Hooks & Sockets

Frank Mittelbach

Prague, July 2024

What's a hook?

And for what is it needed?

Characteristics of (my garage) hooks

Hook status

- A hook can be empty
- or it could hold one or more items

Hook items

- Items can be
  - added
  - removed
  - reordered

Offsite storage

- A hook can be used to store items for future use

Characteristics of (my garage) hooks

Hook status

- A hook can be empty
- or it could hold one or more items

Hook items

- Items can be
  - added
  - removed
  - reordered

Offsite storage

- A hook can be used to store items for future use

Characteristics of (my garage) hooks

Hook status

- A hook can be empty
- or it could hold one or more items

Hook items

- Items can be
  - added
  - removed
  - reordered

Offsite storage

- A hook can be used to store items for future use

History of LaTeX hooks

LaTeX 2.09

- None — only patching of internal commands was possible

LaTeX 2$_\varepsilon$

- A few, mainly `\AtBeginDocument` and `\AtEndDocument`
- No management — first come, first served

Today

- A general hook management
- Hooks in many places (number is growing)
- Hook data can be manipulated from the outside

History of LaTeX hooks

LaTeX 2.09

- None — only patching of internal commands was possible

LaTeX 2_ε

- A few, mainly `\AtBeginDocument` and `\AtEndDocument`
- No management — first come, first served

Today

- A general hook management
- Hooks in many places (number is growing)
- Hook data can be manipulated from the outside

History of LaTeX hooks

LaTeX 2.09

- None — only patching of internal commands was possible

LaTeX 2_ε

- A few, mainly `\AtBeginDocument` and `\AtEndDocument`
- No management — first come, first served

Today

- A general hook management
- Hooks in many places (number is growing)
- Hook data can be manipulated from the outside

The problem with the $\LaTeX\,2_\varepsilon$ hooks . . .

The problem with the LaTeX 2_ε hooks . . .
No alterations possible / no offsite storage / only a few

No alterations possible / no offsite storage / only a few

No alterations possible

- The order of code execution was fixed by the order in which the code was added
- In case of problems the advice therefore was: "Alter the package loading order" but that often didn't work

No offsite storage

- You couldn't provide code for other packages unless they were already loaded
- As a consequence, a package like hyperref had to provide complex conditional code
  - based on packages already loaded;
  - check at \AtBeginDocument if some got loaded later
  - execute different code depending on package combination

No alterations possible / no offsite storage / only a few

No alterations possible

- The order of code execution was fixed by the order in which the code was added
- In case of problems the advice therefore was:
  "Alter the package loading order" but that often didn't work

No offsite storage

- You couldn't provide code for other packages unless they were already loaded
- As a consequence, a package like `hyperref` had to provide complex conditional code
  - based on packages already loaded;
  - check at `\AtBeginDocument` if some got loaded later
  - execute different code depending on package combination

No alterations possible / no offsite storage / only a few

No alterations possible

- The order of code execution was fixed by the order in which the code was added
- In case of problems the advice therefore was:
  "Alter the package loading order" but that often didn't work

No offsite storage

- You couldn't provide code for other packages unless they were already loaded
- As a consequence, a package like `hyperref` had to provide complex conditional code
  - based on packages already loaded;
  - check at `\AtBeginDocument` if some got loaded later
  - execute different code depending on package combination

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot "hook" into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot "hook" into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)
- Thus, packages are often incompatible with others
- Furthermore, updates to the internals would break the patches

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot “hook” into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)
- Thus, packages are often incompatible with others
- Furthermore, updates to the internals would break the patches

Result: Users were often unhappy

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot "hook" into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)
- Thus, packages are often incompatible with others
- Furthermore, updates to the internals would break the patches

Result: Users were often unhappy

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot “hook” into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)
- Thus, packages are often incompatible with others
- Furthermore, updates to the internals would break the patches

Result: Users were often unhappy

No alterations possible / no offsite storage / only a few

Not enough hooks

- Mainly `\AtBeginDocument` and `\AtEndDocument`
- Some packages tried to provide a few additional hooks

For anything else, one had to patch (a.k.a. overwrite) internals

Patching means

- Different packages cannot “hook” into the same place, unless
  - they knew about each other
  - account for the different scenarios (i.e., which packages are present and in what order)
- Thus, packages are often incompatible with others
- Furthermore, updates to the internals would break the patches

Result: Users were often unhappy

The LaTeX maintenance / improvement dilemma

Not fixing bugs / not making improvements

- Users are unhappy when these are needed but unavailable
- The increase in incompatibility over time makes everybody unhappy

Fixing bugs / making improvements

- Makes developers unhappy if their patches are broken by kernel fixes or improvements
- Makes users unhappy — something new always breaks some existing usage somewhere (even if only for a short while)

Our answer until 2016: Keep the kernel frozen

The LaTeX maintenance / improvement dilemma

Not fixing bugs / not making improvements

- Users are unhappy when these are needed but unavailable
- The increase in incompatibility over time makes everybody unhappy

Fixing bugs / making improvements

- Makes developers unhappy if their patches are broken by kernel fixes or improvements
- Makes users unhappy — something new always breaks some existing usage somewhere (even if only for a short while)

Our answer until 2016: Keep the kernel frozen

The LaTeX maintenance / improvement dilemma

Not fixing bugs / not making improvements

- Users are unhappy when these are needed but unavailable
- The increase in incompatibility over time makes everybody unhappy

Fixing bugs / making improvements

- Makes developers unhappy if their patches are broken by kernel fixes or improvements
- Makes users unhappy — something new always breaks some existing usage somewhere (even if only for a short while)

Our answer until 2016: Keep the kernel frozen

The LaTeX maintenance / improvement dilemma (cont.)

Our answer today

- Get rid of patching in packages by providing suitable hooks
- This still makes developers unhappy, as this means changing packages to use these hooks
- However, hopefully only a one-time effort for developers!

The task

- Identify all places where patching was considered necessary
  - For example, `\@footnotetext` is currently patched by 7 packages in 4 different places
- Provide suitable hooks to avoid the need to patch
- Then update the packages to use these hooks

The LaTeX maintenance / improvement dilemma (cont.)

Our answer today

- Get rid of patching in packages by providing suitable hooks
- This still makes developers unhappy, as this means changing packages to use these hooks
- However, hopefully only a one-time effort for developers!

The task

- Identify all places where patching was considered necessary
  - For example, `\@footnotetext` is currently patched by 7 packages in 4 different places
- Provide suitable hooks to avoid the need to patch
- Then update the packages to use these hooks

Hooks and code chunks

Characteristics

- A hook can hold arbitrarily many (labeled) code chunks
- These labeled chunks can be reordered or removed

Names and labels

- Hook ⟨*names*⟩ have to be unique across the document
- Only code chunks with distinct labels can be manipulated.

Defaults

- New hooks are empty and do not alter typesetting
- However, they are by default not transparent to expandable input scanning!
- For full transparency, e.g., in `tabular` a special version of `\UseHook` is needed (without debugging information)

Hooks and code chunks

Characteristics

- A hook can hold arbitrarily many (labeled) code chunks
- These labeled chunks can be reordered or removed

Names and labels

- Hook ⟨*names*⟩ have to be unique across the document
- Only code chunks with distinct labels can be manipulated.

Defaults

- New hooks are empty and do not alter typesetting
- However, they are by default not transparent to expandable input scanning!
- For full transparency, e.g., in `tabular` a special version of `\UseHook` is needed (without debugging information)

Hooks and code chunks

Characteristics

- A hook can hold arbitrarily many (labeled) code chunks
- These labeled chunks can be reordered or removed

Names and labels

- Hook ⟨*names*⟩ have to be unique across the document
- Only code chunks with distinct labels can be manipulated.

Defaults

- New hooks are empty and do not alter typesetting
- However, they are by default not transparent to expandable input scanning!
- For full transparency, e.g., in `tabular` a special version of `\UseHook` is needed (without debugging information)

Key takeaways from the new hook management

Easy and fast

- Packages can easily offer hooks that allow for
  - coordination with other packages
  - safe/controlled extensions
  - easy user customizations
- If a hook is unused, there is nearly no overhead

Improved compatibility

- Different packages can add to the same hook without conflicts
- If code ordering is necessary, rules can be set up
- No destructive patching is needed

Anticipated usage supported

- Add code to a hook even if it doesn't exist yet
  (the defining package may or may not get loaded later)

Key takeaways from the new hook management

Easy and fast

- Packages can easily offer hooks that allow for
  - coordination with other packages
  - safe/controlled extensions
  - easy user customizations
- If a hook is unused, there is nearly no overhead

Improved compatibility

- Different packages can add to the same hook without conflicts
- If code ordering is necessary, rules can be set up
- No destructive patching is needed

Anticipated usage supported

- Add code to a hook even if it doesn't exist yet
  (the defining package may or may not get loaded later)

Key takeaways from the new hook management

Easy and fast

- Packages can easily offer hooks that allow for
    - coordination with other packages
    - safe/controlled extensions
    - easy user customizations
- If a hook is unused, there is nearly no overhead

Improved compatibility

- Different packages can add to the same hook without conflicts
- If code ordering is necessary, rules can be set up
- No destructive patching is needed

Anticipated usage supported

- Add code to a hook even if it doesn't exist yet
  (the defining package may or may not get loaded later)

Hook examples

Putting something into the page background

```
\AddToHookNext{shipout/background}
  {\put(.5\paperwidth,-.5\paperheight)%
       {\makebox(0,0)%
          {\includegraphics{figures/hummingbird.png}}}}
```

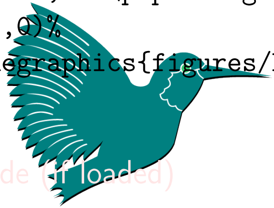
Patching package code (if loaded)

```
\AddToHook{file/dinbrief.cls/after}[firstaid]
   {\FirstAidNeededT{dinbrief}{cls}%
                    {2000/03/02 LaTeX2e class}%
       {\AddToHook{env/document/begin}{\begingroup}}}
```

Hook examples

Putting something into the page background

```
\AddToHookNext{shipout/background}
  {\put(.5\paperwidth,-.5\paperheight)%
      {\makebox(0,0)%
         {\includegraphics{figures/hummingbird.png}}}}
```



Patching package code (if loaded)

```
\AddToHook{file/dinbrief.cls/after}[firstaid]
   {\FirstAidNeededT{dinbrief}{cls}%
                    {2000/03/02 LaTeX2e class}%
       {\AddToHook{env/document/begin}{\begingroup}}}
```

Hook examples

Putting something into the page background

```
\AddToHookNext{shipout/background}
  {\put(.5\paperwidth,-.5\paperheight)%
       {\makebox(0,0)%
          {\includegraphics{figures/hummingbird.png}}}}
```

Patching package code (if loaded)

```
\AddToHook{file/dinbrief.cls/after}[firstaid]
   {\FirstAidNeededT{dinbrief}{cls}%
                    {2000/03/02 LaTeX2e class}%
       {\AddToHook{env/document/begin}{\begingroup}}}
```

Hook examples

Make my document shorter please

```
\AddToHook{para/begin}{\looseness=-1 }
```

```
\newcommand\cancellooseness
  {\AddToHookNext{para/begin}{\looseness=0 }}
```

Notes

- Don't try doing the same with `\looseness=1`
- It will result in many paragraphs with just a single word (or worse a partial word) on the last line!

Hook examples

Make my document shorter please

```
\AddToHook{para/begin}{\looseness=-1 }

\newcommand\cancellooseness
  {\AddToHookNext{para/begin}{\looseness=0 }}
```

Notes

- Don't try doing the same with `\looseness=1`
- It will result in many paragraphs with just a single word (or worse a partial word) on the last line!

Hook examples (cont.)

Record file nesting (from `structuredlog.sty`)

```
\AddToHook{file/before}
   { \__filehook_log_file_record:n { START } }
\AddToHookNext{file/after}
   { \AddToHook{file/after}
        { \__filehook_log_file_record:n { STOP } } }
```

Reorder code chunks in hooks

```
\DeclareHookRule{begindocument}{showkeys}{before}{nameref}
```

Dropping a code chunk

```
\DeclareHookRule{enddocument/info}
      {kernel/testmode}{voids}{kernel/release}
```

Hook examples (cont.)

Record file nesting (from structuredlog.sty)

```
\AddToHook{file/before}
   { \__filehook_log_file_record:n { START } }
\AddToHookNext{file/after}
   { \AddToHook{file/after}
        { \__filehook_log_file_record:n { STOP } } }
```

Reorder code chunks in hooks

```
\DeclareHookRule{begindocument}{showkeys}{before}{nameref}
```

Dropping a code chunk

```
\DeclareHookRule{enddocument/info}
      {kernel/testmode}{voids}{kernel/release}
```

Hook examples (cont.)

Record file nesting (from `structuredlog.sty`)

```
\AddToHook{file/before}
   { \__filehook_log_file_record:n { START } }
\AddToHookNext{file/after}
   { \AddToHook{file/after}
        { \__filehook_log_file_record:n { STOP } } }
```

Reorder code chunks in hooks

```
\DeclareHookRule{begindocument}{showkeys}{before}{nameref}
```

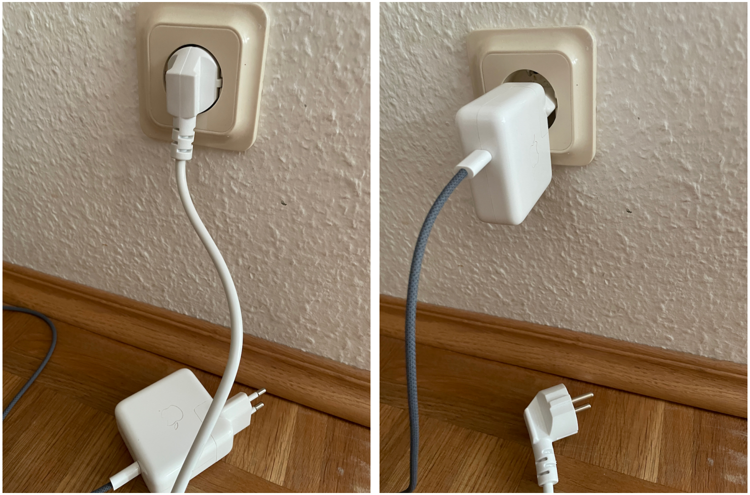
Dropping a code chunk

```
\DeclareHookRule{enddocument/info}
      {kernel/testmode}{voids}{kernel/release}
```

What are sockets?

And what are plugs?

And what are their characteristics?



Code sockets and code plugs

Characteristics

- A socket can have at most one plug inserted at any one time
- In analogy, socket code can be replaced but not augmented

LaTeX sockets and plugs

- A socket defines a named place in the code where a selection of alternatives can be "plugged in"
- These alternative for a socket are therefore called its "plugs"
- Each socket, and its selection of plugs, must be declared before use

Code sockets and code plugs

Characteristics

- A socket can have at most one plug inserted at any one time
- In analogy, socket code can be replaced but not augmented

LaTeX sockets and plugs

- A socket defines a named place in the code where a selection of alternatives can be “plugged in”
- These alternative for a socket are therefore called its “plugs”
- Each socket, and its selection of plugs, must be declared before use

Code sockets and code plugs (cont.)

Names

- Like hooks, sockets (i.e., their ⟨*names*⟩) have to be unique across the document
- Plug ⟨*names*⟩ have to be unique per socket

Defaults

- Each new socket has the plug `noop` plugged in. This means that the socket is ignored (with its arguments, if any)
- Exception: a new socket with exactly one argument has the plug `identity` plugged in, so that its argument is processed (after removing the outer braces)

Code sockets and code plugs (cont.)

Names

- Like hooks, sockets (i.e., their ⟨*names*⟩) have to be unique across the document
- Plug ⟨*names*⟩ have to be unique per socket

Defaults

- Each new socket has the plug `noop` plugged in. This means that the socket is ignored (with its arguments, if any)
- Exception: a new socket with exactly one argument has the plug `identity` plugged in, so that its argument is processed (after removing the outer braces)

Hook or Socket? — When to use which?

Use Hooks

- in places where (general) initialization can happen
- when additions from different packages are likely to be meaningful

Use Sockets

- when code has to be tightly controlled
- in typical "on/off" situations
- when supporting different processing models
  (i.e., one algorithm being replaced with another)

Hook or Socket? — When to use which?

Use Hooks

- in places where (general) initialization can happen
- when additions from different packages are likely to be meaningful

Use Sockets

- when code has to be tightly controlled
- in typical "on/off" situations
- when supporting different processing models (i.e., one algorithm being replaced with another)

Final advice — be careful with the wiring

- … otherwise your users will not know how to use it
- And don't go overboard with it — or it will slow things down

Final advice — be careful with the wiring

- . . . otherwise your users will not know how to use it
- And don't go overboard with it — or it will slow things down

The nitty gritty details (if time permits)

Time Check

The nitty gritty details (if time permits) — otherwise . . .

Documentation for hooks

- `texdoc lthooks-doc` main documentation
- `texdoc ltcmdhooks-doc` generic cmd/env hooks
- Supplementary documentation in
  - `ltfilehook-doc`,
  - `ltmarks-doc`,
  - `ltpara-doc`,
  - `ltshipout-doc`

Documentation for sockets

- `texdoc ltsockets-doc` main documentation

The nitty gritty details (if time permits) — otherwise . . .

Documentation for hooks

- `texdoc lthooks-doc` main documentation
- `texdoc ltcmdhooks-doc` generic cmd/env hooks
- Supplementary documentation in
  - `ltfilehook-doc`,
  - `ltmarks-doc`,
  - `ltpara-doc`,
  - `ltshipout-doc`

Documentation for sockets

- `texdoc ltsockets-doc` main documentation

The new hook management

Declaring hooks

- `\NewHook{`⟨*name*⟩`}`
- `\NewReversedHook{`⟨*name*⟩`}`
- `\NewHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}`
- ... plus a few more

Notes

- ⟨*name*⟩ has to be unique
  Best practice: ⟨*name*⟩ = ⟨*pkg*⟩ / ⟨*identifier*⟩
- Reversed hooks have the code chunks backwards
- ⟨*number*⟩ is the number of arguments

The new hook management

Declaring hooks

- `\NewHook{`⟨*name*⟩`}`
- `\NewReversedHook{`⟨*name*⟩`}`
- `\NewHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}`
- . . . plus a few more

Notes

- ⟨*name*⟩ has to be unique
  Best practice: ⟨*name*⟩ = ⟨*pkg*⟩ / ⟨*identifier*⟩
- Reversed hooks have the code chunks backwards
- ⟨*number*⟩ is the number of arguments

The new hook management

Using hooks

- `\UseHook{`⟨*name*⟩`}`
- `\UseOneTimeHook{`⟨*name*⟩`}`

Using hooks with arguments

- `\UseHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}{`⟨...⟩`}`...
- `\UseOneTimeHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}{`⟨...⟩`}`...

Notes

- Note that the ⟨*number*⟩ of arguments has to be explicitly given for hooks with arguments (for efficiency reasons)
- If a hook is empty it will be therefore bypassed with little overhead

The new hook management

Using hooks

- `\UseHook{`⟨*name*⟩`}`
- `\UseOneTimeHook{`⟨*name*⟩`}`

Using hooks with arguments

- `\UseHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}{`⟨...⟩`}`...
- `\UseOneTimeHookWithArguments{`⟨*name*⟩`}{`⟨*number*⟩`}{`⟨...⟩`}`...

Notes

- Note that the ⟨*number*⟩ of arguments has to be explicitly given for hooks with arguments (for efficiency reasons)
- If a hook is empty it will be therefore bypassed with little overhead

The new hook management

Using hooks

- `\UseHook{⟨name⟩}`
- `\UseOneTimeHook{⟨name⟩}`

Using hooks with arguments

- `\UseHookWithArguments{⟨name⟩}{⟨number⟩}{⟨...⟩}...`
- `\UseOneTimeHookWithArguments{⟨name⟩}{⟨number⟩}{⟨...⟩}...`

Notes

- Note that the ⟨*number*⟩ of arguments has to be explicitly given for hooks with arguments (for efficiency reasons)
- If a hook is empty it will be therefore bypassed with little overhead

The new hook management

Adding code to hooks

- `\AddToHook{`⟨*name*⟩`}[`⟨*label*⟩`]{`⟨*code*⟩`}`
- `\AddToHookNext{`⟨*name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*label*⟩ identifies the code chunk
  default: package/class name; on document-level: `toplevel`
- You can use both commands with hooks taking arguments (if you are not referring to them)
- You can add to a hook that is not yet declared!
- If you add to a one-time hook after it was used, then ⟨*code*⟩ is used immediately

The new hook management

Adding code to hooks

- `\AddToHook{`⟨*name*⟩`}[`⟨*label*⟩`]{`⟨*code*⟩`}`
- `\AddToHookNext{`⟨*name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*label*⟩ identifies the code chunk
  default: package/class name; on document-level: `toplevel`
- You can use both commands with hooks taking arguments (if you are not referring to them)
- You can add to a hook that is not yet declared!
- If you add to a one-time hook after it was used, then ⟨*code*⟩ is used immediately

The new hook management

Adding code to hooks with arguments

- `\AddToHookWithArguments{`⟨*name*⟩`}[`⟨*label*⟩`]{`⟨*code*⟩`}`
- `\AddToHookNextWithArguments{`⟨*name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*code*⟩ can contain #1, #2, …
- Real #'s have to be doubled, i.e., entered as ##
- You can't add to a one-time hook this way after it was used

The new hook management

Adding code to hooks with arguments

- `\AddToHookWithArguments{`⟨*name*⟩`}[`⟨*label*⟩`]{`⟨*code*⟩`}`
- `\AddToHookNextWithArguments{`⟨*name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*code*⟩ can contain #1, #2, . . .
- Real #'s have to be doubled, i.e., entered as ##
- You can't add to a one-time hook this way after it was used

The new hook management

Remove code from hooks

- `\RemoveFromHook{`⟨*name*⟩`}[`⟨*label*⟩`]`

Notes

- Without the optional argument the default ⟨*label*⟩ is used
- Special case: `[*]` remove all code (naughty)

The new hook management

Remove code from hooks

- `\RemoveFromHook{`⟨*name*⟩`}[`⟨*label*⟩`]`

Notes

- Without the optional argument the default ⟨*label*⟩ is used
- Special case: `[*]` remove all code (naughty)

The new hook management

Displaying the hook status

- \ShowHook{⟨*name*⟩}   or   \LogHook{⟨*name*⟩}

Output example

```
-> The hook 'enddocument':
> Code chunks:
>     pgfcore -> \ifpgf@external@grabshipout ...
>     beamerbasemisc -> \clearpage  ...
>     csquotes -> \ifnum \csq@qlevel >\z@ \csq@err@gleft \fi
> Document-level (top-level) code (executed last):
>     ---
> Extra code for next invocation:
>     ---
> Rules:
>     ---
> Execution order:
>     pgfcore, beamerbasemisc, csquotes.
```

The new hook management

Displaying the hook status

- \ShowHook{⟨*name*⟩} or \LogHook{⟨*name*⟩}

Output example

```
-> The hook 'enddocument':
> Code chunks:
>     pgfcore -> \ifpgf@external@grabshipout ...
>     beamerbasemisc -> \clearpage  ...
>     csquotes -> \ifnum \csq@qlevel >\z@ \csq@err@gleft \fi
> Document-level (top-level) code (executed last):
>     ---
> Extra code for next invocation:
>     ---
> Rules:
>     ---
> Execution order:
>     pgfcore, beamerbasemisc, csquotes.
```

The new socket management

Declaring sockets and plugs

- `\NewSocket{`⟨*socket-name*⟩`}{`⟨*number-of inputs*⟩`}`
- `\NewSocketPlug{`⟨*socket-name*⟩`}`
  `{`⟨*socket-plug-name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*socket-name*⟩ has to be unique
  Best practice: ⟨*name*⟩ = ⟨*pkg*⟩ / ⟨*identifier*⟩
- ⟨*socket-plug-name*⟩ has to be unique per socket
  but can be reused in different sockets, e.g., `noop`

The new socket management

Declaring sockets and plugs

- `\NewSocket{`⟨*socket-name*⟩`}{`⟨*number-of inputs*⟩`}`
- `\NewSocketPlug{`⟨*socket-name*⟩`}`
  `{`⟨*socket-plug-name*⟩`}{`⟨*code*⟩`}`

Notes

- ⟨*socket-name*⟩ has to be unique
  Best practice: ⟨*name*⟩ = ⟨*pkg*⟩ / ⟨*identifier*⟩
- ⟨*socket-plug-name*⟩ has to be unique per socket
  but can be reused in different sockets, e.g., `noop`

The new socket management

Assigning plugs to sockets

- `\AssignSocketPlug{`⟨*socket-name*⟩`}{`⟨*socket-plug-name*⟩`}`
- Default assignments are
  - `identity` for sockets with one argument
  - `noop` for all others

Showing sockets

- `\ShowSocket{`⟨*socket-name*⟩`}` or `\LogSocket{`⟨*socket-name*⟩`}`

Using sockets

- `\UseSocket{`⟨*socket-name*⟩`}`
  - Number of arguments is implicit in LaTeX 2_ε
    but explicit in L3 layer, e.g., `\socket_use:nn`

The new socket management

Assigning plugs to sockets

- `\AssignSocketPlug{`⟨*socket-name*⟩`}{`⟨*socket-plug-name*⟩`}`
- Default assignments are
  - `identity` for sockets with one argument
  - `noop` for all others

Showing sockets

- `\ShowSocket{`⟨*socket-name*⟩`}` or `\LogSocket{`⟨*socket-name*⟩`}`

Using sockets

- `\UseSocket{`⟨*socket-name*⟩`}`
  - Number of arguments is implicit in LaTeX 2$_\varepsilon$
    but explicit in L3 layer, e.g., `\socket_use:nn`

The new socket management

Assigning plugs to sockets

- `\AssignSocketPlug{`⟨*socket-name*⟩`}{`⟨*socket-plug-name*⟩`}`
- Default assignments are
  - `identity` for sockets with one argument
  - `noop` for all others

Showing sockets

- `\ShowSocket{`⟨*socket-name*⟩`}` or `\LogSocket{`⟨*socket-name*⟩`}`

Using sockets

- `\UseSocket{`⟨*socket-name*⟩`}`
  - Number of arguments is implicit in LaTeX 2_ε but explicit in L3 layer, e.g., `\socket_use:nn`