# The marks mechanism*

Frank Mittelbach, LaTeX Project†

October 30, 2026

**Abstract**

Marks are used to communicate information about the content of a page to the output routine. For example, in order to construct running headers, the output routine needs information about which section names are present on a page, and this information is passed to it through the mark system. However, marks may also be used for other purposes. This module provides a generalized mechanism for marks of independent classes.

## Contents

**1 Introduction** **2**

**2 Design-level and code-level interfaces** **2**
2.1 Use cases for conditionals . . . . . 4
2.2 Understanding regions . . . . . 5
2.3 Debugging mark code . . . . . 6

**3 Application examples** **7**

**4 Legacy LaTeX 2ε interface** **7**
4.1 Legacy design-level and document-level interfaces . . . . . 7
4.2 Legacy interface extensions . . . . . 8

**5 Notes on the mechanism** **8**

**6 Public interfaces for packages such as multicol** **10**

**7 Internal functions for the standard output routine of LaTeX** **11**

**Index** **12**

*This file has version v1.1f dated 2026-06-26, © LaTeX Project.

†E-mail: latex-team@latex-project.org

# 1 Introduction

The TEX engines offer a low-level mark mechanism to communicate information about the content of the current page to the asynchronous operating output routine. It works by placing `\mark` commands into the source document. When the material for the current page is assembled in box 255, TEX scans for such marks and sets the commands `\topmark`, `\firstmark` and `\botmark`. The `\firstmark` receives the content of the first `\mark` seen in box 255 and `\botmark` the content of the last mark seen. The `\topmark` holds the content of the last mark seen on the previous page or more exactly the value of `\botmark` from the previous page. If there are no marks on the current page then all three are made equal to the `\botmark` from the previous page.

This mechanism works well for simple formats (such as plain TEX) whose output routines are only called to generate pages. It fails, however, in LATEX (and other more complex formats), because here the output routine is sometimes called without producing a page, e.g., when encountering a float and placing it into one of the float regions. In that case the output routine is called, determines where to place the float, alters the goal for assembling text material (if the float was added to the top or bottom region) and then it resumes collecting textual material.

As a result the `\botmark` gets updated and so `\topmark` no longer reflects the situation at the top of the next page when that page is finally boxed.

Another problem for LATEX was that it wanted to use several "independent" marks and in the early implementations of TEX there was only a single `\mark` command available. For that reason LATEX implemented its own mark mechanism where the marks always contained two parts with their own interfaces: `\markboth` and `\markright` to set marks and `\leftmark` and `\rightmark` to retrieve them.

However, this extended mechanism (while supporting scenarios such as chapter/section marks) was far from general. The mark situation at the top of a page (i.e., `\topmark`) remained unusable and the two marks offered were not really independent of each other because `\markboth` (as the name indicates) was always setting both.

The new mechanism overcomes both issues:

- It provides arbitrarily many, fully independent named marks, that can be allocated and, from that point onwards, used.
- It offers access for each such marks to retrieve its top, first, and bottom values separately.
- Furthermore, the mechanism is augmented to give access to marks in different "regions" which may not be just full pages.

# 2 Design-level and code-level interfaces

The interfaces are mainly meant for package developers, but they are usable (with appropriate care) also in the document preamble, for example, when setting up special running headers with fancyhdr, etc. They are therefore available both as CamelCase commands as well as commands for use in the L3 programming layer. Both are described together below.

`\NewMarkClass` `\NewMarkClass {⟨class⟩}`
`\mark_new_class:n` `\mark_new_class:n {⟨class⟩}`

Declares a new ⟨*class*⟩ of marks to be tracked by LaTeX. Each ⟨*class*⟩ must be declared before it is used.

Mark classes can only be declared before `\begin{document}`.

`\InsertMark` `\InsertMark {⟨class⟩} {⟨text⟩}`
`\mark_insert:nn` `\mark_insert:nn {⟨class⟩} {⟨text⟩}`

Adds a mark to the current galley for the ⟨*class*⟩, containing the ⟨*text*⟩.

It has no effect in places in which you can't place floats, e.g., a mark inside a box or inside a footnote never shows up anywhere.

If used in vertical mode it obeys LaTeX's internal `@nobreak` switch, i.e., it does not introduce a breakpoint if used after a heading. If used in horizontal mode it doesn't handle spacing (like, for example, `\index` or `\label` does, so it should be attached to material that is typeset.

`insertmark` `\AddToHook {insertmark} {⟨code⟩}`

When marks are inserted, the mark content may need some special treatment, e.g., by default `\label`, `\index`, and `\glossary` do not expand at this time (but only later if and when the mark content is actually used. In order to allow packages to augment or alter this setup there is a public hook `insertmark` that is executed at this point. It runs in a group so local modification to commands are only applied to the ⟨*text*⟩ argument of `\InsertMark` or `\mark_insert:nn`.

`\TopMark` ⋆ `\TopMark [⟨region⟩] {⟨class⟩}`
`\FirstMark` ⋆ `\FirstMark [⟨region⟩] {⟨class⟩}`
`\LastMark` ⋆ `\LastMark [⟨region⟩] {⟨class⟩}`
`\mark_use_top:nn` ⋆ `\mark_use_top:nn {⟨region⟩} {⟨class⟩}`
`\mark_use_first:nn` ⋆ `\mark_use_first:nn {⟨region⟩} {⟨class⟩}`
`\mark_use_last:nn` ⋆ `\mark_use_last:nn {⟨region⟩} {⟨class⟩}`

These functions expand to the appropriate mark ⟨*text*⟩ for the given ⟨*class*⟩ in the specified ⟨*region*⟩. The default ⟨*region*⟩ in the design-level commands is `page`. Note that with the L3 layer commands there are no optional arguments, i.e., both arguments have to be provided.

**TeXhackers note:** The result is returned within the `\unexpanded` primitive (`\exp_not:n`), which means that the ⟨*text*⟩ does not expand further when appearing in an `x`-type or `e`-type argument expansion.

The "first" and "last" marks are those seen first and last in the current region/page, respectively. The "top" mark is the last mark of the ⟨*class*⟩ seen in an earlier region, i.e., the ⟨*text*⟩ what would be "current" at the very top of the region.

**Important!** The commands are only meaningful inside the output routine, in other places their result is (while not random) unpredictable due to the way LaTeX cuts text material into pages. There is, however, one exception: if you produce multiple columns using the multicol package, it is possible to retrieve mark values from the regions `first-column`, `last-column`, `mcol-1`, `mcol-2`,... directly after the environment has ended. This can, for example, be useful if a `multicols` has been be used inside a box.

Currently, ⟨*region*⟩ is one of `page`, `previous-page`, `column`, `previous-column`, `first-column`, `last-column`, and `mcol-1` (first column in a `multicols`), `mcol-2` (second column in a `multicols`), up to `mcol-20` (twentieth column in a `multicols`). See section 2.2 for discussion of how these regions behave and how one can make use of them.

`\IfMarksEqualTF` ⋆
`\IfMarksEqualT` ⋆
`\IfMarksEqualF` ⋆
`\mark_if_eq:nnnnTF` ⋆
`\mark_if_eq:nnnnnnTF` ⋆

```
\IfMarksEqualTF    [⟨region⟩] {⟨class⟩} {⟨pos1⟩} {⟨pos2⟩} {⟨true⟩} {⟨false⟩}
\mark_if_eq:nnnnTF {⟨region⟩} {⟨class⟩} {⟨pos1⟩} {⟨pos2⟩} {⟨true⟩} {⟨false⟩}
\mark_if_eq:nnnnnnTF {⟨region1⟩} {⟨class1⟩} {⟨pos1⟩}
                     {⟨region2⟩} {⟨class2⟩} {⟨pos2⟩} {⟨true⟩} {⟨false⟩}
```

These conditionals allow you to compare the content of two marks and act based on the result. The commands work in an expansion context, if necessary.

It is quite common when programming with marks to need to interrogate conditions such as whether marks have appeared on a previous page, or if there are multiple marks present on the current page, and so on. The tests above allow for the construction of a variety of typical test scenarios, with three examples presented below.

The first two conditionals cover only the common scenarios. Both marks are picked up from the same ⟨*region*⟩ (by default `page`) and they have to be of the same ⟨*class*⟩.[1] The ⟨$pos_i$⟩ argument can be either `top`, `first`, or `last`.

Important to note is that the comparison is not with respect to the textual content of the marks but whether or not they originated from the same `\InsertMark` command (or the L3 layer version `\mark_insert:nn`).

If you wish to compare marks across different regions or across different classes, you have to do it using the generic test only available in the L3 programming layer or do it manually, i.e., get the marks and then compare the values yourself.[2]

## 2.1 Use cases for conditionals

However, the basic version is enough for the following typical use cases:

**Test for at most one mark of class `myclass` on current page:** If the first and last mark in a region are the same then either there was no mark at all, or there was at most one. To test this on the current page:

```
\NewMarkClass{myclass}
\IfMarksEqualTF{myclass}{first}{last}
                { <zero or one mark> }{ <two or more marks> }
```

**Test for no mark of class `myclass` in the previous page:** If the top mark is the same as the first mark, there is no mark in the region at all. If we wanted to do this test for the previous page:

```
\IfMarksEqualTF[previous-page]{myclass}{top}{first}
                { <no marks> }{ <at least one mark> }
```

Comparing `top` and `last` would give you the same result.

**Test for zero, one, or more than one:** Combining the two tests from above you can test for zero, one or more than one mark.

[1] If an undeclared mark class is used the tests return *true* (not an error).

[2] If two undeclared mark classes are compared the result is always *true*; if a declared and an undeclared mark class is used it is always *false*.

```
\IfMarksEqualTF{myclass}{top}{first}
                { <no marks> }
                {\IfMarksEqualTF{myclass}{first}{last}
                  { <exactly one mark> }{ <more than one mark> }}
```

If you need one of such tests more often (or if you want a separate command for it for readability), then consider defining:

```
\providecommand\IfNoMarkTF[2][page]{\IfMarksEqualTF[#1]{#2}{first}{last}}
```

## 2.2 Understanding regions

If a page has just been finished then the region `page` refers to the current page and `previous-page`, as the name indicates, refers to the page before the current page. This means you are able to access mark information for the current page as well as for the page before (as long as you are inside the output routine) without the need to explicitly save that information beforehand. The `page` region is the region that is most often queried, which is why commands like `\FirstMark` use that region by default.

In single column documents the `column` is the same as the `page` region, but in two-column documents (if not produced by `multicols`), `column` refers to the current column that just got finished and previous-column to the one previously finished. Code for running headers is (in standard LaTeX) evaluated only after both columns have been assembled, which is another way of saying that in that case `previous-column` refers to the left column and `column` to the right column. However, to make these somewhat easier to use, there are also aliased names for these two regions: `first-column` and `last-column`.[3]

Note that you can only look backwards at already processed regions, e.g., in a `twoside` document finishing a recto (odd, right-hand) page you can access the data from the facing verso (left-hand) page, but if you are finishing a left-hand page you can't integrate data from the upcoming right-hand page. If such a scenario needs to be realized then it is necessary to save the left-hand page temporarily instead of finalizing it, process material for the right-hand page and once both are ready, attach running headers and footers and shipout out both in one go.[4]

The situation starts getting rather complex if you allow for multiple columns in the way they are supported by the multicol package. In this case you might have a variable number of "columns" on a single page to be shipped out. And even if not, then a `multicols` might start or end in the middle of the page; in either case, the regions `column` and `previous-column` become rather meaningless and you should therefore not use them.[5] Instead, the algorithm offers `mcol-1`, `mcol-2`, `mcol-3`, etc., to represent the columns in the `multicols` on the current page to be shipped out. If there is more than one `multicols` on the current page then in the output routine only the columns of the last one will be accessible.

[3] The region is called "last-column" not "second-column" in anticipation of extending the mechanism to multiple columns, where first and last would still make sense. There aren't any `previous-first-column` and `previous-last-column` regions to access the corresponding columns from the previous page.

[4] As of now that scenario is not (yet) officially supported but it would be possible to achieve this using the shipout hooks to store the verso page and then on the next shipout use the hook to shipout both with running headers and footers attached.

[5] They return something, because they represent the last two columns of the `multicols` when you are inside the output routine, but that is obviously of little use.

These provisions cover, out of the box, a number of layouts and use cases, but obviously not all. However, more cases can be supported by storing away mark information during the processing. Here is the full algorithm:

- The `column` region is used by the "current column" that is being built (moving through all columns with `previous-column` trailing behind (to handle top marks properly).
- When the `multicols` starts, the `column` region is cleared, i.e., from that point on it looks as if there have not been any marks so far. This will make sure that the top mark in the first column is always empty.
- If the `multicols` extends beyond the current page, then the material designated for the current page is split into columns. The `column` region is used to represent each column in turn.
  - First we copy the current data from `column` to `previous-column`. Then the mark data from the current column is placed into the `column` region. Then we alias `column` to `mcol-1`.
  - These steps are repeated for all columns of the `multicols` environment.
  - Finally, the first and the last column of that page is also made available as `first-column` and `last-column`, respectively.
- All those marks inside any of the columns are also available in the `page` region. Thus, if you are interested in the top, first, or last mark of a specific class on the whole page you simply need to query for it in the `page` region.
- If the `multicols` continues across several pages then this algorithm above is repeated for each page, except that the `column` region is not cleared again. This means that the top mark of the first column of the next page will be the last mark of the last column from the previous page.
- When the `multicols` finishes the remaining material for the current page is balanced to produce columns of roughly equal height.
- Again `column` and `previous-column` are used while this balancing happens and `mcol-1`, `mcol-2`, etc., are used to represent the column regions and `first-column` and `last-column` are set appropriately.
- Then the balanced set of columns is returned back to the page (since there may be space for further material). In addition, all marks inside that material are reinserted so that they become available in the `page` region.
- As a side effect, it is possible (and useful in certain circumstances) to query for mark classes directly after the `multicols` has ended without the need to be inside the output routine. The regions that can be queried this way are `mcol-1`, `mcol-2`, etc. (up to the number of columns the multicol had) and `first-column` and last-column.

## 2.3 Debugging mark code

`\DebugMarksOn`
`\DebugMarksOff`
`\mark_debug_on:`
`\mark_debug_off:`

`\DebugMarksOn ... \DebugMarksOff`

Commands to turn the debugging of mark code on or off. The debugging output is rather coarse and not really intended for normal use at this point in time.

# 3 Application examples

If you want to figure out if a break was taken at a specific point, e.g., whether a heading appears at the top of the page, you can do something like this:

```
\newcounter{breakcounter}
\NewMarkClass{break}
\newcommand\markedbreak[1]{\stepcounter{breakcounter}%
                           \InsertMark{break}{\arabic{breakcounter}%
                           \penalty #1\relax
                           \InsertMark{break}{-\arabic{breakcounter}}
```

To test if the break was taken you can test if `\TopMark{break}` is positive (taken) or negative (not taken) or zero (there was never any marked break so far). The absolute value can be used to keep track of which break it was (with some further coding).

*to be extended with additional application examples*

# 4 Legacy LaTeX 2ε interface

Here we describe the interfaces that LaTeX 2ε offered since the early nineties and some minor extensions.

## 4.1 Legacy design-level and document-level interfaces

`\markboth` `\markboth {⟨left⟩} {⟨right⟩}`
`\markright` `\markright {⟨right⟩}`

LaTeX 2ε uses two marks which aren't fully independent. A "left" mark generated by the first argument of `\markboth` and a "right" mark generated by the second argument of `\markboth` or by the only argument of `\markright`. The command `\markboth` and `\markright` are in turn called from heading commands such as `\chaptermark` or `\sectionmark` and their behavior is controlled by the document class.

For example, in the article class with `twoside` in force the `\sectionmark` will issue `\markboth` with an empty second argument and `\subsectionmark` will issue `\markright`. As a result the left mark will contain chapter titles and the right mark subsection titles.

Note, however, that in one-sided documents the standard behavior is that only `\markright` is used, i.e., there will only be right-marks but no left marks!

\leftmark ⋆ \leftmark
\rightmark ⋆ \rightmark

These functions return the appropriate mark value from the current page and work as before, that is `\leftmark` will get the last (!) left mark from the page and `\rightmark` the first (!) right mark.

In other words they work reasonably well if you want to show the section title that is current when you are about to turn the page and also show the first subsection title on the current page (or the last from the previous page if there wasn't one). Other combinations can't be shown using this interface.

The commands are fully expandable, because this is how they have been always defined in LaTeX. However, this is of course only true if the content of the mark they return is itself expandable and does not contain any fragile material. Given that this can't be guaranteed for arbitrary content, a programmer using them in this way should use `\protected@edef` and *not* `\edef` to avoid bad surprises as far as this is possible, or use the new interfaces (`\TopMark`, `\FirstMark`, and `\LastMark`) which return the ⟨*text*⟩ in `\exp_not:n` to prevent uncontrolled expansion.

### 4.2 Legacy interface extensions

The new implementation adds three mark classes: `2e-left`, `2e-right` and `2e-right-nonempty` and patches `\markboth` and `\markright` slightly so that they also update these new mark classes, so that the new classes work with existing document classes.

As a result you can use `\LastMark{2e-left}` and `\FirstMark{2e-right}` instead of `\leftmark` and `\rightmark`. But more importantly, you can use any of the other retrieval commands to get a different status value from those marks, e.g., `\LastMark{2e-right}` would return the last subsection on the page (instead of the first as returned by `\rightmark`).

The difference between `2e-right` and `2e-right-nonempty` is that the latter will only be updated if the material for the mark is not empty. Thus `\markboth{title}{}` as issued by, say, `\sectionmark`, sets a `2e-left` mark with `title` and a `2e-right` mark with the empty string but does not add a `2e-right-nonempty` mark.

Thus, if you have a section at the start of a page and you would ask for `\FirstMark{2e-right}` you would get an empty string even if there are subsections on that page. But `2e-right-nonempty` would then give you the first or last subsection on that page. Of course, nothing is simple. If there are no subsections it would tell you the last subsection from an earlier page. We therefore need comparison tools, e.g., if top and first are identical you know that the value is bogus, i.e., a suitable implementation would be

```
\IfMarksEqualTF{2e-right-nonempty}{top}{first}
               { <appropriate action if there was no real mark> }
               {\FirstMark{2e-right-nonempty}}
```

## 5 Notes on the mechanism

In contrast to vanilla TeX, ε-TeX extends the mark system to allow multiple independent marks. However, it does not solve the `\topmark` problem which means that LaTeX still needs to manage marks almost independently of TeX. The reason for this is that the more complex output routine used by LaTeX to handle floats (and related structures)

means that `\topmark(s)` remain unreliable. Each time the output routine is fired up, TEX moves `\botmark` to `\topmark`, and while ε-TEX extends this to multiple registers the fundamental concept remains the same. That means that the state of marks needs to be tracked by LATEX itself. An early implementation of this package used TEX's `\botmark` only to ensure the correct interaction with the output routine (this was before the ε-TEX mechanism was even available). However, other than in a prototype implementation for LATEX3, this package was never made public.

The new implementation now uses ε-TEX's marks as they have some advantages, because with them we can leave the mark text within the galley and only extract the marks during the output routine when we are finally shipping out a page or storing away a column for use in the next page. That means we do not have to maintain a global data structure that we have to keep in sync with informational marks in the galley but can rely on everything being in one place and thus manipulations (e.g. reordering of material) will take the marks with them without a need for updating a fragile linkage.

To allow for completely independent marks we use the following procedure:

- For every type of marks we allocate a mark class so that in the output routine TEX can calculate for each class the current top, first, and bottom mark independently. For this we use `\newmarks`, i.e., one marks register per class.
- As already mentioned firing up an output routine without shipping out a page means that TEX's top marks get wrong so it is impossible to rely on TEX's approach directly. What we do instead is to keep track of the real marks (for the last page or more generally last region) in some global variables.
- These variables are updated in the output routine at defined places, i.e., when we do real output processing but not if we use special output routines to do internal housekeeping.
- The trick we use to get correctly updated variables is the following: the material that contains new marks (for example the page to be shipped out) is stored in a box. We then use TEX primitive box splitting functions by splitting off the largest amount possible (which should be the whole box if nothing goes really wrong). While that seems a rather pointless thing to do, it has one important side effect: TEX sets up first and bottom marks for each mark class from the material it has split off. This way we get the first and last marks (if there have been any) from the material in the box.
- The top marks are simply the last marks from the previous page or region. And if there hasn't been a first or bottom mark in the box then the new top mark also becomes new first and last mark for that class.
- That mark data is then stored in global token lists for use during the output routine and legacy commands such as `\leftmark` or new commands such as `\TopMark` simply access the data stored in these token lists.

That's about it in a nutshell. Of course, there are some details to be taken care of—those are discussed in the implementation sections.

# 6 Public interfaces for packages such as multicol

The functions in this section are public so that packages can make use of them. However, this must be done with great care, e.g., `\mark_update_structure_from_material:nn` updates the global mark structure and can therefore be used only in places where such an update is meaningful, e.g., in special output routines. Elsewhere, a change to the mark structure would break the whole mechanism and querying the marks would return incorrect data.

`\mark_update_structure_from_material:nn` `\mark_update_structure_from_material:nn {⟨region⟩} {⟨material with marks⟩}`

Helper function that inspects the marks inside the second argument and assigns new mark values based on that to the ⟨*region*⟩ given in the first argument. For this it first copies the mark structure from ⟨*region*⟩ to `previous-`⟨*region*⟩ and then takes all last mark values currently in the region and makes them the new top mark values. Finally it assigns new first and last values for all mark classes based on what was found in the second argument.

As a consequence, the allowed values for ⟨*region*⟩ are `page` and `column` because only they have `previous-...` counterparts.

Another important aspect to keep in mind is that marks are recognized only if they appear on the top level, e.g., if we want to process material stored in boxes we need to put it unboxed (using `\unvcopy` etc.) into the second argument.

`\mark_copy_structure:nn` `\mark_copy_structure:nn {⟨alias⟩} {⟨source⟩}`

Helper function that copies all mark values in the ⟨*source*⟩ region to ⟨*alias*⟩, i.e., make the structures identical. Used to update the `previous-...` structures inside `\mark_update_structure_from_material:nn` and `first-column` and `last-column` structures inside the internal commands `\__mark_update_singlecol_structures:` or `\__mark__update_dblcol_structures:`.

`\mark_set_structure_to_err:n` `\mark_set_structure_to_err:n {⟨region⟩}`

Helper function that sets all mark values in the ⟨*region*⟩ to an error message. This is currently used for `last-column` at times where using marks from it would be questionable/wrong, i.e., when we have just processed the first column in a two-column document.

`\mark_clear_structure:n` `\mark_clear_structure:n {⟨region⟩}`

Helper function that sets all mark values in the ⟨*region*⟩ to empty. This is currently used for `column` when a multicol environment starts; this is because it wouldn't make sense if the top mark in the first column returned the last mark from a previous multicol (which may have been much earlier, with intermediate material).

\mark_get_marks_for_reinsertion:nNN \mark_get_marks_for_reinsertion:nNN {⟨*source*⟩}
⟨*token-list-var for collecting first marks*⟩
⟨*token-list-var for collecting last marks*⟩

Helper function for extracting marks that would otherwise get lost, for example when they are hidden inside a box. This helper does not update mark structures and can therefore be used outside the output routine as well.

It collects all the top-level marks from inside the ⟨*source*⟩ and then adds suitable `\mark_insert:nn` commands to each of the two token lists. These token lists can then be executed at the right place to reinsert the marks, e.g., directly after the box. This is, for example, going to be used[6] by multicol when a short balanced `multicols` is returned to the galley for typesetting.

If the ⟨*source*⟩ consists of a single vertical box (plus possibly followed by some glue but nothing else) then the box is unpacked and the top-level marks are collected from its content. However, if it is not a vertical box or there are other data then nothing is unpacked and you have to do the unpacking yourself to get at the marks inside.

It is quite likely that one only needs a single token list for returning the `\mark_insert:nn` statements. If that is the case this command may change to take only two arguments.

# 7 Internal functions for the standard output routine of LaTeX

The functions in this section are tied to the output routine and used in the interface to LaTeX 2ε and perhaps at some later time within a new output routine for LaTeX. They are not (yet) meant for general use and are therefore made internal, even though we already use them in multicol. Internal means that `@@` automatically gets replaced in the code (and in the documentation) so we have to give it a suitable value.

```
1 ⟨@@=mark⟩
```

__mark_update_singlecol_structures: __mark_update_singlecol_structures:

LaTeX 2ε integration function in case we are doing single column layouts. It assumes that the page content is already stored in `\@outputbox` and processes the marks inside that box. It is called as part of `\@opcol`.

__mark_update_dblcol_structures: __mark_update_singlecol_structures:

LaTeX 2ε integration function mark used when we are doing double column documents. It assumes that the page content is already stored in `\@outputbox` and processes the marks inside that box. It then does different post-processing depending on the start of the switch `\if@firstcolumn`. If we are in the second column it also has to update page marks, otherwise it only updates column marks. It too is called as part of `\@opcol`.

[6] Probably not before 2025, though.

# Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.


**A**

\AddToHook . . . . . 3

**B**

\botmark . . . . . 2

**C**

\chaptermark . . . . . 7

**D**

\DebugMarksOff . . . . . 6
\DebugMarksOn . . . . . 6

**E**

\edef . . . . . 8
exp commands:
\exp_not:n . . . . . 3

**F**

\FirstMark . . . . . 5
\firstmark . . . . . 2

**G**

\glossary . . . . . 3

**H**

Hooks:
insertmark . . . . . 3

**I**

\IfMarksEqualF . . . . . 4
\IfMarksEqualT . . . . . 4
\IfMarksEqualTF . . . . . 4
\index . . . . . 3
\InsertMark . . . . . 4
insertmark (hook) . . . . . 3
insertmark . . . . . 3

**L**

\label . . . . . 3
\LastMark . . . . . 3
\leftmark . . . . . 8

**M**

\mark . . . . . 2
mark commands:
\mark_clear_structure:n . . . . . 10
\mark_copy_structure:nn . . . . . 10
\mark_debug_off: . . . . . 6
\mark_debug_on: . . . . . 6
\mark_get_marks_for_reinsertion:nNN . . . . . 11
\mark_if_eq:nnnnnnTF . . . . . 4
\mark_if_eq:nnnnTF . . . . . 4
\mark_insert:nn . . . . . 11
\mark_new_class:n . . . . . 3
\mark_set_structure_to_err:n . . . 10
\mark_update_structure_from_material:nn . . . . . 10
\mark_use_first:nn . . . . . 3
\mark_use_last:nn . . . . . 3
\mark_use_top:nn . . . . . 3
mark internal commands:
__mark__update_dblcol_structures: . . . . . 10
__mark_update_dblcol_structures: . . . . . 11
__mark_update_singlecol_structures: . . . . . 10
\markboth . . . . . 7
\markright . . . . . 7

**N**

\NewMarkClass . . . . . 3
\newmarks . . . . . 9

**R**

\rightmark . . . . . 8

**S**

\sectionmark . . . . . 7
\subsectionmark . . . . . 7

**T**

TeX and LaTeX 2ε commands:
\@opcol . . . . . 11
\@outputbox . . . . . 11
\botmark . . . . . 9
\if@firstcolumn . . . . . 11
\protected@edef . . . . . 8
\topmark . . . . . 9
\topmark(s) . . . . . 9
\unexpanded . . . . . 3
\TopMark . . . . . 9
\topmark . . . . . 2

**U**

\unvcopy . . . . . 10