

codemwe 代码框及结果框排版

与 MWE 文件生成宏包 $\Rightarrow$ English Version

耿楠 <nangeng@nwafu.edu.cn>

2026 年 10 月 24 日 v1.1.0 *

目录

1	引言	4
2	用户接口	4
2.1	example 环境	4
2.2	example* 环境	5
2.3	texcode 环境	5
2.4	ccode 环境	6
2.5	\codemwreset 全局选项设置	6
2.6	\codemweNewBox 注册新盒子	6
2.7	\codemweNewLang 注册新语种	7
2.8	\codemweNewBackend 注册新代码后端	7
3	排版模式	8
3.1	水平模式	8
3.2	垂直模式	8
3.3	自定义过渡文字	9
3.4	水平模式下的间距控制	10
3.5	代码框宽度控制	10
4	编号与引用	11
5	仅显示代码	12
6	结果图片	12
6.1	结果图片的宽度与高度控制	13

*https://gitee.com/nwafu_nan/codemwe

7	输出前置内容	13
8	代码后端	14
9	显示行号	16
10	完整示例	17
11	将环境体保存为文件	18
12	语法高亮	19
	12.1 在正文中开关语法高亮	20
13	宏包选项	21
14	未知选项与警告	21
15	盒子类型	22
16	选项说明	23
17	注意事项	27
18	实现简介	28
19	常见问题	28
20	Introduction	30
21	User Interfaces	31
	21.1 The <code>example</code> Environment	31
	21.2 The <code>example*</code> Environment	31
	21.3 The <code>texcode</code> Environment	32
	21.4 The <code>ccode</code> Environment	32
	21.5 <code>\codemweset</code> for Global Settings	33
	21.6 <code>\codemweNewBox</code> for Registering New Boxes	33
	21.7 <code>\codemweNewLang</code> for Registering New Language Varieties	34
	21.8 <code>\codemweNewBackend</code> for Registering New Code Backends	35
22	Typesetting Modes	35
	22.1 Horizontal Mode	35
	22.2 Vertical Mode	36
	22.3 Custom Transition Text	37
	22.4 Spacing in Horizontal Mode	37
	22.5 Code Box Width	38

23	Numbering and References	38
24	Showing the Code Only	39
25	Result Figures	39
25.1	Width and Height of the Result Figure	40
26	Outputting the Preliminary Content	40
27	Code Backends	40
28	Displaying Line Numbers	43
29	A Complete Example	44
30	Saving the Environment Body to a File	45
31	Syntax Highlighting	45
31.1	Switching Highlighting On and Off in the Body	47
32	Package-Level Default Options	47
33	Unknown Options and Warnings	48
34	Box Types	49
35	Options	50
36	Notes	55
37	Implementation Overview	55
38	Frequently Asked Questions	56

1 引言

`codemwe` 是一个基于 `tcolorbox` 宏包用 `Expl3` 开发的示例代码排版宏包，它用于排版示例代码框及其编译结果框，并自动生成可独立编译的 MWE(Minimal Working Example) 文件。它将示例代码置于代码框中，将其编译结果置于结果框中，两者可按水平或垂直方式排列，适合代码与结果的对比阅读。

该宏包提供了 `example`、`example*`、`texcode` 和 `ccode` 四个环境。其中，`example` 环境默认采用水平模式排版，即左侧为代码框、右侧为结果框；`example*` 环境是垂直模式的兼容写法，即上方为代码框、下方为结果框；`texcode` 环境仅排版一个代码框，不编号、不编译，适合插入无需编译的代码片段；`ccode` 环境用于排版 C 语言代码，仅显示代码框并生成 `.c` MWE 源文件。

同时，该宏包还提供了 `\codemwreset` 命令用于设置全局默认选项，`\codemweNewBox` 命令用于注册用户自定义的盒子，`\codemweNewLang` 命令用于注册用户自定义的过渡语、注释词条的语种，以及 `\codemweNewBackend` 命令用于注册需要排版的后端代码语言。

除本节的总体介绍外，后面依次讨论用户接口（第 2 节）、排版模式（第 3 节）、编号与引用、结果图片、导言区内容的输出控制、代码后端（第 8 节）、行号、宏包选项、盒子类型、全部选项说明（第 16 节）、注意事项、实现简介与常见问题。若要快速上手，可先看第 2 节与第 3 节，再按需查阅第 16 节。

注意：为实现中文排版，在手册的中文部分将 `codemwe` 宏包的 `textlang` 语种选项设置为 `zh`，在中文结束后，排版英文手册之前恢复英文。

```
\codemwreset{ textlang = zh }
```

2 用户接口

2.1 `example` 环境

```
\begin{example} \begin{<example>}[<选项>]{<前置内容>}  
\end{example}  <示例代码>  
New: 2026-10-10 \end{<example>}
```

用于排版示例代码及其编译结果，默认以水平模式（左代码框、右结果框）显示，同时自动生成可独立编译的 MWE 源文件。

该环境有一个可选参数 `[<选项>]` 和一个**必选**参数 `{<前置内容>}`。

`[<选项>]` 为逗号分隔的 key-value 列表。有些选项是布尔型，只能取 `true` 或 `false`。对于这些选项，`{<选项>} = true` 中的 “= true” 可以省略。

`{<前置内容>}` 用于写入 MWE 文件的前置内容（如 `LATEX` 文件的导言区），可包含 `\newcommand`、`\tikzset`、`\include <stdio.h>` 等内容。若不需要追加前置内容，则花括号留空 `{}`，但**必须**有一对空花括号。

`example` 环境默认会对示例代码进行编号，若有 `\chapter` 命令，则其编号格式为“章号. 编号”，并会自动生成引用标签（如 `ex:1-1`）；若无 `\chapter` 命令，则其编号格式为“编号”，自动生成的引用标签如 `ex:1`。如：

```

\begin{example}[cls={ctexart}]{ }
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { 猫儿喜欢鱼 }
\tl_use:N \l__test_tl
\ExplSyntaxOff
\end{example}

```

2.2 example* 环境

<code>\begin{example*}</code>	<code>\begin{<example*>}[<选项>]{<前置内容>}</code>
<code>\end{example*}</code>	<code><示例代码></code>
	<code>\end{<example*>}</code>

New: 2026-10-10

用于排版示例代码及其编译结果, 默认以上代码框和下结果框的垂直模式显示。它等价于设置了 `vmode = true` 的 `example` 环境; 如设置了 `[cls={ctexart}, pkg = {tikz}]` 的 `example*` 环境相当于设置了 `[vmode, cls={ctexart}, pkg = {tikz}]` 的 `example` 环境。如:

```

\begin{example*}[cls={ctexart}, pkg={tikz}]{ }
\begin{tikzpicture}
\draw (0, 0) -- (1, 0) -- (1, 1);
\end{tikzpicture}
\end{example*}

```

2.3 texcode 环境

<code>\begin{texcode}</code>	<code>\begin{<texcode>}[<选项>]</code>
<code>\end{texcode}</code>	<code><代码片段></code>
	<code>\end{<texcode>}</code>

New: 2026-10-10

用于排版 L^AT_EX 示例代码, 不编号、不生成引用标签、不编译示例代码, 也不生成 MWE `tex` 文件。它适用于在正文中插入无需编译的示例代码片段。

`[<选项>]` 与 `example` 环境一致。

`texcode` 环境实际上是通过为 `example` 环境预置 `[vmode, nonum, noshow]` 选项实现的, 它等价于 `\begin{example}[vmode, nonum, noshow]{<...>}\end{example}`。

```

\begin{texcode}
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { 猫儿喜欢鱼 }
\tl_use:N \l__test_tl
\ExplSyntaxOff
\end{texcode}

```

2.4 ccode 环境

<code>\begin{ccode}</code>	<code>\begin{<ccode>}[<选项>]</code>
<code>\end{ccode}</code>	<code><C 语言代码片段></code>

New: 2026-10-20

用于排版 C 语言代码，仅显示一个代码框，不编号、不显示结果框，同时按 C 语言格式生成源文件（见第 8 节）。它适用于插入 C 语言代码片段或函数定义。

[<选项>] 与 `example` 环境一致，其中 `backend` 已被预置为 `c`，通常无须再次指定。

`ccode` 环境实际上是通过为 `example` 环境预置 `vmode`、`nonum`、`noshow` 与 `backend = c` 选项实现的，即 `example` 环境加上这四个选项。

```
\begin{ccode}[filename = { hello.c }]{  
#include <stdio.h>  
  
int main(void)  
{  
    printf("Hello, World!\n");  
    return 0;  
}  
\end{ccode}
```

注意：`ccode` 环境的主体部分按原样写入 `c` 文件，因此 `cls`、`pkg` 等 L^AT_EX 专属选项对它没有意义。若代码中需要 `#include` 等前置内容，可直接写入环境体，或写在环境的必选参数（导言区附加内容）中，二者都会出现在生成的源文件里；区别在于后者不会显示在代码框中，除非同时使用 `withextra` 选项。

2.5 \codemweset 全局选项设置

<code>\codemweset</code>	<code>\codemweset{<选项>}</code>
--------------------------	--------------------------------------

New: 2026-10-10

用于在当前作用域内修改后续 `example`、`example*`、`texcode` 和 `ccode` 环境的默认选项，{<选项>} 为逗号分隔的 key-value 列表。如：

```
\codemweset{ vmode, withpkg }
```

2.6 \codemweNewBox 注册新盒子

<code>\codemweNewBox</code>	<code>\codemweNewBox{<盒子环境名>}[<选项>]</code>
-----------------------------	--

New: 2026-10-10

用于向 `codemwe` 注册一个新的盒子环境，注册后即可通过 `boxtype` 选项使用该盒子排版代码框和结果框。

{<盒子环境名>} 必须符合 “0{背景色} d() 0{额外选项}” 形式的参数接口，建议用 tcolorbox 的 \DeclareTColorBox 命令定义。[<选项>] 中目前只支持 withtitle，用于声明该盒子是否支持标题栏，其初始值与默认值均为 true。如：

```
\codemweNewBox{ mybox }[ withtitle = true ]
```

注意：宏包内部维护了一张“盒子环境名称 → 是否支持标题栏”的属性表，6 个内置盒子已预先注册。 \codemweNewBox 只登记属性，它无法实现环境本身的定义，因此须先用 \DeclareTColorBox（或等价方式，如 \NewDocumentEnvironment）定义 {<盒子环境名>}，再调用本命令实现注册。

如果使用了与内置盒子环境名称同名的盒子，则会覆盖内置盒子的设置。

2.7 \codemweNewLang 注册新语种

```
\codemweNewLang \codemweNewLang{<语种名称>}{<8 段内容>}
```

New: 2026-10-13

用于向 codemwe 注册一种新的语种，设置该语种对应的过渡语、注释词等文本，注册后即可通过 textlang 选项使用该新语种排版过渡语、添加的注释内容等。

{<语种名称>} 是新语种的名称。{<8 段内容>} 由 8 段逗号分隔的文本构成（前过渡语，后过渡语，无编号过渡语，文档类注释词，导言区注释词，正文区词，默认文档类名称，标题及 \ref 引用结果引导词）。如：

```
\codemweNewLang{ fr }
{
  Le-résultat-de,
  est-affiché-ci-dessous.,
  Le-résultat-du-code-est-affiché-ci-dessous.,
  classe-de-document,
  préambule,
  corps,
  article,
  exemple
}
```

宏包内置了 en（英文）和 zh（中文）两个语种。如果 \codemweNewLang 定义的语种名称与内置名称重名，则会覆盖内置语种文本内容。

注意：{<8 段内容>} 各段内容中如有逗号则需要用大括号进行保护，空白字段需要留白，不能省略。

2.8 \codemweNewBackend 注册新代码后端

```
\codemweNewBackend \codemweNewBackend{<后端名称>}{<文件扩展名>}{<注释符>}{<能否编译>}[<minted 语言名>]
```

New: 2026-10-20

Updated: 2026-10-23

用于向 `codemwe` 注册一种新的代码后端（源语言），注册后即可通过 `backend` 选项使用该后端排版代码、生成对应格式的源文件。

`{<后端名称>}` 是新后端的名称，也是 `backend` 选项的取值。`{<文件扩展名>}`（不含点号），如 `c`、`py`。`{<注释符>}` 按原样读取，用于源文件的首行注释，如百分号、`//`或`#`号，均无须转义。`{<能否编译>}` 取 `true` 时，源文件按 `LATEX` 文档编译并在结果框中显示结果；取 `false` 时，不对其按 `LATEX` 编译，结果框只能显示由 `result` 选项指定的图片。`[<minted 语言名>]` 是可选参数，给出该语言在 `minted` 宏包中对应的语言名（即 `pygments` 的 `lexer` 名），用于 `minted` 选项下的语法高亮，如 `python`、`javascript`。省略或留空时默认为 `{<后端名称>}` 本身。若不使用 `minted` 选项，该参数不影响任何排版效果。

```
\codemweNewBackend { python } { py } { # } { false }
```

若该后端也要用于语法高亮，可使用可选项给出语言名：

```
\codemweNewBackend { py } { py } { # } { false } [ python ]
```

宏包内置了 `tex`（默认）和 `c` 两种后端。如果`\codemweNewBackend`注册的后端名称与内置名称重名，则会覆盖内置后端的设置。

3 排版模式

3.1 水平模式

水平模式为默认模式，左侧为代码框，右侧为结果框，适合代码与结果对比阅读。水平模式下的宽度分配规则为：若未指定 `width`，则减去左右框之间的间距 `sep` 后，按剩余宽度的 $1/2$ 等宽排版左右两栏。如：

```
\begin{example}{  
  \ExplSyntaxOn  
  \tl_clear_new:N \l__fsttest_tl  
  \tl_set:Nn \l__fsttest_tl  
    { 猫儿喜欢鱼 }  
  \tl_use:N \l__fsttest_tl  
  \ExplSyntaxOff  
}\end{example}
```

的排版结果为：

```
% 例 1  
\ExplSyntaxOn  
\tl_clear_new:N \l__fsttest_tl  
\tl_set:Nn \l__fsttest_tl  
  { 猫儿喜欢鱼 }  
\tl_use:N \l__fsttest_tl  
\ExplSyntaxOff
```

猫儿喜欢鱼

3.2 垂直模式

垂直模式下，上方为代码框，下方为结果框，中间可插入过渡文字。通过 `vmode` 选项实现模式切换。如：

```

\begin{example}[vmode]{}
\ExplSyntaxOn
\tl_clear_new:N \l__sectest_tl
\tl_set:Nn \l__sectest_tl { 猫儿喜欢鱼 }
\tl_use:N \l__sectest_tl
\ExplSyntaxOff
\end{example}

```

的排版结果为：

```

% 例 2
\ExplSyntaxOn
\tl_clear_new:N \l__sectest_tl
\tl_set:Nn \l__sectest_tl { 猫儿喜欢鱼 }
\tl_use:N \l__sectest_tl
\ExplSyntaxOff

```

以下是例 2 编译后得到的结果。

猫儿喜欢鱼

垂直模式下，代码框与结果框之间的过渡文字取自 `textlang` 选项所选语种对应的默认过渡语（见第 16 节的选项说明）。本手册的导言区已用 `\codemwreset` 把语种设为中文，因此上例中出现的过渡文字为“以下是例 2 编译后得到的结果。”（其中引用标签 `\ref{ex:2}` 由宏包自动生成）；若使用了 `nonum` 选项，则为无编号过渡语“以下是该代码编译后得到的结果。”。单个示例也可用 `transition` 选项另行指定过渡文字。

3.3 自定义过渡文字

使用 `transition` 选项可以为单个示例自定义代码框与结果框之间的过渡文字（它会覆盖 `textlang` 所选语种给出的默认过渡语）。如：

```

\begin{example}[
  vmode,
  pkg      = { tikz },
  title     = { 自定义过渡文字 },
  transition = { 下面是本例的排版结果，请核对代码与输出是否一致： },
]{}
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
  \draw (1, 1) -- (2, 0);
\end{tikzpicture}
\end{example}

```

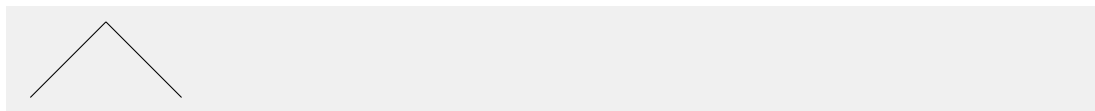
的排版结果为：

```

% 例 3 自定义过渡文字
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
  \draw (1, 1) -- (2, 0);
\end{tikzpicture}

```

下面是本例的排版结果，请核对代码与输出是否一致：



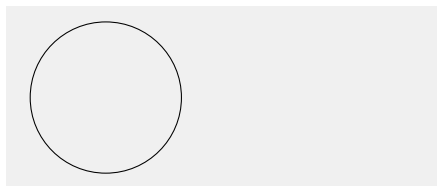
3.4 水平模式下的间距控制

水平模式下，`sep` 选项控制左代码框与右结果框之间的间距，默认 `18pt`，如：下面的示例把间距设置为 `0.2\textwidth`：

```
\begin{example}[
  pkg   = { tikz },
  sep   = { 0.2\textwidth },
  title = { 水平间距加大 },
]{ }
\begin{tikzpicture}
  \draw (0, 0) circle (1);
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
% 例 4 水平间距加大
\begin{tikzpicture}
  \draw (0, 0) circle (1);
\end{tikzpicture}
```



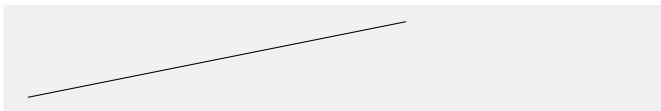
3.5 代码框宽度控制

`width` 选项控制代码框的宽度。未指定时，水平模式按左右均分，垂直模式取 `\textwidth`。下面的示例把代码框宽度设置为 `0.35\textwidth`：

```
\begin{example}[
  pkg   = { tikz },
  width = 0.35\textwidth,
  title = { 指定代码框宽度 },
]{ }
\begin{tikzpicture}
  \draw (0, 0) -- (5, 1);
\end{tikzpicture}
\end{example}
```

其排版结果为：

```
% 例 5 指定代码框宽度
\begin{tikzpicture}
  \draw (0, 0) -- (5, 1);
\end{tikzpicture}
```



`width` 与 `sep` 等长度选项应使用 L^AT_EX 的长度表达方式 (如 `0.35\textwidth`、`4cm`)。若写成 `width = 0.35` 这样的纯小数, T_EX 会补上默认单位 `pt`, 即被解释为 `0.35pt`, 代码框会被压缩成一条细线。

4 编号与引用

默认情况下, 每个 `example` 环境都会编号, 如有 `\chapter` 命令 (`book`, `report` 等文档类), 则编号格式为 “章号. 编号”, 如无 `\chapter` 命令 (`article` 等文档类), 则编号格式为 “编号”。并在代码框首行添加该编号的注释行:

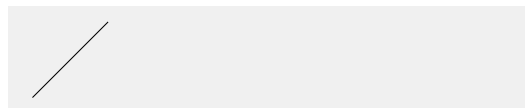
```
% 例 1.3 绘制直线
```

同时每个 `example` 环境都会自动生成引用标签(如 `\label{ex:1-3}`), 可用 `\ref{ex:1-3}` 引用, 显示为 “例 1.3”。若使用 `label` 选项自定义引用标签, 则 `\ref` 按自定义标签解析。如:

```
\begin{example}[
  pkg   = { tikz },
  label = { code:drawline-zh },
  title = { 绘制直线 },
]{ }
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为:

```
% 例 6 绘制直线
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
```



例 6 是用

```
label = { code:drawline-zh }
```

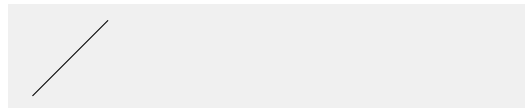
选项自定义引用标签后, 用 `\ref{code:drawline-zh}` 引用的效果。

若不需编号, 可使用 `nonum` (`nonum=true`) 选项进行设置: 此时不占用计数器, 也不生成标签, 但代码框与结果框仍正常排版。如:

```
\begin{example}[
  pkg = { tikz },
  nonum,
]{ }
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
```



5 仅显示代码

若只需代码框，不要结果框，可以使用 `noshow` (`noshow=true`) 选项。

```
\begin{example}[noshow]{}
...
\end{example}
```

此时代码框正常排版，结果框及其过渡文字均不输出，但不影响编号与引用标签。如：

```
\begin{example}[ noshow, vmode, pkg = { tikz } ]{}
\begin{tikzpicture}
  \draw (0, 0) cos (2, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
% 例 7
\begin{tikzpicture}
  \draw (0, 0) cos (2, 1);
\end{tikzpicture}
```

6 结果图片

若示例代码无法直接编译，则可用 `result` 选项指定图片文件，用图片代替编译结果：

注意：使用结果图片时，结果框不再使用指定的背景框，而是使用简单的单线外框进行包围后输出插图。

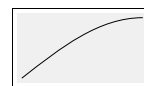
```
\begin{example}[ result = fig/sine.pdf, scale = 0.8 ]{}
...
\end{example}
```

`scale=0.8` 表示按原尺寸的 80% 插入；若需按宽度或高度插入，可改用 `wd` 或 `ht` 选项。如：

```
\begin{example}[
  pkg      = { tikz },
  result   = sine.pdf,
  scale    = 0.8,
]{}
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为:

```
% 例 8
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
```



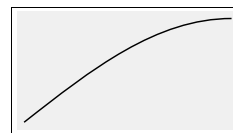
6.1 结果图片的宽度与高度控制

`wd` 用于把结果图片按指定宽度排版, 例如 `wd=0.2\textwidth`:

```
\begin{example}[
  pkg    = { tikz },
  result = sine.pdf,
  wd      = 0.2\textwidth,
]{ }
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为:

```
% 例 9
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
```

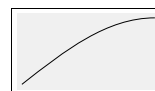


`ht` 用于把结果图片按指定高度排版, 例如 `ht=1cm`:

```
\begin{example}[
  pkg    = { tikz },
  result = sine.pdf,
  ht      = 1cm,
]{ }
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为:

```
% 例 10
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
```



注意: 当 `wd` 被指定后, `ht` 与 `scale` 都被忽略; 当只指定 `ht` 时, `scale` 被忽略。

7 输出前置内容

可以通过 `withcls`、`withpkg`、`withextra`、`nobody` 等逻辑选项的不同组合, 实现是否在示例代码框中输出文档类及其选项、宏包及其选项、导言区额外内容等。如果使用了

noshow 或 nobody 选项，则这些内容会原样输出；否则将会给 cls 和 pkg 的内容添加注释符号%后输出。

下面的示例综合使用了 withcls、withpkg、withextra 和 nobody 选项，可以看到代码框中原样输出了文档类、前置内容（导言区内容），但不输出环境体（nobody），也不输出编译结果：

```
\begin{example}[
  cls      = { article = { 12pt, a4paper } },
  pkg      = { tikz, xcolor = { svgnames } },
  title    = { 绘制直线 },
  pretitle = { 样例 },
  width    = { 0.8\textwidth },
  vmode,
  nonum,
  withcls,
  withpkg,
  withextra,
  nobody,
]{
\newcommand{\mya}{a}
\newcommand{\myb}{b}
}
\begin{tikzpicture}
  \draw (0, 0) node[below] { \mya } -- (1, 1) node[above] { \myb };
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
% 文档类
\documentclass[12pt,a4paper]{article}
% 导言区
\usepackage{tikz}
\usepackage[svgnames]{xcolor}
\newcommand{\mya}{a}
\newcommand{\myb}{b}
```

8 代码后端

codemwe 默认排版 L^AT_EX 示例代码：宏包把代码框中的内容写入临时文件 .aux 文件，并以\input命令将其读入结果框编译，从而得到结果。这一机制依赖于被排版的代码本身就是 L^AT_EX 代码。

为排版 C 语言等其他语言的代码，宏包引入了“代码后端”（backend）的概念，用 backend 选项选择。一个后端登记了以下四项信息：源文件扩展名、首行注释所用的注释符、该语言代码能否被 L^AT_EX 编译、以及该语言在 minted 宏包中对应的语言名。宏包内置了两个后端，如表 1 所示。

最后一列仅在使用 minted 选项（见第 12 节）时才会用到；不使用 minted 时该列内容不影响排版。

表 1: 内置的代码后端

后端	扩展名	注释符	能否编译	minted 语言名
tex	tex	%	是	tex
c	c	//	否	c

`backend = c` 时的排版行为如下:

1. 生成的源文件不是 L^AT_EX 文档, 而是可直接交给 C 编译器处理的纯 C 源文件: 首行为注释 (如// 例 2.2 两数之和), 其后为环境必选参数给出的前置内容与代码框中的代码;
2. 代码框中的注释行使用 C 语言的注释符//, 且不会添加“文档类”“导言区”“正文区”等 L^AT_EX 专有的结构注释;
3. `cls`、`pkg` 及 `withcls`、`withpkg` 等选项对 C 代码没有意义, 因此被忽略;
4. 由于 C 代码不能被 L^AT_EX 编译, 宏包不在排版过程中编译它。若给出了 `result` 选项, 结果框显示指定的图片; 若未给出, 则结果框整体省略, 并给出 `backend-no-result` 警告。

如, 可以用以下代码排版一个 C 函数, 并以插图作为结果。

```
\begin{example}[
  backend = c,
  title   = { 两数之和 },
  result  = { csum.png },
  scale   = 0.3,
  filename = { sum.c },
]{
#include <stdio.h>

int sum(int a, int b);

int main(void)
{
    int a, b;
    a = 3;
    b = 4;

    printf("sum=%d\n", sum(a, b));

    return 0;
}
int sum(int a, int b)
{
    return a + b;
}
\end{example}
```

其排版结果为:

```
// 例 11 两数之和
int sum(int a, int b)
{
    return a + b;
}
```



其中, `filename = { sum.c }` 使源文件保存为 `sum.c`, 其内容为:

```
// 例 11 两数之和
#include <stdio.h>

int sum(int a, int b);

int main(void)
{
    int a, b;
    a = 3;
    b = 4;

    printf("sum=%d\n", sum(a, b));

    return 0;
}

int sum(int a, int b)
{
    return a + b;
}
```

若不关心源文件的名称, 也可省略 `filename`, 此时文件名由 `dir`、`prelabel` 与编号自动构造, 扩展名取当前后端的扩展名 (C 语言为 `c`), 例如 `codes/ex11.c`。

注意: 由于 C 语言代码不在排版时编译, 结果框中的插图必须事先准备好, 并与代码保持同步更新。若未指定 `result` 选项, 可用 `noshow` 选项关闭结果框, 以免出现警告; `ccode` 环境已自动设置了 `noshow`。

需要排版 C 语言之外的语言 (如 Python) 时, 只需用 `\codemweNewBackend` 注册一个新后端, 再通过 `backend` 选项使用它, 无需修改宏包本身。

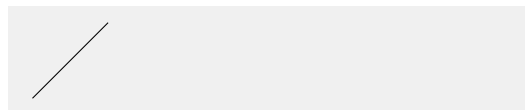
9 显示行号

可以通过使用 `[linenos]` 选项为代码框添加行号, 如:

```
\begin{example}[
    pkg    = { tikz },
    title  = { 绘制直线 },
    linenos,
]{
\begin{tikzpicture}
    \draw (0, 0) -- (1, 1);
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
1 % 例 12 绘制直线
2 \begin{tikzpicture}
3   \draw (0, 0) -- (1, 1);
4 \end{tikzpicture}
```



注意：通过 withcls、withpkg、withextra 添加的内容及自动添加的注释等内容同样会计入行号。

10 完整示例

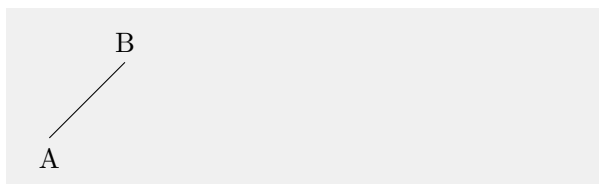
下面给出一个综合运用多种选项的完整示例。它使用 article 文档类、tikz 与 xcolor 宏包，在导言区追加两个自定义命令，并采用垂直模式、指定代码框宽度与自定义标题。如：

```
\begin{example}[
  cls      = { article = { 12pt, a4paper } },
  pkg      = { tikz, xcolor = { svgnames } },
  title    = { 绘制直线 },
  label    = { code:multi-opts-zh },
  pretitle = { 示例 },
  width    = { 0.55\textwidth },
  vmode,
  withextra,
]{
\newcommand{\myA}{A}
\newcommand{\myB}{B}
}
\begin{tikzpicture}
  \draw (0, 0) node[below] { \myA } -- (1, 1)
    node[above] { \myB };
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
% 示例 13 绘制直线
% 导言区
\newcommand{\myA}{A}
\newcommand{\myB}{B}
% 正文区
\begin{tikzpicture}
  \draw (0, 0) node[below] { \myA } -- (1, 1)
    node[above] { \myB };
\end{tikzpicture}
```

以下是示例 13 编译后得到的结果。



示例 13 生成 codes/ex11.tex 文件的 MWE tex 文件，其内容为：

```
% 文档类
\documentclass[12pt,a4paper]{article}

% 导言区
\usepackage{tikz}
\usepackage[svgnames]{xcolor}

\newcommand{\myA}{A}
\newcommand{\myB}{B}

% 正文区
\begin{document}

\begin{tikzpicture}
  \draw (0, 0) node[below] { \myA } -- (1, 1) node[above] { \myB };
\end{tikzpicture}

\end{document}
```

注意：如果需要在 title 选项设置的标题内容中包含类似\tikzset这样的命令，则需要使用 L^AT_EX 的原命令\string实现，如\string\tikzset用于输出\tikzset，以避免在写入注释时被展开。

11 将环境体保存为文件

如果仅需要将环境体保存到指定文件中,可以同时指定 [nonum, filename=test.tex] 两个选项。此时对示例代码不进行编号，但会将环境体保存在 filename 选项指定的文件中，路径前缀仍使用 dir 选项指定的路径。

注意：此时保存的文件不包含文档类、导言区、表示正文区开始的命令 \begin{document} 和表示正文区结束的命令 \end{document} 等内容，该文件无法独立进行编译。

如：

```
\begin{example}[
  vmode,
  noshow,
  nonum,
  title    = { 定义用户接口 },
  filename = test.tex,
]{}
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { 猫儿喜欢鱼 }
\newcommand{\mytest}{ % 定义用户接口
  \tl_use:N \l__test_tl
}
\ExplSyntaxOff
\end{example}
```

的排版结果为：

```
% Test.tex 定义用户接口
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { 猫儿喜欢鱼 }
\newcommand{\mytest}{ % 定义用户接口
  \tl_use:N \l__test_tl
}
\ExplSyntaxOff
```

12 语法高亮

默认情况下，代码框中的代码按原样排版，不做语法高亮。若对代码的可读性有更高要求，可以为宏包或环境设置 `minted` 选项，改用 `minted` 宏包完成示例代码排版，从而得到带语法高亮的示例代码：

```
\usepackage[minted]{codemwe}
```

未指定 `minted` 选项时，宏包使用 `fancyvrb` 宏包排版示例代码；指定该选项后，代码框中的内容改由 `minted` 排版，并统一采用 `\small` 字号、自动去除公共缩进 (`autogobble`)。行号由 `linenos` 选项控制，与不使用 `minted` 时一致。

注意：`minted` 需要调用外部程序 (`latexminted` 或 `pygmentize`) 完成高亮，因此必须以 `-shell-escape` 选项编译文档。若未开启 `shell escape` 或未安装 `minted`，宏包不会中止编译，而会发出一条 `minted-unavailable` 警告并回退到 `fancyvrb` 宏包排版方案，此时文档仍可正常生成，只是没有语法高亮。

此外，`minted` 默认采用 `fasthighlightmode`：首遍编译时只在代码框中插入占位符，从第二遍起才显示出真正的高亮结果。因此启用 `minted` 选项后需要编译两遍以上。

高亮所需的语言名不由 `minted` 选项给出，而是由 `backend` 选项决定：每个代码后端都登记了对应的 `minted` 语言名（见表 1），排版时宏包自动取当前后端的语言名交给 `minted`。例如 `backend = c` 时用 `c` 语法高亮，`backend = tex` 时用 `tex` 语法高亮。

高亮样式由 `mintedstyle` 选项设置，取值为 `minted` (`pygments`) 支持的样式名，默认为 `default`。如改用 `tango` 样式：

```
\usepackage[minted, mintedstyle = tango]{codemwe}
```

可供选择的常用样式有 `default`、`friendly`、`tango`、`monokai`、`bw` 等，完整列表见 `pygments` 的样式文档。若只设置 `mintedstyle` 而未启用 `minted`，宏包会发出 `mintedstyle-unused` 警告，但仍会记录该取值，以便日后启用 `minted` 时生效。

`minted` 与 `mintedstyle` 都可以作为宏包选项，也可以在 `\codemweset` 或具体环境中给出，用法与其他选项相同，如：

```

\begin{example}[
  minted,
  backend      = c,
  mintedstyle = friendly,
  result       = chelloworld.png,
  wd           = 0.45\textwidth,
  title        = { 语法高亮示例 },
]{}
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
\end{example}

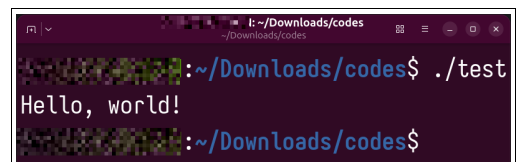
```

的排版结果为：

```

// 例 14 语法高亮示例
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}

```



12.1 在正文中开关语法高亮

`minted` 选项可以像其他布尔选项一样在正文中开关，从而只对个别示例启用高亮：

```

\codemwreset{minted}          % 打开：其后所有示例都做语法高亮
\begin{example}{}
...
\end{example}
\begin{example}[minted = false]{}
...
\end{example}
\codemwreset{minted = false} % 关闭：恢复原有排版方案

```

注意： `minted` 宏包只能在导言区加载。本宏包在被读取时就已经检查系统是否装有 `minted`、编译是否用了 `-shell-escape`，条件满足便直接加载，因此正文中的 `minted`、`minted = false` 随时可用。`minted = false` 只是把高亮关掉，并不会卸载 `minted` 宏包，因此之后再打开无需重新加载。

若缺 `minted` 宏包或未用 `-shell-escape` 编译，加载便不会发生，此时一旦启用 `minted` 选项，会发出 `minted-unavailable` 警告，代码按原有方案排版，不加语法高亮。

注意： 结果框的用途是把示例代码真正编译运行一遍并显示其效果，因此结果框中的代码始终按原样读入编译，不做语法高亮；只有代码框中的代码才会被 `minted` 高亮。对 `c` 这类不能被 `LATEX` 编译的后端，结果框本就只在给出 `result` 选项时才显示，与此无关。

13 宏包选项

宏包支持通过`\usepackage[选项]{<codemwe>}` 传入全局默认选项，这些选项会作为后续 `example`、`example*`、`texcode`、`ccode` 环境的初值。可作为宏包选项传入的键有：`vmode`、`nobody`、`noshow`、`nonum`、`withcls`、`withpkg`、`withextra`、`cls`、`pkg`、`width`、`sep`、`prelabel`、`pretitle`、`dir`、`counter`、`boxtype`、`textlang`、`backend`、`linenos`、`minted` 和 `mintedstyle`。

宏包级选项设置的初值，可以在环境内部通过同名选项覆盖。如：

```
\begin{example}[
  vmode = false,
  pkg    = { tikz },
  title  = { 覆盖宏包级 vmode },
]{}
```

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
\end{example}
```

的排版结果为：

```
% 例 15 覆盖宏包级 vmode
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
```



14 未知选项与警告

若传入未定义的键，宏包会发出一条 `unknown-key` 警告，并忽略该选项。例如编译下列示例时：

```
\begin{example}[foo, pkg={tikz}]{}
```

```
...
```

```
\end{example}
```

会得到：

```
Package codemwe Warning:\
Unknown key-val option 'foo' for package 'codemwe'.
```

这样的警告，但并不会中止编译。

此外，与代码后端相关的两条警告为：

- `unknown-backend`: `backend` 选项给定了未注册的后端名称，宏包回退到 `tex`；
- `backend-no-result`: 当前后端不可被 `LATEX` 编译，又未给出 `result` 选项，结果框被整体省略。补充 `result` 插图或改用 `noshow` 选项即可消除该警告。

与语法高亮相关的三条警告为：

- `minted-unavailable`: 启用了 `minted` 选项，但当前编译未开启 `shell escape`，或系统中没有 `minted` 宏包。此时宏包回退到原有排版方案，代码仍可正常排出，只是没有高亮；
- `mintedstyle-unused`: 设置了 `mintedstyle` 却没有启用 `minted` 选项，该样式暂不生效；
- `minted-no-lexer`、`minted-unknown-lexer`: 当前后端没有登记 `minted` 语言名，或者登记的语言名不被 `pygments` 识别。

15 盒子类型

`boxtype` 选项用于指定代码框和结果框所使用的盒子环境，其默认值为 `syntaxbox`。宏包内置了 6 个盒子环境，详见表 2。其中带星号的三个盒子是各自不带星号盒子的“白底无边框”变体；而 `mwebox/mwebox*` 与 `mweparbox/mweparbox*` 支持标题栏，示例编号与标题会自动成为盒子标题栏的内容，`syntaxbox/syntaxbox*` 不支持标题栏，此时编号与标题以首行注释的形式写在代码框内。

表 2: 内置盒子类型

类型	背景色	边框	标题栏	宏包依赖
<code>syntaxbox</code>	<code>gray!12</code>	无	无	<code>tcolorbox</code>
<code>syntaxbox*</code>	<code>white</code>	无	无	<code>tcolorbox</code>
<code>mwebox</code>	<code>cyan</code>	无	有	<code>tcolorbox</code>
<code>mwebox*</code>	<code>cyan</code>	有	有	<code>tcolorbox</code>
<code>mweparbox</code>	<code>purple</code>	左竖边框	有	<code>tcolorbox</code>
<code>mweparbox*</code>	<code>white</code>	左竖边框	有	<code>tcolorbox</code>

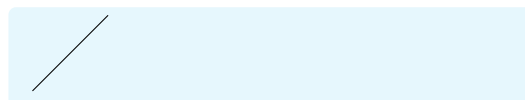
如：

```
\begin{example}[ boxtype = mwebox, label = { code:mwebox-zh }, pkg = { tikz } ]{}
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
\end{example}
```

例 16 是采用 `mwebox` 盒子排版的结果。

例 16

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
```



可以使用 `\codemweNewBox` 命令注册用户自定义的盒子。要求自定义盒子环境的参数接口符合“`0{背景色} d() 0{额外选项}`”的要求，即第一个可选参数 `[]` 给出背景色、第二个可选参数 `()` 给出标题、第三个可选参数 `[]` 给出额外的 `tcolorbox` 选项，就能被 `boxtype` 使用。例如，可以用 `tcolorbox` 宏包的 `\DeclareTColorBox` 命令定义这样一个盒子：

```
% 定义 tcolorbox 盒子（接口与内置盒子一致）
\DeclareTColorBox{ mybox }{ 0{ green!30 } d() 0{ } }
{
  colback = #1,
  colframe = #1,
  boxrule = 0pt,
  arc = 4mm,
  left = 8pt,
  right = 8pt,
  top = 4pt,
  bottom = 4pt,
  nobeforeafter,
  IfValueT = { #2 } { title = #2, coltitle = green!30!black },
  #3,
}
```

因为该盒子会使用 () 参数生成标题栏，所以注册时应声明 `withtitle = true`（这也是其默认值）。

```
% 注册排版盒子
\codemweNewBox{ mybox }[ withtitle = true ]
```

注意：若自定义盒子支持标题栏，`\codemweNewBox`的 `withtitle` 必须为 `true`；反之若盒子不支持标题栏，则应声明 `withtitle = false`。若声明与实际不符，编号与标题有可能被丢失或重复输出。

下面用 `mybox` 排版一个示例（为便于阅读，这里直接用 `\DeclareTColorBox` 命令定义盒子，效果与上面 `\codemweNewBox` 注册后完全一致）：

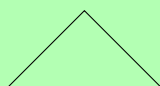
如：

```
\begin{example}[
  pkg = { tikz },
  title = { 自定义盒子 },
  boxtype = mybox,
]{ }
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
\end{example}
```

的排版结果为：

例 17 自定义盒子

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
```



16 选项说明

以下是 `codemwe` 宏包设置的全部 key-value 选项。其中“初始值”是选项在宏包或环境未作任何设置时的取值，“默认值”是该布尔选项写成不带值的简写形式（如 `vmode`）时

的取值。

`vmode = <true/false>` (init: false)

排版模式。`vmode = false` 为水平模式, `vmode = true` 为垂直模式; 布尔值 = `true` 可省略, 仅写 `vmode` 即表示 `vmode = true`。初始值为 `false`。

`noshow = <true/false>` (init: false)

是否显示结果框。使用 `noshow` 则仅显示代码框, 结果框及其过渡文字均不输出。初始值为 `false`。

`nonum = <true/false>` (init: false)

是否编号并生成标签。使用 `nonum` 关闭编号, 此时不占用计数器, 也不生成标签, 但代码框与结果框仍正常排版。初始值为 `false`。

`nobody = <true/false>` (init: false)

是否输出 `example` 环境体。使用 `nobody` 时不输出环境体, 同时也就不输出编译结果 (相当于同时使用了 `noshow` 选项)。初始值为 `false`。

`withcls = <true/false>` (init: false)

在示例代码框中是否加入 `cls` 列出的文档类内容。如果使用了 `noshow` 或 `nobody` 选项, 则把 `cls` 的内容原样输出; 否则在其前面加上注释符号%后输出。初始值为 `false`。

`withpkg = <true/false>` (init: false)

在示例代码框中是否加入 `pkg` 列出的宏包内容。如果使用了 `noshow` 或 `nobody` 选项, 则把 `pkg` 的内容原样输出; 否则为其中的每个宏包加上注释符号%后输出。初始值为 `false`。

`withextra = <true/false>` (init: false)

在示例代码框中是否加入 `example` 必选参数给出的导言区额外内容。初始值为 `false`。

`cls = <文档类>` (init: ctexart)

MWE `tex` 文件所需的文档类。既支持不带选项的 `cls = { article }` 形式, 也支持通过 = 指定文档类选项的 `cls = { article = { 12pt, a4paper } }` 形式, 后者会在 MWE `tex` 文件开头生成 `\documentclass[12pt,a4paper]{article}`。初始值为 `ctexart`。

`pkg = <宏包列表>` (init: none)

MWE `tex` 文件所需的宏包列表。与 `cls` 类似, 用逗号列表的方式给出需要添加的宏包, 如 `pkg = { tikz, tabularray, listings = { numbers = left } }`。初始值为空, 即不添加任何宏包。

`width = <dim>` (init: 0pt)

代码框宽度。初始值为 0pt，表示“未指定”：此时水平模式在扣除间距 `sep` 后按左右均分，垂直模式取 `\textwidth`。在水平模式下，若该值过大导致右结果框宽度为负，会给出 `width-too-large` 警告。该值应写成 L^AT_EX 长度，如 `width = 0.35\textwidth`。

`sep = <dim>` (init: 18pt)

水平模式下，左代码框与右结果框的间距，初始值为 18pt。该值应写成 L^AT_EX 长度，如 `sep = 0.1\textwidth`。

`prelabel = <前缀>` (init: ex)

MWE 文件名与自动引用标签的前缀，初始值为 `ex`。例如第 `ex:1` 个示例的 `label` 默认为 `ex:1`，对应的 MWE 文件名为 `ex1.tex`。

`pretitle = <编号前缀>` (init: example)

示例编号的引导词，用于代码框首行注释与 `\ref` 的显示内容，初始值为 `example`。例如 `pretitle = Example` 时，首行注释为“Example 1.1”，`\ref` 也显示为“Example 1.1”。该值可含换行，换行会被转换为空格。中文排版时可设置为“例”、“示例”等内容。

`dir = <目录>` (init: codes)

MWE 文件的保存目录，默认为当前目录下的 `codes` 文件夹。**指定的路径必须已存在。**

`counter = <计数器名>` (init: example)

计数器名称，初始值为 `example`。若指定的计数器名已被占用，会给出 `counter-exists` 警告并忽略该选项。由于在 `\AtBeginDocument` 时按该名称创建计数器，因此通常不需修改此项。

`boxtype = <盒子环境名>` (init: syntaxbox)

指定代码框和结果框所使用的盒子环境，初始值为 `syntaxbox`。内置盒子见表 2。若指定的盒子未被注册，会给出 `unknown-boxtype` 警告并回退到 `syntaxbox`。

`textlang = <语种名称>` (init: en)

选择语种。`codemwe` 宏包按语种给出默认过渡语、MWE 文件中“文档类/导言区/正文区”三个结构注释词，以及 `cls` 与 `pretitle` 的默认取值，`textlang` 选项用于选择使用哪一套。其取值为语种名称，初始值为 `en`（英文）。

宏包内置了 `en`（英文）和 `zh`（中文）两个语种。若指定的语种未注册，会给出 `unknown-textlang` 警告并回退到 `en`。切换语种通常在导言区或文档开头用 `\codemweset` 完成，如 `\codemweset{textlang = zh}`。单次示例仍可用 `transition` 选项单独覆盖过渡语。

注册新语种用 `\codemweNewLang` 命令，见第 2 节。

backend = $\langle$ 后端名称 $\rangle$ (init: tex)

选择代码后端，即被排版代码所属的语言。后端决定生成源文件的扩展名、首行注释所用的注释符，以及该语言代码能否被 L^AT_EX 编译。其取值为后端名称，初始值为 **tex** (L^AT_EX 示例代码)。

宏包内置了 **tex** 和 **c** (C 语言) 两种后端。选 **c** 时，宏包生成 **c** 源文件，代码框首行注释改用 **//**，且不编译代码：若给出了 **result** 选项则显示该图片，否则省略结果框并给出 **backend-no-result** 警告。若指定的后端未注册，会给出 **unknown-backend** 警告并回退到 **tex**。

注册新后端用 `\codemweNewBackend` 命令，见第 2 节；后端机制的详细说明见第 8 节。

minted = $\langle$ true/false $\rangle$ (init: false)

是否改用 **minted** 宏包排版代码框，从而获得语法高亮。使用 **minted** 时，代码框中的代码由 **minted** 按当前后端登记的语言名高亮，并统一采用 `\small` 字号、自动去除公共缩进 (`autogobble`)；行号仍由 **linenos** 选项控制。初始值为 **false**，即沿用原有的 **fancyvrb** 排版方案，不做语法高亮。

作为宏包选项给出时，全文所有示例都做语法高亮；作为环境选项给出时，只对当前示例启用，环境结束即恢复；用 `\codemweset` 给出时，对其后的所有示例生效，直到再次用 `\codemweset` 改回。因此 **minted = false** 既可关闭宏包级的高亮，也可在正文中随时关掉某一处的高亮。

该选项需要 **minted** 宏包并以 `-shell-escape` 编译文档。本宏包在加载时即检查这两个条件，满足便直接加载 **minted**，所以选项在任何位置（导言区、正文、环境）都能立即生效，无需先在导言区预置一次。若条件不满足，宏包会给出 **minted-unavailable** 警告并自动回退到原有方案，不会中止编译。此外，**minted** 默认首遍只插入占位符，故启用本选项后需编译两遍以上才能看到高亮结果。

详细说明见第 12 节。

mintedstyle = $\langle$ 样式名 $\rangle$ (init: default)

minted 的语法高亮样式名，即 **pygments** 的 **style** 名，初始值为 **default**。常用取值有 **tango**、**friendly**、**monokai**、**bw** 等。

该选项只在启用 **minted** 时起作用。若设置了本选项而未启用 **minted**，宏包会给出 **mintedstyle-unused** 警告，但仍记录该取值，以备日后启用 **minted** 时生效。

title = $\langle$ 标题 $\rangle$ (init: none)

示例的功能标题，如“绘制直线”，初始值为空。该标题在代码框支持标题栏时成为盒子的标题栏，否则成为代码框首行注释的一部分。若需要在 **title** 中包含类似 `\tikzset` 这样的命令，则需要使用 `\string` 原语，即 `\string\tikzset`。

label = $\langle$ 标签名 $\rangle$ (init: none)

该示例的交叉引用标签。未指定时自动生成 $\langle$ prelabel $\rangle$: $\langle$ 章号-编号 $\rangle$ （支持 `\chapter` 的文档类，如 **ex:8-1**）或 $\langle$ prelabel $\rangle$: $\langle$ 编号 $\rangle$ （不支持 `\chapter` 的文档类，如 **ex:1**）。

`filename = <文件名> (init: none)`

MWE 文件名。未指定时自动生成 `<prelabel><章号-编号>.tex` (支持`\chapter`的文档类, 如 `ex8-1.tex`) 或 `<prelabel><编号>.tex` (不支持`\chapter`的文档类, 如 `ex1.tex`)。若同时使用 `nonum` 选项, 则只把环境体保存到该文件中。

`result = <图片文件名> (init: none)`

结果图片文件名。指定后, 用该图片代替示例代码的编译结果, 不再编译示例代码。文件名中可以包含路径 (路径分隔符用`/`)。若文件不存在, 会给出 `result-file-missing` 警告, 但不会中止编译。

`wd = <dim> (init: 0pt)`

结果图片宽度。初始值为 `0pt`, 表示“未指定”。始终优先于 `ht` 与 `scale`: 只要指定了 `wd`, 这两个选项都被忽略。

`ht = <dim> (init: 0pt)`

结果图片高度。初始值为 `0pt`, 表示“未指定”。优先于 `scale`: 在未指定 `wd` 而指定了 `ht` 时, `scale` 被忽略。

`scale = <比例> (init: 1)`

结果图片的缩放比例, 初始值为 `1`。只有在 `wd` 和 `ht` 均未指定时有效。

`transition = <过渡文字> (init: none)`

为当前这一个示例单独指定垂直模式下代码框与结果框之间的过渡文字, 初始值为空。留空时改用 `textlang` 所选语种给出的全局默认过渡语。它可以含`\ref`等命令, 例如 `transition = { 下面是 \tn{ref}{ ex:1 } 的结果: }`。

`linenos = <true/false> (init: false)`

是否在代码框左侧显示行号。使用 `linenos` 时, 代码框的每一行前会加上行号, 行号与代码之间留有 `6pt` 的间距; 不加该选项时不显示行号。初始值为 `false`。

行号由 `fancyvrb` 宏包的`\VerbatimInput`命令的 `numbers=left` 选项实现, 编号从代码框的第一行起算, 与示例自身的编号无关; 使用 `withcls` 或 `withpkg` 时, 被加入的文档类与宏包内容同样计入行号。

注意: `textlang` 决定整篇文档使用的默认过渡语及各语种相关文本(可用`\codemwreset`或宏包选项设置, 适合切换文档语种), `transition` 为单个示例的局部覆盖。可根据需要进行设置。

17 注意事项

1. 只有在未使用 `noshow` 和 `nobody` 选项时, 代码框中的 `cls` 与 `pkg` 内容才会被自动添加注释符号`%`。

2. 水平模式下, 若 `width` 过大, 使右结果框宽度为负, 会给出 `width-too-large` 警告, 此时应减小 `width` 或 `sep`。
3. `boxtype` 指定的盒子必须已注册 (内置或通过 `\codemweNewBox` 注册), 否则会给出 `unknown-boxtype` 警告并回退到 `syntaxbox`。
4. `counter` 选项指定的计数器若已存在, 会给出 `counter-exists` 警告并忽略该选项。
5. 若 `expl3`、`tcolorbox` 等宏包版本过旧, 会给出 `package-old` 警告, 建议把 `TEX` 发行版更新到最新版。
6. `title`、`label`、`transition`、`boxtype` 与 `pretitle` 的值中若含换行, 宏包会把换行转换为空格, 并去掉首尾空白。
7. 结果框在插入 `result` 图片时使用 `\includegraphics`, 因此图片需由 `graphicx` 支持的格式提供 (如 `pdf`、`png`、`jpg`); 用 `XYTEX` 编译时 `eps` 不被支持。

18 实现简介

`codemwe` 的核心思路是将 `example` 环境体写入临时文件 (`jobname-example.aux`), 再通过 `\verbatiminput` 读入该文件, 从而排版出代码框; 结果框则把该临时文件通过 `\input` 置于盒中现场排版, 或插入 `result` 选项指定的图片。MWE 文件按指定后端语言结构拼装后写入 `dir` 指定的目录, 其文件名由 `prelabel`、`counter` (或 `filename` 选项) 决定。

宏包内部用 `Expl3` 的 `l3keys` 机制管理选项: 宏包加载时通过 `\ProcessKeyOptions` 把 `codemwe / pkgopts` 一组键转发到 `codemwe` 键组, `example`、`example*` 与 `texcode` 环境则在各自的组内用 `\keys_set:nn` 设置选项, 并由环境前预置的键 (如 `example*` 的 `vmode = true`) 提供默认值。由于环境体在 `\group_begin:` 与 `\group_end:` 之间处理, 各示例的选项互不影响。语种相关的文本 (默认过渡语、结构注释词等) 集中存放在一张 “语种 → 文本” 的属性表中, 由 `textlang` 选项选用, `\codemweNewLang` 用于向该表登记新语种。更底层的实现细节可直接查阅 `codemwe.sty` 源码。

19 常见问题

为什么结果框是空的?

若未使用 `result` 选项, 结果框会通过 `\input` 编译代码临时文件得到; 请确认示例代码本身可编译。若使用了 `result` 选项, 请确认图片文件存在且路径正确。

为什么代码框中的宏包名被加了注释符?

这是 `withpkg` (以及 `withcls`) 的默认行为: 当输出编译结果时, 文档类与宏包内容会被加上注释符号%, 仅作为说明之用。若希望原样输出, 可配合 `noshow` 或 `nobody` 选项。

如何把示例代码单独保存成一个文件？

同时使用 `nonum` 和 `filename` 选项，即可只把环境体保存到指定文件中（路径前缀仍取 `dir` 选项）。注意该文件不含文档类与导言区，也不含 `\begin{document}` 与 `\end{document}`，不能独立编译。

为什么 MWE 文件名与我在选项中写的 `label` 不一致？

MWE 文件名只由 `prelabel`、`counter` 与 `filename` 决定，与 `label` 选项无关。`label` 只改变交叉引用标签，不改变文件名。

设置了标题，却看不到标题栏？

标题栏是否出现取决于当前盒子类型。`syntaxbox` 与 `syntaxbox*` 不支持标题栏，此时编号与标题会作为首行注释写在代码框内容里；若要看到独立的标题栏，请改用 `mwebox`、`mwebox*`、`mweparbox` 或 `mweparbox*`。

为什么结果框里没有 C 语言的运行结果？

C 语言代码不能在 $\text{\LaTeX}$ 中编译，宏包不会运行它，结果框只显示 `result` 选项指定的图片。若希望结果框中出现真实输出，需要先用 C 编译器运行生成的 `c` 文件，再把输出（或截图）保存为图片，通过 `result` 选项插入。参见第 8 节。

The `codemwe` Package: Typesetting Code with a Result Box and Generating MWE Files $\Rightarrow$ 中文版本

Nan Geng <nangeng@nwafu.edu.cn>

2026 年 10 月 24 日

20 Introduction

`codemwe` is a code-typesetting package built on the `tcolorbox` package and developed with `Expl3`. It typesets an example code box together with its compiled result box, and automatically generates an independently compilable MWE (Minimal Working Example) file. The example code is placed in a code box and its compiled result in a result box; the two boxes can be arranged horizontally or vertically, which is convenient for comparing the code with its output.

The package provides four environments: `example`, `example*`, `texcode` and `ccode`. The `example` environment uses the horizontal mode by default, i.e. the code box on the left and the result box on the right; the `example*` environment is a compatibility spelling of the vertical mode, i.e. the code box on the top and the result box at the bottom; the `texcode` environment typesets a code box only, without numbering or compilation, and is suitable for code fragments that need not be compiled; the `ccode` environment typesets C code, showing a code box only and generating a `c` source file.

In addition, the package provides the `\codemweset` macro for setting global default options, the `\codemweNewBox` macro for registering user-defined boxes, the `\codemweNewLang` macro for registering user-defined language varieties for transition phrases and comment terms, and the `\codemweNewBackend` macro for registering code backends other than C.

Besides this overview, the following sections discuss the user interfaces (Section 21), the typesetting modes (Section 22), numbering and references, result figures, the output of the preliminary content, code backends (Section 27), line numbers, package-level options, unknown options and warnings, box types, the complete option list (Section 35), notes, an implementation overview and frequently asked questions. For a quick start, read Sections 21 and 22 first, and consult Section 35 when needed.

NOTE: The default language variety of the package is English (`textlang = en`). For documents written in another language it can be switched with the `textlang` option (see Section 35), as is done in the Chinese part of this manual. The `textlang` option also selects the wording of the structural comments “document class / preamble / body” written into the MWE `tex` file.

*https://gitee.com/nwafu_nan/codemwe

21 User Interfaces

21.1 The `example` Environment

```
\begin{example} \begin{<example>}[<options>]{<preliminary content>}
\end{example}  <example code>
\end{example}
```

New: 2026-10-10

Typesets the example code and its compiled result, in the horizontal mode by default (code box on the left, result box on the right), and generates an independently compilable MWE file.

The environment takes one optional argument [*<options>*] and one **mandatory** argument {<preliminary content>}.

The [*<options>*] is a comma-separated key-value list. Some options are boolean and accept only **true** or **false**; for these options the “= **true**” part of {<option>} = **true** may be omitted.

{<preliminary content>} is written into the preliminary part of the MWE file (e.g. the preamble of a `LATEX` file) and may contain `\newcommand`, `\tikzset`, `#include <stdio.h>` and similar material. If no preliminary content is needed, an empty pair of braces {} must still be given; the empty pair of braces is **mandatory**.

Each `example` environment is numbered by default. In document classes with a `\chapter` command (`book`, `report`, etc.) the number looks like “chapter.number” and a reference label such as `ex:1-1` is generated automatically; in classes without a `\chapter` command (e.g. `article`) the number is a plain number and the label looks like `ex:1`. For instance:

```
\begin{example}{ }
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { cat~likes~fish }
\tl_use:N \l__test_tl
\ExplSyntaxOff
\end{example}
```

21.2 The `example*` Environment

```
\begin{example*} \begin{<example*>}[<options>]{<preliminary content>}
\end{example*}  <example code>
\end{example*}
```

New: 2026-10-10

Typesets the example code and its compiled result in the vertical mode (code box on the top, result box at the bottom). It is equivalent to giving the `example` environment the `vmode = true` option; for instance, the [`pkg = {tikz}`] option of `example*` corresponds to [`vmode, pkg = {tikz}`] of `example`. For instance:

```

\begin{example*}[pkg={tikz}]{}
\begin{tikzpicture}
  \draw (0, 0) -- (1, 0) -- (1, 1);
\end{tikzpicture}
\end{example*}

```

21.3 The `texcode` Environment

```

\begin{texcode} \begin{<texcode>}[<options>]
\end{texcode}   <code fragment>
\end{<texcode>}

```

New: 2026-10-10

Typesets a code box only. The content is not numbered, no reference label is generated, the example code is not compiled and no MWE `tex` file is generated. It is suitable for inserting code fragments that need not be compiled.

The [*<options>*] are the same as those of the `example` environment.

The `texcode` environment is in fact implemented by presetting the [`vmode`, `nonum`, `noshow`] options of `example`. It is equivalent to

```

\begin{example}[vmode, nonum, noshow]{<...>}
\end{example}

```

```

\begin{texcode}
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { cat~likes~fish }
\tl_use:N \l__test_tl
\ExplSyntaxOff
\end{texcode}

```

21.4 The `ccode` Environment

```

\begin{ccode} \begin{<ccode>}[<options>]
\end{ccode}   <C code fragment>
\end{<ccode>}

```

New: 2026-10-20

Typesets C code: it shows a code box only, without numbering or a result box, and generates a source file in the C format (see Section 27). It is suitable for inserting C code fragments or function definitions.

The [*<options>*] are the same as those of the `example` environment, except that `backend` has already been preset to `c` and normally need not be given again.

The `ccode` environment is in fact implemented by presetting the options `vmode`, `nonum`, `noshow` and `backend = c` for the `example` environment.

```

\begin{ccode}[filename = { hello.c }]{ }
#include <stdio.h>

int main(void)
{
    printf("Hello, World!\n");
    return 0;
}
\end{ccode}

```

NOTE: The body of the `ccode` environment is written verbatim to the `c` file, so $\LaTeX$ -specific options such as `cls` and `pkg` are meaningless for it. If the code needs preliminary content such as `#include` lines, they may be written directly in the environment body or given in the mandatory argument (extra preamble) of the environment; both end up in the generated source file, but the latter is not shown in the code box unless the `withextra` option is used as well.

21.5 `\codemweset` for Global Settings

`\codemweset` `\codemweset{<options>}`

New: 2026-10-10

Modifies the default options of the following `example`, `example*`, `texcode` and `ccode` environments in the current scope. The `{<options>}` is a comma-separated key-value list. For instance:

```
\codemweset{ vmode, withpkg }
```

21.6 `\codemweNewBox` for Registering New Boxes

`\codemweNewBox` `\codemweNewBox{<box environment name>}[<options>]`

New: 2026-10-10

Registers a new box environment with `codemwe`; once registered, the box can be used for the code box and the result box through the `boxtype` option.

`{<box environment name>}` must obey the argument interface

```
O{background colour} d() O{extra options}
```

It is recommended to define the box with the `\DeclareTColorBox` command of `tcolorbox`. Currently only `withtitle` is supported in `[<options>]`, which declares whether the box supports a title bar;

```
\codemweNewBox{ mybox }[ withtitle = true ]
```

NOTE: Internally the package maintains a property list mapping a “box environment name” to “whether it supports a title bar”, in which the six built-in boxes are pre-registered. `\codemweNewBox` only records the properties; it does not define the environment itself. Hence one must first define $\{\langle box\ environment\ name\rangle\}$ with `\DeclareTColorBox` (or an equivalent means, such as `\NewDocumentEnvironment`) and then call this command to register it.

If a box with the same name as a built-in box environment is used, it will override the settings of the built-in box.

21.7 `\codemweNewLang` for Registering New Language Varieties

`\codemweNewLang` $\{\langle variety\ name\rangle\}\{\langle eight\ pieces\ of\ content\rangle\}$

New: 2026-10-13

Registers a new language variety with `codemwe`, supplying the transition phrases, comment words and other texts for that variety; once registered, the variety can be selected through the `textlang` option to typeset the transition text, the structural comments and so on.

$\{\langle variety\ name\rangle\}$ is the name of the new variety. $\{\langle eight\ pieces\ of\ content\rangle\}$ consists of eight comma-separated pieces of text (pre-transition text, post-transition text, unnumbered transition text, document-class comment word, preamble comment word, body comment word, default document class, leading word for the title and the `\ref` result). For example:

```
\codemweNewLang{ fr }
{
  Le-résultat-de,
  est-affiché-ci-dessous.,
  Le-résultat-du-code-est-affiché-ci-dessous.,
  classe-de-document,
  préambule,
  corps,
  article,
  exemple
}
```

The package ships with two built-in varieties: `en` (English) and `zh` (Chinese). If the variety name defined by `\codemweNewLang` duplicates a built-in name, the built-in text content is overwritten.

NOTE: A comma inside any of the $\{\langle eight\ pieces\ of\ content\rangle\}$ must be protected with braces, and an empty field must be left blank rather than omitted.

21.8 `\codemweNewBackend` for Registering New Code Backends

<code>\codemweNewBackend</code>	<code>\codemweNewBackend{<backend name>}{<file extension>}{<comment character>}{<compilable>}</code> <code>[<minted language name>]</code>
---------------------------------	---

New: 2026-10-20

Updated: 2026-10-23

Registers a new code backend (source language) with `codemwe`; once registered, the backend can be used through the `backend` option to typeset code and generate a source file of the corresponding format.

`{<backend name>}` is the name of the new backend and also the value of the `backend` option. `{<file extension>}` is given without the dot, e.g. `c` or `py`. `{<comment character>}` is read verbatim and is used for the first-line comment of the source file: the percent sign of `LATEX`, `//` of C, or the number sign of Python, none of which needs to be escaped. `{<compilable>}` takes the value `true` if the source file is to be compiled as a `LATEX` document with its result shown in the result box, and `false` if the package is not to compile it, in which case the result box can only show the image given by the `result` option. `[<minted language name>]` is optional and gives the name that the `minted` package uses for that language (i.e. the `pygments` lexer name), which is needed for the syntax highlighting of the `minted` option, e.g. `python` or `javascript`. When omitted or left empty it defaults to `{<backend name>}`. If the `minted` option is not used, this argument has no effect on the output at all.

```
\codemweNewBackend { python } { py } { # } { false }
```

If the backend is also to be highlighted, give the language name as well:

```
\codemweNewBackend { py } { py } { # } { false } [ python ]
```

The package ships with two built-in backends: `tex` (the default) and `c`. If the backend name registered by `\codemweNewBackend` duplicates a built-in name, the settings of the built-in backend are overwritten.

22 Typesetting Modes

22.1 Horizontal Mode

In the horizontal mode the code box is on the left and the result box on the right, which is convenient for comparing the code with its output. The width allocation rule of the horizontal mode is: if `width` is not specified, the spacing `sep` between the two boxes is subtracted from the line width first and the remaining width is divided equally into two columns. For instance:

```

\begin{example}{}
\ExplSyntaxOn
\tl_clear_new:N \l__fsttest_tl
\tl_set:Nn \l__fsttest_tl
{ cat~likes~fish }
\tl_use:N \l__fsttest_tl
\ExplSyntaxOff
\end{example}

```

which is typeset as:

```

% Example 18
\ExplSyntaxOn
\tl_clear_new:N \l__fsttest_tl
\tl_set:Nn \l__fsttest_tl
{ cat~likes~fish }
\tl_use:N \l__fsttest_tl
\ExplSyntaxOff

```

cat likes fish

22.2 Vertical Mode

In the vertical mode the code box is on the top, the result box is at the bottom, and an optional transition text may be inserted in between. The mode is switched with the `vmode` option. For instance:

```

\begin{example}[vmode]{}
\ExplSyntaxOn
\tl_clear_new:N \l__sectest_tl
\tl_set:Nn \l__sectest_tl { cat~likes~fish }
\tl_use:N \l__sectest_tl
\ExplSyntaxOff
\end{example}

```

which is typeset as:

```

% Example 19
\ExplSyntaxOn
\tl_clear_new:N \l__sectest_tl
\tl_set:Nn \l__sectest_tl { cat~likes~fish }
\tl_use:N \l__sectest_tl
\ExplSyntaxOff

```

The result of **example 19** is shown below.

cat likes fish

In the vertical mode the transition text between the code box and the result box is taken from the **default transition text** of the language variety selected by the `textlang` option (see Section 35). Since this manual has set the variety to English, the transition text shown in the example above is “The result of **example 19** is shown below.”, in which the reference `\ref{ex:19}` is generated automatically; if the `nonum` option is used, the

non-numbered text “The result of the code is shown below.” is used. A single example can also specify its own transition text with the `transition` option.

22.3 Custom Transition Text

The `transition` option customises the transition text between the code box and the result box for a single example (it overrides the default transition text of the variety selected with `textlang`). For instance:

```
\begin{example}[
  vmode,
  pkg      = { tikz },
  title     = { Custom transition text },
  transition = { The typeset result of this example is shown below;
                check that the code and the output agree: },
]{ }
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
  \draw (1, 1) -- (2, 0);
\end{tikzpicture}
\end{example}
```

which is typeset as:

```
% Example 20 Custom transition text
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
  \draw (1, 1) -- (2, 0);
\end{tikzpicture}
```

The typeset result of this example is shown below; check that the code and the output agree:



22.4 Spacing in Horizontal Mode

In the horizontal mode the `sep` option controls the spacing between the left code box and the right result box; its initial value is `18pt`. Like `width`, the value should be a $\text{\LaTeX}$ dimension such as `sep = 0.2\textwidth`; writing `sep = 0.3` would be read as `0.3pt` because $\text{\TeX}$ appends the default unit `pt`. For instance:

```

\begin{example}[
  pkg    = { tikz },
  sep    = { 0.2\textwidth },
  title  = { Wider spacing },
]{ }
\begin{tikzpicture}
  \draw (0, 0) circle (1);
\end{tikzpicture}
\end{example}

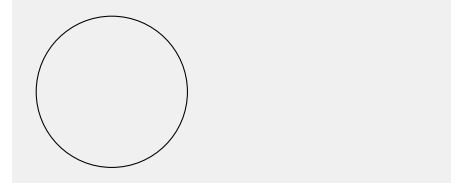
```

which is typeset as:

```

% Example 21 Wider spacing
\begin{tikzpicture}
  \draw (0, 0) circle (1);
\end{tikzpicture}

```



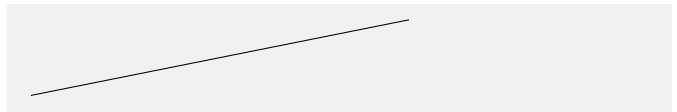
22.5 Code Box Width

The `width` option controls the width of the code box. If it is not specified, the horizontal mode divides the width equally while the vertical mode uses `\textwidth`. The following example sets the code box width to `0.35\textwidth`:

```

% Example 22 set box width
\begin{tikzpicture}
  \draw (0, 0) -- (5, 1);
\end{tikzpicture}

```



Dimension options such as `width` and `sep` should be given as L^AT_EX dimensions (e.g. `0.35\textwidth` or `4cm`). A bare decimal such as `width = 0.35` is read as `0.35pt` because T_EX appends the default unit `pt`, which collapses the code box to a hairline.

23 Numbering and References

By default every `example` environment is numbered. In document classes with a `\chapter` command (`book`, `report`, etc.) the number looks like “chapter.number”; otherwise (e.g. `article`) it is a plain number. A comment line with the number is added to the first line of the code box:

```

% Example 1.3 Draw a line

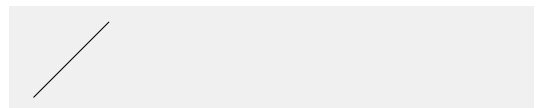
```

At the same time a reference label such as `\label{ex:1-3}` is generated automatically and can be referenced with `\ref`, which prints the leading word given by `pretitle` (`example` by default) followed by the number: “Example 1.3”. If the `label` option customises the label, `\ref` resolves the customised one.

```

% Example 23 Draw a line
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}

```



The **example 23** is the effect of customising the reference label with the

```
label = { code:drawline-en }
```

option and referencing it with `\ref{code:drawline-en}`.

If no numbering is wanted, use the `nonum=true` or `nonum` option: the counter is not advanced and no label is generated, but the code box and the result box are still typeset normally.

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1);
\end{tikzpicture}
```



24 Showing the Code Only

If only the code box is wanted, use the `noshow=true` or `noshow` option:

```
\begin{example}[noshow]{}
  ...
\end{example}
```

The code box is then typeset normally while the result box and its transition text are omitted; numbering and reference labels are unaffected.

```
% Example 24
\begin{tikzpicture}
  \draw (0, 0) cos (2, 1);
\end{tikzpicture}
```

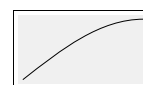
25 Result Figures

If the example code is inconvenient to compile inside the document, the `result` option can specify an image file to be used as the compiled result:

```
\begin{example}[ result = fig/sine.pdf, scale = 0.8 ]{}
  ...
\end{example}
```

`scale=0.8` inserts the image at 80% of its natural size. To insert it by width or height, use the `wd` or `ht` option instead.

```
% Example 25
\begin{tikzpicture}
  \draw (0, 0) sin (2, 1);
\end{tikzpicture}
```



25.1 Width and Height of the Result Figure

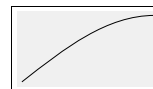
The `wd` option typesets the result figure at a given width, e.g. `wd=0.2\textwidth`:

```
% Example 26
\begin{tikzpicture}
\draw (0, 0) sin (2, 1);
\end{tikzpicture}
```



The `ht` option typesets the result figure at a given height, e.g. `ht=1cm`:

```
% Example 27
\begin{tikzpicture}
\draw (0, 0) sin (2, 1);
\end{tikzpicture}
```



NOTE: When `wd` is given, both `ht` and `scale` are ignored; when only `ht` is given, `scale` is ignored.

26 Outputting the Preliminary Content

Different combinations of the logical options `withcls`, `withpkg`, `withextra` and `nobody` control whether the document class, the packages with their options and the preliminary content are printed in the example code box. If `noshow` or `nobody` is used, this content is printed as is; otherwise a comment sign `%` is prepended to the `cls` and `pkg` content.

The following example combines `withcls`, `withpkg`, `withextra` and `nobody`: the document class and the preliminary content (the preamble) are printed as is in the code box, while the environment body (`nobody`) and the compiled result are not printed:

```
% document class
\documentclass[12pt,a4paper]{article}
% preamble
\usepackage{tikz}
\usepackage[svgnames]{xcolor}
\newcommand{\mya}{a}
\newcommand{\myb}{b}
```

27 Code Backends

By default `codemwe` typesets $\text{\LaTeX}$ example code: the content of the code box is written into the temp `.tex` file, which is read into the result box with `\input` and compiled there to produce the result. This mechanism relies on the code being $\text{\LaTeX}$ code itself.

In order to typeset code of other languages such as C, the package introduces the notion of a *code backend*, selected with the `backend` option. A backend records four pieces of information: the file extension of the source file, the comment character used for its

first-line comment, whether the code of that language can be compiled by $\text{\LaTeX}$, and the language name that the `minted` package uses for that language. Two backends are built in, as shown in Table 3.

Table 3: The built-in code backends				
Backend	Extension	Comment	Compilable	minted language
<code>tex</code>	<code>tex</code>	<code>%</code>	yes	<code>tex</code>
<code>c</code>	<code>c</code>	<code>//</code>	no	<code>c</code>

The last column is used only when the `minted` option is enabled (see Section 31); without `minted` it has no effect on the output.

With `backend = c` the behaviour is as follows:

1. the generated source file is not a $\text{\LaTeX}$ document but a plain C source file that can be passed directly to a C compiler: its first line is a comment (e.g. `// Example 2.2 Sum of two numbers`), followed by the preliminary content given in the mandatory argument of the environment and the code in the code box;
2. the comment lines of the code box use the C comment character `//`, and the $\text{\LaTeX}$ -specific structural comments “document class”, “preamble” and “body” are not added;
3. options such as `cls`, `pkg`, `withcls` and `withpkg` are meaningless for C code and are therefore ignored;
4. since C code cannot be compiled by $\text{\LaTeX}$, the package does not compile it during typesetting. If the `result` option is given, the result box shows the given image; otherwise the result box is omitted altogether and an `backend-no-result` warning is issued.

For example, a C function can be typeset with the following code, using an image as the result:

```

\begin{example}[
  backend = c,
  title   = { Sum of two numbers },
  result  = { csum.png },
  scale   = 0.3,
  filename = { sum.c },
]{
#include <stdio.h>

int sum(int a, int b);

int main(void)
{
  int a, b;
  a = 3;
  b = 4;

  printf("sum=%d\n", sum(a, b));

  return 0;
}
}
int sum(int a, int b)
{
  return a + b;
}
\end{example}

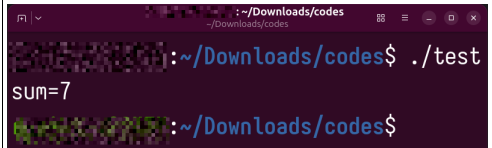
```

which is typeset as:

```

// Example 28 Sum of two numbers
int sum(int a, int b)
{
  return a + b;
}

```



```

~/Downloads/codes$ ./test
SUM=7
~/Downloads/codes$

```

Here filename = { sum.c } saves the source file as sum.c, whose content is:

```
// Example 28 Sum of two numbers
#include <stdio.h>

int sum(int a, int b);

int main(void)
{
    int a, b;
    a = 3;
    b = 4;

    printf("sum=%d\n", sum(a, b));

    return 0;
}

int sum(int a, int b)
{
    return a + b;
}
```

If the name of the source file does not matter, `filename` may be omitted; the name is then constructed from `dir`, `prelabel` and the number, with the extension of the current backend (C uses `c`), for instance `codes/ex28.c`.

NOTE: Since C code is not compiled during typesetting, the image shown in the result box must be prepared in advance and kept in step with the code. If `result` is not given, the result box can be turned off with the `noshow` option to avoid the warning; the `ccode` environment has already set `noshow`.

To typeset a language other than C (e.g. Python), it suffices to register a new backend with `\codemweNewBackend` and use it through the `backend` option; no modification of the package itself is needed.

28 Displaying Line Numbers

Line numbers can be added to the code box with the `[linenos]` option, for instance:

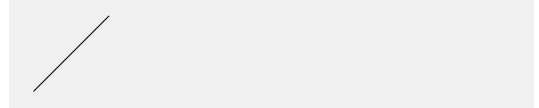
```
\begin{example}[
    pkg    = { tikz },
    title  = { Draw a line },
    linenos,
]{
\begin{tikzpicture}
    \draw (0, 0) -- (1, 1);
\end{tikzpicture}
\end{example}
```

which is typeset as:

```

1 % Example 29 Draw a line
2 \begin{tikzpicture}
3   \draw (0, 0) -- (1, 1);
4 \end{tikzpicture}

```



NOTE: The content added by `withcls`, `withpkg` and `withextra`, as well as the automatically inserted comments, is counted in the line numbers as well.

29 A Complete Example

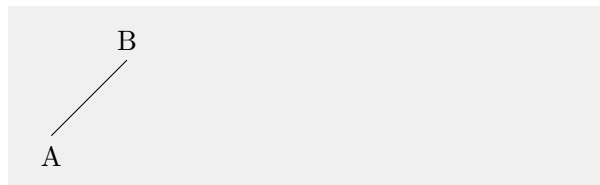
The following example combines several options. It uses the `article` class together with the `tikz` and `xcolor` packages, adds two user macros to the preamble, and uses the vertical mode with a given code box width and a customised title:

```

% Example 30 Draw a line
% preamble
\newcommand{\myA}{A}
\newcommand{\myB}{B}
% body
\begin{tikzpicture}
  \draw (0, 0) node[below] { \myA } -- (1, 1)
    node[above] { \myB };
\end{tikzpicture}

```

The result of [example 30](#) is shown below.



The [example 30](#) generates the MWE `tex` file `codes/ex25.tex`, whose content is:

```

% document class
\documentclass[12pt,a4paper]{article}

% preamble
\usepackage{tikz}
\usepackage[svgnames]{xcolor}

\newcommand{\myA}{A}
\newcommand{\myB}{B}

% body
\begin{document}

\begin{tikzpicture}
  \draw (0, 0) node[below] { \myA } -- (1, 1) node[above] { \myB };
\end{tikzpicture}

\end{document}

```

NOTE: The structural comment lines “% document class”, “% preamble” and “% body” at the top of each of the three blocks are provided by the language variety selected with the `textlang` option (here English, the default). They can simply be deleted when the MWE file is used.

NOTE: If a macro such as `\tikzset` is needed inside the title given by the `title` option, the `\string` primitive must be used, i.e. `\string\tikzset` prints `\tikzset`, so that the macro is not expanded when the comment line is written.

30 Saving the Environment Body to a File

If only the environment body needs to be saved to a file, give the two options [`\nonum, filename=test.tex`] together. The example is then not numbered, but the body is saved to the file given by the `filename` option, with the path prefix still taken from the `dir` option.

NOTE: The saved file contains neither the document class nor the preamble, nor the `\begin{<document>}` and `\end{<document>}` lines, so it cannot be compiled independently.

```
% Test.tex Define a user interface
\ExplSyntaxOn
\tl_clear_new:N \l__test_tl
\tl_set:Nn \l__test_tl { cat~likes~fish }
\newcommand{\mytest}{ % define a user interface
  \tl_use:N \l__test_tl
}
\ExplSyntaxOff
```

31 Syntax Highlighting

By default the code in the code box is typeset as is, without syntax highlighting. If readability of the code matters, the `minted` option can be passed to the package so that the box is typeset with the `minted` package and the code is highlighted:

```
\usepackage[minted]{codemwe}
```

Without the `minted` option the package keeps using the original `fancyvrb` scheme, so the behaviour is the same as in previous versions. With it, the content of the code box is typeset by `minted` at `\small` size with `autogobble` enabled, so that the common indentation is removed. Line numbers are still controlled by the `linenos` option, exactly as without `minted`.

NOTE: `minted` invokes an external program (`latexminted` or `pygmentize`) to do the highlighting, so the document must be compiled with `-shell-escape`. If shell escape is disabled or `minted` is not installed, the package does not abort the run: it issues a `minted-unavailable` warning and falls back to the original scheme. The document is still produced, just without highlighting.

Moreover, `minted` enables `fasthighlightmode` by default: on the first run only a placeholder is inserted into the code box, and the highlighted result appears from the second run onwards. Two or more runs are therefore needed once `minted` is enabled.

The language name needed for highlighting is not given by the `minted` option but by the `backend` option: every code backend records its `minted` language name (see Table 3), and the package passes the name of the current backend to `minted` automatically. With `backend = c`, for instance, the code is highlighted as `c`, and with `backend = tex` as `tex`.

The highlighting style is set with the `mintedstyle` option, whose value is a style name understood by `minted` (pygments); the initial value is `default`. For example, to use the `tango` style:

```
\usepackage[minted, mintedstyle = tango]{codemw}
```

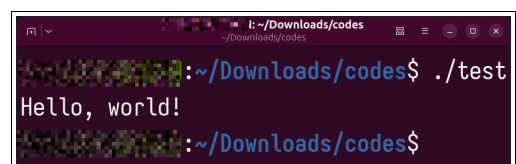
Common styles include `default`, `friendly`, `tango`, `monokai` and `bw`; the complete list is given in the style documentation of `pygments`. If `mintedstyle` is set without enabling `minted`, the package issues a `mintedstyle-unused` warning but still records the value, so that it takes effect as soon as `minted` is enabled.

Both `minted` and `mintedstyle` may be given as package options, through `\codemwreset`, or in an individual environment, just like any other option:

```
\begin{example}[
  backend      = c,
  minted,
  mintedstyle = friendly,
  result       = chelloworld.png,
  wd           = 0.45\textwidth,
  title        = { Syntax highlighting },
]{\}
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
\end{example}
```

which is typeset as:

```
// Example 31 Syntax highlighting
#include <stdio.h>
int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```



31.1 Switching Highlighting On and Off in the Body

Like any other boolean option, `minted` can be switched on and off in the document body, so that only some examples are highlighted:

```
\codemwreset{minted}          % on: every following example is highlighted
\begin{example}{}
...
\end{example}
\begin{example}[minted = false]{}
...
\end{example}
\codemwreset{minted = false} % off: back to the original scheme
```

NOTE: The `minted` package can be loaded in the preamble only. This package performs the check at load time (when it is necessarily still in the preamble): if `minted` is installed and the document is compiled with `-shell-escape`, it loads `minted` right away. `minted` and `minted = false` therefore work anywhere — in the preamble, in the body, or as an environment option — without any need to enable the option once in the preamble beforehand. Note that `minted = false` merely switches the highlighting off, it does not unload the `minted` package, so no reloading is needed when it is switched on again later.

If `minted` is missing or `-shell-escape` is not used, the load does not happen; enabling `minted` then issues a `minted-unavailable` warning and the code is typeset with the original scheme, without highlighting.

NOTE: The purpose of the result box is to actually compile and run the example code and show the outcome, so the code in the result box is always read in and compiled as is, never highlighted; only the code in the code box is highlighted by `minted`. For backends such as `c`, which cannot be compiled by `LATEX`, the result box is shown only when the `result` option is given anyway, so this makes no difference.

32 Package-Level Default Options

The package accepts global default options through `\usepackage[options]{\codemwe}`; they serve as the initial values of the following `example`, `example*`, `texcode` and `ccode` environments. The keys that may be given as package options are `vmode`, `nobody`, `noshow`, `nonum`, `withcls`, `withpkg`, `withextra`, `cls`, `pkg`, `width`, `sep`, `prelabel`, `pretitle`, `dir`, `counter`, `boxtype`, `textlang`, `backend`, `linenos`, `minted` and `mintedstyle`.

The initial values set at package level can be overridden inside an environment by an option of the same name. For instance:

```

\begin{example}[
  vmode = false,
  pkg    = { tikz },
  title  = { Override package's vmode },
]{}
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
\end{example}

```

which is typeset as:

```

% Example 32 Override package's vmode
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}

```



33 Unknown Options and Warnings

If an undefined key is given, the package issues an **unknown-key** warning and ignores the option. Compiling the following example

```

\begin{example}[foo, pkg={tikz}]{}
...
\end{example}

```

produces a warning like

```

Package codemwe Warning:\
Unknown key-val option 'foo' for package 'codemwe'.

```

but the compilation is not aborted.

In addition, two warnings are related to the code backend:

- **unknown-backend**: the **backend** option names a backend that is not registered, and the package falls back to **tex**;
- **backend-no-result**: the current backend cannot be compiled by **L^AT_EX** and no **result** option is given, so the result box is omitted altogether. Giving a **result** image or using the **noshow** option removes the warning.

Three further warnings relate to syntax highlighting:

- **minted-unavailable**: the **minted** option is enabled but shell escape is disabled for the current run or the **minted** package is not installed. The package falls back to the original scheme, so the code is still typeset, just without highlighting;
- **mintedstyle-unused**: **mintedstyle** is set while the **minted** option is not enabled, so the style has no effect for the time being;
- **minted-no-lexer**, **minted-unknown-lexer**: the current backend records no **minted** language name, or the recorded name is not recognised by **pygments**.

34 Box Types

The `boxtype` option selects the box environment used for the code box and the result box; its default value is `syntaxbox`. The built-in box types are listed in Table 4.

Table 4: Built-in box types

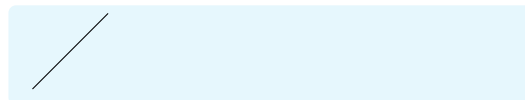
Type	Background	Border	Title bar	Dependency
<code>syntaxbox</code>	gray!12	none	no	<code>tcolorbox</code>
<code>syntaxbox*</code>	white	none	no	<code>tcolorbox</code>
<code>mwebox</code>	cyan	none	yes	<code>tcolorbox</code>
<code>mwebox*</code>	cyan	full border	yes	<code>tcolorbox</code>
<code>mweparbox</code>	purple	left bar	yes	<code>tcolorbox</code>
<code>mweparbox*</code>	white	left bar	yes	<code>tcolorbox</code>

The three starred boxes are white-background, borderless variants of their unstarred counterparts. The `mwebox`/`mwebox*` and `mweparbox`/`mweparbox*` boxes support a title bar: the example number and the title automatically become the content of the title bar. The `syntaxbox` and `syntaxbox*` boxes do not support a title bar; in that case the number and the title are written as a first-line comment inside the code box.

The [example 33](#) is typeset with the `mwebox` box.

Example 33

```
\begin{tikzpicture}
\draw (0, 0) -- (1, 1);
\end{tikzpicture}
```



A user-defined box can be registered with `\codemweNewBox`. As long as the box environment obeys the interface

```
O{background colour} d() O{extra options}
```

i.e. the first optional argument [...] takes a background colour, the second optional argument (...) takes a title and the third optional argument [...] takes extra `tcolorbox` options, it can be used by `boxtype`. For example, such a box can be defined with the `\DeclareTColorBox` command of `tcolorbox`:

```
% Define a tcolorbox box (same interface as the built-in boxes)
\DeclareTColorBox{ mybox }{ 0{ green!30 } d() 0{ } }
{
  colback = #1,
  colframe = #1,
  boxrule = 0pt,
  arc = 4mm,
  left = 8pt,
  right = 8pt,
  top = 4pt,
  bottom = 4pt,
  nobeforeafter,
  IfValueT = { #2 } { title = #2, coltitle = green!30!black },
  #3,
}
```

Since the box uses its () argument to produce a title bar, it should be registered with `withtitle = true` (which is also the default).

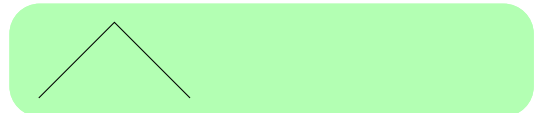
```
% Register the box for typesetting
\codemweNewBox{ mybox }[ withtitle = true ]
```

NOTE: If a custom box supports a title bar, `withtitle` in `\codemweNewBox` must be `true`; if it does not, declare `withtitle = false`. If the declaration does not match the actual behaviour, the number and the title may be lost or printed twice.

The following example is typeset with `mybox` (for readability the box is defined here directly with `\DeclareTColorBox`; the result is identical to defining and registering it as shown above):

Example 34 A custom box

```
\begin{tikzpicture}
  \draw (0, 0) -- (1, 1) -- (2, 0);
\end{tikzpicture}
```



35 Options

All key-value options of `codemwe` are listed below. The “initial value” is the value an option has before any setting; for boolean options the “default value” is the value taken when the option is written without a value (e.g. `vmode`).

`vmode` = `<true/false>` (init: `false`)

The typesetting mode. `vmode = false` is the horizontal mode and `vmode = true` the vertical mode; the boolean “= `true`” may be omitted, so writing `vmode` alone means `vmode = true`. The initial value is `false`.

`noshow` = `<true/false>` (init: `false`)

Whether the result box is shown. With `noshow` only the code box is shown; the result box and its transition text are omitted. The initial value is `false`.

`nonum = <true/false>` (init: false)

Whether the example is numbered and labelled. With `nonum` the numbering is switched off: the counter is not advanced and no label is generated, but the code box and the result box are still typeset normally. The initial value is `false`.

`nobody = <true/false>` (init: false)

Whether the body of the `example` environment is printed. With `nobody` the body is not printed, and hence the compiled result is not printed either (equivalent to also using `noshow`). The initial value is `false`.

`withcls = <true/false>` (init: false)

Whether the document class content given by `cls` is added to the example code box. If `noshow` or `nobody` is used, the content of `cls` is printed as is; otherwise a comment sign `%` is prepended to it. The initial value is `false`.

`withpkg = <true/false>` (init: false)

Whether the package content given by `pkg` is added to the example code box. If `noshow` or `nobody` is used, the content of `pkg` is printed as is; otherwise a comment sign is prepended to each package. The initial value is `false`.

`withextra = <true/false>` (init: false)

Whether the preliminary content given by the mandatory argument of `example` is added to the example code box. The initial value is `false`.

`cls = <document class>` (init: article)

The document class required by the MWE `tex` file. Both the form without class options, `cls = { article }`, and the form with options given through `=`, `cls = { article = { 12pt, a4paper } }` are supported. The `\documentclass[12pt,a4paper]{article}` will be generated latter at the beginning of the MWE `tex` file. The initial value is `article`.

`pkg = <package list>` (init: none)

The list of packages required by the MWE `tex` file. Similar to `cls`, a comma-separated list of packages is given, e.g. `pkg = { tikz, tabulararray, listings = { numbers = left } }`. The initial value is empty, i.e. no package is added.

`width = <dim>` (init: Opt)

The width of the code box. The initial value `Opt` means “not specified”: in the horizontal mode the width is then divided equally after subtracting `sep`, and in the vertical mode `\textwidth` is used. In the horizontal mode, if the value is so large that the right result box would have a negative width, a `width-too-large` warning is issued. The value should be a $\text{\LaTeX}$ dimension, e.g. `width = 0.35\textwidth` .

`sep = <dim>` (init: 18pt)

The spacing between the left code box and the right result box in the horizontal mode; the initial value is 18pt. The value should be a L^AT_EX dimension, e.g. `sep = 0.1\textwidth` .

`prelabel = <prefix>` (init: ex)

The prefix of the MWE file names and of the automatic reference labels; the initial value is ex. For instance, the default `label` of the first example is `ex:1` and the corresponding MWE file name is `ex1.tex`.

`pretitle = <number prefix>` (init: example)

The leading word of the example number, used in the first comment line of the code box and in the text printed by `\ref`; the initial value is `example`. For instance, with `pretitle = Example` the first comment line is “Example 1.1” and `\ref` also prints “Example 1.1”. The value may contain line breaks, which are converted to spaces. In a non-English document it may be changed to an equivalent word in that language, e.g. “example” or “demo”.

`dir = <directory>` (init: codes)

The directory in which the MWE files are saved; the initial value is the `codes` folder in the current directory. **The given path must already exist.**

`counter = <counter name>` (init: example)

The counter name; the initial value is `example`. If the given counter name is already in use, a `counter-exists` warning is issued and the option is ignored. Since the package creates the counter at `\AtBeginDocument`, this option normally needs no change.

`boxtype = <box env name>` (init: syntaxbox)

Selects the box environment used for the code box and the result box; the initial value is `syntaxbox`. The built-in boxes are listed in Table 4. If the given box is not registered, an `unknown-boxtype` warning is issued and the box falls back to `syntaxbox`.

`textlang = <variety name>` (init: en)

Selects the language variety. For each variety the `codemwe` package supplies a default transition text, the structural used in the MWE file, and the default values of `cls` and `pretitle`; the `textlang` option selects which set is used. Its value is a variety name and its initial value is `en` (English).

The package ships with two built-in varieties: `en` (English) and `zh` (Chinese). If the given variety is not registered, an `unknown-textlang` warning is issued and the variety falls back to `en`. Switching the variety is normally done in the preamble or at the beginning of the document with `\codemweset`, e.g. `\codemweset{ textlang = zh }`. A single example can still override the transition text with the `transition` option.

A new variety is registered with the `\codemweNewLang` command; see Section 21.

backend = $\langle$ backend name $\rangle$ (init: `tex`)

Selects the code backend, i.e. the language of the code being typeset. The backend determines the extension of the generated source file, the comment character used for its first-line comment, and whether the code of that language can be compiled by L^AT_EX. Its value is a backend name and its initial value is `tex` (L^AT_EX example code).

The package ships with two built-in backends: `tex` and `c` (C). With `c`, a `c` source file is generated, the first-line comment of the code box uses `//`, and the code is not compiled: if the `result` option is given its image is shown, otherwise the result box is omitted and an `backend-no-result` warning is issued. If the given backend is not registered, an `unknown-backend` warning is issued and the backend falls back to `tex`.

A new backend is registered with the `\codemweNewBackend` command; see Section 21. The backend mechanism is described in detail in Section 27.

minted = $\langle$ true/false $\rangle$ (init: `false`)

Whether the code box is typeset with the `minted` package so as to obtain syntax highlighting. With `minted`, the code in the box is highlighted by `minted` according to the language name recorded by the current backend, at `\small` size and with `autogobble` enabled, so that the common indentation is removed; line numbers are still controlled by the `linenos` option. The initial value is `false`, i.e. the original `fancyvrb` scheme is used and no highlighting is applied.

Given as a package option, every example in the document is highlighted; given as an environment option, only the current example is highlighted and the setting is restored when the environment ends; given through `\codemweset`, it applies to all following examples until changed back by another `\codemweset`. The value `minted = false` thus both turns off package-level highlighting and switches off a single place in the body at any time.

This option requires the `minted` package and a compilation with `-shell-escape`. This package checks both conditions at load time and loads `minted` straight away if they hold, so the option takes effect immediately wherever it is given — preamble, body or environment — with no need to enable it once in the preamble first. If the conditions are not met, the package issues a `minted-unavailable` warning and falls back to the original scheme without aborting the run. Since `minted` inserts only a placeholder on the first run by default, two or more runs are needed to see the highlighted result.

See Section 31 for details.

mintedstyle = $\langle$ style name $\rangle$ (init: `default`)

The syntax highlighting style of `minted`, i.e. a style name of `pygments`; the initial value is `default`. Common values are `tango`, `friendly`, `monokai` and `bw`.

The option takes effect only when `minted` is enabled. If it is set without `minted`, the package issues a `mintedstyle-unused` warning but still records the value, so that it applies once `minted` is enabled.

title = $\langle title \rangle$ (init: none)

The functional title of the example, e.g. “Draw a line”; the initial value is empty. If the current box supports a title bar, the title becomes the title bar; otherwise it becomes part of the first-line comment of the code box. If a macro such as `\tikzset` is needed inside **title**, the `\string` primitive must be used, i.e. `\string\tikzset`.

label = $\langle label\ name \rangle$ (init: none)

The cross-reference label of the example. If it is not given, $\langle prelabel \rangle : \langle chapter-number \rangle$ is generated automatically (classes with `\chapter`, e.g. `ex:8-1`), or $\langle prelabel \rangle : \langle number \rangle$ (classes without `\chapter`, e.g. `ex:1`).

filename = $\langle file\ name \rangle$ (init: none)

The name of the MWE file. If it is not given, $\langle prelabel \rangle \langle chapter-number \rangle .tex$ is generated automatically (classes with `\chapter`, e.g. `ex8-1.tex`), or $\langle prelabel \rangle \langle number \rangle .tex$ (classes without `\chapter`, e.g. `ex1.tex`). If the `nonum` option is used at the same time, only the environment body is saved to that file.

result = $\langle figure\ file \rangle$ (init: none)

The file name of the result figure. Once given, the figure replaces the compiled result of the example code and the code is no longer compiled. The file name may contain a path (use `/` as the path separator). If the file does not exist, a `result-file-missing` warning is issued, but the compilation is not aborted.

wd = $\langle dim \rangle$ (init: 0pt)

The width of the result figure. The initial value `0pt` means “not specified”. It always takes precedence over **ht** and **scale**: once **wd** is given, both of them are ignored.

ht = $\langle dim \rangle$ (init: 0pt)

The height of the result figure. The initial value `0pt` means “not specified”. It takes precedence over **scale**: when **wd** is not given but **ht** is, **scale** is ignored.

scale = $\langle scale\ factor \rangle$ (init: 1)

The scale factor of the result figure; the initial value is 1. It is effective only when neither **wd** nor **ht** is given.

transition = $\langle transition\ text \rangle$ (init: none)

Specifies the transition text between the code box and the result box in the vertical mode **for this single example**; the initial value is empty. When it is left empty, the global default transition text of the language variety selected by `textlang` is used. It may contain commands such as `\ref`, e.g. `transition = \{ The result of \tn {ref}\{ex:1\} is shown below: \}`.

NOTE: The division of work between `transition` and `textlang` is: `textlang` determines the **default transition text** and the other variety-dependent texts of the whole document (settable with `\codemwreset` or as a package option, suitable for switching the variety of a document), while `transition` is a local override for a single example. Choose either of them as needed.

`linenos` = $\langle true/false \rangle$ (init: false)

Whether line numbers are shown on the left of the code box. With `linenos`, every line of the code box is prefixed with its line number, separated from the code by a gap of 6pt; without this option no line numbers are shown. The initial value is `false`.

The line numbers are implemented with the `numbers=left` option of the `\VerbatimInput` command of the `fancyvrb` package. The numbering starts from the first line of the code box and is independent of the numbering of the example itself; when `withcls` or `withpkg` is used, the added document class and package content is counted in the line numbers as well.

36 Notes

1. Only when neither `noshow` nor `nobody` is used are the `cls` and `pkg` contents commented out with `%` in the code box.
2. In the horizontal mode, if `width` is too large so that the right result box would have a negative width, a `width-too-large` warning is issued; decrease `width` or `sep` in that case.
3. The box given by `boxtype` must be registered (built-in or via `\codemweNewBox`); otherwise an `unknown-boxtype` warning is issued and the box falls back to `syntaxbox`.
4. If the counter given by `counter` already exists, a `counter-exists` warning is issued and the option is ignored.
5. If packages such as `expl3` or `tcolorbox` are too old, a `package-old` warning is issued; updating the T_EX distribution is recommended.
6. If the values of `title`, `label`, `transition`, `boxtype` or `pretitle` contain line breaks, the breaks are converted to spaces and leading/trailing spaces are stripped.
7. The result figure is inserted with `\includegraphics`, so it must be in a format supported by `graphicx` (e.g. `pdf`, `png` or `jpg`); `eps` is not supported when compiling with X_YT_EX.

37 Implementation Overview

The central idea of `codemwe` is to write the body of the `example` environment into a temporary file (`\meta{jobname}-example.aux`), which is then read back with `\verbatiminput` inside a `tcolorbox` box to typeset the code box. The result box either

typesets that temporary file in place via `\input` or inserts the figure given by the `result` option. The MWE file is assembled in the structure and written to the directory given by `dir`; its file name is determined by `prelabel` and `counter` (or by the `filename` option).

Internally the package manages its options with the `l3keys` mechanism of `Expl3`: at load time `\ProcessKeyOptions` forwards the `codemwe / pkgopts` key group to the `codemwe` group, and the `example`, `example*` and `texcode` environments set their options inside their own groups with `\keys_set:nn`, with the keys preset by the environment (e.g. `vmode = true` for `example*`) providing the defaults. Since each environment body is processed between `\group_begin:` and `\group_end:`, the options of different examples do not interfere with each other. The variety-dependent texts (the default transition text, the structural comment words, etc.) are kept together in a single “variety $\rightarrow$ text” property list, from which the `textlang` option selects; `\codemweNewLang` registers a new variety in that list. For lower-level details, please refer to the source of `codemwe.sty` directly.

38 Frequently Asked Questions

Why is the result box empty?

If the `result` option is not used, the result box is produced by `\input`-ing the temporary code file; make sure that the example code itself compiles. If the `result` option is used, make sure the figure file exists and the path is correct.

Why are the package names in the code box commented out?

This is the default behaviour of `withpkg` (and `withcls`): when the compiled result is printed, the document class and the package content are prepended with the comment sign `%` and serve only as an explanation. To print them as is, combine with the `noshow` or `nobody` option.

How do I save the example code as a separate file?

Use the `nonum` and `filename` options together and only the environment body is saved to the given file (the path prefix is still taken from `dir`). Note that the file contains neither the document class nor the preamble, nor the `\begin{<document>}/\end{<document>}` lines, and cannot be compiled independently.

Why does the MWE file name differ from the label I set?

The MWE file name is determined only by `prelabel`, `counter` and `filename`; the `label` option does not influence it. `label` only changes the cross-reference label, not the file name.

I set a title, but no title bar appears.

Whether a title bar appears depends on the current box type. The `syntaxbox` and `syntaxbox*` boxes do not support a title bar; the number and the title are then written as a first-line comment inside the code box. To get a real title bar, use `mwebox`, `mwebox*`, `mweparbox` or `mweparbox*`.

Why does the result box show no output of the C program?

C code cannot be compiled by L^AT_EX, so the package does not run it; the result box only shows the image given by the `result` option. To show the real output, run the generated `c` file with a C compiler first, save the output (or a screenshot of it) as an image, and include that image with the `result` option. See Section [27](#).