB*⟩][␣⟨*IndexC*⟩]...[`*`⟨*OutputA*⟩][`*`⟨*OutputB*⟩]...
> [`*`⟨*OutputA*⟩][␣⟨*IndexA*⟩][␣⟨*IndexB*⟩][`*`⟨*OutputC*⟩]...
> `*`

where the order of arguments can be arbitrarily mixed (except for the first ⟨*IndexA*⟩ in the first form). ⟨*IndexA*⟩ ⟨*IndexB*⟩ ... ⟨*OutputA*⟩ ... describe the format of the index entries resp. of the output into the running text. The output is generated in the given order (this might be important, if a pagebreak occurs during the output into the running text). The last case is exceptional and equivalent to "`*␣`". The following characters/strings are allowed in the format entries:

token	meaning
`1`–`9`	⟨*entry1*⟩–⟨*entry9*⟩
`0`	⟨*entry10*⟩
`~`	a placeholder sign (default is $\sim$)
`=`	a secondary placeholder sign (default is $\approx$)
`_`	the space character "␣"
`s`	The token "`\space`" (but "␣" is used for the sort entry)
`.`	No space will follow (see below)
`,`	The character "`,`" (space will follow)
`-`	The character "`-`" (no space will follow)
`!`	Create a new sublevel (subitem).

All other tokens in this argument are forbidden. (Note that the magic symbol `+` in ⟨*format*⟩ is handled by the `varindextwo` macros, and not by `varindex`).
The token "`!`" above makes no sense for the output in the running text and thus is forbidden behind `*`.
By a heuristic rule, a space is automatically inserted between two entries which generate non-empty text. If the heuristic rule fails, you may always force a space by either "`_`" or "`s`", or forbid a space by "`.`".
If a format is empty, all entries are taken in the order of input. Note that TeX eliminates concatenated spaces, and so you are able to produce an empty format entry only at the end of ⟨*format*⟩ or in front of a "`*`". If you want to force an empty *output* (is this ever useful?), you may use just "`.`" as the entry.

A sort entry is only written to the `*.idx` file if it differs from the text entry.

7 Description of the `varindextwo` macros

`\varindextwoScan`
`\varindextwo`

The macros

```
\varindextwoScan \varindextwo
```

can be used together as parameters ⟨*scan program*⟩ resp. ⟨*main program*⟩ for the `\varindex` command. If `\varindex` is called with these macros, and no explicit format argument is given, these macros generate a format depending on the number of entries in ⟨*main entries*⟩:

a) If there is only one entry in ⟨*main entries*⟩, then the format “`1`” resp. “`*1␣1`” is generated (depending whether the `*` flag was used or not), i. e. the entry is put into the index, and printed if `\varindex` was called with the `*` flag.

b) If there are two entries in ⟨*main entries*⟩, then ⟨*format*⟩ becomes “`␣1!~2␣1!2~`”. For example, if ⟨*main entries*⟩ is “`graulty bazola`”, then the `\index` entries

   graulty
      ∼ bazola, 17
   bazola
      graulty ∼, 17

   are produced.

   You can modify the first format entry with the following flags:

flags	format used	with “`*`” additionally
`_`	`12`	`*12`
`-`	`1-2`	`*1-2`
`^`	`1!~-2` resp. `1!-2`	`*1-2`
`-^`	`1-!~.2` resp. `1-!~2`	`*1-2`
`.`	`1.2`	`*1.2`
`.^`	`1!~.2`	`*1.2`
`/`	`1!2` (without `~`)	`*12`
`<`	no entry	`*12`

   Here, the combinations `-^` and `.^` mean that both flags are used (the order plays no role). The output for `^` respectively `-^` is determined by the switches

`\varindextwoDashTildetrue`
`\varindextwoDashTildefalse`

   `\varindextwoDashTildetrue` (default)
   `\varindextwoDashTildefalse`

   and

`\varindextwoDashSpacetrue`
`\varindextwoDashSpacefalse`

   `\varindextwoDashSpacetrue`
   `\varindextwoDashSpacefalse` (default)

   respectively. The last entry in the above table is an additional format entry which is generated if also the “`*`” flag is used.

   You can modify the second format entry with the following flags:

flags	format used
`,`	`2,1~` resp. `2,1`
`~`	`21`
`=`	`2!1-~`
`,=`	`2,1-~` resp. `2,1-`
`+`	`2-1`
`:`	`2!1.~`
`,:`	`2,1.~` resp. `2,1`
`;`	`2.1`
`>`	no entry
`!`	Append `~` (if not already there)
`?`	Without trailing `~`

Whether the first or the second alternatives in the above forms is used depends on the status of the switch

`\varindextwoCommaTildetrue`
`\varindextwoCommaTildefalse`

`\varindextwoCommaTildetrue` (default)
`\varindextwoCommaTildefalse`

We point out that `\varindextwoCommaTildefalse` was in earlier versions of `varindex` the default (and could not be changed). Note that this switch has no effect if the ! or ? flags are used.

Hint for remembering the symbols: The symbols `_ . -` for the first entry are the same flags as for the output and the same flags which are used in the ⟨*format*⟩ argument. The corresponding symbols `~ :` (and `;`) `=` (and `+`) for the second entry "look similar". The `,` flag is only useful in the second entry (and is the same symbol as in the ⟨*format*⟩ argument). The two exceptional symbols `>` and `<` can be read as "generate only the 'forward-directed' resp. 'backward-directed entry' ".

c) If there are three entries in ⟨*main entries*⟩, then ⟨*format*⟩ becomes "`␣1!~23␣3!12~`". For example, if ⟨*main entries*⟩ is "`graulty of bazola`", then the following two `\index` entries are produced.

graulty
    ∼ of bazola, 17
bazola
    graulty of ∼, 17

The flags have an analogous effect to b). In addition, if the flags `` ` `` resp. `'` are used, the second entry (in our example: "of") is omitted from the output in the first resp. in the second index entry. If the flags `*` and `` ` `` are used together, the second entry is also omitted from the output into the running text.

d) If there are four entries in ⟨*main entries*⟩, then ⟨*format*⟩ becomes "`␣1!~24␣4!31~`". For example, if ⟨*main entries*⟩ is "`graulty of@ OF bazola`", then the following two `\index` entries are produced.

graulty
  ∼ of bazola, 17   (*sorted like* "bazola")
bazola
  graulty OF ∼, 17   (*sorted like* "graulty OF")

In other words, we have a similar effect as in c) except that for the first entry the third word is skipped, and for the second entry the second word is skipped. All flags as in c) can be used with an analogous meaning. Also, if the `*` flag is used, the output into the running text is analogous to c) (the third word is skipped).

e) If there are more than four entries in ⟨*main entries*⟩, then ⟨*formatA*⟩ resp. ⟨*format*A*⟩ is used.

If an explicit ⟨*format*⟩ argument is given to `\varindex` (together with the `\varindextwo` macro as ⟨*main program*⟩), then this format is used, except when it contains the symbol `+`. In this case, this symbol `+` is replaced by the format which would be generated by the rules a)–e). If additionally the `*` flag is used, there is a special rule: If the explicit format contains a `*`, then no additional `*`-output is produced. Otherwise, the `*`-output from a)–e) is prepended to the given ⟨*format*⟩ (with a space at the end, unless the format string is empty). This means that "normally" you have the output from a)–e), unless you write an own explicit `*`-part in the ⟨*format*⟩.
If you do not like the tokens chosen for the default symbols, you can easily choose your own tokens by just replacing `\varindextwoScan` with your own macro (or defining your own "magic" tokens after `\varindextwoScan`, see the example in Section 5.9).

8 Primitive customization

`\varindexUsePlaceholderAfalse`
`\varindexUsePlaceholderBfalse`

You may use the command

```
\varindexUsePlaceholderAfalse
\varindexUsePlaceholderBfalse
```

to ignore the symbol `~` resp. `=` in the format entry. You can easily restore the default by

`\varindexUsePlaceholderAtrue`
`\varindexUsePlaceholderBtrue`

```
\varindexUsePlaceholderAtrue
\varindexUsePlaceholderBtrue
```

If you want to switch off the placeholder only at the beginning of a new entry (resp. of a new subentry), you can use the commands

`\varindexLeadingPlaceholderAfalse`
`\varindexLeadingPlaceholderBfalse`

```
\varindexLeadingPlaceholderAfalse
\varindexLeadingPlaceholderBfalse
```

`\varindexLeadingPlaceholderAtrue`
`\varindexLeadingPlaceholderBtrue`

The default is restored with

```
\varindexLeadingPlaceholderAtrue
\varindexLeadingPlaceholderBtrue
```

`\varindexCommaLeadsfalse`

By default, the "`,`" in the format string is in this connection treated as a symbol generating a new "subentry". You can change this default with the command

```
\varindexCommaLeadsfalse
```

`\varindexCommaLeadstrue`

You may switch back with

```
\varindexCommaLeadstrue
```

`\varindexPlaceholderAText`
`\varindexPlaceholderASort`
`\varindexPlaceholderBText`
`\varindexPlaceholderBSort`

The content of the macros

```
\varindexPlaceholderAText
\varindexPlaceholderASort
\varindexPlaceholderBText
\varindexPlaceholderBSort
```

is used as the placeholder `~` resp. `=` in the index resp. sort entry. Note that if one of these entries expands empty, different rules for the automatic insertion of spaces apply (for the index and sort entry separately).
You may prefer that the placeholder text resp. sort content changes at run time to the context of the previous `!` entry. For example, you may want that the command `\varindex{Gibble Gobble}{2!~2}` produces the index entry

Gibble

  Gibble Gobble, 17

(or is at least *sorted* as such an entry).

`\varindexPreviousATexttrue`
`\varindexPreviousASorttrue`
`\varindexPreviousBTexttrue`
`\varindexPreviousBSorttrue`

You can achieve this with the commands

```
\varindexPreviousATexttrue
\varindexPreviousASorttrue
\varindexPreviousBTexttrue
\varindexPreviousBSorttrue
```

for the text and sort entry and the placeholders `~` and `=`, respectively. With these commands, the content of `\varindexPlaceholderAText` etc is only used as the default if no previous `!` entry was given. You can switch back to the default mode with the respective commands

`\varindexPreviousATextfalse`
`\varindexPreviousASortfalse`
`\varindexPreviousBTextfalse`
`\varindexPreviousBSortfalse`

```
\varindexPreviousATextfalse
\varindexPreviousASortfalse
\varindexPreviousBTextfalse
\varindexPreviousBSortfalse
```

`\varindexCommaExpandfalse`

By default, the "`,`" in the format entry is considered in this connection similar to "`!`". If you want to change this, use the command

```
\varindexCommaExpandfalse
```

`\varindexCommaExpandtrue`

You may switch back with

```
\varindexCommaExpandtrue
```

`\varindexOutSortDashBeg`

Since version 2.2, the dash "`-`" occurring at the beginning of entry (or after "`!`" or even after "`,`" if `\varindexCommaExpandtrue` is in effect) is ignored for sorting. You can switch off this feature with the command

```
\let\varindexOutSortDashBeg\varindexOutSortDash
```

To restore the new default, use

```
\let\varindexOutSortDashBeg\toolboxEmpty
```

The commands

`\varindexSetWordString`
`\varindexSetSortString`
`\varindexSetPlainString`

`\varindexSetWordString{|}`	(Use `|` as separator for entries instead of spaces)
`\varindexSetSortString{>}`	(default: `@`)
`\varindexSetPlainString{<}`	(default: `!`)

set the default "magic" strings used for ⟨*main entries*⟩. The argument of the above macros is intentionally *not* expanded (so that you do not have troubles with active characters like `~`). To force expansion, use e. g.

```
\expandafter\varindexSetWordString\expandafter{\MacroToExpand}
```

or some similar construct.
If you use a different separator than "space" for the entries, you may want to change the rule how braces are eliminated. With the commands

`\varindexEliminateOuterBracetrue`
`\varindexEliminateInnerBracetrue`
`\varindexEliminateOuterBracefalse`
`\varindexEliminateInnerBracefalse`

```
\varindexEliminateOuterBracetrue
\varindexEliminateInnerBracetrue
\varindexEliminateOuterBracefalse
\varindexEliminateInnerBracefalse
```

you may allow resp. forbid the elimination of braces around the entries resp. around ⟨*indextext*⟩ ⟨*sorttext*⟩ and ⟨*outputtext*⟩ With the flag "`1`", ⟨*main entries*⟩ is considered as one single entry, but if `\varindexEliminateOuterBracetrue` is set (which is the default) also in this case an additional pair of braces around main entry is eliminated.
Similarly as above,

`\varindexSetIndexString`	`\varindexSetIndexString{|}`	default: space
`\varindexSetOutputString`	`\varindexSetOutputString{<}`	default: `*`
`\varindexSetTildeAString`	`\varindexSetTildeAString{~}`	(is default)
`\varindexSetTildeBString`	`\varindexSetTildeBString{=}`	(is default)
`\varindexSetSpaceString`	`\varindexSetSpaceString{_}`	(is default)
`\varindexSetSpaceTokString`	`\varindexSetSpaceTokString{s}`	(is default)
`\varindexSetOmitString`	`\varindexSetOmitString{.}`	(is default)
`\varindexSetCommaString`	`\varindexSetCommaString{,}`	(is default)
`\varindexSetDashString`	`\varindexSetDashString{-}`	(is default)
`\varindexSetExclamString`	`\varindexSetExclamString{!}`	(is default)
`\varindexSetStringForNr`	`\varindexSetStringForNr 1{a}`	default: `1`
	`\varindexSetStringForNr{11}{b}`	No default!

set the default "magic" strings used for ⟨*format*⟩. In contrast to before, the arguments are fully expanded (with `\varindexedef`, see Section 9). Note that the last command provides a way to access more than 10 entries!
If you use `\varindexSetIndexString{|}` (with some non-space token), you may still use spaces in the format which then are just ignored.
Avoid that one string is a prefix of another string: In this case, strange effects may happen, and this behavior may even change in future releases of this package. Note that the above effect may arise unintentionally if you use active chars. For this reason, "~" is defined to expand to the letter "~", before the expansion is executed. Maybe in later releases of this package there will be further such redefinitions. You can explicitly use this feature in your own macros by using `\varindexedef` instead of the usual `\edef`, see Section 9.
You can change the output for the text resp sort entry of the various symbols used in the format string. More precisely, you can redefine

`\varindexOutExclam`	`\varindexOutExclam`	Text output for `!`
`\varindexFollowsExclam`	`\varindexFollowsExclam`	Decide whether magic space after `!` is inserted
`\varindexOutTextComma`	`\varindexOutTextComma`	Text output for `,`
`\varindexOutSortComma`	`\varindexOutSortComma`	Sort output for `,`
`\varindexFollowsComma`	`\varindexFollowsComma`	Decide whether magic space after `,` is inserted
`\varindexPreceedsComma`	`\varindexPreceedsComma`	Decide whether magic space before `,` is erased
`\varindexOutTextSpace`	`\varindexOutTextSpace`	Text output for `␣`
`\varindexOutSortSpace`	`\varindexOutSortSpace`	Sort output for `␣`
`\varindexFollowsSpace`	`\varindexFollowsSpace`	Decide whether magic space after `␣` is inserted
`\varindexPreceedsSpace`	`\varindexPreceedsSpace`	Decide whether magic space before `␣` is erased
`\varindexOutTextSpTok`	`\varindexOutTextSpTok`	Text output for `s`
`\varindexOutSortSpTok`	`\varindexOutSortSpTok`	Sort output for `s`
`\varindexFollowsSpTok`	`\varindexFollowsSpTok`	Decide whether magic space after `s` is inserted
`\varindexPreceedsSpTok`	`\varindexPreceedsSpTok`	Decide whether magic space before `s` is erased
`\varindexOutTextDash`	`\varindexOutTextDash`	Text output for `-`
`\varindexOutSortDash`	`\varindexOutSortDash`	Sort output for `-`
`\varindexOutSortDashBeg`	`\varindexOutSortDashBeg`	Sort output for `-` if nothing preceeds.
`\varindexFollowsDash`	`\varindexFollowsDash`	Decide whether magic space after `-` is inserted
`\varindexPreceedsDash`	`\varindexPreceedsDash`	Decide whether magic space before `-` is erased

`\varindexOutTextOmit` `\varindexOutTextOmit` Text output for `.`

`\varindexOutSortOmit` `\varindexOutSortOmit` Sort output for `.`

`\varindexFollowsOmit` `\varindexFollowsOmit` Decide whether magic space after `.` is inserted

`\varindexPreceedsOmit` `\varindexPreceedsOmit` Decide whether magic space before `.` is erased

The meaning is as follows: `\varindexOut...` should just expand to the tokens which should be written into the text resp. sort output. The corresponding macro `\varindexFollows...` is typically defined with one of the following commands:

```
\let\varindexFollows...\varindexNextSpace
\let\varindexFollows...\varindexNoNextSpace
```

In the first case, a space is 'magically' inserted in front of a subsequent token (unless this token erases the magic space). In the second case, no space follows, of course. There is an alternative call:

`\varindexSpaceTexttrue`
`\varindexSpaceTextfalse`
`\varindexSpaceSorttrue`
`\varindexSpaceSortfalse`

```
\def\varindexFollows...{\varindexSpaceTexttrue\varindexSpaceSortfalse}
```

This definition achieves that for the text output a space should follow, but not for the sort output. Of course, you can also use similarly the commands `\varindexSpaceTextfalse` and/or `\varindexSpaceSorttrue` in the above definition (the effect should be obvious). In particular, `\varindexNextSpace` is equivalent to `\varindexSpaceTexttrue\varindexSorttrue`.
The macro `\varindexPreceeds...` is similarly as `\varindexFollows...` with the difference that it is executed *before* the token in question is output. In particular, you can ignore a previous 'magic space'. All of the 'magic space' commands are defined by default with

```
\let\varindexFollows...\toolboxEmpty
\let\varindexPreceeds...\varindexNoNextSpace
```

with the following two exceptions:

```
\let\varindexFollowsExclam\varindexNextSpace
\let\varindexFollowsComma\varindexNextSpace
```

9 Documented Features

The title "documented features" means that these are hacks which allow further customization but which are "documented" in the sense that these hacks will also be available in later versions. (You use an "undocumented" feature if you must use a macro name of the `varindex` package which contains the symbol `@`). If a feature described in this section does not work in the documented way, this is a bug and might be repaired in a later version of `\varindex`.
(In a future release, `varindex` will cook your coffee, too).

The argument ⟨*main entries*⟩ is *never* expanded, so you can actually write macros into the `*.idx` file. The command actually used to write the index is

`\varindexIndex` `\varindexIndex`

(which by default is of course defined as `\index`). Since some implementations of the `\index` command still (partially) expand their argument (which might be considered as a bug), the argument of `\varindexIndex` is translated before the actual call with the aid of `\toolboxMakeHarmless`. If you want to redefine `\varindexIndex` to a personal `\index`-output function, you might want to skip the `\toolboxMakeHarmless` step. This is arranged with the command

`\varindexNoConvert`

```
\varindexNoConvert
```

You can cancel the effect of this command with

`\varindexNoConvert`

```
\varindexNoConvert
```

Even if `\varindexNoConvert` is not in effect, ⟨*main entries*⟩ is not expanded (and in particular, the argument of `\varindexIndex` consists of the corresponding entries in an unexpanded form).

The whole `\varindex` expansion takes place in a group, so all your variables are local to a single call (unless you use `\gdef` (and friends) of course).
There are no `\global` directives used in any macro of this package. In particular, if you call the above macros in a group (or redefine `\varindexIndex`), the effect holds only until the end of the group.

In contrast to ⟨*main entries*⟩, the ⟨*format*⟩ argument is expanded (with `\varindexedef`, see below) (and also ⟨*formatA*⟩ resp. ⟨*format*A*⟩ is expanded before), so you can define abbreviations (even simple conditional abbreviations) for ⟨*format*⟩. Note, however, that the expansion is essentially only evaluated with `\edef`. So, you will probably not want to use e. g. `\relax`, since this command expands to itself (and not into nothing – use `\toolboxEmpty` if you want the latter).
If you want more complex abbreviations, use ⟨*main program*⟩ instead.
In order to help you write conditional abbreviations, the following variables are defined when your macro is expanded (and in particular also in ⟨*main program*⟩). You may of course redefine them appropriately:

`\varindexAppend`

`\varindexAppend` The argument ⟨*pagestyleA*⟩ resp. ⟨*pagestyleB*⟩.

`\varindexCount`

`\varindexCount` A TEX counter containing the number of entries in ⟨*main entries*⟩.

`\varindexLastEntry`

`\varindexLastEntry` This is a macro (*not* a counter) which is usually the value of `\the\varindexCount`. See below.

`\varindexEntry1-...`

`\varindexEntry1` `\varindexEntry2` ... The (index) text occurring in ⟨*Entry1*⟩ ⟨*Entry2*⟩ ...

`\varindexSort1-...`

`\varindexSort1` `\varindexSort2` ... The corresponding sort entry. If no sort entry was given, this is the same as the corresponding `\varindexEntry1-...`

`\varindexPlain1-...`

`\varindexPlain1` `\varindexPlain2` ... The text which should be output in the text. If no such entry was given, this is the same as the corresponding `\varindexEntry1-...`

`\varindexCount` may be larger than 10, and correspondingly there may be also more than 10 different `\varindexEntry/Sort/Plain` macros. If you *add* entries, you have to increase `\varindexCount` correspondingly, otherwise an error is produced if the format string contains an entry larger then `\varindexCount`. However, your additional entries are *not* output for *empty* formats: For empty formats the entries output are 1–`\varindexLastEntry`. So if you want to output also your additional entries in empty formats, you have to set `\varindexLastEntry` to your modified value of `\varindexCount` in ⟨*main program*⟩. You may e. g. achieve this with the following lines:

```
\expandafter
\def\expandafter\varindexLastEntry\expandafter{\the\varindexCount}%
```

It is admissible that `\varindexLastEntry` is larger than `\varindexCount`: In this case all entries until `\varindexLastEntry` are output in empty formats without producing an error (provided, of course, that the corresponding variables `\varindexEntry.../Sort.../Plain...` are defined).
Note that numbers in TEX command names require special handling, i. e., you have to use something like

```
\csname varindexPlain2\endcsname
```

`\toolboxLet`
`\toolboxWithNr`

to access variables. To avoid this, you may want to use the macros `\toolboxLet` and `\toolboxWithNr` of the `toolbox` package. Examples are

```
\toolboxWithNr 1\let{varindexEntry}\toolboxEmpty
\toolboxWithNr {10}\def{varindexSort}{Foo}
\toolboxLet    \mymacro{varindexPlain\the\mycount}
```

These commands set `\varindexEntry1` to `\toolboxEmpty`, `\varindexSort10` to Foo, and `\mymacro` to the content of `\varindexPlain5` (if `\mycount=5`).
All these variables are also available when ⟨*main program*⟩ is expanded. In ⟨*main program*⟩ also the following functions are useful:

`\varindexFormat`

`\varindexFormat` This macro expands to the actual ⟨*format*⟩ which is used. The main purpose of ⟨*main program*⟩ will usually be to (re)define this macro. After ⟨*main program*⟩ has been called, this macro is modified in several ways:

1. `\varindexFormat` is expanded (with `\varindexedef`, see below). (thus, `\varindexFormat` is expanded *before and after* the call of ⟨*main program*⟩).
2. `\ifvarindexStar` is true (see below) a "`*`" resp. "`*␣`" is prepended.
3. If it is only "`*`", it is replaced by "`*␣`".

Note that before execution of ⟨*main program*⟩, no tests on the validity of the format are done: You may use your own symbols to 'communicate' with ⟨*main program*⟩ (provided that ⟨*main program*⟩ eliminates these symbols afterwards).

`\varindexFormatDefault`

`\varindexFormatDefault` This macro expands to ⟨*formatA*⟩ resp. ⟨*format*A*⟩ (in the expanded form) depending whether the `*` flag has been used in the call. Note that this expansion was executed *before* the optional format argument is expanded for the first time.

\ifvarindexstar `\ifvarindexstar` ⟨*ifpart*⟩ [`\else` ⟨*elsepart*⟩] `\fi`

\varindexStarfalse
\varindexStartrue

If the optional `*` was given, ⟨*ifpart*⟩ is executed, otherwise ⟨*elsepart*⟩. This is a TeX `if` command. In particular, by `\varindexStarfalse` resp. `\varindexStartrue` you may change the behavior for future `if`'s. This can be used to prevent/force that a "`*`" resp. "`*␣`" is prepended to the format after the execution of ⟨*main program*⟩. Setting of this variable has no effect in ⟨*scan program*⟩.

\ifvarindexExplicitFormat `\ifvarindexExplicitFormat` This is a TeX `if` command (see above) which is true if the optional format argument ⟨*formatB*⟩ was given.

To "compose" the format, you may want to use the macros

\toolboxDef `\toolboxDef\macrotodefine{`⟨*argumentlist*⟩`}`

\toolboxAppend `\toolboxAppend\macrotoappend{`⟨*argumentlist*⟩`}`

\varindexedef `\varindexedef\macrotodefine{`⟨*argumentlist*⟩`}`

All commands work similar to `\def` resp. `\edef` with two differences:
For `\toolboxDef\macro`, ⟨*argumentlist*⟩ is expanded precisely by one level (for details, see the documentation of the `toolbox` package). Of course, for `\toolboxAppend`, the new definition is appended to the old definition of `\macro`.
`\varindexedef` fully expands ⟨*argumentlist*⟩. However, some active symbols (currently only `~`, but additional symbols might follow in a future release) are deactivated before expansion, so that `~` actually expands to `~` and not to a strange command sequence.
To "decompose" the format, you may want to use one of the macros

\toolboxSplitAt `\toolboxSplitAt{`⟨*arg*⟩`}{`⟨*critical*⟩`}{`⟨*command*⟩`}`

\toolboxMakeSplit `\toolboxMakeSplit{`⟨*critical*⟩`}{`⟨*command*⟩`}`

\varindexMakeSplitExpand `\varindexMakeSplitExpand{`⟨*macros expanding to critical*⟩`}{`⟨*command*⟩`}`

\varindexMakeVarSplit `\varindexMakeVarSplit\variable{`⟨*macros expanding to critical*⟩`}{`⟨*command*⟩`}`

The first two macros are described in the `toolbox` package. The remaining two macros are similar to `\varindexMakeSplit` with the difference that the argument ⟨*critical*⟩ is obtained by expanding ⟨*macros expanding to critical*⟩ with the aid of `\varindexedef`. In the last form, additionally `\variable` is (re)defined to expand to ⟨*critical*⟩ (`\variable` is any free name).
The following instances of a command created by `\toolboxMakeSplit` exist (the content of ⟨*critical*⟩ should be obvious):

\varindexSplitAtIndex `\varindexSplitAtIndex` (generated by `\varindexSetIndexString`)

\varindexSplitAtOutput `\varindexSplitAtOutput` (generated by `\varindexSetOutputString`)

\varindexSplitAtTildeA `\varindexSplitAtTildeA` (generated by `\varindexSetTildeAString`)

\varindexSplitAtTildeB `\varindexSplitAtTildeB` (generated by

`\varindexSetTildeBString`)

\varindexSplitAtSpace
: `\varindexSplitAtSpace` (generated by `\varindexSetSpaceString`; by default, ⟨*critical*⟩ is `_`)

\varindexSplitAtSpaceTok
: `\varindexSplitAtSpaceTok` (generated by `\varindexSetSpaceTokString`; by default, ⟨*critical*⟩ is `s`)

\varindexSplitAtOmit
: `\varindexSplitAtOmit` (generated by `\varindexSetOmitString`)

\varindexSplitAtComma
: `\varindexSplitAtComma` (generated by `\varindexSetCommaString`)

\varindexSplitAtDash
: `\varindexSplitAtDash` (generated by `\varindexSetDashString`)

\varindexSplitAtExclam
: `\varindexSplitAtExclam` (generated by `\varindexSetExclamString`)

\varindexSplitAtNr??
: `\varindexSplitAtNr1` (generated by `\varindexSetStringForNr`)

`\varindexSplitAtNr2`

...

Note that you must use `\csname ....\endcsname` to call e. g. the macro `\varindexSplitAtNr1`. Only those numbers are admissible for `\varindexSplitAtNr??` which have been introduced with `\varindexSetStringForNr` (by default, this is the case for 1–10).

\varindexSplitSpace
There is also the instance

`\varindexSplitSpace` (to be distinguished from `\varindexSplitAtSpace`)

where ⟨*critical*⟩ is the space symbol.
In addition, you may use the variables

\varindexIndexString `\varindexIndexString`

\varindexOutputString `\varindexOutputString`

\varindexTildeAString `\varindexTildeAString`

\varindexTildeBString `\varindexTildeBString`

\varindexSpaceString `\varindexSpaceString`

\varindexSpaceTokString `\varindexSpaceTokString`

\varindexOmitString `\varindexOmitString`

\varindexCommaString `\varindexCommaString`

\varindexDashString `\varindexDashString`

\varindexExclamString `\varindexExclamString`

\varindexNr??String `\varindexNr1String \varindexNr2String ...`

which expand to the corresponding strings.
All the previous macros should not be redefined “by hand”. They are implicitly redefined by the `\varindexSet...` commands (which partially also do other tasks).

\toolboxDropBrace To drop possible braces, use the command

```
\toolboxDropBrace\variable
```

of the `toolbox` package.
In ⟨*scan program*⟩ you may already want to call `\toolboxMakeSplit`: In this way, the choices for the "magic" tokens are made in the (usually small) argument ⟨*scan program*⟩, and in this sense, you can keep your (possibly rather complex) macro ⟨*main program*⟩ "customizable" (i. e. you can use the same macro also with a different set of "magic" strings). However, the main task of ⟨*scan program*⟩ is to introduce new flags with

\varindexNewFlag

```
\varindexNewFlag⟨token⟩\position[⟨program⟩\varindexEndOfFlag]
```

or

```
\varindexNewFlag⟨token⟩\position[\programmacro]
```

Here, `\position` is an (undefined) macro, and ⟨*token*⟩ an arbitrary token. The `[program(macro)]` part is optional and is explained later. If ⟨*token*⟩ appears in ⟨*flags*⟩, then `\position` is defined to expand to a (literally) number, namely the (last) position where token was given inside the ⟨*flags*⟩ list (counting from 0). For example, if

```
\varindexNewFlag ,\commapos
\varindexNewFlag -\minuspos
\varindexNewFlag .\pointpos
```

are used in `\programA`, then in the call

```
\varindex(){\programA}{\programB},-**-{}
```

the variable `\commapos` will expand in `\programB` to `0`, while `\minuspos` will expand to `4` (the last position is taken). Finally, `\pointpos` is `\undefined` (unless you defined `\pointpos` differently *after* the call of `\varindexNewFlag`; in this case, this is the default). If `\varindexNewFlag` is called multiple times with the same token, only the *last* call with this token takes effect.

\varindexStarPos
\varindexOneEntry

The flags `*` and `1` are introduced in this way with

```
\varindexNewFlag *\varindexStarPos
\varindexNewFlag 1\varindexOneEntry
```

before ⟨*scan program*⟩ is executed. This means:

1. Usually, `\varindexStarPos` contains the (last) position of `*` (resp. is `\undefined`). Moreover, if you define `\varindexStarPos` in ⟨*scan program*⟩ or in some flag, you get the same effect as if `*` had been used. An analogous remark holds for `\varindexOneEntry`.

2. If you introduce `*` with `\varindexNewFlag`, the `*` looses its original meaning. The same holds for `1`.

If you have added a `[program(macro)]` part in the call of `\varindexNewFlag`, this part is expanded whenever the flag introduced by `\token` is used in the call of `\varindex` (note that it is not excluded that this happens several times within one call). More precisely, program is expanded *after* the variable `\position` has been set to the corresponding value, so you may already use `\position` in the program part.

`\varindexEndOfFlag` **Important:** The last token expanded in program *must* be `\varindexEndOfFlag`. This is not nice but has to do with the way TEX parses its arguments. Also if you use the form `\programmacro`, the *very last* token expanded must be `\varindexEndOfFlag`. Even a construction like

```
\def\myprogrammacro{\ifx....
  \CallSomeMacroWithvarindexEndOfFlagAtTheEnd
\fi
\varindexEndOfFlag}
```

is forbidden: In `\CallSomeMacroWithvarindexEndOfFlagAtTheEnd` an error would occur at the end, since still the tokens `\fi\varindexEndOfFlag` are in the tokenlist when EndOfFlag is reached there. As a workaround, you may e. g. use

```
\def\myprogrammacro{\ifx...
    \def\execute{....\varindexEndOfFlag}%
    \else
    \def\execute{\varindexEndOfFlag}%
    \execute}
```

If you use the form `[\programmacro]`, your macro may even read additional parameters. These parameters are expected in the call of `\varindex` *behind* the flag you have introduced. So you may actually use flags with parameters. For example, if `\scanprogram` contains a macro like

```
\varindexNewFlag -\minuspos[\readpara]
```

and you have defined

```
\def\readpara#1{\def\merk{#1}\varindexEndOfFlag}
```

then the call

```
\varindex(){\scanprogram}{\mainprg}*-{Foo}-{Foo 2}*{Entries}
```

is admissible, and in ⟨*main program*⟩, `\merk` will have the value “Foo 2”.
If you are more familiar with TEX, you can even allow *optional* arguments following your flag: The value of the next (non-space) token is at the call of `\programmacro` already saved into the macro

`\varindexNextToken`

```
\varindexNextToken
```

so you can just use it to test for e. g. “`[`”. In this connection, you may also want to use the commands

`\varindexTestAndExec`
`\varindexSkipAndExec`

```
\varindexTestAndExec
\varindexSkipAndExec
```

Please, see the program text how these commands are applied.

Since version 2.1 you can also hack in personal extensions of the format string. To do this, use the command

`\varindexNewFormat`

`\varindexNewFormat\`⟨*splitcommand*⟩`{`⟨*action*⟩`}`

Here, `\`⟨*splitcommand*⟩ is a command generated by `\toolboxMakeSplit` or friends (preferably by `\varindexMakeSplitExpand`, because the command should act on the format string which is expand with `\varindexedef`). The string where it splits is the new string you can use in the format argument after this call. For each occurrence of the corresponding string in the format argument, ⟨*action*⟩ will be executed. If `\`⟨*splitcommand*⟩ splits at a string which already had a previous meaning in the format string (or which is a prefix to such a string) the old meaning of this string in the format string is overridden.
Typically, action will contain the following commands: One action will probably be to output a desired token (sequence) via

`\varindexTokensOut`

`\varindexTokensOut{`⟨*text token*⟩`}{`⟨*sort token*⟩`}`

or

`\varindexTokensOutExpand`

`\varindexTokensOutExpand`⟨*macro for text token*⟩⟨*macro for sort token*⟩

In the first form, ⟨*text token*⟩ resp. ⟨*sort token*⟩ is the token sequence put into the index or running text respectively into the sort entry of the index for the corresponding format entry. The second form is similar with the only difference that the arguments must be single macros which expand to ⟨*text token*⟩ and ⟨*sort token*⟩, respectively.

`\ifvarindexIndexMode`

The variable

```
\ifvarindexIndexMode
```

can be used to test whether the output goes into the running text or into the index (i. e. whether a "`*`" preceeded the current entry). For output into the text, ⟨*sort token*⟩ is ignored, of course).
Another action in `\varindexNewFormat` will probably be to take care of the magic space flags. This is achieved by a call of `\varindexNextSpace` or `\varindexNoNextSpace` (or separately via `\varindexSpaceTexttrue/false` resp. `\varindexSpaceTexttrue/false`); see the earlier description. The magic space flags are taken into account in `\varindexTokensOut`. Thus, if you want to ignore the previous flag for some reason you should set them correspondingly before this call. However, after the call you should also set them correspondingly for further processing.
Example:

```
\toolboxMakeSplit{:}{splitAtColon}
  \varindexNewFormat\splitAtColon{%
     \varindexNoNextSpace
     \ifvarindexIndexMode
        \varindexTokensOut{:}{}\varindexSpaceTexttrue
     \else
        \varindexTokensOut{---}{}\varindexNoNextSpace
     \fi}
```

defines a new format entry "`:`" which has the meaning that a colon (automagically followed by a space) is put into the index but not into the sort entry. Moreover, in the running text, the colon appears as a long dash with no space followed. In any case, there is no magic space output in front of the colon.

As an alternative action in \varindexNewFormat, you can also call the default commands for the format entries. The corresponding macros are

\varindexAddSpace	\varindexAddSpace	␣
\varindexAddSpTok	\varindexAddSpTok	s
\varindexAddOmit	\varindexAddOmit	.
\varindexAddDash	\varindexAddDash	-
\varindexAddComma	\varindexAddComma	,
\varindexAddExclam	\varindexAddExclam	!
\varindexAddTildeA	\varindexAddTildeA	~
\varindexAddTildeB	\varindexAddTildeB	=
\varindexAddNumber	\varindexAddNumber{⟨*number*⟩}	0-9 (⟨*number*⟩ is 1, 2, ...)

The precise meaning of these macros is intentionally *not* documented, because some details or side effects might change in a future release of varindex. But just for this reason, it might be advantageous to use the above macros instead of writing personal substitutes which may fail to have such side effects.
Example:

```
\varindexMakeSplitExpand{~-}{splitPhrase}% It is important to expand
                                          % (to have correct ~ catcode)
\varindexNewFormat\splitPhrase{%
   \varindexAddTildeA
   \let\remember\varindexPreceedsDash
   \let\varindexPreceedsDash\toolboxEmpty
   \varindexAddDash
   \let\varindexPreceedsDash\remember}
```

After the above command, “~” and “-” have their usual meaning in the format string except when they follow immediately in the form “~-”. In this case, the behavior of - changes as if \varindexPreceedsDash were empty (which has the effect that the output looks like “∼ -” instead of “∼-”).
Note that although ~ is a prefix to ~-, the converse is not true: Thus, the above command does not change the previous meaning of ~ (and of course also not of -).

Some hack: TeX ignores leading spaces in the argument list of a “normal” macro. This has the effect that you should be able to insert spaces *between* any of your arguments without any trouble. If this does not work the expected way, you can use the command

\varindexArgumentSpace
\varindexSkipTricky

```
\let\varindexArgumentSpace\varindexSkipTricky
```

which implements an own macro which does this task. Sadly, this can only be done with some catcode trickery which in turn might bring you in trouble with some (*very*) exotic packages. You can restore the default with

\varindexSkipDefault

```
\let\varindexArgumentSpace\varindexSkipDefault
```

All above customization commands/variables may be used anytime between two `\varindex` calls. They take effect with the next `\varindex` call.
The macros `\varindextwoScan` and `\varindextwo` are considered as a (very useful) *example* for the customization of the `\varindex` command. If you need further customization, have a look at their definition first.