memoize-ext: Extensions for memoize

Clea F. Rees*

11782 2026-01-19

Abstract

memoize-ext provides extensions for Živanović's package memoize (2024). In particular, it supports the memoization of content in tagged PDFs and presentations produced with Wright's ltx-talk (2026).

Contents

*Bug tracker: codeberg.org/cfr/memoize-ext/issues | Code: codeberg.org/cfr/memoize-ext | Mirror: github.com/cfr42/memoize-ext

Part I
Usage

1 Basics

Usage is simple.

`\documentclass{`⟨*class*⟩`}`

`\usepackage{memoize-ext}`

The package will automatically load memoize and pass any unrecognised options onto that package.

Note that to create a tagged presentation with ltx-talk, the package should be loaded *after* the class[1].

```
\DocumentMetadata{tagging=on,lang=en-GB,pdfversion=2.0,pdfstandard=UA-2,}
\documentclass{ltx-talk}
\usepackage{memoize-ext}
```

If for any reason it is necessary to load the package *prior* to the class in a tagged PDF, then tagging must be explicitly requested. For example,

```
\DocumentMetadata{tagging=on,lang=cy,pdfversion=2.0,pdfstandard=ua-2,}
\RequirePackage[tag]{memoize-ext}
\documentclass{book}
```

If necessary, a small number of package options are available to customise which code is loaded.

`expl3` (*opt.*) `=true|false`

Loads code supporting expl3 syntax.

Default is `true`. Initially `false`.

`l3draw` (*opt.*) `=true|false`

Loads code supporting l3draw, if the package is loaded.

Default is `true`. Initially `true`.

`tag` (*opt.*) `=true|false`

Loads code supporting tagged PDF, if LaTeX's tagging code is activated when the package is loaded. Note that this is *not* true prior to `\documentclass`.

Default is `true`. Initially `true` if tagging is activated; `false` otherwise.

`talk` (*opt.*) `=true|false`

Loads code supporting ltx-talk, if the class is loaded.

[1] Note that ltx-talk diverges from beamer here, a point to which I was oblivious when I wrote the initial version of this documentation.

Default is `true`. Initially `true`.

Note that the additional code is not loaded if a different class is used, regardless of this setting. The option is provided in case it is necessary to disable support for the class, without disabling other parts of memoize-ext.

2 expl3

`replicate expl fn` (*pgfkey*) Sets up advice to 'replicate' an expl3 function.

This works similarly to the builtin support for commands created with `\NewDocumentCommand` etc. This means that it is not necessary to specify `args`.

Functions with w-type arguments are NOT supported. Attempting to use this key with such a function will result in an error. Such cases require custom handling and can be configured using the standard memoize keys.

`memoize-ext-l3draw.sty` demonstrates use of `replicate expl fn`:

```
\mmzset{%
  auto~csname={draw_begin:}{memoize,collector=\AdviceCollectDrawArguments,},
  auto~csname={__draw_record_origin:}{run~if~memoizing,replicate~expl~fn,},
}
```

This code sets up a custom 'collector' and installs 'advice' which memoizes `\draw_begin:`. It further advises `\__draw_record_origin:` so that if this function is found during memoization, it will be replicated in the `ccmemo`. This ensures that the origin is recorded correctly when the memoized picture is utilised, since we do not want to record its position when memoized.

The net result of this is that auto-memoization of l3draw pictures should (hopefully) 'just work'.

3 l3draw

sec:draw «< l3draw pictures are auto-memoized by default. `memoize-ext-l3draw.sty` mostly exists to demonstrate use of `replicate expl fn`. »> sec:draw

4 ltx-talk

The code is based on that provided by memoize for beamer and supports the same options, except that '`talk`' is substituted for '`bemaer`'.

`per overlay` (*pgfkey*) Equivalent to the beamer option of the same name.

`talk mode to prefix` (*pgfkey*) Equivalent to beamer mode to prefix.

The code uses and/or changes internal code from both ltx-talk and memoize. While the public interface for memoize is fairly stable, the internals may not be, and ltx-talk is highly experimental. The latter also uses a large number of experimental packages and makes extensive use of experimental LaTeX features.

The justification for publishing this part of memoize-ext is essentially that anybody using ltx-talk and memoize is already playing with fire, so it is better to have an unreliable extinguisher to hand than none at all.

A few things you should know, even if you do not want to:

- the code uses an internal ltx-talk boolean to drive extern creation and utilisation;
- `talk mode to prefix` relies on an internal ltx-talk string;
- to workaround incompatibilities between memoize and pdfmanagement, the code redefines an internal memoize macro[2].

5 Tagged pdf

If using the package to produce tagged PDF, note that the tagging support

- redefines the internal macro `\mmzIncludeExtern` during utilisation;
- relies on an internal string variable in ltsockets.

Correct tagging *requires* that unmemoizable code be marked as such, either manually or automatically. The package does this automatically for `tikzpictures` which use an unsupported tagging plug, but does nothing in any other case. So if your picture uses `remember picture`, for example, you *must* mark the code as unmemoizable or disable tagging for the affected code. The package will warn you about this, but that is all it does.

5.1 Ti*k*Z pictures

If the content you wish to memoize is a Ti*k*Z picture, you probably do not need to do anything special, but note that the default latex-lab plug is *not* supported. You must use one of `alt`, `actualtext` or `artifact`.

If you use alt or `actualtext` in the optional argument to `tikzpicture`, the value will be recorded in the *ccmemo* for use during utilisation. If you set the value outside the `tikzpicture`, this is not necessary. In the latter case, the *extern* will not depend on the value given (unless you request that specifically).

Note that if you change the selected plug *and* you set this *outside* the picture, you must manually tell memoize it should recompile the picture, since the plug is recorded in the ccmemo, but the hash will not have changed.

Note that tagging is disabled during memoization and additionally *disabled for content which has just been memoizsed*. So when a run produces an extern, the memoized code will not be tagged at all.

[2]The redefinition injects code into the box memoize ships out which resets opacity before and after the memoized code is executed. This is required because memoize relies on primitive shipout, whereas the implementation of opacity in pdfmanagement relies on LaTeX's shipout routine.

Note that this package does *not* support forest. If your document uses forest (Živanović 2017), you should either disable memoization for these pictures or load forest-ext[3] (Rees 2026).

The Ti*k*Z support is implemented by replacing plugs provided by latex-lab with versions designed for memoized content (LaTeX Project 2025b). Code is also installed into the same hooks latex-lab uses with rules to ensure this package's has priority.

`mmzx` (*plug*) Plug for `tagsupport/tikz/picture/init`

If memoization is not active, the plug executes the latex-lab `default` plug.

If some option for this package is specifically configured, it is used. Otherwise, the code initialisation code at the start of the picture attempts to find a match for any configured latex-lab plug. In effect, this means that you should not need to change anything in your document if you use one of the three supported plugs.

If memoization is enabled but no suitable plug is found, a warning is issued and memoization aborted. Otherwise, code is inserted into the ccmemo to emulate the appropriate latex-lab plug. In most cases, this code simply calls the relevant latex-lab plugs.

Plugs for `tagsupport/tikz/picture/begin` and `tagsupport/tikz/picture/end`:

`mmzx/actualtext` (*plug*) Sets up the ccmemo to use the latex-lab `actualtext` plugs.

`mmzx/artifact` (*plug*) Sets up the ccmemo to use the latex-lab `artifact` plugs.

`mmzx/alt` (*plug*) This pair of plugs is the exception. Rather than writing a ccmemo which will invoke the latex-lab `alt` plugs, these plugs write a ccmemo which uses an alternative implementation of those plugs. The reimplementation uses *properties* (provided by the LaTeX format) rather than *rememberpicture* (provided by PGF/Ti*k*Z)[4].

If everything looks OK, tagging is disabled for the current picture. This is efficient if memoization is successful, but may be problematic if memoization is aborted or fails. In this case, it may be necessary to mark the content as unmemoizable or to disable memoization for particular pictures, in order to ensure content is tagged correctly[5].

5.2 Other content

If the content you wish to memoize is *not* a Ti*k*Z picture, you may need to read the remainder of this section.

Generic support is provided in the form of two sockets which are used directly before and directly after an extern is included during utilisation. By default, the sockets do nothing, but they may be used to inject code which wraps the included extern in a suitable tagging structure.

Plugs may be assigned to the sockets either by writing suitable code to the ccmemo or in the document itself. The Ti*k*Z support, for example, writes commands to the ccmemo which assign plugs analogous to the latex-lab plugs available for non-memoized pictures.

[3] This is not necessary if you use prooftrees, which will load the package automatically if required.

[4] I considered using the support provided for `\includegraphic`, but this would require more intrusive changes to the internals of memoize and would essentially duplicate bounding box calculations already completed during memoization.

[5] It would be possible to disable tagging only if memoization succeeds, but I am not sure whether the structure will be right in this case?

`tagsupport/memoize/include/extern/before`

(*socket*) This socket receives three arguments during extern utilisation: the width, height and depth of the memoized content. The `alt` plug for Ti*k*Z, for example, uses these values to calculate the bounding box required to create a `Figure` structure with `alt` text.

This socket is used just before the extern is included in the document.

`tagsupport/memoize/include/extern/after`

(*socket*) This socket absorbs no arguments. During extern utilisation, it is used immediately after inclusion of an extern.

Part II

Implementation

A double underscore (`__`) *or* an 'at' (`@`) indicates an internal macro or key. These are liable to change without notice and should not be used elsewhere. Some additional macros are categorised in the same way, but are named differently to simplify use in memos[6].

memoize-ext

<*sty> <@@=mmzx>

```
\NeedsTeXFormat{LaTeX2e}[2021-11-15]%
```

copied verbatim, excepting format from Joseph Wright's `siunitx.sty` under LPPL

```
\@ifundefined{ExplLoaderFileDate}{%
  \RequirePackage{expl3}%
}{}
```

almost verbatim from `siunitx.sty`

should check date requirement (copied from chronos)

```
\@ifl@t@r\ExplLoaderFileDate{2022-02-24}{%
}{%
  \PackageError{memoize-ext}{Support package expl3 too old}
  {%
    You need to update your installation of the bundles 'l3kernel' and
    'l3packages'.\MessageBreak
    Loading memoize-ext will abort!%
  }%
  \endinput
}%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\GetIdInfo $Id: memoize-ext.dtx 11782 2026-03-17 14:49:00Z cfrees $ {Extensions for
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName}{\ExplFileDate}
⟨!debug⟩     {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
```

[6] This follows memoize's own practice.

```
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-debug}
⟨debug⟩      {\ExplFileDate}{v0.3.4 \ExplFileVersion}{\ExplFileDescription}
%
\str_new:N \g__mmzx_name_str
\str_gset:NV \g__mmzx_name_str \ExplFileName
%
⟨!debug⟩    \disable@package@load {memoize-ext-debug}
⟨debug⟩    \disable@package@load {memoize-ext}
{
  Only one of memoize-ext and memoize-ext-debug should be loaded.
  Since
⟨!debug⟩    memoize-ext
⟨debug⟩    memoize-ext-debug
   had been loaded,I will ignore your request for
⟨debug⟩    memoize-ext
⟨!debug⟩    memoize-ext-debug
.}
\SetDefaultHookLabel{memoize-ext}
```

`\l__mmzx_opt_tag_bool` (*var.*) Set according to activation status by default.

```
\bool_new:N \l__mmzx_opt_tag_bool
\tag_if_active:TF
  { \bool_set_true:N \l__mmzx_opt_tag_bool }
  { \bool_set_false:N \l__mmzx_opt_tag_bool }
```

`l__mmzx_opt_draw_bool` (*var.*)
`l__mmzx_opt_expl_bool` (*var.*) Other bools.

```
\keys_define:nn {memoize-ext}
{
⟨*!debug⟩
  debug       .code:n     = {
    \PackageWarning{memoize-ext}{
      To load the debugging code,use memoize-ext-debug instead of this package.
    }
  },
⟨/!debug⟩
  expl3       .bool_set:N = \l__mmzx_opt_expl_bool,
  expl3       .default:n  = true,
  expl3       .initial:n  = false,
  l3draw      .bool_set:N = \l__mmzx_opt_draw_bool,
  l3draw      .default:n  = true,
  l3draw      .initial:n  = true,
  tag         .bool_set:N = \l__mmzx_opt_tag_bool,
  tag         .default:n  = true,
  talk        .bool_set:N = \l__mmzx_opt_talk_bool,
  talk        .default:n  = true,
  talk        .initial:n  = true,
}
\DeclareUnknownKeyHandler{
  \PassOptionsToPackage{\CurrentOption}{memoize}
}
```

`\IfFormatAtLeastTF` Joseph Wright: from `siunitx.sty` ; https://chat.stackexchange.com/transcript/message/

64327823#64327823

```
\providecommand \IfFormatAtLeastTF { \@ifl@t@r \fmtversion }

\IfFormatAtLeastTF { 2022-06-01 }
{
  \ProcessKeyOptions [ memoize-ext ]
}{
  \RequirePackage { l3keys2e }
  \ProcessKeysOptions { memoize-ext }
}

\IfFormatAtLeastTF { 2020-10-01 }{
}{
  \RequirePackage { xparse }
  \providecommand \ExpandArgs [1]
  { \cs_if_exist_use:c { exp_args:N #1 } }
}
```

Should specify next version here, most probably. Or conditionalise input switch for ccmemos?

```
\RequirePackage{memoize}
⟨debug⟩    \mmzset{
⟨debug⟩      trace,
⟨debug⟩      include context in ccmemo,
⟨debug⟩    }
```

Fix for `\toksapp` and friends courtesy of Max Chernoff https://github.com/sasozivanovic/memoize/pull/57/commits.

```
\sys_if_engine_luatex:F
{
  \clist_map_inline:nn {\toksapp,\etoksapp,\gtoksapp,\xtoksapp}
  {
    \tl_if_head_eq_meaning:VNF #1 \protected
    {
      \expandafter\protected\expandafter\def\expandafter#1\expandafter{#1}
    }
  }
}
```

temporary variables, quarks

```
\bool_new:N \l__mmzx_tmpa_bool
\fp_new:N \l__mmzx_tmpa_fp
\int_new:N \l__mmzx_tmpa_int
\quark_new:N \q__mmzx_stop
\tl_new:N \l__mmzx_tmpa_tl
\tl_new:N \l__mmzx_tmpb_tl
\tl_new:N \l__mmzx_tmpc_tl
\seq_new:N \l__mmzx_tmpa_seq
\str_new:N \l__mmzx_tmpa_str
\str_new:N \l__mmzx_tmpb_str
⟨*debug⟩
\cs_new_protected:Npn \__mmzx_debug:n #1
{
```

```
  \iow_log:n {[mmzx debug]:: #1}
}
\cs_generate_variant:Nn \__mmzx_debug:n {e}
\cs_new_protected:Npn \__mmzx_debug:N #1
{
  \__mmzx_debug:e {\cs_to_str:N #1: \exp_args:NV \exp_not:n #1}
}
⟨/debug⟩
```

`\__mmzx_noop:` `\__mmzx_noop:n` Do nothing successfully.

```
\cs_new:Npn \__mmzx_noop: {}
\cs_new:Npn \__mmzx_noop:n {}
```

tag, expl, l3draw, talk loaded conditionally

```
\bool_if:NT \l__mmzx_opt_tag_bool
{
⟨!debug⟩      \RequirePackage{\g__mmzx_name_str -tag}
⟨debug⟩      \RequirePackage{\g__mmzx_name_str -tag-debug}
  \hook_gput_code:nnn {package/forest/after}{.}
  {
    \hook_gput_code:nnn {begindocument/before}{.}
    {
      \IfPackageLoadedF {forest-lib-ext.tagging}
      {
        \IfPackageLoadedF {forest-lib-ext.tagging-debug}
        {
          \msg_warning:nnnnnn {memoize-ext}{unsupported}{forest}
          {forest-lib-ext.tagging.sty}{forest-ext}
          {forest trees will not be correctly tagged and may cause fatal
          compilation errors.}
        }
      }
    }
  }
}
```

memoize-ext-expl3[-debug]

```
\bool_if:NT \l__mmzx_opt_expl_bool
{
⟨!debug⟩      \RequirePackage{\g__mmzx_name_str -expl3}
⟨debug⟩      \RequirePackage{\g__mmzx_name_str -expl3-debug}
}
```

memoize-ext-l3draw[-debug]

```
\hook_gput_code:nnn {package/l3draw/after}{.}
{
  \bool_if:NT \l__mmzx_opt_draw_bool
  {
⟨!debug⟩        \RequirePackage {\g__mmzx_name_str -l3draw}
⟨debug⟩      \__mmzx_debug:n {Loading memoize-ext-l3draw-debug.}
⟨debug⟩        \RequirePackage {\g__mmzx_name_str -l3draw-debug}
  }
}
```

memoize-ext-talk[-debug]

```
\hook_gput_code:nnn {class/ltx-talk/after} {.}
{
  \bool_if:NT \l__mmzx_opt_talk_bool
  {
⟨!debug⟩      \RequirePackage{memoize-ext-talk}
⟨debug⟩       \__mmzx_debug:n {Loading memoize-ext-talk-debug.}
⟨debug⟩       \RequirePackage{memoize-ext-talk-debug}
  }
}
```

NaCl | halen | salt

```
\toksapp\mmzSalt{
  Tagging status: \tag_if_active_p:
}
```

messages

```
\msg_new:nnnn {memoize-ext}{unsupported}
{
  \msg_warning_text:n {memoize-ext}:
  Non-existent or inappropriate version of #2 from #3 \msg_line_context:.
  #4
} {
  memoize-ext#1 requires an appropriate version of #2 from #3.
}
```

</sty>

memoize-ext-expl3

Clea F. Rees

11782 2026-01-19

Abstract

Provides memoize-ext-expl3 and memoize-ext-expl3-common. Part of memoize-ext.

Contents

<@@=mmzx> <*common>

```
\GetIdInfo $Id: memoize-ext-expl3.dtx 11782 2026-03-17 14:49:00Z cfrees $ {Extensio
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName-common}{\ExplFileDate}{v0.0 %
⟨!debug⟩     \ExplFileVersion}{\ExplFileDescription}
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-common-debug}{\ExplFileDate}{v0.0 %
⟨debug⟩      \ExplFileVersion}{\ExplFileDescription}
%
⟨!debug⟩    \disable@package@load {memoize-ext-expl3-common-debug}
⟨debug⟩    \disable@package@load {memoize-ext-expl3-common}
{ Only one of memoize-ext-expl3-common and memoize-ext-expl3-common-debug
  should be loaded.
  Since
⟨!debug⟩    memoize-ext-expl3-common
⟨debug⟩    memoize-ext-expl3-common-debug
   has been loaded,I will ignore your request for
⟨debug⟩    memoize-ext-expl3-common
⟨!debug⟩    memoize-ext-expl3-common-debug
.}

⟨!debug⟩    \RequirePackage{memoize-ext}
⟨debug⟩    \RequirePackage{memoize-ext-debug}
%
```

We don't want inconsistent names in hooks.

```
\SetDefaultHookLabel{memoize-ext}
```

`\l__mmzx_replicating_bool` (*var.*) Internal variable to track whether currently replicating.

```
\bool_new:N \l__mmzx_replicating_bool
\bool_set_false:N \l__mmzx_replicating_bool
```

`mmzx_if_replicating:TF` (*fn.*)
`mmzx_if_replicating_p:` (*fn.*)

```
\prg_new_conditional:Npnn \__mmzx_if_replicating: {p,T,TF,F}
{
  \if_bool:N \l__mmzx_replicating_bool
⟨debug⟩    \__mmzx_debug:n {Replicating true.}
    \prg_return_true:
  \else:
⟨debug⟩    \__mmzx_debug:n {Replicating false.}
    \prg_return_false:
  \fi:
}
```

`\AdviceRunIfNotReplicating` Run condition.

```
\cs_new:Npn \AdviceRunIfNotReplicating
{
  \__mmzx_if_replicating:F
  {
⟨debug⟩    \__mmzx_debug:n {Not replicating,so proceeding.}
    \ifmemoizing \AdviceRuntrue \fi
  }
}
```

`if not replicating` (*pgfkey*) Style.

```
\mmzset{
  auto/run if not replicating/.style = {
    run conditions={\AdviceRunIfNotReplicating},
  },
}
```

Constants

```
\cctab_const:Nn \c__mmzx_expl_at_cctab {
  \cctab_select:N \c_code_cctab
  \makeatletter
}
\cctab_const:Nn \c__mmzx_nexpl_at_cctab {
  \cctab_select:N \c_code_cctab
  \makeatletter
  \int_set:Nn \tex_endlinechar:D { 13 }
  \char_set_catcode_space:n      { 9 }
  \char_set_catcode_space:n      { 32 }
  \char_set_catcode_active:n     { 126 } % tilde
}
```

expl3, memoizing côd ynddo

hooks instead of TEX SE: David Carlisle

`\l__mmzx_expl_bool` (*var.*) A boolean to track whether `expl3` syntax is active or not. yn lle ateb | in place of 748807 by David Carlisle.

```
\bool_new:N \l__mmzx_expl_bool
```

`_restore_ccmemo_input:` (*fn.*)
`saved_mmzxExplAtBegin:` (*fn.*)
`x_saved_mmzxExplAtEnd:` (*fn.*)

Initialise.

```
\cs_new_protected_nopar:Npn \__mmzx_restore_ccmemo_input: {}
\cs_new_protected_nopar:Npn \__mmzx_saved_mmzxExplAtBegin: {}
\cs_new_protected_nopar:Npn \__mmzx_saved_mmzxExplAtEnd: {}
```

Auto-switching for `expl3` syntax.

```
\hook_gput_code:nnn {begindocument/end}{.}
{
  \bool_set_false:N \l__mmzx_expl_bool
}

\hook_gput_code:nnn {begindocument/end}{.}
{
⟨debug⟩    \__mmzx_debug:n {Tracking expl3 syntax changes.}
  \bool_set_false:N \l__mmzx_expl_bool
  \hook_gput_code:nnn {cmd/ExplSyntaxOn/before} { . }
  {
    \bool_if:NF \l__mmzx_expl_bool
    {
      \bool_set_true:N \l__mmzx_expl_bool
      \ifmmz@direct@ccmemo@input
        \relax
      \else
        \cs_set_protected_nopar:Npn \__mmzx_restore_ccmemo_input:
        {
          \mmz@direct@ccmemo@inputfalse
        }
      \fi
      \mmz@direct@ccmemo@inputtrue
      \cs_set_eq:NN \__mmzx_saved_mmzxExplAtBegin: \mmzxExplAtBegin
      \cs_set_eq:NN \__mmzx_saved_mmzxExplAtEnd: \mmzxExplAtEnd
      \__mmzx_expl_at_start:
    }
  }
```

`\ExplSyntaxOn` rewrites `\ExplSyntaxOff`, so it doesn't work to add hook code to `\ExplSyntaxOff` directly.

```
  \hook_gput_code:nnn {cmd/ExplSyntaxOn/after} { . }
  {
    \hook_gput_code:nnn {cmd/ExplSyntaxOff/before} { . }
    {
      \bool_set_false:N \l__mmzx_expl_bool
      \cs_set_eq:NN \mmzxExplAtBegin \__mmzx_saved_mmzxExplAtBegin:
      \cs_set_eq:NN \mmzxExplAtEnd \__mmzx_saved_mmzxExplAtEnd:
      \__mmzx_restore_ccmemo_input:
    }
  }
}
```

fns mewnol

`\__mmzx_cctab_end:` (*fn.*) just for symmetry ...

```
\cs_new_eq:NN  \__mmzx_cctab_end: \cctab_end:
```

`\__mmzx_expl_at_begin:` (*fn.*)
`__mmzx_nexpl_at_begin:` (*fn.*)
`\__mmzx_expl_at_start:` (*fn.*)
`__mmzx_nexpl_at_start:` (*fn.*)
`\__mmzx_cctab_stop:` (*fn.*)

Convenience wrappers for cat code changes.

```
\cs_new_protected_nopar:Npn \__mmzx_expl_at_begin:
{
  \cctab_begin:N \c__mmzx_expl_at_cctab
}
\cs_new_protected_nopar:Npn \__mmzx_nexpl_at_begin:
{
  \cctab_begin:N \c__mmzx_nexpl_at_cctab
}
\cs_new_nopar:Npn \__mmzx_expl_at_start:
{
  \cs_set_nopar:Npn \mmzxExplAtBegin {__mmzx_expl_at_begin:}
  \cs_set_nopar:Npn \mmzxExplAtEnd {__mmzx_cctab_end:}
}
\cs_new_nopar:Npn \__mmzx_nexpl_at_start:
{
  \cs_set_nopar:Npn \mmzxExplAtBegin {__mmzx_nexpl_at_begin:}
  \cs_set_nopar:Npn \mmzxExplAtEnd {__mmzx_cctab_end:}
}
\cs_new_protected_nopar:Npn \__mmzx_cctab_stop:
{
  \cs_set_nopar:Npn \mmzxExplAtBegin {relax}
  \cs_set_nopar:Npn \mmzxExplAtEnd {relax}
}
```

toks memoize (cyhoeddus yn unig)

The code here is constant but the meaning changes, so what is added to the memos reflects the configuration at the time.

```
\appto\mmzAtBeginMemoization{
⟨debug⟩    \__mmzx_debug:n {Appending expl begin to ccmemo at end
⟨debug⟩      \string\mmzAtBeginMemoization.}
  \xtoksapp\mmzCCMemo
  {
    \exp_not:N \csname \mmzxExplAtBegin \exp_not:N \endcsname
  }
⟨debug⟩    \exp_args:No \__mmzx_debug:n {\the\mmzCCMemo}
}
⟨debug⟩    \cs_log:N \mmzAtBeginMemoization
\preto\mmzAtEndMemoization{
⟨debug⟩    \__mmzx_debug:n {Appending expl end to ccmemo at start
⟨debug⟩      \string\mmzAtEndMemoization.}
  \xtoksapp\mmzCCMemo
  {
    \exp_not:N \csname \mmzxExplAtEnd \exp_not:N \endcsname
  }
}
⟨debug⟩    \cs_log:N \mmzAtEndMemoization
```

init

```
\hook_gput_code:nnn { begindocument/end } {mmzx}
{
% % % % % % % % %     \__mmzx_cctab_stop:
% }
```

</common>

<*sty>

```
\GetIdInfo $Id: memoize-ext-expl3.dtx 11782 2026-03-17 14:49:00Z cfrees $ {Extensio
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName}{\ExplFileDate}
⟨!debug⟩     {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-debug}{\ExplFileDate}
⟨debug⟩      {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
%
⟨!debug⟩    \disable@package@load {memoize-ext-expl3-debug}
⟨debug⟩    \disable@package@load {memoize-ext-expl3}
{ Only one of memoize-ext-expl3 and memoize-ext-expl3-debug
  should be loaded.
  Since
⟨!debug⟩    memoize-ext-expl3
⟨debug⟩    memoize-ext-expl3-debug
   has been loaded,I will ignore your request for
⟨debug⟩    memoize-ext-expl3
⟨!debug⟩    memoize-ext-expl3-debug
.}

⟨!debug⟩    \RequirePackage{memoize-ext}
⟨debug⟩    \RequirePackage{memoize-ext-debug}
⟨!debug⟩    \RequirePackage{memoize-ext-expl3-common}
⟨debug⟩    \RequirePackage{memoize-ext-expl3-common-debug}
%
```

We don't want inconsistent names in hooks.

```
\SetDefaultHookLabel{memoize-ext}
```

expl3 replicators

todo: figure out how to maintain correct grouping ...

`\g__mmzx_expl_replicate__bb_tl` (*var.*)
`\g__mmzx_expl_replicate__ba_tl` (*var.*)
`\g__mmzx_expl_replicate__tb_tl` (*var.*)

Variables

```
\tl_new:N \g__mmzx_expl_replicate__bb_tl
\tl_new:N \g__mmzx_expl_replicate__ba_tl
\tl_new:N \g__mmzx_expl_replicate__tb_tl
\tl_gput_right:NV \g__mmzx_expl_replicate__bb_tl \c_left_brace_str
\tl_gput_right:Nn \g__mmzx_expl_replicate__bb_tl {#}
\tl_gput_right:NV \g__mmzx_expl_replicate__ba_tl \c_right_brace_str
\tl_gput_right:Nn \g__mmzx_expl_replicate__tb_tl {#}
```

`l_replicate_fn_aux:nnN` (*fn.*)
`_expl_replicate__aux:n` (*fn.*)

Generic auxiliary functions for replication. The first does the actual replicating; the second delegates details according to the argument specification.

```
\cs_new:Npn \__mmzx_expl_replicate_fn_aux:nnN #1#2#3
{
  \cs_if_exist:cF { __mmzx_rep_#1:#2 }
  {
    \int_zero:N \l__mmzx_tmpa_int
    \tl_clear:N \l__mmzx_tmpa_tl
    \tl_clear:N \l__mmzx_tmpc_tl
    \tl_map_function:nN {#2} \__mmzx_expl_replicate__aux:n
```

```
    \cs_if_exist:cF { __mmzx_rep_#1:\l__mmzx_tmpc_tl }
    {
      \cs_gset_protected:ce {__mmzx_rep_#1:\l__mmzx_tmpc_tl}
      {
        \xtoksapp\mmzCCMemo{
          \exp_after:wN \exp_not:N \cs:w #1:#2 \cs_end: \l__mmzx_tmpa_tl
        }
        \exp_not:N \expandonce \exp_not:N \AdviceOriginal \l__mmzx_tmpa_tl
        \exp_not:N \group_end:
        \__mmzx_tmp_cctab_stop:
      }
    }
    \str_if_eq:eeF {#2}{\l__mmzx_tmpc_tl}
    { % ych!
      \cs_new_eq:cc {__mmzx_rep_#1:#2} {__mmzx_rep_#1:\l__mmzx_tmpc_tl}
    }
  }
  \use:c { __mmzx_rep_#1:#2 }
}
\cs_new:Npn \__mmzx_expl_replicate__aux:n #1
{
  \int_incr:N \l__mmzx_tmpa_int
  \str_case:nnF { #1 }
  {
    {c} { \__mmzx_expl_replicate__b: }
    {e} { \__mmzx_expl_replicate__b: }
    {o} { \__mmzx_expl_replicate__b: }
    {p} { }
    {n} { \__mmzx_expl_replicate__b: }
    {v} { \__mmzx_expl_replicate__b: }
    {D} { }
    {F} { \__mmzx_expl_replicate__b: }
    {N} { \__mmzx_expl_replicate__t: }
    {T} { \__mmzx_expl_replicate__b: }
    {V} { \__mmzx_expl_replicate__t: }
    {w} { \__mmzx_expl_replicate__e: }
  }{
    \__mmzx_expl_replicate__e:
  }
}
```

`mzx_expl_replicate__b:` (*fn.*)
`mzx_expl_replicate__t:` (*fn.*)
`mzx_expl_replicate__e:` (*fn.*)

Type-specific auxiliaries. We don't need to be very specific here. We just need to distinguish argument specifiers which expect braced groups from those which expect single tokens and from those we cannot automate.

```
\cs_new_nopar:Npn \__mmzx_expl_replicate__b:
{
  \tl_put_right:Nn \l__mmzx_tmpc_tl {n}
  \tl_put_right:NV \l__mmzx_tmpa_tl \g__mmzx_expl_replicate__bb_tl
  \tl_put_right:Ne \l__mmzx_tmpa_tl { \int_to_arabic:n { \l__mmzx_tmpa_int } }
  \tl_put_right:NV \l__mmzx_tmpa_tl \g__mmzx_expl_replicate__ba_tl
}
\cs_new_nopar:Npn \__mmzx_expl_replicate__t:
{
  \tl_put_right:Nn \l__mmzx_tmpc_tl {N}
  \tl_put_right:NV \l__mmzx_tmpa_tl \g__mmzx_expl_replicate__tb_tl
```

```
  \tl_put_right:Ne \l__mmzx_tmpa_tl { \int_to_arabic:n { \l__mmzx_tmpa_int } }
}
\cs_new_nopar:Npn \__mmzx_expl_replicate__e:
{
  \PackageError{mmzx}{No do,sorry. Use replicate with args instead.}{}
}
```

`mzx_expl_replicate_fn:` (*fn.*) For replicating `expl3` *functions.*
`__mmzx_tmp_cctab_stop:` (*fn.*)
`\mmzx@expl@replicate@fn`

```
\cs_new_eq:NN \__mmzx_tmp_cctab_stop: \__mmzx_noop:
\cs_new:Npn \__mmzx_expl_replicate_fn:
{
  \bool_set_false:N \l__mmzx_tmpa_bool
  \str_if_eq:eeT {\mmzxExplAtEnd} {relax}
  {
    \bool_set_true:N \l__mmzx_tmpa_bool
    \__mmzx_nexpl_at_start:
    \xtoksapp\mmzCCMemo {
      \exp_not:N \csname \mmzxExplAtBegin \exp_not:N \endcsname
    }
    \cs_set:Npn \__mmzx_tmp_cctab_stop:
    {
      \xtoksapp\mmzCCMemo {
        \exp_not:N \csname \mmzxExplAtEnd \exp_not:N \endcsname
      }
      \__mmzx_cctab_stop:
    }
  }{
    \cs_set_eq:NN \__mmzx_tmp_cctab_stop: \__mmzx_noop:
  }
  \group_begin:
    \bool_set_true:N \l__mmzx_replicating_bool
    \exp_last_unbraced:Ne \__mmzx_expl_replicate_fn_aux:nnN
      { \exp_args:NV \cs_split_function:N \AdviceReplaced }
}
\cs_new_eq:NN \mmzx@expl@replicate@fn \__mmzx_expl_replicate_fn:
```

`o/replicate expl fn` (*pgfkey*) By default we handle `\tex_savepos:D` since this commonly requires replication in
`/replicate expl var` (*pgfkey*) the kinds of environments typically subject to memoization. Another candidate is `\int_gincr:N`, but that results in a large number of additions and it is not at all clear these are generally required or desirable. Don't do this. It breaks stuff.

```
\mmzset{% config <<<
  auto/replicate expl fn/.style={
    run if not replicating,
    outer handler=\mmzx@expl@replicate@fn,
  },
}% >>>
```

</sty>

memoize-ext-l3draw

Clea F. Rees

11782 2026-01-19

Abstract

Support for auto-memoization of content created with l3draw (LaTeX Project 2025a). memoize-ext-l3draw is part of memoize-ext.

Contents

<*sty> <@@=mmzx>

```
\GetIdInfo $Id: memoize-ext-l3draw.dtx 11785 2026-03-17 17:32:51Z cfrees $ {Extensi
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName}{\ExplFileDate}
⟨!debug⟩     {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-debug}{\ExplFileDate}
⟨debug⟩      {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
%
⟨!debug⟩    \disable@package@load {memoize-ext-l3draw-debug}
⟨debug⟩    \disable@package@load {memoize-ext-l3draw}
{ Only one of memoize-ext-l3draw and memoize-ext-l3draw-debug
  should be loaded.
  Since
⟨!debug⟩    memoize-ext-l3draw
⟨debug⟩    memoize-ext-l3draw-debug
   has been loaded,I will ignore your request for
⟨debug⟩    memoize-ext-l3draw
⟨!debug⟩    memoize-ext-l3draw-debug
.}
%
⟨!debug⟩    \RequirePackage{memoize-ext}
⟨debug⟩    \RequirePackage{memoize-ext-debug}
⟨!debug⟩    \RequirePackage{memoize-ext-expl3}
⟨debug⟩    \RequirePackage{memoize-ext-expl3-debug}
```

l3draw

`mzx_draw_id_begin_int` (*var.*) Temporary `int`.

```
\int_new:N \g__mmzx_draw_id_begin_int
```

`ce_collect_draw_args:w` (*fn.*) The l3draw picture environment is essentially a function with a weird argument specification, so we use a custom collector.

```
\cs_new:Npn \__mmzx_advice_collect_draw_args:w #1 \draw_end:
{
  \toks0={ #1 \draw_end: }
  \exp_args:No \AdviceInnerHandler {\the\toks0}
}
```

\AdviceCollectDrawArguments Public wrapper.

```
\cs_new:Npn \AdviceCollectDrawArguments{
  \toks0 = {}
  \__mmzx_advice_collect_draw_args:w
}
```

Default configuration. This replicates changes to `\g_draw_id_int`, but does so by executing code at the start and end of *every* memoization. There must be a more efficient way to do do this ....

```
\mmzset{%
  gincr draw id/.style={
    at begin memoization={
      \gtoksapp\mmzCCMemo
      {
        \UseName{int_gincr:c} {g_draw_id_int}
      }
    },
  },
  auto csname={draw_begin:}{memoize,collector=\AdviceCollectDrawArguments,gincr draw id,},
  auto csname={__draw_record_origin:}{run if memoizing,replicate expl fn,},
}
```

</sty>

memoize-ext-sockets

Clea F. Rees

11782 2026-01-19

Abstract

memoize-ext-sockets is part of memoize-ext.

Contents

<*sty> <@@=mmzx>

ltsockets

`socket_assigned_plug:n` (*fn.*) Returns the name of the plug assigned to the specified socket. This ought not use a variable internal to the format's code, but there does not seem to be a public interface. So it may break, but for now it works.

```
\cs_new_nopar:Npn \__mmzx_socket_assigned_plug:n #1
{ % rhybudd: fn mewnol
  \str_use:c { l__socket_#1_plug_str }
}
```

</sty>

memoize-ext-tag

Clea F. Rees

11782 2026-01-19

Abstract

memoize-ext-tag is part of memoize-ext. It supports tagging memoized content/the memoization of tagged content.

Contents

Uses and/or redefines the following internals, primitives and other nefarious methods:

- `\l__socket_assigned_plug:n` thing
  - If a public interface for retrieving the plug is provided at some point, I will see if I can use that. However, they do not wish it to be expandable. This is manageable, but it is very nice having an expandable function here, so I will see if I realise, remember and can do it not-too-painfully, I guess.
- latex-lab variables, keys etc.
  - This is fairly fragile: patching patches to make them work with patched patches ....
  - But I guess the l3keys and pgfkeys are public. I'm not sure about the sockets/plugs. Are these supposed to be used by external code?
  - The use of `l__tikz_tagging_alt_tl` and `l__tikz_tagging_actualtext_tl` is definitely ungood, but I do not currently see a better way. Hopefully most people will use the pgfkeys interface, as that's much more natural here?
- `\tex_savepos:D`
  - This is more-or-less routine and actually documented, so no worries really here?
- `\mmzIncludeExtern`
  - Documented as internal, certainly not something to redefine, but maybe I can persuade Sašo to add the sockets I need here?

<*sty> <@@=mmzx>

```
\GetIdInfo $Id: memoize-ext-tag.dtx 11782 2026-03-17 14:49:00Z cfrees $ {Extensions
```

```
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName}{\ExplFileDate}
⟨!debug⟩     {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-debug}{\ExplFileDate}
⟨debug⟩      {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
%
⟨!debug⟩    \disable@package@load {memoize-ext-tag-debug}
⟨debug⟩    \disable@package@load {memoize-ext-tag}
{ Only one of memoize-ext-tag and memoize-ext-tag-debug
  should be loaded.
  Since
⟨!debug⟩    memoize-ext-tag
⟨debug⟩    memoize-ext-tag-debug
   has been loaded,I will ignore your request for
⟨debug⟩    memoize-ext-tag
⟨!debug⟩    memoize-ext-tag-debug
.}

⟨!debug⟩    \RequirePackage{memoize-ext}
⟨debug⟩    \RequirePackage{memoize-ext-debug}
%
```

We don't want inconsistent names in hooks.

```
\SetDefaultHookLabel{memoize-ext}
%
\cs_generate_variant:Nn \socket_assign_plug:nn {nV}
```

`\g__mmzx_tagpic_int` (*var.*)
`\l__mmzx_toks_tl` (*var.*)
`\l__mmzx_ok_bool` (*var.*)
`g__mmzx_plug_orig_str` (*var.*)
`\mmzxtagtoks` (*toks*)

Tracking & storage variables.

```
\bool_new:N \l__mmzx_ok_bool
\int_new:N \g__mmzx_tagpic_int
\str_new:N \g__mmzx_plug_orig_str
\tl_new:N \l__mmzx_toks_tl
\newtoks\mmzxtagtoks
```

`nclude/extern/before` (*socket*)
`include/extern/after` (*socket*)

New sockets for extern utilisation.

```
\socket_new:nn {tagsupport/memoize/include/extern/before} {3}
\socket_new:nn {tagsupport/memoize/include/extern/after} {0}
```

Ulrike's pgf/tikz keys don't do anything with the arguments they receive? It only seems they do because `init` passes them also to the l3keys?

And so we have to pass them from tikz to l3keys and back to tikz ....

This creates a problem of synchronisation. It would be nice to use module-specific names to track latex-lab internal variables for ease of maintenance, but I don't think this is possible. If I let a new name to a token list variable, I'll just get the current value.

On the one hand, if users use pgfkeys to set the values for `alt` and `actualtext`, we can easily avoid latex-lab internals, at least. But we do that by introducing additional variables and then have the problem of synchronisation.

On the other hand, we could avoid the synchronisation problems by using latex-lab internals more consistently, but then even the pgfkeys interface will be dependent on the internal implementation[1].

[1] I get the prohibition on internals. I really do. But there are cases where it just makes things much

Note: will need revising when the pgfkeys version of the latex-lab code gets published.

```
\hook_gput_code:nnn {package/tikz/after} {.}
{
  \pgfqkeys{/tikz}{
    alt/.forward to=/mmz/alt,
    actualtext/.forward to=/mmz/actualtext,
    artifact/.forward to=/mmz/artifact,
    tagging-setup/.forward to=/mmz/tagging-setup,
    /mmzx/tagging@setup/.is family,
    alt/.belongs to family=/mmzx/tagging@setup,
    artifact/.belongs to family=/mmzx/tagging@setup,
    actualtext/.belongs to family=/mmzx/tagging@setup,
    tagging-setup/.belongs to family=/mmzx/tagging@setup,
  }
  \mmzset{
    alt/.code={
      \keys_set:nn {tikz/tagging}{alt={#1}}
      \bool_set_true:N \l__mmzx_ok_bool
      \str_gset:Nn \g__mmzx_plug_orig_str {alt}
      \exp_args:Ne \mmzxtagtoks { \text_purify:n {#1} }
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/before} {mmzx/alt}
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/after} {mmzx/alt}
⟨debug⟩     \tl_log:e {\exp_args:No \exp_not:n {\the\mmzCCMemo}}
    },
    actualtext/.code={
      \keys_set:nn {tikz/tagging}{actualtext={#1}}
      \bool_set_true:N \l__mmzx_ok_bool
      \str_gset:Nn \g__mmzx_plug_orig_str {actualtext}
      \exp_args:Ne \mmzxtagtoks { \text_purify:n {#1} }
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/before} {mmzx/actualtext}
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/after} {mmzx/actualtext}
⟨debug⟩     \tl_log:e {\exp_args:No \exp_not:n {\the\mmzCCMemo}}
    },
    artifact/.code={
      \keys_set:nn {tikz/tagging}{artifact}
      \bool_set_true:N \l__mmzx_ok_bool
      \str_gset:Nn \g__mmzx_plug_orig_str {artifact}
      \mmzxtagtoks {}
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/before} {mmzx/artifact}
      \socket_assign_plug:nn
        {tagsupport/memoize/include/extern/after} {mmzx/artifact}
⟨debug⟩     \tl_log:e {\exp_args:No \exp_not:n {\the\mmzCCMemo}}
    },
    tagging-setup/.is choice,
    tagging-setup/alt/.forward to=/mmz/alt,
    tagging-setup/actualtext/.forward to=/mmz/actualtext,
    tagging-setup/artifact/.forward to=/mmz/artifact,
    tagging-setup/.unknown/.code =
```

more complicated and error-prone. And sometimes it just doesn't not seem worth the additional fragility that introduces, even at the cost of some additional fragility due to the use of internals.

```
    {
      \exp_args:Nno
      \keys_set:nn {tikz/tagging}{\pgfkeyscurrentname={#1}}
    },
    alt/.belongs to family=/mmzx/tagging@setup,
    actualtext/.belongs to family=/mmzx/tagging@setup,
    artifact/.belongs to family=/mmzx/tagging@setup,
    tagging-setup/.belongs to family=/mmzx/tagging@setup,
    tagging-setup/alt/.belongs to family=/mmzx/tagging@setup,
    tagging-setup/actualtext/.belongs to family=/mmzx/tagging@setup,
    tagging-setup/artifact/.belongs to family=/mmzx/tagging@setup,
  }
}
```

So it is possible to do without this, but it is *much* more straightforward to do with.

The basic idea is this:

- At the start of memoization, we redefine the (internal) command `\mmzIncludeExtern`.
- In its place, we use simple wrapper around a copy of the original.
- The wrapper defines a socket before and after executing the original.

This lets us wrap utilisation of the extern in an appropriate tagging structure. At least, that's the idea.

It would be nice to assign the plug here just based on whichever plugs are installed, but it is too early, while `\mmzAtEndMemoization` is too late.

```
\appto\mmzAtBeginMemoization{
  \gtoksapp\mmzCCMemo{
    \let\mmzxIncludeExternOrig\mmzIncludeExtern
    \let\mmzIncludeExtern\mmzxIncludeExtern
  }
}
```

`\__mmzx_noop:nNnnnnnnn` (*fn.*)
`clude_extern:nNnnnnnnn` (*fn.*)
`\mmzxIncludeExtern`
`\mmzxIncludeExternOrig`

`\mmzIncludeExtern` only seems to be defined during utilisation, so we set up a command `\mmzxIncludeExternOrig` and set it to noop here, then redefine it locally when the extern is utilised. `\mmzIncludeExtern` is then redefined to `\mmzxIncludeExtern`, which wraps `\mmzxIncludeExternOrig` in a tagging structure[2].

Note this not only uses, but redefines an internal command which the manual says users should never need to even use ....

On the other hand, this is exactly the kind of thing memoize does to *other* packages' internals and, indeed, to the LaTeX Project's. Which is more than can be said to defend my use of their internals ....

But does that lessen my guilt? :-)

```
\cs_new_protected_nopar:Npn
  \__mmzx_noop:nNnnnnnnn #1#2#3#4#5#6#7#8#9 {}
```

[2]Note to self: check output of tests visually if you do not want documents to be inaccessible to that tiny group of people who rely on vision to read.

```
\cs_new_protected_nopar:Npn \__mmzx_include_extern:nNnnnnnnn #1#2#3#4#5#6#7#8#9
{
  \int_gincr:N \g__mmzx_tagpic_int
⟨debug⟩    \__mmzx_debug:n {Args: #1; #2; #3; #4; #5; #6; #7; #8; #9}
  \socket_use:nnnn {tagsupport/memoize/include/extern/before}
    {#3}{#4}{#5}
⟨debug⟩    \__mmzx_debug:n {Executing original inclusion macro.}
  \mmzxIncludeExternOrig {#1}#2{#3}{#4}{#5}{#6}{#7}{#8}{#9}
⟨debug⟩    \__mmzx_debug:n {Closing tagging structures.}
  \socket_use:n {tagsupport/memoize/include/extern/after}
}
\cs_new_eq:NN \mmzxIncludeExtern \__mmzx_include_extern:nNnnnnnnn
\cs_new_eq:NN \mmzxIncludeExternOrig \__mmzx_noop:nNnnnnnnn
```

`xtern/before mmzx/alt` (*plug*) pgf/tikz addaswyd o latex-lab-testphase-tika.sty «<

```
\socket_new_plug:nnn {tagsupport/memoize/include/extern/before} {mmzx/alt}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/alt in socket
⟨debug⟩      tagsupport/memoize/include/extern/before.}
  \mode_if_vertical:T
  {
    \if@inlabel
      \mode_leave_vertical:
    \else
      \tag_socket_use:n {para/begin}
    \fi
  }
  \tag_mc_end_push:
  \tl_set:No \l__mmzx_toks_tl {\the\mmzxtagtoks}
  \tag_struct_begin:n
  {
    tag=Figure,
    alt=\l__mmzx_toks_tl,
  }
  \tag_mc_begin:n {tag=Figure}
  \cs_new:cpe {mmzx@tag@tikz@mark@pos@\int_to_arabic:n {\g__mmzx_tagpic_int}}
  {
    \__mmzx_pgftikz_tag_bbox:ennn {mmzx-id\int_to_arabic:n {\g__mmzx_tagpic_int}}
    {#1}{#2}{#3}
  }
⟨debug⟩    \cs_log:c {mmzx@tag@tikz@mark@pos@\int_to_arabic:n
⟨debug⟩      {\g__mmzx_tagpic_int}}
  \tag_struct_gput:ene
  {\tag_get:n {struct_num}}
  {attribute}
  {
    /O /Layout /BBox
    [
      \use:c
      {mmzx@tag@tikz@mark@pos@\int_to_arabic:n {\g__mmzx_tagpic_int}}
    ]
  }
⟨debug⟩    \__mmzx_debug:e {Recording xpos,
⟨debug⟩      ypos of mmxc-id\int_to_arabic:n {\g__mmzx_tagpic_int}.}
  \tex_savepos:D
```

```
  \__mmzx_property_record_orig:ee
    {mmzx-id\int_to_arabic:n {\g__mmzx_tagpic_int}}
    {xpos,ypos}
  \tex_savepos:D
}
```

»>

extern/after mmzx/alt (*plug*) pgf/tikz addaswyd o latex-lab-testphase-tika.sty bod yn onest, cafodd ei ddwyn o latex-lab yn hollol «<

```
\socket_new_plug:nnn {tagsupport/memoize/include/extern/after} {mmzx/alt}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/alt in socket
⟨debug⟩      tagsupport/memoize/include/extern/after.}
  \tag_mc_end:
  \tag_struct_end:
  \tag_mc_begin_pop:n {}
}
```

»>

efore mmzx/actualtext (*plug*) «< addaswyd o latex-lab-testphase-tika.sty
after mmzx/actualtext (*plug*)

```
\socket_new_plug:nnn {tagsupport/memoize/include/extern/before} {mmzx/actualtext}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/actualtext in socket
⟨debug⟩      tagsupport/memoize/include/extern/before.}
  \keys_set:nn {tikz/tagging} {actualtext:o = {\the\mmzxtagtoks},}
  \socket_use:n {tagsupport/tikz/picture/begin}
}
\socket_new_plug:nnn {tagsupport/memoize/include/extern/after} {mmzx/actualtext}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/actualtext in socket
⟨debug⟩      tagsupport/memoize/include/extern/after.}
  \socket_use:n {tagsupport/tikz/picture/end}
}
```

»>

/before mmzx/artifact (*plug*) «< addaswyd o latex-lab-testphase-tika.sty
n/after mmzx/artifact (*plug*)

```
\socket_new_plug:nnn {tagsupport/memoize/include/extern/before} {mmzx/artifact}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/artifact in socket
⟨debug⟩      tagsupport/memoize/include/extern/before.}
  \keys_set:nn {tikz/tagging} {artifact,}
  \socket_use:n {tagsupport/tikz/picture/begin}
}
\socket_new_plug:nnn {tagsupport/memoize/include/extern/after} {mmzx/artifact}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx/artifact in socket
⟨debug⟩      tagsupport/memoize/include/extern/after.}
  \socket_use:n {tagsupport/tikz/picture/end}
}
```

»>

`_pgftikz_tag_bbox:nnnn` (*fn.*)
`_pgftikz_tag_bbox:ennn` (*fn.*)

A memoize-friendly alternative to get the bounding box for the `alt` plug.

```
\cs_new_nopar:Npn \__mmzx_pgftikz_tag_bbox:nnnn #1#2#3#4
{
  \__mmzx_pgftikz_tag_bbox_aux:eennn
  {
    \mmzx_property_ref_orig:ee {#1}{xpos}
  }
  {
    \mmzx_property_ref_orig:ee {#1}{ypos}
  }
  {#2}{#3}{#4}
}
\cs_generate_variant:Nn \__mmzx_pgftikz_tag_bbox:nnnn {ennn}
```

`ikz_tag_bbox_aux:nnnnn` (*fn.*)
`ikz_tag_bbox_aux:eennn` (*fn.*)

Auxiliary.

```
\cs_new_nopar:Npn \__mmzx_pgftikz_tag_bbox_aux:nnnnn #1#2#3#4#5
{
  \dim_to_decimal_in_bp:n {#1sp}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#2sp-#5}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#1sp+#3}
  \c_space_tl
  \dim_to_decimal_in_bp:n {#2sp+#4+#5}
}
\cs_generate_variant:Nn \__mmzx_pgftikz_tag_bbox_aux:nnnnn {eennn}

\hook_gput_code_with_args:nnn {cmd/tikz@picture/before} {mmzx}
{
  \tag_if_active:T
  {
⟨debug⟩    \__mmzx_debug:n {Executing mmzx code in hook cmd/tikz@picture/before.}
    \socket_assign_plug:nn {tagsupport/tikz/picture/init} {mmzx}
  }
}

»>
```

`ikz/picture/init mmzx` (*plug*)

«< Originally, I made something much more complicated, which worked, but meh. Here the idea is that *if we are memoizing, we do not tag at all.* There's no real downside to this: a document which includes code memoized during the current run is not usable anyway and tagging the memoized code is pointless and inefficient. (Actually, so is executing the original code for use during the current run, which is, I believe, how it works. But that is not my fault.)

Instead, we tag *only the utilisation of memos.* We don't need to get the bounding box — memoize already does that. All we need do is get the position on the page during utilisation. Everything else is for free.

Note that this code uses multiple format/latex-lab internals. If the support for tikz used exclusively pgfkeys, this would be easily avoided: we could stick to the public interface for all but one of these usages. But because it supports also l3keys, I don't see a way to avoid depending on internal variables there, too. l3keys does not have a `.forward_to:n` or similar. There is `.inherit:n`, but that does not work at all in the same way. Filtering

and family code (below) is copied from the code I suggested for latex-lab's Ti*k*Z support (#2004).

```
\socket_new_plug:nnn {tagsupport/tikz/picture/init} {mmzx}
{
⟨debug⟩    \__mmzx_debug:n {Executing plug mmzx in socket
⟨debug⟩      tagsupport/tikz/picture/init.}
  \bool_set_false:N \l__mmzx_ok_bool
  \legacy_if:nT {memoizing}
  {
⟨debug⟩    \__mmzx_debug:n {Arg to tagsupport/tikz/picture/init is #1.}
    \pgfkeyssavekeyfilterstateto\mmzx@tagging@setup@saved@filterstate
    \pgfkeysinstallkeyfilter{/pgf/key filters/active families}{}
    \pgfqkeysactivatesinglefamilyandfilteroptions{/mmzx/tagging@setup}{/tikz}{#1}
⟨debug⟩    \__mmzx_debug:e { Restoring filter state: \exp_not:V
⟨debug⟩      \mmzx@tagging@setup@saved@filterstate }
    \mmzx@tagging@setup@saved@filterstate
    \bool_if:NF \l__mmzx_ok_bool
    {
⟨debug⟩            \__mmzx_debug:n {
⟨debug⟩              No plug set for utilisation. Trying to match latex-lab plug.
⟨debug⟩            }
```

If the argument to the picture set the plug, it is already in the code hash. Otherwise, we add the value of the latex-lab plug to the context, using `\mmzContextExtra` and appending globally as we are memoizing by hypothesis.

```
      \xtoksapp\mmzContextExtra {
        tagging plug=\__mmzx_socket_assigned_plug:n {tagsupport/tikz/picture/begin}
      }
      \str_case_e:nn
      {\__mmzx_socket_assigned_plug:n {tagsupport/tikz/picture/begin}}
      {
        {alt} {
          \exp_args:Ne \mmzset{
            alt = \exp_not:V \l__tikz_tagging_alt_tl,
          }
⟨debug⟩            \__mmzx_debug:n {Using alt plug.}
        }
        {actualtext} {
          \exp_args:Ne \mmzset{
            actualtext = \exp_not:V \l__tikz_tagging_actualtext_tl,
          }
⟨debug⟩            \__mmzx_debug:n {Using actualtext plug.}
        }
        {artifact} {
          \mmzset{artifact,}
⟨debug⟩            \__mmzx_debug:n {Using artifact plug.}
        }
        {text} {
          \bool_set_false:N \l__mmzx_ok_bool
⟨debug⟩            \__mmzx_debug:e {Found unsupported
⟨debug⟩              text
⟨debug⟩              {tagsupport/tikz/picture/begin} plug.}
          \PackageWarning{memoize-ext}{Unsupported tag config for tikz picture.
            Please use alt,actualtext,artifact or another supported plug.
            Note that text (the latex-lab default) is NOT supported.
```

```
            Aborting memoization and marking code unmemoizable.
          }
          \mmzUnmemoizable
        }
      }
    }
  }
  \bool_if:NT \l__mmzx_ok_bool
  {
    \xtoksapp\mmzCCMemo{
      \exp_not:N \mmzxtagtoks
      =
      \c_left_brace_str
      \the\mmzxtagtoks
      \c_right_brace_str
      \exp_not:N \AssignSocketPlug
      \c_left_brace_str
      tagsupport/memoize/include/extern/before
      \c_right_brace_str
      \c_left_brace_str
      \__mmzx_socket_assigned_plug:n {tagsupport/memoize/include/extern/before}
      \c_right_brace_str
      \exp_not:N \AssignSocketPlug
      \c_left_brace_str
      tagsupport/memoize/include/extern/after
      \c_right_brace_str
      \c_left_brace_str
      \__mmzx_socket_assigned_plug:n {tagsupport/memoize/include/extern/after}
      \c_right_brace_str
    }
⟨debug⟩     \__mmzx_debug:n {Disabling tagging for current tikz picture during
⟨debug⟩       current run.}
    \socket_assign_plug:nn {tagsupport/tikz/picture/begin} {noop}
    \socket_assign_plug:nn {tagsupport/tikz/picture/end} {noop}
    \socket_assign_plug:nn {tagsupport/tikz/picture/text/begin} {noop}
    \socket_assign_plug:nn {tagsupport/tikz/picture/text/end} {noop}
  }
}

\hook_gput_code:nnn {cmd/mmzAbort/after} {.}
{
  \legacy_if:nT {memoizing} {
    \PackageWarning{memoize-ext}{
      Memoization has been aborted. If something like a reference needs
      resolving,just compile again. If the content is unmemoizable --
      e.g. contains remember picture,a breakable tcolorbox or suchlike,
      you **must** mark the content unmemoizable or the tagging will be
      incorrect. You may do this automatically or manually. Aborted
    }
  }
}
```

»> pgf/tikz addaswyd o latex-lab-testphase-tika.sty

Hook rules to ensure our substitutes override the format's. «<

```
\hook_gset_rule:nnnn {cmd/tikz@picture/before}{latex-lab-testphase-tikz}{>}{.}
```

```
\hook_gset_rule:nnnn {cmd/tikz@picture/before}{tagpdf}{>}{.}
\hook_gset_rule:nnnn {cmd/tikz@picture/before}{tikz}{>}{.}
\hook_gset_rule:nnnn {cmd/endpgfpicture/after}{latex-lab-testphase-tikz}{>}{.}
\hook_gset_rule:nnnn {cmd/endpgfpicture/after}{tikz}{>}{.}
\hook_gset_rule:nnnn {cmd/endpgfpicture/after}{tagpdf}{>}{.}
\hook_gset_rule:nnnn {package/tikz/after} {.} {>} {latex-lab-testphase-tikz}
\hook_gset_rule:nnnn {package/tikz/after} {.} {>} {tagpdf}
\hook_gset_rule:nnnn {package/tikz/after} {.} {>} {tikz}
⟨debug⟩       \hook_log:n {package/tikz/after}
\hook_gput_code:nnn {begindocument/end} {.}
{
⟨debug⟩       \hook_log:n {cmd/tikz@picture/before}
⟨debug⟩       \hook_log:n {cmd/endpgfpicture/after}
```

»>

`x_property_ref_orig:nn` (*fn.*) «< Avoid dependency on memoize-ext-properties.

```
  \cs_if_free:NT \mmzx_property_ref_orig:nn
  {
    \cs_new_eq:NN \mmzx_property_ref_orig:nn \property_ref:nn
    \cs_generate_variant:Nn \mmzx_property_ref_orig:nn {ee}
  }
```

»>

`roperty_record_orig:nn` (*fn.*) «< Avoid dependency on memoize-ext-properties.

```
  \cs_if_free:NT \__mmzx_property_record_orig:nn
  {
    \cs_new_eq:NN \__mmzx_property_record_orig:nn \property_record:nn
    \cs_generate_variant:Nn \__mmzx_property_record_orig:nn {ee}
  }
```

»>

```
}
```

</sty>

memoize-ext-talk

Clea F. Rees

11782 2026-01-19

Abstract

memoize-ext-talk enables memoization of ltx-talk presentations (Wright 2026). The package is part of memoize-ext.

memoize-ext-talk is essentially a 'translation' of memoize's support for beamer, together with modifications for differences in the way the classes implement overlays and changes to the implementation of opacity when pdfmanagement-testphase (LaTeX Project 2026) is loaded.

The package tries to avoid utilising the internals of either memoize or ltx-talk, but it was not entirely possible to do so without making the code both more fragile and unduly convoluted. See the main manual for memoize-ext for details.

The user interface is identical to that provided by memoize for beamer (Živanović 2024).

Contents

<*sty> <@@=mmzx>

```
\GetIdInfo $Id: memoize-ext-talk.dtx 11782 2026-03-17 14:49:00Z cfrees $ {Extension
⟨!debug⟩    \ProvidesExplPackage{\ExplFileName}{\ExplFileDate}
⟨!debug⟩     {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
⟨debug⟩    \ProvidesExplPackage{\ExplFileName-debug}{\ExplFileDate}
⟨debug⟩      {v0.3.4 \ExplFileVersion}{\ExplFileDescription}
%
⟨!debug⟩    \disable@package@load {memoize-ext-talk-debug}
⟨debug⟩    \disable@package@load {memoize-ext-talk}
{ Only one of memoize-ext-talk and memoize-ext-talk-debug
  should be loaded.
  Since
⟨!debug⟩    memoize-ext-talk
⟨debug⟩    memoize-ext-talk-debug
   has been loaded,I will ignore your request for
⟨debug⟩    memoize-ext-talk
⟨!debug⟩    memoize-ext-talk-debug
.}
```

We have to load this (I think) prior to `\documentclass`, so cannot use `tag_if_active:TF` here. We could test for `\DocumentMetadata`, but ltx-talk already requires that. But maybe I should be testing that anyway?

```
⟨!debug⟩    \RequirePackage{memoize-ext-tag}
⟨debug⟩    \RequirePackage{memoize-ext-tag-debug}
```

We don't want inconsistent names in hooks.

```
\SetDefaultHookLabel{memoize-ext}
```

BEGIN ltx-talk addaswyd o memoize-beamer.code.tex & other memoize code

```
\hook_gput_code:nnn {class/ltx-talk/after}{.}
{
⟨debug⟩    \__mmzx_debug:n {Enabling ltx-talk support.}
  \mmzset{
    per overlay/.code={},
    talk mode to prefix/.style={
      prefix=\mmz@prefix@dir\mmz@prefix@name\l__mmzx_talk_mode_str -,
    },
  }
```

`mmzx_talk_opacity_seq` (*var.*)
`alk_saved_opacity_seq` (*var.*)
`_mmzx_talk_opacity_tl` (*var.*)

```
\seq_new:N \g__mmzx_talk_opacity_seq
\seq_new:N \g__mmzx_talk_saved_opacity_seq
\tl_new:N \l__mmzx_talk_opacity_tl
```

`\g__mmzx_talk_frame_int`
`\g__mmzx_talk_slide_int`
`\g__mmzx_talk_pauses_int`
`\l__mmzx_talk_mode_str`

```

\cs_new_eq:NN \g__mmzx_talk_frame_int \c@frame
\cs_new_eq:NN \g__mmzx_talk_slide_int \c@slide
\cs_new_eq:NN \g__mmzx_talk_pauses_int \c@pauses
\cs_new_eq:NN \l__mmzx_talk_mode_str \l__talk_mode_str
```

`\opacity_select:V` (*fn.*) ltx-talk does this too late (at least, I get an error)

```
\cs_generate_variant:Nn \opacity_select:n {V}
```

Initialise sequence.

```
\seq_gpush:Nn \g__mmzx_talk_opacity_seq {1}
```

`mzx_talk_opacity_save:` (*fn.*)

```
\cs_new_protected_nopar:Npn \__mmzx_talk_opacity_save:
{
  \seq_gset_eq:NN \g__mmzx_talk_opacity_saved_seq \g__mmzx_talk_opacity_seq
  \seq_get:NN \g__mmzx_talk_opacity_seq \l__mmzx_talk_opacity_tl
  \exp_args:NV \tl_if_eq:NNTF \l__mmzx_talk_opacity_tl \q_no_value
  {
    \seq_gpush:Nn \g__mmzx_talk_opacity_saved_seq {1}
    \opacity_select:n {1}
  } {
    \seq_gpush:NV \g__mmzx_talk_opacity_saved_seq \l__mmzx_talk_opacity_tl
    \opacity_select:V \l__mmzx_talk_opacity_tl
  }
}
```

_talk_opacity_restore: (*fn.*)

```
\cs_new_protected_nopar:Npn \__mmzx_talk_opacity_restore:
{
  \seq_gpop:NN \g__mmzx_talk_opacity_saved_seq \l__mmzx_talk_opacity_tl
  \opacity_select:V \l__mmzx_talk_opacity_tl
  \seq_gset_eq:NN \g__mmzx_talk_opacity_seq \g__mmzx_talk_opacity_saved_seq
}
```

\mmzSingleExternDriver Redefine driver Even if this does not have an internal name, the redefinition uses internals in spades .... We could define a new one & I guess that's what should be done ...

```
\long\def\mmzSingleExternDriver#1{
  \xtoksapp\mmzCCMemo{\mmz@maybe@quitvmode}
  \setbox\mmz@box\mmz@capture{
    \__mmzx_talk_opacity_save:
    #1
    \__mmzx_talk_opacity_restore:
  }
  \mmzExternalizeBox\mmz@box\mmz@temptoks
  \xtoksapp\mmzCCMemo{\the\mmz@temptoks}
  \mmz@maybe@quitvmode\box\mmz@box
}
```

The code below is iffy, I guess, since the begin/end thing here is rather illusory ...

```
\hook_gset_rule:nnnn {begindocument} {ltx-talk} {<} {.}
\hook_gput_code_with_args:nnn {cmd/opacity_select:n/before} {.}
{
  \seq_gpush:Nn \g__mmzx_talk_opacity_seq {#1}
}
\hook_gput_code:nnn {cmd/opacity_end:/after}{.}
{
⟨debug⟩    \__mmzx_debug:n{Opacity after}
  \seq_gpop:NN \g__mmzx_talk_opacity_seq \l_tmpa_tl
⟨debug⟩    \seq_log:N \g__mmzx_talk_opacity_seq
}
\mmzset{
  at begin memoization={
    \socket_use:n {mmzx/talk/memoization/begin}
  },
  at end memoization={
    \socket_use:n {mmzx/talk/memoization/end}
  },
  per overlay/.style={
    /mmz/context={
      overlay=\csname theslide\endcsname,
      pauses=\ifmemoizing
        \mmzxTalkPauses
      \else
        \expandafter\the\csname c@pauses\endcsname
      \fi
    },
    /utils/exec={
      \str_if_eq:eeF {peroverlay}
      {\__mmzx_socket_assigned_plug:n {mmzx/talk/memoization/begin}}
```

```
944         {
945           \socket_assign_plug:nn {mmzx/talk/memoization/begin}{peroverlay}
946           \socket_assign_plug:nn {mmzx/talk/memoization/end}{peroverlay}
```

This is required to allow per overlay to be used in the options for `tikzpictures` etc.

```
947           \legacy_if:nT {memoizing} {
948             \socket_use:n {mmzx/talk/memoization/begin}
949           }
950         }
951       },
```

Is resetting the style meant to prevent duplication in memos etc.? Because it obviously won't ...?

```
952       /mmz/per overlay/.code={},
953     },
954   }
```

`k/memoization/begin` (*socket*) Sockets.
`alk/memoization/end` (*socket*)

```
955   \socket_new:nn {mmzx/talk/memoization/begin}{0}
956   \socket_new:nn {mmzx/talk/memoization/end}{0}
```

`tion/begin peroverlay` (*plug*) Plugs.
`zation/end peroverlay` (*plug*)

```
957   \socket_new_plug:nnn {mmzx/talk/memoization/begin}{peroverlay}
958   {
959 ⟨debug⟩     \__mmzx_debug:n {Executing plug peroverlay in socket
960 ⟨debug⟩       mmzx/talk/memoization/begin.}
961     \xdef\mmzxTalkPauses{
962       \int_to_arabic:n {\g__mmzx_talk_pauses_int}
963     }
964     \xtoksapp\mmzCMemo{
965       \noexpand\mmzxSetTalkOverlays{\mmzxTalkPauses}{
966         \int_to_arabic:n {\g__mmzx_talk_slide_int}
967       }
968     }
969     \gtoksapp\mmzCCMemo{
970       \only<all:\mmzxTalkOverlays>{}
971     }
972     \seq_get:NNTF \g__mmzx_talk_opacity_seq \l__mmzx_opacity_tl
973     {
974       \fp_gset:NV \l__mmzx_tmpa_fp \l__mmzx_opacity_tl
975     } {
976       \fp_gset:Nn \l__mmzx_tmpa_fp {1}
977     }
978     \xtoksapp\mmzContextExtra{
979       opacity=\fp_to_decimal:N \l__mmzx_tmpa_fp
980     }
981   }
982   \socket_new_plug:nnn {mmzx/talk/memoization/end}{peroverlay}
983   {
984 ⟨debug⟩     \__mmzx_debug:n {Executing plug peroverlay in socket
985 ⟨debug⟩       mmzx/talk/memoization/end.}
986     \xtoksapp\mmzCCMemo{
987       \exp_not:N \setcounter{pauses}
```

```
      {\int_to_arabic:n {\g__mmzx_talk_pauses_int}}
    }
  }
```

`\mmzxSetTalkOverlays` Note the addition of code to set `g__talk_slide_continue_bool`. This isn't necessary for beamer and is not 'allowed' for ltx-talk, but is necessary for frames with overlay specifications in memoized content.

```
  \cs_new_nopar:Npn \mmzxSetTalkOverlays#1#2{
⟨debug⟩    \__mmzx_debug:e {Executing \cs_to_str:N \mmzxSetTalkOverlays}
    \int_compare:nNnTF {\g__mmzx_talk_pauses_int} = {#1}
    {
      \gdef\mmzxTalkOverlays{#2}
      \int_compare:nNnTF {\g__mmzx_talk_slide_int} < {#2}
      {
        \bool_set_true:N \l__mmzx_tmpa_bool
      }{
        \bool_set_false:N \l__mmzx_tmpa_bool
      }
    }{
      \bool_set_true:N \l__mmzx_tmpa_bool
    }
    \bool_if:NT \l__mmzx_tmpa_bool
    {
      \bool_gset_true:N \g__talk_slide_continue_bool
      \appto\mmzAtBeginMemoization{
        \gtoksapp\mmzCMemo{\mmzxSetTalkOverlays{#1}{#2}}
      }
    }
  }

}
```

</sty>

References

LaTeX Project (2025a). *The l3draw Package: Core Drawing Support*. 2025-10-09. 9th Oct. 2025. CTAN: l3experimental.

— (2025b). *The latex-lab-tikz Package: Support for the Tagging of TikZ Pictures*. v0.80d. 27th Sept. 2025. CTAN: latex-lab.

— (2026). *The LaTeX PDF Management Bundle*. 0.96y. 23rd Jan. 2026. CTAN: pdfmanagement-testphase.

Rees, Clea F. (2026). *forest-ext: A Collection of forest Libraries*. 0.2. 20th Feb. 2026. CTAN: forest-ext.

Wright, Joseph (2026). *ltx-talk: A Class for Typesetting Presentations*. 0.4.4. 10th Feb. 2026. CTAN: ltx-talk.

Živanović, Sašo (2017). *Forest: A PGF/TikZ-Based Package for Drawing Linguistic Trees*. 2.1.5. 14th July 2017. CTAN: forest.

— (2024). *Memoize*. 1.4.1. 2nd Dec. 2024. CTAN: memoize.

Change History

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

Symbols

A

B

C

D

E

F

G