

DIE OSGDOC TOOLS

v0.5.0 2026-09-08

Unterstützung für L^AT_EX-Paket-Dokumentationen

Matthias WERNER¹

<https://github.com/werner-matthias/osglecture>

Die OSGDOC-Tools sind Teil des [osglecture](#) Bundles, jedoch auch unabhängig von den anderen Paketen des Bundles einsetzbar. Sie beinhalten einerseits einen Patch-Wrapper für Clemens NIEDERBERGERS großartige CNLTX-DOC-Klasse, die aber leider nicht mehr mit den aktuellen L^AT_EX kompatibel ist. Zum anderen werden ein paar Makros zur Unterstützung von Dokumentationen definiert.

Inhaltsverzeichnis

1 Nutzung	1 3 Implementation	4
1.1 osgdoc.cls	1 3.1 Class OSGDOC.cls	4
1.2 osgdoc.sty	2 3.2 Package OSGDOC.sty	7
2 Bugs und Einschränkungen	3 Index	34

1 Nutzung

Das OSGDOC-Paket besteht aus der Klassendatei `osgdoc.cls` und der Paketdatei `osgdoc.sty`. Letztere wird durch die Klasse geladen, man kann sie aber auch unabhängig verwenden, wenn nur die dort definierten Befehle gebraucht werden.

1.1 osgdoc.cls

Die Klasse wird auf dem üblichen Weg geladen: `\documentclass[⟨options⟩]{osgdoc}`

Die Optionen und zur Verfügung gestellten Befehle der Basisklasse CNLTX-DOC entnehmen Sie bitte der Dokumentation des [cnltx](#)¹-Pakets. Die eigentliche Klasse OSGDOC macht nichts, außer einige Befehle zu patchen und einige Standardeinstellungen wie das Farbschema zu ändern. Zusätzlich stehen noch die im Abschnitt 1.2 beschriebenen Befehle zur Verfügung.

Die folgende Option wird sowohl von der Klasse als auch vom Paket verstanden.

1. matthias.werner@informatik.tu-chemnitz.de

1. auf CTAN als `cnltx`: <https://ctan.org/tex-archive/macros/latex/contrib/cnltx>

`readprov = true|false`

Voreinstellung: false

Verwendet `readprov` als Backend für `\UseVersionOf` und `\UseDateOf`. Die gewünschten Dateien müssen zuvor mit einem der `readprov`-Befehle wie `\ReadFileInfos` eingelesen werden.

1.2 osgdoc.sty

Das Paket ergänzt folgende Befehle und Umgebungen:

`\AutoDocInput[⟨mode⟩]{⟨file⟩}{⟨guard⟩}`

Leichtgewichtige Möglichkeit für die Aufnahme dokumentierten Quellcodes, analog dem `doc`²/`docstrip`³ Mechanismus. Im Unterschied zu diesem sind jedoch (wie in `gmdoc`) keine `\begin{macrocode}... \end{macrocode}` notwendig. Außerdem kann Lua-Code dokumentiert und zur Textauszeichnung statt L^AT_EX eine Art Mini-Markdown verwendet werden.

`mode`

Voreinstellung: tex

Der Modus hat die Form `⟨Guard-Syntax⟩/⟨Kommentar-Syntax⟩/⟨Markup-Format⟩`. Eine fehlende oder leere Kommentar-Syntax übernimmt die Guard-Syntax; ein fehlendes oder leeres Markup-Format ist tex. Damit bestimmt beispielsweise `lua//ascii` Lua für Guards und Kommentare sowie ASCII-Auszeichnung für den Dokumentations-text.

Feld	Bedeutung	Werte	Vorgabe
1	Guard-Syntax	tex, lua	tex
2	Kommentar-Syntax	tex, lua	Feld 1
3	Markup-Format	tex, ascii	tex

Eingabe	Vollständiger Modus
tex	tex/tex/tex
lua	lua/lua/tex
tex/lua	tex/lua/tex
tex//ascii	tex/tex/ascii
lua//ascii	lua/lua/ascii
lua/tex/ascii	lua/tex/ascii

Da das zweite Feld die Kommentar-Syntax bezeichnet, ist beispielsweise `tex/ascii` kein Kürzel für ASCII-Text; hierfür ist `tex//ascii` zu verwenden. Für die einzelnen Elemente gilt folgende Syntax:

Guards

`tex`: `%<⟨Guard⟩> / %</⟨Guard⟩>`

`lua`: `--<⟨Guard⟩> / --</⟨Guard⟩>`

Kommentare

`tex`: genau eine %-Marke, optional gefolgt von einem Leerzeichen. Zeilen mit %% oder mehr Prozentzeichen werden als Quelltext dargestellt

2. auf CTAN als doc: <https://ctan.org/pkg/doc>

3. auf CTAN als docstrip: <https://ctan.org/pkg/docstrip>

lua: mindestens zwei --Marken, optional gefolgt von einem Leerzeichen, oder ein Lua-Langkommentar

Auszeichnung

tex: L^AT_EX, wobei für eine bessere Kompatibilität mit L^AT_EX₃ nicht maskierte Unterstriche innerhalb der Argumente von `\cs` und `\cs*` maskiert werden. Dadurch können beispielsweise `\keys_set:nn` und `\__module_function:N` direkt geschrieben werden. Der Makroname muss vollständig in einer Zeile stehen.

ascii: Formatierung in einer kleinen, zeilenorientierten Auszeichnungssprache.

`*text*`: **text**

`_text_`: *text*

``code``: code

`\macro` oder ``\macro``: `\macro`

`===...:` _____ (Trennlinie)

`###:` (am Zeilenanfang mit folgendem Leerzeichen): **Überschriften**

`-/*` (am Zeilenanfang mit folgendem Leerzeichen): • Listenpunkte

file

Name der Datei, die eingebunden werden soll.

guard

Name des Guards. Es sind nur einfache Namen zulässig, keine komplexen Guardausdrücke.

`\OnlyDescription`

`\MaybeStop{<text>}`

Die beiden Makros erfüllen die gleiche Funktion wie die gleichnamigen Makros aus dem `doc`-Paket.

`\UseVersionOf{<file>}`

`\UseDateOf{<file>}`

Liest die durch `<file>` angegebene Datei und sucht nach `\ProvidesExplPackage`, `\ProvidesPackage`, `\ProvidesExplClass`, `\ProvidesClass`, `\ProvidesExplFile` oder `\ProvidesFile` und gibt die gesuchte Information zurück (Version oder Datum).

`\begin{infobox}{<title>}`

`\begin{warnbox}{<title>}`

Kästchenumgebungen für Information und Warnungen.

2 Bugs und Einschränkungen

- `OSGDOC` ist zum gegenwärtigen Zeitpunkt als experimentell zu betrachten, die Versionsnummer weist darauf hin.

- Die `OSGDOC`-Klasse ist nur ein dünner Wrapper für die `CNLTX-DOC`-Klasse aus dem `cnltx`-Paket. Sie sorgt nur für syntaktische Korrektheit im Standardfall. Es ist durchaus möglich, dass bei der Vielzahl von Optionen und Möglichkeiten, die `CNLTX-DOC` bietet, weitere Probleme mit dem heutigen L^AT_EX existieren, die `OSGDOC` nicht abfängt.
- Die Dateinformation bei den klassischen (nicht L^AT_EX₃) `\Provide...`-Befehlen beruhen auf einem einfachen Lua-Musterabgleich über dem optionalen Argument. Wenn sicher ist, dass Dateien *nicht* im L^AT_EX₃-Syntax geschrieben sind, kann mit der Option `readprov` stattdessen `readprov`⁴ benutzt werden.
- Obwohl das `osglecture`-Bundle bemüht ist, die Barrierefreiheit mittels Tagging zu unterstützen, erzeugt `OSGDOC` *keine* korrekt getaggten PDFs.

3 Implementation

3.1 Class `osgdoc.cls`

```

1 \NeedsTeXFormat{LaTeX2e}[2022/06/01]
2 \ProvidesExplClass
3 {osgdoc}
4 {2026-09-08}
5 {v0.5.0}
6 {A thin wrapper around cnltx-doc}
7 \ExplSyntaxOff
8 \ExplSyntaxOn

```

Zunächst müssen wir `cnltx-doc` patchen. Dazu muss der nicht mehr gültige `\AfterPackage!`-Ruf abgefangen werden.

```

9 \AddToHook{class/scrartcl/after}{
10   \NewCommandCopy{\originalAfterPackage}{\AfterPackage}
11   \RenewDocumentCommand{\AfterPackage}{t! m m}{
12     \IfBooleanTF{#1}{#3}{%
13       \originalAfterPackage{#2}{#3}
14     }
15   }
16 }
17 \AddToHook{class/cnltx-doc/before}{
18   \ClassInfo{osgdoc}{Patches~applied~for~cnltx-doc.cls}
19 }

```

Die Pakete `multicol` und `ragged2e` müssen explizit geladen werden, um nicht den Fehler „*Loading in a group*“ zu triggern.

4. auf CTAN als `readprov`: <https://ctan.org/tex-archive/macros/latex/contrib/fileinfo>

```

20 \RequirePackage{multicol,ragged2e}
21 \sys_if_engine luatex:T{
22   \RequirePackage{luatex85}
23 }

```

Die example-Umgebung braucht `shellesc`.

```

24 \RequirePackage{shellesc}

```

`cnltx-doc` versucht für die Logo-Kerning-Anpassung an `libertine` das (nichtveröffentlichte) Paket `libertinehologopatch` zu laden. Dies wird per Dateisubstitution und unterbunden und stattdessen `metalogox` genutzt, das dieselbe Aufgabe für viele Schriften übernimmt.

```

25 \declare@file@substitution{libertinehologopatch.sty}{}

```

Die Option `load-preamble` von `cnltx-doc` (oder eine ihrer Varianten) lädt `babel` und mit fest einprogrammierten Optionen. Das kollidiert, sobald das Dokument selbst (etwa über `langselect`) `babel` mit anderen Optionen laden will. Wir unterbinden daher den Ladeversuch temporär per Dateisubstitution und holen ihn mit denselben von `cnltx-doc` konfigurierten Optionen erst am Ende der Präambel nach, und zwar nur dann, wenn bis dahin noch niemand `babel` (oder `polyglossia`) geladen hat.

```

26 \declare@file@substitution{babel.sty}{}

```

```

27

```

Damit `cnltx-docs` eigene Font-/Microtype-/Babel-Logik (Option `load-preamble+`) unverändert greift, wird sie als Vorgabe an `cnltx-doc` durchgereicht, ebenso alle weiteren vom Anwender explizit übergebenen Optionen (etwa `load-fonts=false`, `add-index`, `babel-options`).

```

28 \PassOptionsToClass{load-preamble+}{cnltx-doc}
29 \DeclareOption{readprov}{\PassOptionsToPackage{readprov}{osgdoc}}
30 \DeclareOption*{\PassOptionsToClass{\CurrentOption}{cnltx-doc}}
31 \ProcessOptions\relax
32 \LoadClass{cnltx-doc}
33 \undecclare@file@substitution{babel.sty}

```

`cnltx-docs` nutzt `\textsu` für die Redefinition von `\@makefnmark` ohne Bereitstellung von `superiors`. Wir stellen einen minimalen Ersatz bereit.

```

34 \providecommand\textsu[1]{\textsuperscript{#1}}

```

Der Ersatz für `libertinehologopatch`: `metalogox` erkennt die aktive Schrift (hier `libertine`) automatisch und wählt ein passendes Kerning für die T_EX-Familie-Logos.

```

35 \RequirePackage{metalogox}

```

Das eigentliche Nachladen von `babel`, siehe oben.

3 Implementation

```
36 \AtEndPreamble{
37   \@ifpackageloaded{babel}{}{
38     \@ifpackageloaded{polyglossia}{}{
39       \ifbool{cnltx@load@preamble}{
40         \RequirePackage[\cnltx@babel@options]{babel}
41       }{}
42     }
43   }
44 }
45 \RequirePackage[hidelinks]{hyperref}
46 \RequirePackage{osgdoc}
47 \RequirePackage{imakeidx}
48
```

Manchmal ist ein Paket Teil eines Bundles, so dass das Verzeichnis einen anderen Namen hat. Für diese Fälle wird für die `\needXXX`-Macros zusätzlich eine Sternvariante zur Verfügung gestellt, die bei der URL nur das optionale Argument (Pfad) ausgibt. Außerdem sollte nur noch `https` genutzt werden.

```
49 \DeclareTranslationFallback{on-CTAN}{on~\ctan{}}~as}
50 \DeclareTranslation{English}{on-CTAN}{on~\ctan{}}~as}
51 \DeclareTranslation{German}{on-CTAN}{auf~\ctan{}}~als}
52 \RenewDocumentCommand{\CTANurl}{s 0{tex-archive/macros/latex/contrib} m}{
53   \GetTranslation{on-CTAN}~\cnltx@weblink@font{#3}:~
54   \IfBooleanTF{#1}{
55     \url{https://ctan.org/#2}
56   }{
57     \url{https://ctan.org/#2/#3}
58   }
59 }
60
61 \RenewDocumentCommand{\needpackage}{s 0{tex-archive/macros/latex/contrib} m}{
62   \pkg{#3}\footnote{
63     \IfBooleanTF{#1}{
64       \CTANurl*{#2}{#3}
65     }{
66       \CTANurl{#2}{#3}
67     }
68   }
69 }
70
71 \RenewDocumentCommand{\needclass}{s 0{tex-archive/macros/latex/contrib} m}{
72   \cls{#3}\footnote{
73     \IfBooleanTF{#1}{
74       \CTANurl*{#2}{#3}

```

```

75     }{
76     \CTANurl[#2]{#3}
77     }
78   }
79 }
80

```

Da die Titellei symbolische Fußnotenzeichen nutzt, wird die Nummer zurückgesetzt.

```

81 \AddToHook{cmd/cnltx@title@information/after}{
82   \setcounter{footnote}{0}
83 }
84 \def\gobble{\@gobble}

```

3.2 Package **OSGDOC.sty**

```

1 \NeedsTeXFormat{LaTeX2e}[2022/06/01]
2 \ProvidesExplPackage
3   {osgdoc}
4   {2026-09-08}
5   {v0.5.0}
6   {small tools for documentation}
7 \newif\ifosgdoc@readprov
8 \DeclareOption{readprov}{\osgdoc@readprovtrue}
9 \ProcessOptions\relax
10 \ifosgdoc@readprov
11   \RequirePackage{readprov}
12 \fi

```

Wir beginnen mit den Funktionalitäten, die nicht Lua braucht.

Farbboxen

```

13 \RequirePackage{tcolorbox}
14 \newtcolorbox{warnbox}[1]{
15   colback=red!5!white,
16   colframe=red!75!black,fonttitle=\bfseries,
17   title={#1}
18 }
19 \newtcolorbox{infobox}[1]{
20   colback=cnltxformalblue!5!white,
21   colframe=cnltxformalblue,fonttitle=\bfseries,
22   title={#1}
23 }

```

Farbboxen

```

24 \newbool{osgdocimpl}
25 \booltrue{osgdocimpl}
26 \NewDocumentCommand{\OnlyDescription}{}{
27   \boolfalse{osgdocimpl}
28 }
29 \NewDocumentCommand{\MaybeStop}{+m}{
30   \ifbool{osgdocimpl}{
31     \def\osgterminate{\relax}
32   }{
33     #1
34     \def\osgterminate{\end{document}\endinput}
35   }
36 \osgterminate
37 }

```

Wenn die `osgdoc`-Klasse (und damit auch die `cnltx-doc`-Klasse) nicht geladen ist, sollte das `cnltx`-Paket geladen werden.

```

38 \@ifclassloaded{osgdoc}{}{
39   \RequirePackage{example,names}{cnltx}
40 }

```

Noch ein paar Einstellungen, die ich bevorzuge.

```

41 \mdfdefinestyle{cnltxmdframed}{
42   backgroundcolor = cnltxbg ,
43   linecolor = cnltx ,
44   roundcorner = 5pt
45 }
46 \newcommand\osgdoc@example@pre{%
47   \begin{mdframed}[
48     backgroundcolor=white,
49     hidealllines=true,
50     innerleftmargin=0pt,
51     innerrightmargin=0pt,
52     innertopmargin=0pt,
53     innerbottommargin=0pt
54   ]
55 }
56 \newcommand\osgdoc@example@after{%
57   \end{mdframed}
58 }
59 \AddToHook{env/example/begin}{
60   \setcnltx{

```

3 Implementation

```

61     pre-output    = \osgdoc@example@pre,
62     after-output  = \osgdoc@example@after,
63   }%
64 }
65 \setcni{
66   color-scheme=blue
67 }
68 \renewcommand{\packageformat}{\color{cni}formalblue}\sffamily}
69 \renewcommand{\classformat}{\color{cni}formalblue}\sffamily}
70 \renewcommand{\bundleformat}{\color{cni}formalblue}\sffamily}

```

Zusätzlich zu den Befehlen wie `\cls` oder `\pkg` wird `\File` definiert. Das Macro wird entgegen der Konvention groß geschrieben, um einen Clash mit dem `dirtree`-Paket zu vermeiden.

```

71
72 \newcommand*{\fileformat}{\color{cni}formalblue}\ttfamily}
73 \newcommand{\File}[1]{\color{cni}formalblue}\ttfamily#1}

```

Die restlichen Funktionen brauchen Lua. Um keine harten Fehler zu generieren, werden Stubs zur Verfügung gestellt.

```

74 \ExplSyntaxOn
75 \let \ifLuaTF=\sys_if_engine luatex:TF
76 \ExplSyntaxOff
77 \def\odnext{\relax}
78 \ifLuaTF{
79   \RequirePackage{luacode}
80 }{
81   \NewDocumentCommand{\AutoDocInput}{0{tex} m m}{
82     \textcolor{red}{For use \string\AutoDocInput, you have to translate this
83       ↪ document with Lua\LaTeX.}
84   }
85   \ifosgdoc@readprov\else
86     \NewDocumentCommand{\UseVersionOf}{m}{
87       \textcolor{red}{For use \string\UseVersionOf, you have to translate this
88         ↪ document with Lua\LaTeX.}
89     }
90     \NewDocumentCommand{\UseDateOf}{m}{
91       \textcolor{red}{For use \string\UseDateOf, you have to translate this
92         ↪ document with Lua\LaTeX.}
93     }
94   \fi
95   \let\odnext\endinput
96 }

```

AutoDocInput

Für `\AutoDocInput` brauchen wir neben `luacode` für das Parsing noch `fancyverb/fvextra` für die Darstellung.

```

96 \RequirePackage{fvextra}
97 \fvset{formatcom=\color{blue},numbers=left, breaklines, breakafter=~\{ \}
98 \begin{luacode*}
99     function texassert(v, msg)
100         if v then
101             return v
102         end
103         tex.error("Error", {msg or "assertion failed"})
104     end
105 \end{luacode*}

    \AutoDocInput[\mode]{\file}{\guard}

106 \NewDocumentCommand{\AutoDocInput}{0{tex} m m}{
107     \directlua{
108         autodoc.autodoc_process(
109             \luastring{#2},
110             \luastring{#3},
111             \luastring{#1}
112         )
113     }
114 }
115
116 \begin{luacode*}
117 autodoc = autodoc or {}
118
119 local verbtempfiles = {}

Syntaxmodi

120 -- TODO: mode auto?
121 local function get_syntax(mode)
122     mode = mode or "tex"
123     local fields = {}
124
125     for field in (mode .. "/"):gmatch("(.)/") do
126         fields[#fields + 1] = field
127     end
128
129     local function invalid(message)
130         tex.error(
131             "AutoDocInput: invalid syntax mode `" .. tostring(mode) .. "'",

```

3 Implementation

```
132     {
133         message,
134         "Expected: guard/comment/format",
135         "Guard and comment syntax: tex or lua",
136         "Text format: tex or ascii",
137     }
138 )
139
140 return {
141     guard_prefix = "%%",
142     comment_kind = "tex",
143     documentation_format = "tex",
144 }
145 end
146
147 if #fields > 3 then
148     return invalid("A syntax mode has at most three fields.")
149 end
150
151 local guard_kind = fields[1]
152 if guard_kind == nil or guard_kind == "" then
153     guard_kind = "tex"
154 end
155
156 local comment_kind = fields[2]
157 if comment_kind == nil or comment_kind == "" then
158     comment_kind = guard_kind
159 end
160
161 local documentation_format = fields[3]
162 if documentation_format == nil or documentation_format == "" then
163     documentation_format = "tex"
164 end
165
166 if guard_kind ~= "tex" and guard_kind ~= "lua" then
167     return invalid("Unknown guard syntax `" .. guard_kind .. "'")
168 end
169
170 if comment_kind ~= "tex" and comment_kind ~= "lua" then
171     return invalid("Unknown comment syntax `" .. comment_kind .. "'")
172 end
173
174 if documentation_format ~= "tex" and documentation_format ~= "ascii" then
175     return invalid(
```

3 Implementation

```
176     "Unknown documentation text format `" .. documentation_format .. "'."
177   )
178   end
179
180   return {
181     guard_prefix = guard_kind == "tex" and "%%" or "%-%-",
182     comment_kind = comment_kind,
183     documentation_format = documentation_format,
184   }
185   end
186
```

Helper für Lua-Patterns

```
187 local function quote_pattern(text)
188   return (text:gsub("(%W)", "%%%1"))
189 end
190
191 local function guard_contains_symbol(expression, symbol)
192   if not expression then
193     return false
194   end
195
196   local atomchars = "%W_%- "
197   local quoted_symbol = quote_pattern(symbol)
198
199   local pattern =
200     "%f[" .. atomchars .. "]"
201     .. quoted_symbol
202     .. "%f[^" .. atomchars .. "]"
203
204   return expression:find(pattern) ~= nil
205 end
206
```

Guarderkennung

```
207 local function guard_expression(line, syntax, guard_kind)
208   local operator
209
210   if guard_kind == "open" then
211     operator = "%*"
212   elseif guard_kind == "close" then
213     operator = "/"
214   else
215     error("Unknown guard kind: " .. tostring(guard_kind))
216   end
217
```

```

216     end
217
218     return line:match(
219         "^%S*"
220         .. syntax.guard_prefix
221         .. "%S*<%S*"
222         .. operator
223         .. "%S*(.-%S*>%S*$"
224     )
225 end
226
227 local function is_opening_guard(line, syntax, symbol)
228     local expression = guard_expression(line, syntax, "open")
229     return guard_contains_symbol(expression, symbol)
230 end
231
232 local function is_closing_guard(line, syntax, symbol)
233     local expression = guard_expression(line, syntax, "close")
234     return guard_contains_symbol(expression, symbol)
235 end
236

```

Erkennung von Kommentaren

```

237 local function tex_comment_text(line)
238     -- A guard is not a documentation comment.
239     if line:match("^%S%%S*<") then
240         return nil
241     end
242
243     local markers, text = line:match("^%S*(%%+)%S?(.*)$")
244
245     if markers == "%" then
246         return text
247     end
248
249     return nil
250 end
251
252 autodoc.tex_comment_text = tex_comment_text
253
254 local function lua_line_comment_text(line)
255     -- Again: a Lua guard is not a documentation comment.
256     if line:match("^%S*%-%-S*<") then
257         return nil

```

258 end

259

Lua-Langkommentare werden gesondert behandelt

260 if line:match("^%s*%-%-%(=*)%(") then

261 return nil

262 end

263

264 if line:match("^%s*%-%-[%s*\$") then

265 return nil

266 end

267

268 return line:match("^%s*%-%-+s?(.*)\$")

269 end

270

271 local function comment_text(line, syntax)

272 if syntax.comment_kind == "tex" then

273 return tex_comment_text(line)

274 end

275

276 return lua_line_comment_text(line)

277 end

278

Lua-Langkommentare. Folgende Varianten werden erkannt:

- -- [[...]]
- -- [= [...]=]
- -- [== [...]==]

279

280 local function lua_long_comment_start(line)

281 local equals, rest =

282 line:match("^%s*%-%-%(=*)%[%s?(.*)\$")

283

284 if equals ~= nil then

285 return {

286 pseudo = false,

287 equals = equals,

288 text = rest,

289 }

290 end

291

292 local rest_pseudo =

293 line:match("^%s*%-%-[%s?(.*)\$")

294

3 Implementation

```
295     if rest_pseudo ~= nil then
296         return {
297             pseudo = true,
298             equals = "",
299             text = rest_pseudo,
300         }
301     end
302
303     return nil
304 end
305
306 local function lua_long_comment_line(line, long_comment)
307     if long_comment.pseudo then
308         local text = line:match("^(.)%s*%-[%]s*$")
309
310         if text ~= nil then
311             return text, true
312         end
313
314         return line, false
315     end
316
317     local delimiter =
318         "%]"
319     .. quote_pattern(long_comment.equals)
320     .. "%]"
321
322     local text =
323         line:match("^(.)%s*" .. delimiter .. "%s*$")
324
325     if text ~= nil then
326         return text, true
327     end
328
329     return line, false
330 end
331
```

Nicht maskierte Unterstriche in \cs-Argumenten maskieren.

```
332 local function is_escaped(text, position)
333     local backslashes = 0
334     local index = position - 1
335
336     while index >= 1 and text:sub(index, index) == "\\" do
```

3 Implementation

```
337     backslashes = backslashes + 1
338     index = index - 1
339 end
340
341     return backslashes % 2 == 1
342 end
343
344 local function skip_spaces(text, position)
345     while position <= #text and text:sub(position, position):match("%s") do
346         position = position + 1
347     end
348
349     return position
350 end
351
352
353 local function skip_balanced(text, position, opening, closing)
354     local depth = 0
355
356     for index = position, #text do
357         local character = text:sub(index, index)
358
359         if not is_escaped(text, index) then
360             if character == opening then
361                 depth = depth + 1
362             elseif character == closing then
363                 depth = depth - 1
364
365                 if depth == 0 then
366                     return index + 1
367                 end
368             end
369         end
370     end
371
372     return nil
373 end
374
375
376 local function escape_cs_underscores(text)
377     local result = {}
378     local cursor = 1
379     local search = 1
380
```

3 Implementation

```
381 while search <= #text do
382     local command_start, after_name = text:find("\\cs", search, true)
383
384     if command_start == nil then
385         break
386     end
387
388     local following = text:sub(after_name + 1, after_name + 1)
389
390     if is_escaped(text, command_start)
391         or following:match("%a@")
392     then
393         search = after_name + 1
394     else
395         local position = skip_spaces(text, after_name + 1)
396
397         if text:sub(position, position) == "*" then
398             position = skip_spaces(text, position + 1)
399         end
400
401         if text:sub(position, position) == "[" then
402             position = skip_balanced(text, position, "[", "]")
403
404             if position == nil then
405                 break
406             end
407
408             position = skip_spaces(text, position)
409         end
410
411         if text:sub(position, position) ~= "{" then
412             search = after_name + 1
413         else
414             local argument_end = skip_balanced(text, position, "{", "{")
415
416             if argument_end == nil then
417                 break
418             end
419
420             result[#result + 1] = text:sub(cursor, position)
421
422             for index = position + 1, argument_end - 2 do
423                 local character = text:sub(index, index)
424             end
425         end
426     end
427 end
```

3 Implementation

```
425         if character == "_" and not is_escaped(text, index) then
426             result[#result + 1] = "\\"
427         end
428
429         result[#result + 1] = character
430     end
431
432     result[#result + 1] = "}"
433     cursor = argument_end
434     search = argument_end
435 end
436 end
437 end
438
439 result[#result + 1] = text:sub(cursor)
440 return table.concat(result)
441 end
442
```

Kleine ASCII-Auszeichnungssprache für Dokumentationskommentare.

```
443 local function ascii_escape_text(text)
444     local replacements = {
445         ["#"] = "\\#",
446        ["$"] = "\\$",
447         ["%"] = "\\%",
448         ["&"] = "\\&",
449         [{""] = "\\{",
450         ["}"] = "\\}",
451         ["~"] = "\\textasciitilde{}",
452         ["^"] = "\\textasciicircum{}",
453         ["_"] = "\\_",
454     }
455
456     return (text:gsub("#$%&{~^_", replacements))
457 end
458
459 local function ascii_escape_code(text)
460     local replacements = {
461         ["\\"] = "\\textbackslash{}",
462         ["#"] = "\\#",
463        ["$"] = "\\$",
464         ["%"] = "\\%",
465         ["&"] = "\\&",
466         [{""] = "\\{",

```

3 Implementation

```
467     ["}"] = "\\}",
468     ["~"] = "\\textasciitilde{}",
469     ["^"] = "\\textasciicircum{}",
470     ["_"] = "\\_",
471 }
472
473 return (text:gsub("[\\#$$%&{}~^_]", replacements))
474 end
475
476 local function ascii_delimiter_can_open(text, position, marker)
477     if marker == "`" then
478         return true
479     end
480
481     local before = text:sub(position - 1, position - 1)
482     local after = text:sub(position + 1, position + 1)
483     return not before:match("[%w_]") and not after:match("%s")
484 end
485
486 local function ascii_delimiter_can_close(text, position, marker)
487     if marker == "`" then
488         return true
489     end
490
491     local before = text:sub(position - 1, position - 1)
492     local after = text:sub(position + 1, position + 1)
493     return not before:match("%s") and not after:match("[%w_]")
494 end
495
496 local function ascii_find_closing(text, marker, start)
497     local position = start
498
499     while true do
500         position = text:find(marker, position, true)
501
502         if position == nil then
503             return nil
504         end
505
506         if not is_escaped(text, position)
507             and ascii_delimiter_can_close(text, position, marker)
508         then
509             return position
510         end
511     end
512 end
```

3 Implementation

```
511
512     position = position + 1
513 end
514 end
515
516 local function ascii_inline_to_tex(text)
517     local result = {}
518     local plain = {}
519     local position = 1
520
521     local function flush_plain()
522         if #plain > 0 then
523             result[#result + 1] = ascii_escape_text(table.concat(plain))
524             plain = {}
525         end
526     end
527
528     while position <= #text do
529         local character = text:sub(position, position)
530
531         if character == "\\" then
532             local following = text:sub(position + 1, position + 1)
533             local name = text:sub(position + 1):match("^([%a@:][%w@:]*)")
534
535             if name ~= nil then
536                 flush_plain()
537                 result[#result + 1] = "\\cs{" .. name .. "}"
538                 position = position + #name + 1
539             elseif following == "`" or following == "*" or following == "_" then
540                 flush_plain()
541                 if following == "_" then
542                     result[#result + 1] = "\\_"
543                 elseif following == "`" then
544                     result[#result + 1] = "\\textasciigrave{"
545                 else
546                     result[#result + 1] = following
547                 end
548                 position = position + 2
549             else
550                 flush_plain()
551                 result[#result + 1] = "\\textbackslash{"
552                 position = position + 1
553             end
554         end
555     end
556 end
```

3 Implementation

```
555     elseif (character == "\"" or character == "*" or character == "_")
556       and ascii_delimiter_can_open(text, position, character)
557     then
558       local closing = ascii_find_closing(text, character, position + 1)
559
560       if closing ~= nil and closing > position + 1 then
561         local contents = text:sub(position + 1, closing - 1)
562         flush_plain()
563
564         if character == "\"" then
565           local cs_name = contents:match("^\\([%a@_:[%w@_:]*)$")
566
567           if cs_name ~= nil then
568             result[#result + 1] = "\\cs{" .. cs_name .. "}"
569           else
570             result[#result + 1] = "\\code{" .. ascii_escape_code(contents) ..
571               ↳ "}"
572             end
573           elseif character == "*" then
574             result[#result + 1] = "\\textbf{" .. ascii_inline_to_tex(contents) ..
575               ↳ "}"
576             else
577               result[#result + 1] = "\\emph{" .. ascii_inline_to_tex(contents) ..
578                 ↳ "}"
579             end
580
581             position = closing + 1
582           else
583             plain[#plain + 1] = character
584             position = position + 1
585           end
586         end
587       else
588         plain[#plain + 1] = character
589         position = position + 1
590       end
591     end
592
593     flush_plain()
594     return table.concat(result)
595 end
596
597 autodoc.ascii_inline_to_tex = ascii_inline_to_tex
598
```

Temporäre Verbatim-Dateien

```

596 local function unique_temp_name()
597     local name = string.format(
598         "osgdoc-%s-%04d.tmp",
599         tex.jobname,
600         #verbtempfiles + 1
601     )
602     verbtempfiles[#verbtempfiles + 1] = name
603     return name
604 end
605

```

Hauptfunktion für AutoDocInput

```

606 function autodoc.autodoc_process(filename, guard, mode)
607     local syntax = get_syntax(mode)
608     local first = true
609     local input = io.open(filename, "r")
610
611     if not input then
612         tex.error(
613             "AutoDocInput: cannot open `" .. filename .. "'"
614         )
615         return
616     end

```

Zum Parsen wird ein kleiner Zustandsautomat genutzt. Dabei haben die Zustände folgende Bedeutungen:

- **OUTSIDE**: Außerhalb eines ausgewählten Guard-Bereichs; keine Verbatim-Datei offen.
- **OUTSIDE_VERBATIM**: Außerhalb eines ausgewählten Guard-Bereichs; eine Verbatim-Datei ist weiterhin offen. Der nächste passende Guard setzt denselben Codeblock fort.
- **START**: Direkt hinter einem öffnenden Guard; noch keine Entscheidung zwischen Dokumentation und Code getroffen.
- **VERBATIM**: Innerhalb eines ausgewählten Guard-Bereichs und mit geöffneter Verbatim-Datei.
- **COMMENT**: Innerhalb eines ausgewählten Guard-Bereichs und in Dokumentation.
- **LONG_COMMENT**: Innerhalb eines mehrzeiligen Lua-Dokumentationskommentars.

```

617 local OUTSIDE = 0
618 local OUTSIDE_VERBATIM = 1
619 local START = 2

```

3 Implementation

```
620 local VERBATIM = 3
621 local COMMENT = 4
622 local LONG_COMMENT = 5
623
624 local state = OUTSIDE
625
626 local long_comment = nil
627 local long_comment_return_state = nil
628
629 local verbatim_file = nil
630 local verbatim_filename = nil
631
632 local line_number = 0
633 local documentation_list_open = false
634
635 local function warn(message)
636     tex.warning(
637         string.format(
638             "AutoDocInput: %s in %s:%d",
639             message,
640             filename,
641             line_number
642         )
643     )
644 end
645
646 local function close_documentation_list()
647     if documentation_list_open then
648         tex.print("\\end{itemize}")
649         documentation_list_open = false
650     end
651 end
652
653 local function print_documentation(text)
654     text = text or ""
655
656     if syntax.documentation_format ~= "ascii" then
657         tex.print(escape_cs_underscores(text))
658         return
659     end
660
661     local heading_marks, heading = text:match("^%s*(##?)%s+(.+)%s*$")
662
663     if heading ~= nil then
```

3 Implementation

```
664     close_documentation_list()
665     local command = heading_marks == "#" and "section" or "subsection"
666     tex.print(
667         "\\\" .. command .. "{"
668         .. escape_cs_underscores(ascii_inline_to_tex(heading))
669         .. "}"
670     )
671     return
672 end
673
674 local item = text:match("^%s*[%-%*]%s+(.)$")
675
676 if item ~= nil then
677     if not documentation_list_open then
678         tex.print("\\begin{itemize}")
679         documentation_list_open = true
680     end
681
682     tex.print("\\item \" .. escape_cs_underscores(ascii_inline_to_tex(item)))
683     return
684 end
685
686 close_documentation_list()
687
688 if text:match("^%s*===+%s*$") then
689     tex.print("\\par\\smallskip\\noindent\\hrule\\smallskip")
690 else
691     tex.print(escape_cs_underscores(ascii_inline_to_tex(text)))
692 end
693 end
694
695 local function open_verbatim(first_line)
696     if verbatim_file ~= nil then
697         tex.error(
698             "AutoDocInput: internal error",
699             {
700                 string.format(
701                     "Attempt to open a second verbatim file in %s:%d.",
702                     filename,
703                     line_number
704                 )
705             }
706         )
707     return false
```

3 Implementation

```
708     end
709
710     verbatim_filename = unique_temp_name()
711
712     verbatim_file = io.open(verbatim_filename, "w")
713
714     if not verbatim_file then
715         tex.error(
716             "AutoDocInput: cannot open temporary file",
717             {verbatim_filename}
718         )
719         verbatim_filename = nil
720         return false
721     end
722
723     verbatim_file:write(first_line, "\n")
724     return true
725 end
726
727 local function write_verbatim(line)
728     if verbatim_file == nil then
729         tex.error(
730             "AutoDocInput: internal parser error",
731             {
732                 string.format(
733                     "VERBATIM state without an open file in %s:%d.",
734                     filename,
735                     line_number
736                 )
737             }
738         )
739         return false
740     end
741
742     verbatim_file:write(line, "\n")
743     return true
744 end
745
746 local function close_verbatim()
747     if verbatim_file == nil then
748         return
749     end
750
751     verbatim_file:close()
```

3 Implementation

```
752     verbatim_file = nil
753
```

Die Zeilennummer entspricht der logischen Zeilennummer der Implementation, d. h. generierter Boilerplate wird nicht berücksichtigt.

```
754     if first then
755         tex.print(
756             "\\VerbatimInput[firstnumber=1]{\"
757             .. verbatim_filename
758             .. \"}\"
759         )
760         first = false
761     else
762         tex.print(
763             "\\VerbatimInput[firstnumber=last]{\"
764             .. verbatim_filename
765             .. \"}\"
766         )
767     end
768     verbatim_filename = nil
769 end
770
771 local function begin_long_comment(start, return_state)
772     long_comment = start
773     long_comment_return_state = return_state
774     state = LONG_COMMENT
775
776     if start.text == "" then
777         return
778     end
779
780     local text, finished =
781         lua_long_comment_line(start.text, long_comment)
782
783     print_documentation(text)
784
785     if finished then
786         state = long_comment_return_state
787         long_comment_return_state = nil
788         long_comment = nil
789     end
790 end
791
792 for line in input:lines() do
```

Der eigentliche Automat

```

793     line_number = line_number + 1
794
795     if state == LONG_COMMENT then
796         local text, finished =
797             lua_long_comment_line(line, long_comment)
798
799         print_documentation(text)
800
801         if finished then
802             state = long_comment_return_state
803             long_comment_return_state = nil
804             long_comment = nil
805         end
806
807     else
808         local opening =
809             is_opening_guard(line, syntax, guard)
810
811         local closing =
812             is_closing_guard(line, syntax, guard)
813
814         local documentation =
815             comment_text(line, syntax)
816
817         local long_start = nil
818
819         if syntax.comment_kind == "lua" then
820             long_start = lua_long_comment_start(line)
821         end
822
823         if state == OUTSIDE then
824             if opening then
825                 state = START
826             end
827
828         elseif state == OUTSIDE_VERBATIM then
829             if opening then
830                 if verbatim_file == nil then
831                     tex.error(
832                         "AutoDocInput: internal parser error",
833                         {
834                             string.format(
835                                 "OUTSIDE_VERBATIM without an open file in %s:%d.",

```

3 Implementation

```
836         filename,
837         line_number
838     )
839 }
840 )
841 state = START
842 else
843     state = VERBATIM
844 end
845 end
846
847 elseif state == START then
848     if closing then
849         close_documentation_list()
850         -- Leerer Guard-Bereich; es wurde keine Datei geöffnet.
851         state = OUTSIDE
852
853     elseif opening then
854         warn("cascading opening guards")
855
856     elseif long_start then
857         begin_long_comment(long_start, COMMENT)
858
859     elseif documentation ~= nil then
860         print_documentation(documentation)
861         state = COMMENT
862
863     else
864         close_documentation_list()
865         if open_verbatim(line) then
866             state = VERBATIM
867         else
868             state = OUTSIDE
869         end
870     end
871
872 elseif state == VERBATIM then
873     if closing then
874         state = OUTSIDE_VERBATIM
875
876     elseif opening then
877         warn("cascading opening guards")
878
879     elseif long_start then
```

3 Implementation

```
880         close_verbatim()
881         begin_long_comment(long_start, COMMENT)
882
883     elseif documentation ~= nil then
884         close_verbatim()
885         print_documentation(documentation)
886         state = COMMENT
887
888     else
889         write_verbatim(line)
890     end
891
892 elseif state == COMMENT then
893     if closing then
894         close_documentation_list()
895         state = OUTSIDE
896
897     elseif opening then
898         warn("cascading opening guards")
899
900     elseif long_start then
901         begin_long_comment(long_start, COMMENT)
902
903     elseif documentation ~= nil then
904         print_documentation(documentation)
905
906     else
907         close_documentation_list()
908         if open_verbatim(line) then
909             state = VERBATIM
910         else
911             state = OUTSIDE
912         end
913     end
914
915 else
916     tex.error(
917         "AutoDocInput: invalid parser state",
918         {
919             string.format(
920                 "Invalid state %s in %s:%d.",
921                 tostring(state),
922                 filename,
923                 line_number
```

3 Implementation

```
924         )
925     }
926 )
927     break
928 end
929 end
930 end
931
932 if state == LONG_COMMENT then
933     warn("unterminated Lua long comment")
934 end
935
936 close_documentation_list()
937
938 -- A file can be open in both, VERBATIM or OUTSIDE_VERBATIM
939 if verbatim_file ~= nil then
940     close_verbatim()
941 end
942
943 input:close()
944 end
945
```

Temporäre Dateien aufräumen.

```
946 luatexbase.add_to_callback(
947     "stop_run",
948     function()
949         for _, filename in ipairs(verbtempfiles) do
950             os.remove(filename)
951         end
952     end,
953     "cleanup osgdoc verbatim temporary files"
954 )
955 \end{luacode*}
956
957
```

Dateiinformatoren

`\UseVersionOf` und `\UseDateOf` nutzen beide die Lua-Funktion `useversion`.

```
958 \ifosgdoc@readprov\else
959     \NewExpandableDocumentCommand{\UseVersionOf}{m}{%
960         \directlua{fileinfo.useversion(\luastring{#1})}%
961     }
962 \end{luacode*}
963
```

3 Implementation

```

961 }
962
963 \NewExpandableDocumentCommand{\UseDateOf}{m}{%
964   \directlua{fileinfo.usedate(\luastring{#1})}%
965 }
966 \fi
967
968 \begin{luacode*}
969 fileinfo = fileinfo or {}
970
971 fileinfo.data = {}
972
973 function fileinfo.getinfo(filename)
974   local cached = fileinfo.data[filename]
975   if cached ~= nil then
976     return cached
977   end
978
979   local resolved_filename = kpse.find_file(filename)
980   local f = texassert(
981     resolved_filename and io.open(resolved_filename, "r"),
982     "Can't read '" .. filename .. "'"
983   )
984   local s = f:read("*a")
985   f:close()
986   local ftype = nil
987   local name,date,version,descr = s:match(
988     "\\ProvidesExplPackage%s*{([^}]*)}%s*{([^}]*)}%s*{([^}]*)}%s*{([^}]*)}")
989   ftype = "package"
990   if name == nil then
991     name,date,descr = s:match(
992       "\\ProvidesPackage{
993         ↪ ([^}]*)}%s*[(%d%d%d%d[%-/]%d%d[%-/]%d%d)%s?([^%]*)%)"
994   if name ~= nil then
995     descr = descr or "??"
996     version = descr:match("^v?(%d+[%p%d]*%l*)%s")
997   end
998 end
999
1000 if name == nil then
1001   name,date,version,descr = s:match(
1002     "\\ProvidesExplClass%s*{([^}]*)}%s*{([^}]*)}%s*{([^}]*)}%s*{([^}]*)}")
1003   ftype = "class"
1004   if name == nil then

```

3 Implementation

```

1004     name,date,descr = s:match(
1005         "\\ProvidesClass{
1006             ↪ ([^}]*)}%s*[(%d%d%d[d[-/]%d%d[-/]%d%d)%s?([^\%]*)%]"
1007     if name ~= nil then
1008         descr = descr or "???"
1009         version = descr:match("^v?(%d+(%p%d+)*%l*)%s")
1010     end
1011 end
1012 if name == nil then
1013     name,date,version,descr = s:match(
1014         "\\ProvidesExplFile%s*{([^\%]*)}%s*{([^\%]*)}%s*{([^\%]*)}%s*{([^\%]*)}"
1015     ftype = "file"
1016     if name == nil then
1017         name,date,descr = s:match(
1018             "\\ProvidesFile{
1019                 ↪ ([^}]*)}%s*[(%d%d%d[d[-/]%d%d[-/]%d%d)%s?([^\%]*)%]"
1020     if name ~= nil then
1021         descr = descr or "???"
1022         version = descr:match("^v?(%d+(%p%d+)*%l*)%s")
1023     end
1024 end
1025 if name ~= nil then
1026     date = date or "???"
1027     version = version or "???"
1028     cached = {
1029         name = name,
1030         date = date,
1031         version = version,
1032         descr = descr,
1033         type = ftype,
1034     }
1035     fileinfo.data[filename] = cached
1036 end
1037
1038     return cached
1039 end
1040
1041 function fileinfo.useversion(filename)
1042     local info = fileinfo.getinfo(filename)
1043     tex.sprint(info and info.version or "??")
1044 end
1045

```

3 *Implementation*

```
1046 function fileinfo.usedate(filename)
1047     local info = fileinfo.getinfo(filename)
1048     tex.sprint(info and info.date or "?")
1049 end
1050 \end{luacode*}
1051
```

Index

A

`ascii` 3
`\AutoDocInput` 2, 10

B

`babel` (Paket) 5

C

`\cls` 9
`cnltx` (Paket) 1, 4, 8
`cnltx-doc` (Klasse) 8
`\cs` 3, 15
`\cs*` 3
`CTAN` 1 f., 4

D

`doc` (Paket) 2 f.
`docstrip` (Paket) 2

E

`example` (Umgebung) 5

F

`fancyverb` (Paket) 10
`\File` 9
`fvextra` (Paket) 10

G

`gmdoc` (Paket) 2

I

`infobox` (Umgebung) 3

K

`\keys_set:nn` 3

L

`langselect` (Paket) 5
`libertine` (Paket) 5
`libertinehologopatch` (Paket) 5
`lua` 2 f.
`luacode` (Paket) 10

M

`\macro` 3
`\MaybeStop` 3
`metalogox` (Paket) 5
`multicol` (Paket) 4

O

`\OnlyDescription` 3
`osgdoc` (Klasse) 8

P

`\pkg` 9
`polyglossia` (Paket) 5
`\ProvidesClass` 3
`\ProvidesExplClass` 3
`\ProvidesExplFile` 3
`\ProvidesExplPackage` 3
`\ProvidesFile` 3
`\ProvidesPackage` 3

R

`ragged2e` (Paket) 4
`\ReadFileInfos` 2
`readprov` 2
`readprov` (Paket) 2, 4

S

`shellestc` (Paket) 5
`superiors` (Paket) 5

T

`tex` 2 f.
`\textsu` 5

U

`\UseDateOf` 2 f., 30
`\UseVersionOf` 2 f., 30

W

`warnbox` (Umgebung) 3