

# The monoref package\*

Srikanth Mohankumar  
srikanthmohankumar@gmail.com

July 7, 2026

## Abstract

`monoref` resolves forward cross-references, page references, the total page count and a multi-level table of contents in a *single* Lua $\text{\LaTeX}$  run. No second pass, no `latexmk`, and no `.aux` round-trip are required. It holds every finished page in memory and back-patches the still unknown values at `\end{document}`. When `hyperref` is loaded, the references and contents entries become clickable.

## 1 Introduction

Normally  $\text{\LaTeX}$  needs at least two runs to settle cross-references: the first writes label values, section numbers and page numbers to the `.aux` file and the second reads them back. This package removes that round-trip for a useful subset of the problem by never shipping a page out immediately. Every page is *held* in memory (using the `shipout/before` hook and `\DiscardShipoutBox`); at the end of the document all values are known, so they can be patched into the held pages before the pages are finally shipped.

Two ideas make it work.

**Reserved slots.** A value that is not yet known (a forward `\ref`, any `\pageref`, `\lastpage`) is typeset as a reserved box. At flush the box collapses to the value's natural width and the enclosing line is re-justified to its original width, so the interword glue absorbs any freed space: short values (a single digit) leave no gap. The reserved width therefore only guides the original line break, not the final spacing.

**Shipout-accurate pages.** `\label` drops an invisible `\special` marker; when a page is held it is scanned for markers and each label is stamped with the page it really landed on. This avoids the off-by-one you get from reading `\value{page}` at `\label` time, because  $\text{\TeX}$ 's page builder is asynchronous.

## 2 Requirements

`monoref` requires Lua $\text{\LaTeX}$ . It stops with an error on any other engine.

---

\*This document corresponds to `monoref` v1.3, dated July 7, 2026.

### 3 Usage

```
\documentclass{article}
\usepackage{hyperref} % optional; enables clickable links
\usepackage{monoref} % load AFTER hyperref
\begin{document}
\section{Introduction}\label{sec:intro}
See Section~\ref{sec:results} on page~\pageref{sec:results}.
This document has \lastpage\ body pages.
...
\section{Results}\label{sec:results}
...
\end{document}
```

- `\ref` Prints the number of the referenced `\label`. A backward reference is typeset immediately; a forward reference is emitted as a slot and filled at the end.
- `\pageref` Prints the page of the referenced `\label`. It is always a slot, because a label's true page is known only once its page is assembled.
- `\lastpage` Prints the total number of body pages (front matter excluded).

### 4 Slot width

Forward values are reserved with a font-relative width taken from a template string. This width only guides the original line break; at flush the slot collapses to the value's natural width and the line is re-justified, so the reserved width never shows up as a gap. `\monorefreftemplate` (default 0.0.0) sizes `\ref` slots; it must be at least as wide as the widest forward (sub)section number. `\monorefpagetemplate` (default 000) sizes `\pageref` and `\lastpage` slots. Redefine either with `\renewcommand`, e.g. `\renewcommand\monorefreftemplate{0.0}` for a document with no forward subsection references. Setting the length `\monorefslotwidth` to a positive value overrides both templates with a fixed width.

### 5 Hyperlinks

If `hyperref` is loaded, `\ref`, `\pageref` and the contents entries become clickable. The links use the package's own named destinations (derived directly from the label names), *not* `hyperref`'s `.aux`-based anchors, so linking stays single-pass and works for forward references too. Load `monoref` *after* `hyperref`.

### 6 The table of contents

Without the `notoc` option a contents list covering `\section`, `\subsection` and `\subsubsection` is collected during the run and prepended at the end with roman folios, while the body keeps arabic numbering; prepending the contents therefore does not renumber the body and there is no circular dependency. Use `notoc` to switch it off.

## 7 Limitations

- It holds the whole document in memory until `\end{document}`.
- It redefines `\label`, `\ref`, `\pageref`, `\section`, `\subsection` and `\subsubsection`; because it replaces `hyperref`'s versions of these, `hyperref` features that depend on them (`\autoref`, `\nameref`, `cleveref`) are not available. It also conflicts with `titlesec` and `varioref`.
- A forward value wider than its slot template overflows.
- Numbered-equation `\ref` works, including with `amsmath`; but `\eqref` (the parenthesised form) and lists of figures/tables are not provided.
- Because it intercepts every `\shipout`, it is incompatible with other packages that manipulate `shipout`.

## 8 Examples

The distribution ships a set of runnable examples in the `examples/` directory; compile each once with `LuaLaTeX`:

**01-minimal** the smallest possible use.

**02-article** three heading levels, a numbered equation, forward and backward `\ref/\pageref`, a page total, and a front TOC.

**03-hyperref-links** clickable links via `hyperref` (loaded first, with `hidelinks`).

**04-notoc** the `notoc` option.

**05-page-of-total** a “Page X of Y” footer with `fancyhdr` and `\lastpage`.

**06-custom-slot-width** widening the slot templates.

## 9 Implementation: the style file

```
1 (*package)
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{monoref}%
4   [2025/07/07 v1.3 Single-pass cross-references, page totals and TOC (LuaLaTeX)]
5
6 \RequirePackage{iftex}
7 \RequireLuaTeX
8 \RequirePackage{xparse}
9
10 \newif\ifmonoref@dotoc \monoref@dotoctrue
11 \DeclareOption{notoc}{\monoref@dotocfalse}
12 \ProcessOptions\relax
13
14 \directlua{if not pcall(require, "monoref") then
15   dofile("monoref.lua") end}
16
17 \newbox\monoref@shipbox
```

```

18 \directlua{monoref.shipreg = \number\monoref@shipbox}
19 \attributedef\monoref@slotattr=\directlua{tex.print(monoref.slotattr)}\relax
20 \attributedef\monoref@markattr=\directlua{tex.print(monoref.markattr)}\relax
21
22 \newcommand\monorefreftemplate{0.0.0}
23 \newcommand\monorefpagetemplate{000}
24 \newlength\monorefslotwidth \setlength\monorefslotwidth{0pt}
25 \newlength\monoref@tmpwd
26
27 \newcommand\monoref@slot[2]{% #1 = slot id #2 = width template
28 \begingroup
29 \ifdim\monorefslotwidth>\z@ \monoref@tmpwd=\monorefslotwidth
30 \else \settowidth\monoref@tmpwd{#2}\fi
31 \monoref@slotattr=#1\relax \hbox to \monoref@tmpwd{\hfil}%
32 \endgroup}
33
34 \newcommand\monoref@mark[1]{\monoref@markattr=#1\relax \special{}}
35
36 \newif\ifmonoref@hyper \monoref@hyperfalse
37 \newcommand\monoref@target[1]{\ifmonoref@hyper\hypertarget{monoref:#1}\fi}
38 \newcommand\monoref@link[2]{\ifmonoref@hyper\hyperlink{monoref:#1}{#2}\else#2\fi}
39
40 \newcommand\lastpage{\monoref@slot{\directlua{monoref.newslot("", "total")}}{\monorefpagetempl
41
42 \newcommand\monoref@refinner[1]{%
43 \ifnum\directlua{monoref.known("#1")}>\z@
44 \directlua{monoref.ref_value("#1")}%
45 \else
46 \monoref@slot{\directlua{monoref.newslot("#1", "ref")}}{\monorefreftemplate}%
47 \fi}
48
49 \newcommand\monoref@record[3]{% #1 level #2 number #3 title
50 \protected@edef\monoref@nm{#2}%
51 \protected@edef\monoref@tl{#3}%
52 \monoref@mark{\directlua{monoref.toc_begin(#1,
53 "\luaescapestring{\monoref@nm}", "\luaescapestring{\monoref@tl}")}}%
54 \ifmonoref@hyper
55 \edef\monoref@ti{\directlua{monoref.toc_count()}}%
56 \hypertarget{monoref:toc:\monoref@ti}{}%
57 \fi}
58
59 \def\monoref@activate{%
60 \@ifpackageloaded{hyperref}{\monoref@hypertrue}{\monoref@hyperfalse}%
61 \let\monoref@origlabel\label
62 \let\monoref@origsection\section
63 \let\monoref@origsubsection\subsection
64 \let\monoref@origsubsubsection\subsubsection
65 \RenewDocumentCommand\label{m}{%
66 \protected@edef\monoref@c1{\@currentlabel}%
67 \monoref@mark{\directlua{monoref.label_begin("##1", "\luaescapestring{\monoref@c1}")}}%
68 \monoref@target{lbl:##1}%
69 \monoref@origlabel{##1}}%
70 \RenewDocumentCommand\ref{m}{\monoref@link{lbl:##1}{\monoref@refinner{##1}}}%
71 \RenewDocumentCommand\pageref{m}{%

```

```

72   \monoref@link{lbl:##1}{\monoref@slot{\directlua{monoref.newsloc("##1","page")}}{\monoref@
73 \RenewDocumentCommand\section{s o m}{%
74   \IfBooleanTF{##1}{\monoref@origsection*{##3}}%
75   {\IfValueTF{##2}{\monoref@origsection[##2]{##3}}{\monoref@origsection{##3}}%
76   \monoref@record{1}{\thesection}{##3}}%
77 \RenewDocumentCommand\subsection{s o m}{%
78   \IfBooleanTF{##1}{\monoref@origsubsection*{##3}}%
79   {\IfValueTF{##2}{\monoref@origsubsection[##2]{##3}}{\monoref@origsubsection{##3}}%
80   \monoref@record{2}{\thesubsection}{##3}}%
81 \RenewDocumentCommand\subsubsection{s o m}{%
82   \IfBooleanTF{##1}{\monoref@origsubsubsection*{##3}}%
83   {\IfValueTF{##2}{\monoref@origsubsubsection[##2]{##3}}{\monoref@origsubsubsection{##3}}%
84   \monoref@record{3}{\thesubsubsection}{##3}}%
85 \@ifpackageloaded{amsmath}{\let\ltx@label\label}{}}%
86 }
87 \AtBeginDocument{\monoref@activate}
88
89 \newcommand\monoref@indent[1]{\ifcase#1\or\or\hspace{1.5em}\or\hspace{3em}\fi}
90 \newcommand\monoref@tocline[5]{% #1 idx #2 level #3 number #4 title #5 page
91 \par\noindent\monoref@indent{#2}%
92 \monoref@link{toc:#1}{\makebox[2.6em][l]{#3#4}\dotfill\ #5}
93
94 \newcount\monoref@nt \newcount\monoref@k
95 \newcommand\monoref@buildtoc{%
96 \directlua{monoref.target = "toc"}%
97 \pagenumbering{roman}%
98 \monoref@nt=\directlua{monoref.toc_count()}\relax
99 \ifnum\monoref@nt>\z@
100 \section*{\contentsname}\par\medskip
101 \monoref@k=\z@
102 \loop\ifnum\monoref@k<\monoref@nt
103 \advance\monoref@k\@ne
104 \monoref@tocline
105 {\the\monoref@k}%
106 {\directlua{monoref.toc_level(\the\monoref@k)}}%
107 {\directlua{monoref.toc_num(\the\monoref@k)}}%
108 {\directlua{monoref.toc_title(\the\monoref@k)}}%
109 {\directlua{monoref.toc_page(\the\monoref@k)}}%
110 \repeat
111 \clearpage
112 \fi}
113
114 \newif\ifmonoref@flushing
115 \AddToHook{shipout/before}{%
116 \ifmonoref@flushing\else
117 \directlua{monoref.hold(\number\ShipoutBox, \the\value{page})}%
118 \DiscardShipoutBox
119 \fi}
120
121 \newcount\monoref@i
122 \AtEndDocument{%
123 \clearpage
124 \ifmonoref@dotoc \monoref@buildtoc \fi
125 \directlua{monoref.finalize()}%

```

```

126 \monoref@i=\directlua{monoref.qcount()}\relax
127 \monoref@flushingtrue
128 \loop\ifnum\monoref@i>\z@
129   \directlua{monoref.qpop()}%
130   \shipout\box\monoref@shipbox
131   \advance\monoref@i\m@ne
132 \repeat
133 \monoref@flushingfalse}
134
135 \end{package}

```

## 10 Implementation: the Lua backend

The backend is a plain Lua module, generated verbatim into `monoref.lua`. Its own comment header carries the licence (a `docstrip %%` header would be invalid Lua), so it is shipped standalone rather than generated by the installer.

```

136 (*lua)
137 -- monoref.lua
138 -- Lua backend for the monoref package.
139 --
140 -- Copyright (C) 2025 Srikanth Mohankumar <srikanthmohankumar@gmail.com>
141 -- This work may be distributed and/or modified under the conditions of
142 -- the LaTeX Project Public License, version 1.3c or later.
143 -- http://www.latex-project.org/lppl.txt
144
145 monoref = monoref or {}
146
147 monoref.labels = {} -- name -> {ref=, page=}
148 monoref.slots = {} -- id -> {name=, font=, kind=} kind: ref | page | total
149 monoref.marks = {} -- id -> {kind=, key=} kind: label | toc
150 monoref.toc = {} -- ordered list of {level=, num=, title=, page=}
151 monoref.held = {} -- body pages (arabic)
152 monoref.toc_pages = {} -- front-matter pages (roman)
153 monoref.queue = {} -- final ship order
154 monoref.target = "body"
155 monoref.sid, monoref.mid, monoref.total = 0, 0, 0
156
157 monoref.slotattr = luatexbase.new_attribute("monorefslot")
158 monoref.markattr = luatexbase.new_attribute("monorefmark")
159
160 local HLIST = node.id("hlist")
161 local VLIST = node.id("vlist")
162 local GLYPH = node.id("glyph")
163 local GLUE = node.id("glue")
164
165 -----
166 -- Registration (all print/sprint straight back into the TeX stream)
167 -----
168 function monoref.known(name)
169   tex.print(monoref.labels[name] and 1 or 0)
170 end
171
172 function monoref.newslot(name, kind)
173   monoref.sid = monoref.sid + 1

```

```

174 monoref.slots[monoref.sid] = { name = name, font = font.current(), kind = kind }
175 tex.print(monoref.sid)
176 end
177
178 local function newmark(kind, key)
179   monoref.mid = monoref.mid + 1
180   monoref.marks[monoref.mid] = { kind = kind, key = key }
181   return monoref.mid
182 end
183
184 function monoref.label_begin(name, ref)
185   monoref.labels[name] = { ref = ref, page = 0 }
186   tex.print(newmark("label", name))
187 end
188
189 function monoref.ref_value(name)
190   local L = monoref.labels[name]
191   tex.sprint(L and L.ref or "?")
192 end
193
194 function monoref.toc_begin(level, num, title)
195   monoref.toc[#monoref.toc + 1] = { level = level, num = num, title = title, page = 0 }
196   tex.print(newmark("toc", #monoref.toc))
197 end
198
199 function monoref.toc_count() tex.print(#monoref.toc)          end
200 function monoref.toc_level(i) tex.print(monoref.toc[i].level) end
201 function monoref.toc_num(i)   tex.sprint(monoref.toc[i].num)   end
202 function monoref.toc_title(i) tex.sprint(monoref.toc[i].title) end
203 function monoref.toc_page(i)  tex.print(monoref.toc[i].page)   end
204
205 -----
206 -- Marker scan: stamp every label / TOC entry with the TRUE page it is on
207 -----
208 local function scan(head, pg)
209   local n = head
210   while n do
211     local m = node.get_attribute(n, monoref.markattr)
212     if m and monoref.marks[m] then
213       local r = monoref.marks[m]
214       if r.kind == "label" and monoref.labels[r.key] then
215         monoref.labels[r.key].page = pg
216       elseif r.kind == "toc" and monoref.toc[r.key] then
217         monoref.toc[r.key].page = pg
218       end
219     end
220     local id = n.id
221     if id == HLIST or id == VLIST then scan(n.head, pg) end
222     n = n.next
223   end
224 end
225
226 function monoref.hold(reg, pg)
227   local b = tex.getbox(reg)

```

```

228 if not b then return end
229 local c = node.copy(b)
230 scan(c.head, pg)
231 local dest = (monoref.target == "toc") and monoref.toc_pages or monoref.held
232 dest[#dest + 1] = c
233 end
234
235 -----
236 -- Slot filling at flush (fixed width kept -> never reflows)
237 -----
238 local function glyphs_of(s, fid)
239 local fnt = font.getfont(fid)
240 local first, last
241 for i = 1, #s do
242 local code = s:byte(i)
243 local g = node.new(GLYPH)
244 g.font, g.char, g.subtype = fid, code, 0
245 if fnt and fnt.characters and fnt.characters[code] then
246 local ci = fnt.characters[code]
247 g.width, g.height, g.depth = ci.width or 0, ci.height or 0, ci.depth or 0
248 end
249 if first then last.next = g; g.prev = last; last = g
250 else first, last = g, g end
251 end
252 return first, last
253 end
254
255 local function value_of(rec)
256 if rec.kind == "total" then return tostring(monoref.total) end
257 local L = monoref.labels[rec.name]
258 if not L then return "??" end
259 if rec.kind == "page" then
260 return (L.page and L.page > 0) and tostring(L.page) or "??"
261 end
262 return L.ref
263 end
264
265 -- Fill one placeholder hbox to the value's NATURAL width (the enclosing
266 -- line is re-justified afterwards so the reserved slack disappears).
267 local function fill_slot(hbox, id)
268 local val = value_of(monoref.slots[id])
269 if hbox.head then node.flush_list(hbox.head); hbox.head = nil end
270 local gf = glyphs_of(val, monoref.slots[id].font)
271 local packed = node.hpack(gf or node.new(GLUE))
272 hbox.head = packed.head
273 hbox.width = packed.width
274 hbox.height = packed.height
275 hbox.depth = packed.depth
276 hbox.glue_set = 0
277 hbox.glue_sign = 0
278 hbox.glue_order = 0
279 packed.head = nil
280 node.free(packed)
281 end

```

```

282
283 -- Re-pack a line to its original width so its interword glue absorbs the
284 -- space freed by shrinking a slot (keeps the line justified, no gap).
285 local function rejustify(hbox)
286   local packed = node.hpack(hbox.head, hbox.width, "exactly")
287   hbox.head      = packed.head
288   hbox.glue_set  = packed.glue_set
289   hbox.glue_sign = packed.glue_sign
290   hbox.glue_order = packed.glue_order
291   hbox.height    = packed.height
292   hbox.depth     = packed.depth
293   packed.head = nil
294   node.free(packed)
295 end
296
297 -- Patch every slot in a box; re-justify each line that contained one.
298 local function patch_box(box)
299   local direct = false
300   local n = box.head
301   while n do
302     local id = n.id
303     if id == HLIST or id == VLIST then
304       local a = node.get_attribute(n, monoref.slotattr)
305       if a and monoref.slots[a] then
306         fill_slot(n, a)
307         direct = true
308       else
309         patch_box(n)
310       end
311     end
312     n = n.next
313   end
314   if direct and box.id == HLIST then rejustify(box) end
315 end
316
317 -----
318 -- Finalisation and shipping
319 -----
320 function monoref.finalize()
321   monoref.total = #monoref.held
322   for i = 1, #monoref.toc_pages do patch_box(monoref.toc_pages[i]) end
323   for i = 1, #monoref.held do patch_box(monoref.held[i]) end
324   monoref.queue = {}
325   for i = 1, #monoref.toc_pages do monoref.queue[#monoref.queue + 1] = monoref.toc_pages[i] e
326   for i = 1, #monoref.held do monoref.queue[#monoref.queue + 1] = monoref.held[i] e
327 end
328
329 function monoref.qcount() tex.print(#monoref.queue) end
330 function monoref.qpop() tex.setbox(monoref.shipreg, table.remove(monoref.queue, 1)) end
331
332 return monoref
333 </lua>

```