Contents

1 Hook Management

1.1 add/remove

```
\newcommand\cmda{Content}
\AddToHook{cmd/cmda/before}{Before\ }
% Before Content

\AddToHook{cmd/cmda/after}{\ After}
% Before Content After

\RemoveFromHook{cmd/cmda/before}
% Content After

\RemoveFromHook{cmd/cmda/after}
% Content
```

1.2 create hook

There are 2 commands to create a hook:

- `\NewHook{<hook>}`: create a normal hook
- `\NewReversedHook{<hook>}`: create a reversed hook

the code chunk can have arguments as well, see section: "Hooks with Args".
Remark: To add code chunks to hook, there is only one command: `\AddToHook{<hook>}{code}`, and the `<label>` is essential for reversed hook. See the last subsection in this section.

1.3 where is the code chunk ?

To see where this code chunk is stored, and what is the difference between the

```
\pretocmd{<cmd>}{code}{<success code>}{<failure code>}
\apptocmd{<cmd>}{code}{<success code>}{<failure code>}
```

There is 2 simple examples about etoolbox and LaTeX Hook Management.
1. TOOLS in etoolbox

```
\pretocmd{\cmda}{(CODE-CHUNK) }{}{}
\meaning\cmda\par
\cmda{}

% \long macro:#1->(CODE-CHUNK) CMD-A:#1
% (HOOK-CHUNK) CMD-A:
```

2. TOOLS in hook management

```
\AddToHook{cmd/cmda/before}{(HOOK-CHUNK) }
\meaning\cmda
\cmda{}
```

```
4
5 % \long macro:#1->\UseHookWithArguments {cmd/cmda/before}{1}{#1}CMD-A:#1
6 % (HOOK-CHUNK) CMD-A:
```

1.4 hook label

Original Content of `\cmda`: `Before-1 Before-2 Content`, the below contents listing the result of each hook label

```
1 \RemoveFromHook{cmd/cmda/before}
2 % Before-1 Before-2 Content
3
4 \RemoveFromHook{cmd/cmda/before}[bf-1]
5 % Before-2 Content
6
7 \RemoveFromHook{cmd/cmda/before}[bf-2]
8 % Before-1 Content
```

If use the default hook label frequently, you can consider using the "environment":

```
1 \PushDefaultHookLabel {<default label>}
2 % <code>
3 \PopDefaultHookLabel
```

'Push' to alter the default lable, while 'Pop' reverts it.
Or you can using:

```
1 \SetDefaultHookLabel {<default label>}
```

The effect holds until the label is changed again or until the next `\PopDefaultHookLabel`.

1.5 one-off hook

Execute code only once, i.e., just the next time a hook is called. Because this is one-off code, it is not labeled.

```
1 \AddToHookNext{hook}{code}
2 \ClearHookNext{hook}
```

A simple example:

```
1 \newcommand\cmda{Content}
2 \AddToHook{cmd/cmda/before}[bf-1]{Before-1\ }
3 \AddToHookNext{cmd/cmda/before}{Before-Once\ }
4 \cmda\par
5 \cmda\par
6 % Before-1 Before-Once Content
7 % Before-1 Content
```

Typical use cases for `\AddToHookNext` are the hooks related to shipping out pages; e.g., you may want to use a special background on the next page.

1.6 reverse hook

Some hooks (whether normal or one-time) come in pairs, the hooks `file/before` and `file/after`, which are executed before and after the loading of every file. The second of such hooks is called **Reverse Hook**, means: the execution order in the reversed hook is exactly the opposit
An example to see the difference between the normal and reversed hooks:

```
% Original: Content
\AddToHook{cmd/cmda/before}[bf-1]{Before-1\ }
\AddToHook{cmd/cmda/before}[bf-2]{Before-2\ }
\AddToHook{cmd/cmda/after}[af-1]{\ After-1}
\AddToHook{cmd/cmda/after}[af-2]{\ After-2}
\cmda

% Before-1 Before-2 Content After-2 After-1
```

Then the `cmd/<name>/after` hook is a reversed hook.
Another example of creating your own reversed hook:

```
\documentclass{article}
\NewHook{zlatex/thmstyle/before}
\NewReversedHook{zlatex/thmstyle/after}
\newenvironment{test}
  {\UseHook{zlatex/thmstyle/before}}
  {\UseHook{zlatex/thmstyle/after}}


\AddToHook{zlatex/thmstyle/before}[bf-1]{aaa}
\AddToHook{zlatex/thmstyle/before}[bf-2]{bbb}
\AddToHook{zlatex/thmstyle/after}[af-1]{AAA}
\AddToHook{zlatex/thmstyle/after}[af-2]{BBB}
\begin{document}
\begin{test}
Hello
\end{test}
\end{document}

% aaabbb Hello AAABBB
```

Remark: The labels('af-1', 'af-2') for the reversed hook is essential, or it will work as a normal hook. See "lthooks-doc:2.1.4 Updating code for hooks":
If there already exists code under the `<label>` then the new `<code>` is appended to the existing one (even if this is a **reversed** hook).

2 Common Hooks

2.1 environment hooks

Every environment offers a set of four generic hooks.

- Outer hooks: `before` and `after`
- Inner hooks: `begin` and `end`

Only the `/after` hook is implemented as a reversed hook; Generic environment hooks are never one-time hooks even with environments that are supposed to appear only once in a document.
The hooks are executed only if `\begin{env}` and `\end{env}` are used. If the en- vironment code is executed via low-level calls to `\⟨env⟩` and `\end⟨env⟩` (e.g., to avoid the environment grouping), they are not available. If you want them available in code using this method, you would need to add them yourself, i.e., write something like

```
\UseHook{env/quote/before}\quote
...
\endquote\UseHook{env/quote/after}
```

to add the outer hooks, etc.
Remark:Loading order among some `\begin{document}` commands:

```
1. \AddToHook{env/document/begin} % still in preamble

2. \document\normalsize ...

3. \AtBeginDocument(= \AddToHook{begindocument})
```

Reference: AtBeginDocument VS AddToHook{env/document/begin}

2.2 command hooks

Similar to environments there are two generic hooks available for any LaTeX (document-level) command — in theory at least. In practice there are restrictions, and especially the `/after` hooks work only with a subset of commands.
For example, if you try to hook the `\section` command as below, you will get an error message:

```
\AddToHook{cmd/section/after}{AFTER SECTION}
```

Compile it, you will get the error message:

```
1. Argument of \hook_use:nnw has an extra }.
<inserted text>

2. Paragraph ended before \hook_use:nnw was complete.
<to be read again>
```

3 File Hooks

3.1 file/package/class hooks

An example:

```
\AddToHook{package/amsmath/after}{%
  \AddToHook{cmd/colon/after}{\!}%
}
```

This is essential to make our this example involving `package/amsmath/after` work: if the package was already loaded, then the code in immediately applied. If we had used a file hook instead (which is a normal hook), the code would have been stored away, waiting forever for a second invocation that would never happen.
Some usefule hooks:

- `file/before`, `file/<file>/before`, `file/<file>/after`, `file/after`
- `package/before`, `package/<name>/before`, `package/<name>/after`, `package/after`
- `class/before`, `class/<name>/before`, `class/<name>/after`, `class/after`

As you may have guessed, the last two are reversed hooks. The `⟨file⟩` name has to be given with its extension to be recognized, even if it is `.tex`, so this is different from the behavior of the `\input` command.

3.2 include hooks

The final group of file-related hooks are those specific to files loaded with an `\include` command.

4 Shipout Hooks

4.1 Background and Foreground

During the shipout the `background picture` is printed first, then the `content of the page`, and finally the `foreground picture`, each overwriting the other.
With the starting point being the top-left corner of the page, your vertical co- ordinate values should be negative.
The hooks should therefore contain only `\put` commands or other commands suitable within a picture environment;


```
% 1. foreground
\AddToHookNext{shipout/foreground}
{\put(.5\paperwidth, -.5\paperheight)
  {\makebox(0, 0){\includegraphics
    [width=.5\paperwidth]{latex-logo}}}}

% 2. background
\AddToHookNext{shipout/background}
  {\put(.5\paperwidth, -.5\paperheight)
    {\makebox(0, 0){\includegraphics
      [width=.5\paperwidth]{vs_logo}}}}

% 3. remove shipout picture
\RemoveFromHook{shipout/foreground}
\RemoveFromHook{shipout/background}
```

For `foreground` case, you may need the `transparent` package to make the image transparent, an simple example:

```
% \usepackage{transparent}
\transparent{.5}\includegraphics{<pic-name>}
```

For XeTEX, see: `https://github.com/ho-tex/transparent/issues/6`:

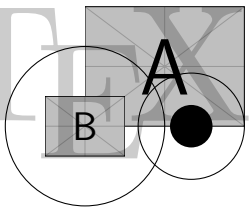
```
\RequirePackage{pdfmanagement-testphase}
\DeclareDocumentMetadata{}

\documentclass{book}
\usepackage{transparent,lipsum}
\begin{document}
\lipsum[1-3]
\transparent{0.5}
\lipsum[1-3]
\end{document}
```

4.2 review pict2e

Review of the `pict2e` which is extention of the `picture` environment:

```
\begin{picture}(0,0)
  \put(0,0) {\includegraphics[width=6em]{example-image-a}}
  \put(0,0) {\makebox(0, 0){
    \includegraphics[width=3em]{example-image-b}
  }}
  \put(0,0) {\circle{60}}
  \put(40,0){\circle{40}} \put(40,0){\circle*{16}}
\end{picture}
```



4.3 tikz code

Using tikz code for watermark:

```
\AddToHook{shipout/background}{%
\put(1cm,-\paperheight)
{\begin{tikzpicture}[remember picture]%
```

```
% tikz code
\end{tikzpicture}}%
}
```

For example:

```
\AddToHook{shipout/background}{%
\put(.5\paperwidth, -.5\paperheight)
{\makebox(0, 0){\begin{tikzpicture}[remember picture]%
  \draw[pink, fill=pink] (0,0) rectangle (5, 5);
\end{tikzpicture}}}%
}
```

4.4 Other Shipout Hooks

All related hooks:

- `shipout/background`
- `shipout/foreground`
- `shipout/firstpage`: can also be set using the command `\AtBeginDvi`
- `shipout/lastpage`
- `shipout/before`: can be used to manipulate the collected page box before it is being shipped out (or even discard it)
- `shipout`: is executed right in front of the page being shipped out, i.e., after any foreground or background material has been added.
- `shipout/after`: is called after the current page has been shipped out.

Caution: Note that it is not possible (or advisable) to try to use these hooks to typeset material with the intention of returning it to the main vertical list. It will go wrong and give unexpected results in many cases

5 Declaring Hooks

5.1 basic intro

There is one further category: some hooks are so called "**Generic Hooks**"(the name of these hooks can be anything valid assigned by user, like: "`mypackage-sub1-testA`", "`hello|world`", "`./hook`", "`./hook/sub`" etc). So these hooks has to be explicitly declared before it can be used in code.
These generic hook can be seen as a place holder, you can put these holders(-tag) into a specific place in some command or environment. When you want to replace these holders(-tag) by some code, you can use the `\AddToHook` command to replace them. How to place these holders(-tag) into the command or environment? You can use `\AddToHook` with `\UseHook` commands to place them into the places you want to put.
Remark: something like place some macros with default value **empty** there, and assign these macros using `\def<macro name>{content}`.

Remark: Remember that the default label for code chunks is the current package or class name when used without specifying an explicit label. Thus "`./hook`" is equvilent to "`mypackage/hook`" if the package or class name is `mypackage`. The `top-level` label, which by default is used to label code added in the preamble or in the document body to a hook, is special in that it is always executed last in a normal hook (and first in a reversed hook)

5.2 declaration and call

We now look briefly at what is necessary to define new hooks and use them in your own package code.

```
\NewHook{hook}       \NewReversedHook{hook}
\NewMirroredHookPair{hook1}{hook2}
```

How to call (normal) hooks ?

- `\UseHook{hook}`: execute a normal hook code
- `\UseOneTimeHook{hook}`: If the hook is intended to be a one-time hook, you call it with this. This has the effect that any further calls to `\UseOneTimeHook` or `\UseHook` with that `hook` name as the argument do nothing.

5.3 simple example

An example using generic hooks:

```
\newcommand\cmdb{Generic-Content}
\NewHook{top-second-BF}
\NewHook{mypackage/AF}
\AddToHook{cmd/cmdb/before}{\UseHook{top-second-BF}}
\AddToHook{cmd/cmdb/after}{\UseHook{mypackage/AF}}
\cmdb\par
% Generic-Content

\AddToHook{top-second-BF}{Fisrt-New-Hook\ }
\AddToHook{mypackage/AF}{\ Second-New-Hook}
\cmdb\par
% Fisrt-New-Hook Generic-Content Second-New-Hook
```

5.4 Reverse hook

Reverse hook in this case is as follows:

```
\newcommand\cmdb{Generic-Content}
\NewHook{top-second-BF}
\NewHook{mypackage/AF}
\AddToHook{cmd/cmdb/before}{\UseHook{top-second-BF}}
\AddToHook{cmd/cmdb/after}{\UseHook{mypackage/AF}}
\AddToHook{top-second-BF}{Fisrt-1\ }
\AddToHook{mypackage/AF}{\ Second-1}
```

```
\AddToHook{top-second-BF}{Fisrt-2\ }
\AddToHook{mypackage/AF}{\ Second-2}
\cmdb\par
% Fisrt-1 Fisrt-2 Generic-Content Second-1 Second-2
```

Thus the hook `top-second-BF` is not a reverse hook even though `cmd/cmdb/after` is a reverse hook. The following code alter `mypackage/AF` to a reverse hook (the hook label for `mypackage/AF` is essential):

```
\newcommand\cmdb{Generic-Content}
\NewHook{top-second-BF}
\NewReversedHook{mypackage/AF}
\AddToHook{cmd/cmdb/before}{\UseHook{top-second-BF}}
\AddToHook{cmd/cmdb/after}{\UseHook{mypackage/AF}}
\AddToHook{top-second-BF}{Fisrt-1\ }
\AddToHook{mypackage/AF}[af-1]{\ Second-1}
\AddToHook{top-second-BF}{Fisrt-2\ }
\AddToHook{mypackage/AF}[af-2]{\ Second-2}
\cmdb\par
% Fisrt-1 Fisrt-2 Generic-Content Second-2 Second-1
```

5.5 hook activation

For a truly generic hook, with a **variable part** in the hook name, an up-front activation would be difficult or impossible, such as `babel` package, it may want to provides:

```
babel/<language>/afterextras
```

So doing the activation should by the user who wants to use a particular hook. Best practice is (Once this declaration is given, the hook is activated):

```
\ActivateGenericHook{hook}
```

In contrast to `\NewHook` (which also activates a hook), this declaration can be used multiple times. An example of `babel`(such declarations can happen even before babel is loaded):

```
\AddToHook{babel/ngerman/afterextras}{\color{blue}}
\ActivateGenericHook{babel/ngerman/afterextras}

\usepackage{color} \usepackage[ngerman,english]{babel}
```

Another example about activation hooks:

```
\newcommand\cmdb{Generic-Content}
\AddToHook{cmd/cmdb/after}{\UseHook{mypackage/AF}}
\AddToHook{mypackage/AF}{\ Second-1}
\AddToHook{mypackage/AF}{\ Second-2}
\cmdb\par
% Generic-Content

\ActivateGenericHook{mypackage/AF}
\cmdb\par
% Generic-Content Second-1 Second-2
```

Remark: For some commands such patching is not possible, and to avoid the user getting a low-level failure, LaTeX has the declaration `\DisableGenericHook`. Thus it is important to explicitly declare the hook with either `\NewHook` or `\NewReversedHook` to make it "nongeneric" after `\UseHook` call.

5.6 Reorder code chunks

The default assumption is that if several packages add data to the same hook, the **order** of the code execution is **of no importance**.
But sometimes the excution order of each code chunk is important. You can declare a rule that tells the hook management to arrange for the required order. If the other package is not loaded, the rule is simply ignored.

```
1 \DeclareHookRule{hook}{label-1}{relation}{label-2}
```

Remark: There can be only a single relation; a later `\DeclareHookRule` overwrites any previous declaration. Optional relations are:

- `before` or `<`: Code for `label-1` comes before code for `label-2`
- `after` or `>`: Code for `label-1` comes after code for `label-2`
- `incompatible-warning`: only 1 code chunk can be executed for underlying incompatiblity. A warning is raised if both labels appear in the same hook.
- `incompatible-error`: like the above, but it throws an error.
- `voids`: Code for `label-1` overwrites code for `label-2`.
- `unrelated`: Undo an incorrect rule specified earlier. Or you can use `\ClearHookRule`

If there are many hooks for which this relationship holds, you can use this:

```
1 \DeclareDefaultHookRule{label-1}{relation}{label-2}
```

6 Hooks with Args

6.1 introduction

Sometimes it is necessary to pass **contextual information**[1] to a hook, and, for one reason or another, it is **not feasible to store such information in macros**. To serve this purpose, hooks can be declared with arguments, so that the programmer can pass along the data necessary for the code in the hook to function properly.
A hook with arguments works mostly like a regular hook, and most commands that work for regular hooks, also work for hooks that take arguments. The differences are:

- use `\NewHookWithArguments` instead of `\NewHook`
- use `\AddToHookWithArguments` instead of `\AddToHook`
- use `\UseHookWithArguments` instead of `\UseHook`

[1] See example in "SECOND EXAMPLE": the contextual information for `\cmda` is "CMDA-ARGS"

Command argument specification:

```
\NewHookWithArguments{<hook>}{<number>}
```

All code added to that hook can then use `#1` to access the first argument, `#2` to access the second, and so on ...
All the command `\cmda` in this section is defined as:

```
\newcommand\cmda[1]{CMD-A:#1}
```

6.2 why we need hook with args ?

See the following example, the first is a hook without arguments, the second is a hook with arguments.
FIRST EXAMPLE

```
\NewHook{self/noargs/pre}
\AddToHook{self/noargs/pre}{\gdef\hello{#1}}
\AddToHook{cmd/cmda/before}{\UseHook{self/noargs/pre}}

\meaning\cmda\par
\cmda <---> \hello
```

Compile it, you will get the (error) message:

```
long macro:#1->\UseHookWithArguments{cmd/cmda/before}{1}{#1}CMD-A:#1

% if you use the \hello command you will get error messages:
1. Illegal parameter number in definition of \hello.
<to be read again>
2. You can't use `macro parameter character #' in horizontal mode.
\hello ->##
```

SECOND EXAMPLE

```
\NewHookWithArguments{self/args/pre}{1}
\AddToHookWithArguments{self/args/pre}{\gdef\hello{#1}}
\AddToHookWithArguments{cmd/cmda/before}
  {\UseHook{self/args/pre}{#1}}

\meaning\cmda\par
\cmda{CMDA-ARGS} <---> \hello
```

Compile it, you will get the result:

```
long macro:#1->\UseHookWithArguments{cmd/cmda/before}{1}{#1}CMD-A:#1
CMD-A:CMDA-ARGS <---> CMDA-ARGS
```

6.3 examples

An simple example from the TLC-3rd Edition, you can see this code snippets as a way of debug hook that has argument(s), for that `\UseHook` will exccute the code chunk in a Hook.

```
\NewHookWithArguments{test}{1}
\AddToHookWithArguments{test}{%
  \typeout{Defining foo with "#1"}
  \def\foo##1{Hello, ##1! Some text after: #1}%
}
\UseHook{test}{Howdy!}
\ShowCommand\foo
```

Running the code above prints in the terminal:

```
Defining foo with "Howdy!"
> \foo=macro:
#1->Hello, #1! Some text after: Howdy!.
```

There is a more specific example about hook with arguments:

```
% 1.define a hook that has argument
\NewHookWithArguments{self/args/pre}{1}
% 2. define how to use this hook that has argument
\AddToHookWithArguments{self/args/pre}{\gdef\hello{#1}}
% 3. test this hook that has argument
\UseHook{self/args/pre}{WORLD}
\meaning\hello\par
% 4. add this hook to a exsit command:\cmda
\AddToHookWithArguments{cmd/cmda/before}
  {\UseHook{self/args/pre}{#1}}
\meaning\cmda\par
\cmda{CMDA-ARGS} <---> \hello


% result
macro:->WORLD
long macro:#1->\UseHookWithArguments{cmd/cmda/before}{1}{#1}CMD-A:#1
CMD-A:CMDA-ARGS <---> CMDA-ARGS
```

7 Diagnose Hooks

7.1 display hook

The \ShowHook command gives you an overview about the **code chunks** that have been added to a particular hook including information about the **order** they are executed.
An simple example:

```
\newcommand\cmdb{Generic-Content}
\NewReversedHook{mypackage/AF}
\AddToHook{cmd/cmdb/after}{\UseHook{mypackage/AF}}
\AddToHook{mypackage/AF}[af-1]{\ Second-1}
\AddToHook{mypackage/AF}[af-2]{\ Second-2}
\ShowHook{mypackage/AF}

```

```
% compile result as below:
-> The hook 'mypackage/AF':
> Code chunks:
>     af-1 -> \ Second-1
>     af-2 -> \ Second-2
> Document-level (top-level) code (executed first):
>     ---
> Extra code for next invocation:
>     ---
> Rules:
>     ---
> Execution order (after reversal):
>     af-2, af-1.
```

The \LogHook{hook} produces the same information but simply writes it into the transcript file.

7.2 check hook

```
\IfHookEmptyTF{<hook>}{<true code>}{<false code>}
\IfHookExistsTF{<hook>}{<true code>}{<false code>}
```

7.3 debug hook

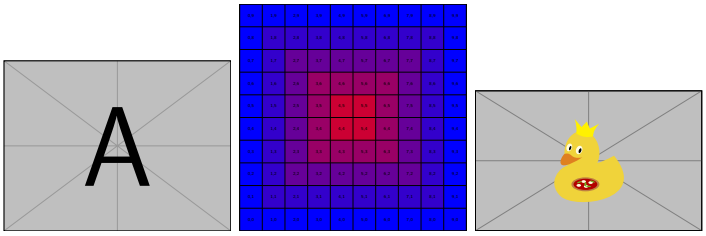
Turn the debugging of hook code on or off. This displays most changes made to the hook data structures. The output is rather coarse and **not really intended for normal use**.

```
\DebugHooksOn
...
\DebugHooksOff
```

8 Appendix

Example image in graphicx package:

```
\includegraphics[width=3cm]{example-image-a}
\includegraphics[width=3cm]{example-grid-100x100pt}
\includegraphics[width=3cm]{example-image-duck}
```



9 Reference

[1] lthooks-code.pdf
[2] ltfilehook.pdf
[3] ltshipout-doc.pdf
[4] The LaTeX Companion - 3rd edition