{robust-externalize}

Cache anything (Ti*k*Z, python…), in a robust, efficient and pure way.

Léo Colisson   Version 1.0+unstable

github.com/leo-colisson/robust-externalize

Contents

WARNING: This library is young and has not been tested extensively (and is an important rewrite of a previous version). Even if we try to stay backward compatible, the only guaranteed way to be immune to changes is to copy/paste the library in your main project folder. Please report any bug to `https://github.com/leo-colisson/robust-externalize`, or let us know if it works!

1   A taste of this library

This library allows you to cache any language: not only LaTeX documents and TikZ images, taking into account depth and overlays:

The next picture is cached My node that respects baseline ❤. and you can see that overlay and depth works.

```
\robExtConfigure{
  add to preset={tikz}{
    add to preamble={\usepackage{pifont}}
  }
}
The next picture is cached %
\begin{tikzpictureC}[baseline=(A.base)]
  \node[fill=red, rounded corners](A){My node that respects baseline \ding{164}.};
  \node[fill=red, rounded corners, opacity=.3,overlay] at (A.north east){I am an overlay text};
\end{tikzpictureC} and you can see that overlay and depth works.
```

but also arbitrary code (e.g. python). You can also define arbitrary compilation commands, inclusion commands, and presets to fit you need. For instance, you can create a preset to obtain:

> **The for loop**
>
> ```
> 1  for name in ["Alice", "Bob"]:
> 2      print(f"Hello {name}")
> ```
>
> Output:
>
> ```
> Hello Alice
> Hello Bob
> ```

```latex
\begin{CacheMeCode}{python print code and result, set title={The for loop}}
for name in ["Alice", "Bob"]:
    print(f"Hello {name}")
\end{CacheMeCode}
```

Actually, we also provide this style by default (and explain how to write it yourself), just make sure to load:

```latex
\usepackage{pythonhighlight}
\usepackage{tcolorbox}
```

You can also cache any environment and command using something like:

- For environments:

A forest tree automatically cached

And one not cached:
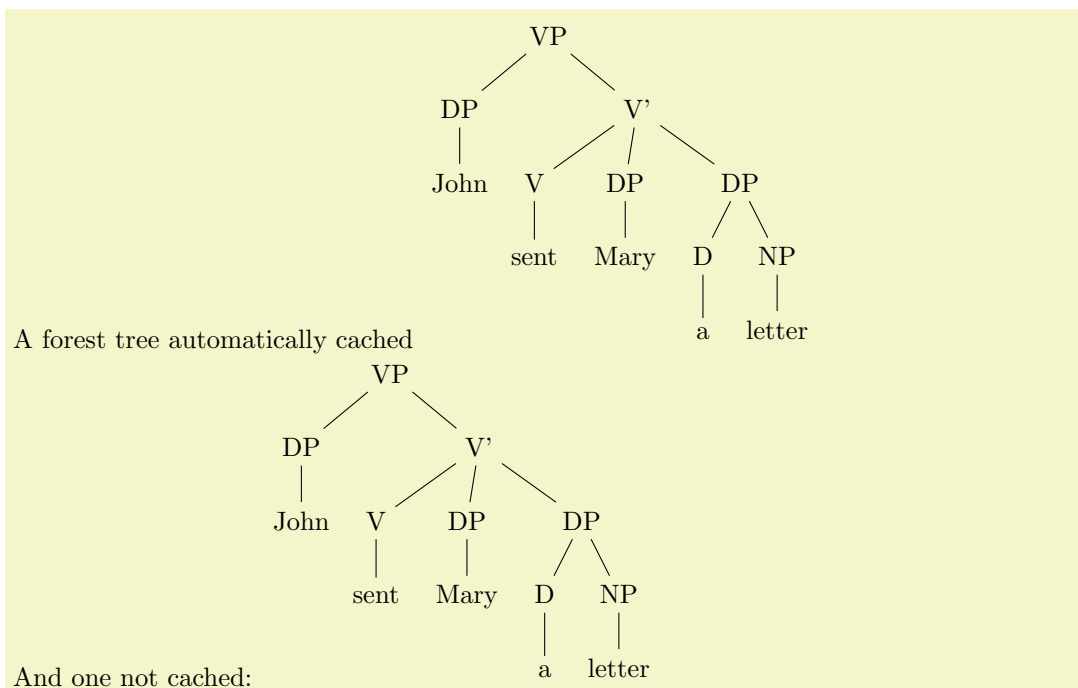
```latex
% We cache the environment forest
\cacheEnvironment{forest}{latex, add to preamble={\usepackage{forest}}}

A forest tree automatically cached %
\begin{forest}
  [VP
    [DP[John]]
    [V'
      [V[sent]]
      [DP[Mary]]
      [DP[D[a]][NP[letter]]]
    ]
  ]
\end{forest}\\
And one not cached: %
\begin{forest}<disable externalization>
  [VP
    [DP[John]]
    [V'
      [V[sent]]
      [DP[Mary]]
      [DP[D[a]][NP[letter]]]
    ]
  ]
\end{forest}
```

- For commands:

A ZX figure ⟨diagram⟩, we can disable externalization: ⟨diagram⟩ or add more options: ⟨diagram⟩
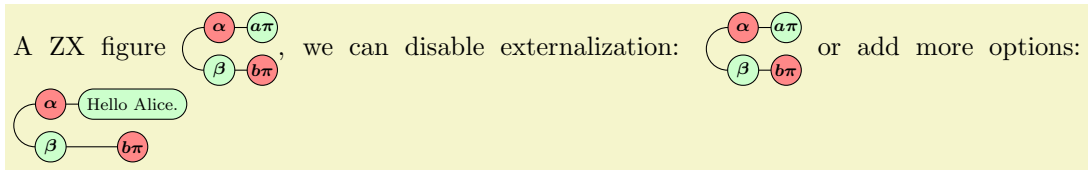
```
% We cache the command \zx
% O{} = optional argument, m = mandatory argument. Autodetected if command defined with xparse (NewDocumentCommand)
\cacheCommand{zx}[O{}O{}O{}m]{latex, add to preamble={\usepackage{zx-calculus}}}

A ZX figure %
\zx{
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar           & \zxX*{b\pi}
}, we can disable externalization: %
\zx<disable externalization>{
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar           & \zxX*{b\pi}
} or add more options: %
\zx<add to preamble={\usepackage{amsmath}\def\hello#1{Hello #1.}}>{
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{\hello{Alice}}} \\
  \zxZ{\beta} \rar           & \zxX*{b\pi}
}
```

You can also cache all tikz pictures by default, except those that are run from some commands (as remember picture does not work yet, this is important). For instance, this code will allow you to freely use the `todonotes` package:

```
\cacheCommand{todo}[O{}m]{disable externalization}
\todo{Check how to use cacheCommand and cacheEnvironment}
```

will gives you

> Check how to use cacheCommand and cacheEnvironment

To achieve a pleasant and configurable interface, we introduced placeholders, that can be of independent interest.

2 Introduction

2.1 Why do I need to cache (a.k.a. externalize) parts of my document?

One often wants to cache (i.e. store pre-compiled parts of the document, like figures) operations that are long to do: For instance, TikZ is great, but TikZ figures often take time to compile (it can easily take a few seconds per picture). This can become really annoying with documents containing many pictures, as the compilation can take multiple minutes: for instance my thesis needed roughly 30mn to compile as it contains many tiny figures, and LaTeX needs to compile the document multiple times before converging to the final result. But even on much smaller documents you can easily reach a few minutes of compilation, which is not only high to get a useful feedback in real time, but worse, when using online LaTeX providers (e.g. overleaf), this can be a real pain as you are unable to process your document due to timeouts.

Similarly, you might want to cache the result of some codes, for instance a text or an image generated via python and matplotlib, without manually compiling them externally.

2.2 Why not using TikZ's externalize library?

TikZ has an externalize library to pre-compile these images on the first run. Even if this library is quite simple to use, it has multiple issues:

- If you add a picture before existing pre-compiled pictures, the pictures that are placed after will be recompiled from scratch. This can be mitigated by manually adding a different prefix to each picture, but it is highly not practical to use.

- To compile each picture, TikZ's externalize library reads the document's preamble and needs to process (quickly) the whole document. In large documents (or in documents relying on many packages), this can result in a significant loading time, sometimes much bigger than the time to compile the document without the externalize library: for instance, if the document takes 10 seconds to be processed, and if you have 200 pictures that take 1s each to be compiled, the first compilation with the TikZ's externalize library will take roughly half an hour instead of 3mn without the library. And if you add a single picture at the beginning of the document... you need to restart everything from scratch. For these reasons, I was not even able to compile my thesis with TikZ's external library in a reasonable time.

- If two pictures share the same code, it will be compiled twice

- Little purity is enforced: if a macro changes before a pre-compiled picture that uses this macro, the figure will not be updated. This can result in different documents depending on whether the cache is cleared or not.

- As far as I know, it is made for TikZ picture mostly, and is not really made for inserting other stuff, like matplotlib images generated from python etc...

- According to some maintainers of TikZ, "the code of the externalization library is mostly unreadable gibberish[1]", and therefore most of the above issues are unlikely to be solved in a foreseeable future.

2.3 FAQ

What is supported? You can cache most things, including tikz pictures, (including ones with overlays (but not with remember picture), with depth etc.), python code etc. We tried to make the library as customizable as possible to be useful in most scenarios. You can also send some data (e.g. : the current page) to the compiled pictures, and feed some data back to the main document (say that you want to compute a value that takes time to compute, or compute the number of pages of the produced document in order to increase the number of pages accordingly...). Since v1.1, you can also cache automatically any environment.

What is not supported? We do not yet support remember picture, and you can't use (yet) cross-references inside your images (at least not without further hacks) or links as links are stripped when the pdf is included. I might have some ideas to solve this[2], meanwhile you can just disable locally the library on problematic figures. Note that this library is quite young, so expect untested things.

What OS are supported? I tested the library mostly on Linux systems, but the library should work on all OS. Please let me know if it fails for you.

Do I need to compile using -shell-escape? Since we need to compile the images via an external command, the simpler option is to add the argument `-shell-escape` to let the library run the compilation command automatically (this is also the case of TikZ's externalize library). However, people worried by security issues of `-shell-escape` (that allows arbitrary code execution if you don't trust the LaTeX code) might be interested by these facts:

- You can compile manually the images: all the commands that are left to be executed are listed in `robExt-compile-missing-figures.sh` and you can just inspect and run them, either with `bash robExt-compile-missing-figures.sh` or by typing them manually (most of the time it's only a matter of running `pdflatex somefile.tex`).

- If you allowed:

    - `pdflatex` (needed to compile latex documents)

---

[1] https://github.com/pgf-tikz/pgf/issues/758
[2] https://tex.stackexchange.com/questions/695277/clickable-includegraphics-for-cross-reference-data

- – cd (not needed when using `no cache folder`)
- – and `mkdir` (not needed when using `no cache folder` or if the cache folder that defaults to `robustExternalize` is already created)

  to run in restricted mode (so without enabling `-shell-escape`), then there is no need for `-shell-escape`. In that case, set `force compilation` and this library will compile even if `-shell-escape` is disabled.

- If images are all already cached, you don't need to enable `-shell-escape` (this might be interesting e.g. to send a pre-cached document to the arxiv or to a publisher: just make sure to include the cache folder).

- You can choose to display a dummy content until you choose to compile them, see `enable fallback to manual mode`.

- We use very few commands to compile latex files, basically only pdflatex, mkdir (to create the cache folder if needed) and cd (if the cache folder is not present).

Is it working on overleaf? Yes: overleaf automatically compiles documents with `-shell-escape`, so nothing special needs to be done there (of course, if you use this library to run some code, the programming language might not be available, but I heard that python is installed on overleaf servers for instance, even if this needs to be doubled checked). If the first compilation of the document to cache images times out, you can just repeat this operation multiple times until all images are cached.

Do you have some benchmarks? On an early draft of a small paper containing 76 small tikz-cd based pictures (from my other zx-calculus library), we measured:

- 35 seconds for a normal compilation without externalization

- 75 seconds for the first compilation with this library

- 2.4 seconds for the next runs

So during the first compilation, we lost a x2 factor (roughly an additional time of .5 seconds per picture coming from the time to start LaTeX, it seems like on average a picture takes .5 seconds to be built in my benchmark), but then we have a speedup of x15 (2.43s instead of 34.63s) for all subsequent runs. And I expect this to be even higher with more pictures and more complex documents.

Can I use version-control to keep the cached files in my repository? Sure, each cached figure is stored in a few files (typically one pdf and one LaTeX file, plus the source) having the same prefix (the hash), avoiding collision between runs. Just commit these files and you are good to go.

Can you deal with baseline position ? Yes, the depth of the box is automatically computed and used to include the figure by default.

How is purity enforced? Purity is the property that if you remove the cached files and re-compile your document, you should end-up with the same output. To enforce purity, we compute the hash of the final program, including the compilation command and the dependency files used for instance in `\input{include.tex}` (unless you prefer not to, for instance to keep parts of the process impure for efficiency reasons), and put the code in a file named based on this hash. Then we compile it if it has not been used before, and include the output. Changing a single character in the file, the tracked dependencies, or the compilation command will lead to a new hash, and therefore to a new generated picture.

What if I don't want purity for all files? If you do not want your files to be recompiled if you modify a given file, then just do not add this file to the list of dependencies.

Can I extend it easily? We tried to take a quite modular approach in order to allow easy extensions. Internally, to support a new cache scheme, we only expect a string containing the program (possibly produced using a template), a list of dependencies, a command to compile this program (e.g. producing a pdf and possibly a tex file with the properties (depth...) of the pdf), and a command to load the result of the compilation into the final document (called after loading the previously mentioned optional tex file). Thanks to pgfkeys, it is then possible to create simple pre-made settings to automatically apply when needed.

How does it compare with `https://github.com/sasozivanovic/memoize`? I recently became aware of the great `https://github.com/sasozivanovic/memoize`. While me aim to solve a similar goal, our approaches are quite different. While we focus on purity, and therefore create a different file for each picture, the above project puts all pictures in a single file and compile them all at once to avoid losing time to run the latex command for each picture (this mostly makes a difference for the first compilation). Our understanding of the main differences is the following:

Pros of `https://github.com/sasozivanovic/memoize`:

- The above library is likely to be quicker on the first run since it packs everything in a single file (so you save the time to run latex for each picture). On the other hand, we can load in the preamble exactly what is needed for the picture (we do not load all the preamble of the main file), so our startup time is not huge (it adds .5s per picture in my tests when using the `zx-calculus` library), and you can still commit the cached file to help with recompiling the document elsewhere.

- It seems to be easier to setup as they do not need to specify the preamble of the file to compile (the preamble of the main file is used), and tikz picture are automatically memoized (we provide `\cacheTikz` for that, but you still need to specify the preamble once).

Cons of `https://github.com/sasozivanovic/memoize`:

- The purity is not enforced so strongly since all images are in the same file. Notably, the hash only depends on the picture, but not on its context. So for instance if you define `\def\mycolor{blue}` before the picture, and use `\mycolor` inside the picture, if you change later the color to, say, `\def\mycolor{red}`, the picture will not be recompiled (so cleaning the cache and recompiling would produce a different result). In our case, the purity is always strictly enforced (unless you choose not to).

- As a result, the above library has poor support for contexts (in our case, you can easily, for instance, make a picture depend on the current page, and recompile the picture only if the current page changes: you can also do it the other way, and change some counters, say, depending on the cached file).

- The above library only focuses on LaTeX while our library works for any language

- The above library can only produce pdf formats, while we can generate any format (text, tex, jpg... and even videos that I include in my beamer presentation).

- We have an arguably more complete documentation.

Note that `remember picture` is not working in both libraries.

3 Tutorial

3.1 Installation

To install the library, just copy the `robust-externalize.sty` file into the root of the project. Then, load the library using:

```
\usepackage{robust-externalize}
```

If you want to display a message in the pdf on how to manually compile the file if `-shell-escape` is disabled, you can also load this configuration (only available on v1.1):

```
\robExtConfigure{
  enable fallback to manual mode,
}
```

(otherwise, it will give you an error if `-shell-escape` is disabled and if some pictures are not yet cached)

If you forget/do not want to enable `-shell-escape`, this will give you this kind of message:

Draft mode: either compile with `-shell-escape` or compile:
`robustExternalize/robExt-9A58BBFD5C2B6C079891CF68E406829D.tex`
via
`cd robustExternalize/ && pdflatex -halt-on-error "robExt-9A58BBFD5C2B6C079891CF68E406829D.tex"`
or call
`bash robust-externalize-robExt-compile-missing-figures.sh`
to compile all missing figures.

3.2 Caching a tikz picture

If you only care about TikZ's picture, you have 3 options:

1. Call once `\cacheTikz` that will redefine `tikzpicture` and `\tikz` to use our library (if you use this solution, make sure to read how to disable externalization (section 4.3.6) as we do not support for instance `remember picture`). Then, configure the default preamble for cached files as explained below by extending the `tikz` or `tikzpicture` presets (that first loads `tikz`).

2. Use the generic commands that allows you to wrap an arbitrary environment or command:

   ```
   %        name environment ---v            v----- preset options
   \robExtCacheEnvironment{tikzpicture}{tikzpicture}
   %     name command ---v    v------v-----signature and preset options
   \robExtCacheCommand{tikz}[O{}m]{tikz}
   ```
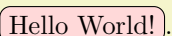
   Note that the signature is not needed if the macro is defined using xparse.

   This is not recommended for the tikz command as the actual definition of `\tikz` is more complicated since it has numerous syntax. `\cacheTikz` on the other hand parses tikz correctly. For simpler macros, this is not a problem.

3. Use `tikzpictureC` instead of `tikzpicture` (this is mostly done to easily convert existing code to this library, but works only for `tikz` pictures).

4. Use the more general `CacheMe` environment, that can cache TikZ, LaTeX, python, and much more.

These 3 options are illustrated below (note that the newly defined `tikzpicture` and `tikzpictureC` accept a second optional argument that contains the options to pass to `CacheMe` after loading the `tikzpicture` preset, that loads itself the `tikz` preset first):
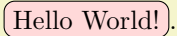
Option 1:

I am a cached picture: Hello World! .

```
%% We override the default tikzpicture environment
%% to externalize all pictures
%% Warning: it will cause troubles with pictures relying on |remember pictures|
\cacheTikz

I am a cached picture: \begin{tikzpicture}[baseline=(A.base)]
  \node[draw,rounded corners,fill=pink!60](A){Hello World!};
\end{tikzpicture}.
```

Option 2:

I am a cached picture: [Hello World!].

```
\cacheEnvironment{tikzpicture}{tikzpicture}
I am a cached picture: \begin{tikzpicture}[baseline=(A.base)]
  \node[draw,rounded corners,fill=pink!60](A){Hello World!};
\end{tikzpicture}.
```

Option 3:

I am a cached picture: [Hello World!].

```
I am a cached picture: \begin{tikzpictureC}[baseline=(A.base)]
  \node[draw,rounded corners,fill=pink!60](A){Hello World!};
\end{tikzpictureC}.
```
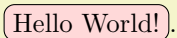
Option 4:

I am a cached picture: [Hello World!].

```
I am a cached picture: \begin{CacheMe}{tikzpicture}[baseline=(A.base)]
  \node[draw,rounded corners,fill=pink!60](A){Hello World!};
\end{CacheMe}.
```

Since `CacheMe` is more general as it applies also to non-tikz pictures (just replace `tikzpicture` with the style of your choice), we will mostly use this syntax from now.

3.3 Custom preamble

Note that the pictures are compiled in a separate document, with a different preamble and class (we use the standalone class). This is interesting to reduce the compilation time of each picture (loading a large preamble is really time consuming) and to avoid unnecessary recompilation (do you want to recompile all your pictures when you add a single new macro?) without sacrificing the purity. But of course, you need to provide the preamble of the pictures. The easiest way is probably to modify the `tikz` preset (since `tikzpicture` loads this style first, it will also apply to tikzpictures. Note that you can also modify the `latex` preset if you want the change to apply to all LaTeX documents):

See, tikz's style now packs the `shadows` library by default:
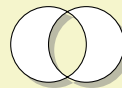
```
\robExtConfigure{
  add to preset={tikz}{
    add to preamble={\usetikzlibrary{shadows}},
  },
}

See, tikz's style now packs the |shadows| library by default: %
\begin{CacheMe}{tikzpicture}[even odd rule]
  \filldraw [drop shadow,fill=white] (0,0) circle (.5) (0.5,0) circle (.5);
\end{CacheMe}
```

and you can also create a new preset, and even mix multiple presets together:
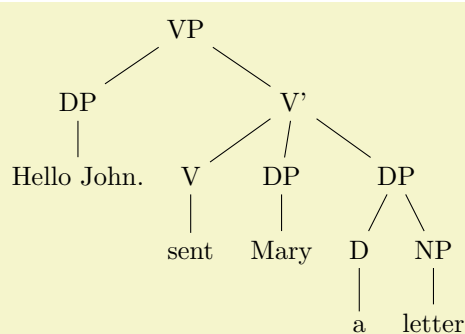
```
\robExtConfigure{
  new preset={load forest}{
    latex,
    add to preamble={\usepackage{forest}},
  },
  new preset={load hello}{
    add to preamble={\def\hello#1{Hello #1.}},
  },
}

\begin{CacheMe}{load forest, load hello}
  \begin{forest}
    [VP
      [DP[\hello{John}]]
      [V'
        [V[sent]]
        [DP[Mary]]
        [DP[D[a]][NP[letter]]]
      ]
    ]
  \end{forest}
\end{CacheMe}
```

Note: the `add to preset` and `new preset` directives have been added in v1.1, together with the `tikzpicture` preset. In v1.0, you would do `tikz/.append style={...}` (and you can still do this if you prefer), the difference is that `.append style` and `.style` require the user to double all hashes like `\def\mymacro##1{Hello ##1.}` which can lead to confusing errors.

You can also choose to overwrite it for a single picture (or even a block of picture if you run the `\robExtConfigure` and `CacheMe` inside a group `{ ... }`):
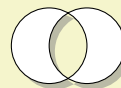

```
See, you can add something to the preamble of a single picture: %
\begin{CacheMe}{tikzpicture, add to preamble={\usetikzlibrary{shadows}}}[even odd rule]
  \filldraw [drop shadow,fill=white] (0,0) circle (.5) (0.5,0) circle (.5);
\end{CacheMe}
```

Note that if you use the `tikzpictureC` or `tikzpicture` syntax, you want to add the options right after the name of the command or environment, enclosed in `<>` (by default):
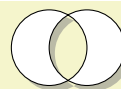

```
See, you can add something to the preamble of a single picture: %
\begin{tikzpictureC}<add to preamble={\usetikzlibrary{shadows}}>[even odd rule]
  \filldraw [drop shadow,fill=white] (0,0) circle (.5) (0.5,0) circle (.5);
\end{tikzpictureC}
```

Important: Note that the preset options can be specified in a number of places: in

`\robExtConfigure` (possibly in a group, or inside a preset), in the options of the picture, in the default options of `cacheEnvironment` and `cacheCommand` etc. Most options can be used in all these places, the only difference being the scope of the options.

3.4 Dependencies

It might be handy to have a file that is loaded in both the main document and in the cached pictures. For instance, if you have a file `common_inputs.tex` that you want to input in both the main file and in the cached files, that contains, say:

`\def\myValueDefinedInCommonInputs{42}`

then you can add it as a dependency this way (here we use the `latex` preset that does not wrap the code inside a `tikzpicture` only to illustrate that we can also cache things that are not generated by tikz):

The answer is 42.

```
\begin{CacheMe}{latex,
    add dependencies={common_inputs.tex},
    add to preamble={\input{__ROBEXT_WAY_BACK__common_inputs.tex}}}
  The answer is \myValueDefinedInCommonInputs.
\end{CacheMe}
```

Note that the placeholder `__ROBEXT_WAY_BACK__` contains the path from the cache folder (containing the `.tex` that will be cached) to the root folder, and will be replaced when creating the file. This way, you can easily input files contained in the root folder. You can also create your own placeholders, read more below.

You can note that we used `add dependencies={common_inputs.tex}`: this allows us to recompile the files if `common_inputs.tex` changes. If you do not want this behavior (e.g. `common_inputs.tex` changes too often and you do not want to recompile everything at every change), you can remove this line, but beware: if you do a breaking changes in `common_inputs.tex` (e.g. redefine 42 to 43), then the previously cached picture will not be recompiled! (So you will still read 42 instead of 43.)

3.5 Wrap arbitrary environments

You can wrap arbitrary environments using the already presented `cacheEnvironment` and `cacheCommand`, where the first mandatory argument is the name of the environment/macro, and the second mandatory argument contains the default preset. `cacheEnvironment` works for any environment while `cacheCommand` might need a bit of help to determine the signature of the macro if the function is defined via xparse. Long story short, `O{foo}` is an optional argument with default value foo, `m` is a mandatory argument. By default, you can pass an optional arguments via the first argument enclosed in `<>`:

```
\cacheCommand{zx}[O{}O{}O{}m]{latex, add to preamble={\usepackage{zx-calculus}\def\hello#1{Hello #1.}}}
\zx<add to preamble={\usepackage{amsmath}\def\bye#1{Bye #1.}}>[mbr=1]{ % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{\hello{Alice}}} \\
  \zxZ{\beta} \rar           & \zxZ{\text{\bye{Bob}}}
}
```

you can also disable externalization for some commands using this same command, here is for instance the code to use `todonotes`:

```
\cacheCommand{todo}[O{}m]{disable externalization}
\todo{Check how to use cacheCommand and cacheEnvironment}
```

Check how to use cacheCommand and cacheEnvironment

Which gives you

See section 4.7 for more details.

3.6 Disabling externalization

You can use `disable externalization` to disable externalization (which is particularly practical if you set `\cacheTikz`). You can configure the exact command run in that case using `command if no externalization/.code={...}`, but most of the time it should work out of the box (see section 4.3.6 for details).

Point to me if you can

This figure is not externalized. This way, it can use remember picture.

This figure is not externalized. This way, it can use remember picture.

This figure is not externalized. This way, it can use remember picture.

This figure is externalized, but cannot use remember picture.

```
% In theory all pictures should be externalized (so remember picture should fail)
\tikz[remember picture,baseline=(pointtome1.base)]
  \node[rounded corners, fill=orange](pointtome1){Point to me if you can};\\
\cacheTikz
% But we can disable it temporarily
\begin{tikzpicture}<disable externalization>[remember picture]
  \node[rounded corners, fill=red](A){This figure is not externalized.
    This way, it can use remember picture.};
  \draw[->,overlay] (A) to[bend right] (pointtome1);
\end{tikzpicture}\\

% You can also disable it globally/in a group:
{
  \robExtConfigure{disable externalization}

  \begin{tikzpicture}[remember picture]
    \node[rounded corners, fill=red](A){This figure is not externalized.
      This way, it can use remember picture.};
    \draw[->,overlay] (A.west) to[bend left] (pointtome1);
  \end{tikzpicture}\\

  \begin{tikzpicture}[remember picture]
    \node[rounded corners, fill=red](A){This figure is not externalized.
      This way, it can use remember picture.};
    \draw[->,overlay] (A.east) to[bend right] (pointtome1);
  \end{tikzpicture}\\
}

\begin{tikzpicture}
  \node[rounded corners, fill=green](A){This figure is externalized, but cannot use remember picture.};
\end{tikzpicture}
```

You can also disable externalization for some kinds of commands. For instance, the package `todonotes` requires `remember picture` and is therefore not compatible with externalization provided by this package. To disable externalization on all `\todo`, you can do:

```
\cacheCommand{todo}[O{}m]{disable externalization}
\todo{Check how to use cacheCommand and cacheEnvironment}
```

Which gives you

Check how to use cacheCommand and cacheEnvironment

3.7 Feeding data from the main document to the picture

You can feed data from the main document to the cached file using placeholders, since `set placeholder eval={__foo__}{\bar}` will evaluate `\bar` and put the result in `__foo__`. For instance, if the picture depends on the current page, you can do:

The current page is 12.

```
\begin{tikzpictureC}<set placeholder eval={__thepage__}{\thepage}>
  \node[rounded corners, fill=red]{The current page is __thepage__.};
\end{tikzpictureC}
```

3.8   Defining macros

You can define macros in the preamble:

Hello my friend.

```
\cacheMe[latex, add to preamble={\def\sayhello#1{Hello #1.}}]{
  \sayhello{my friend}
}
```

3.9   Feeding data back into the main document

For more advanced usage, you might want to compute a data and cache the result in a macro that you could use later. This is possible if you write into the file \jobname-out.tex during the compilation of the cached file (by default, we already open \writeRobExt to write to this file). This file will be automatically loaded before loading the pdf (but you can customize all these operations, for instance if you do not want to load the pdf at all; the only requirement is that you should generate a .pdf file to specify that the compilation is finished).

For instance:

We computed the cached value 1.61803.

```
\begin{CacheMe}{latex, add to preamble={\usepackage{tikz}}, do not include pdf}
We compute this data that is long to compute:
\pgfmathparse{(1 + sqrt(5))/2}% result is stored in \pgfmathresult
% We write the result to the -out file (\string\foo writes \foo to the file without evaluating it,
% so this will write "\gdef\myLongResult{1.61803}"):
% Note that CacheMe is evaluated in a group, so you want to use \gdef to define it
% outside of the group
\immediate\write\writeRobExt{%
  \string\gdef\string\myLongResult{\pgfmathresult}%
}
\end{CacheMe}

We computed the cached value \myLongResult.
```

3.10   For non-LaTeX code

Due to the way LaTeX works, non-LaTeX code can't be reliably read inside macros and some environments that parse their body (e.g. align) as some characters are removed (e.g. percent symbols are comments and are removed). For this reason, we sometimes need to separate the time where we define the code and where we insert it (this is done using placeholders, see `PlaceholderFromCode`), and we need to introduce new environments to populate the template (see section 4.5 for more details, to generate them from filename, to get the path of a file etc).

The environment `CacheMeCode` can be used for this purpose.

3.10.1   Python code

Generate an image   For instance, you can use the default `python` template to generate an image with python. The following code:
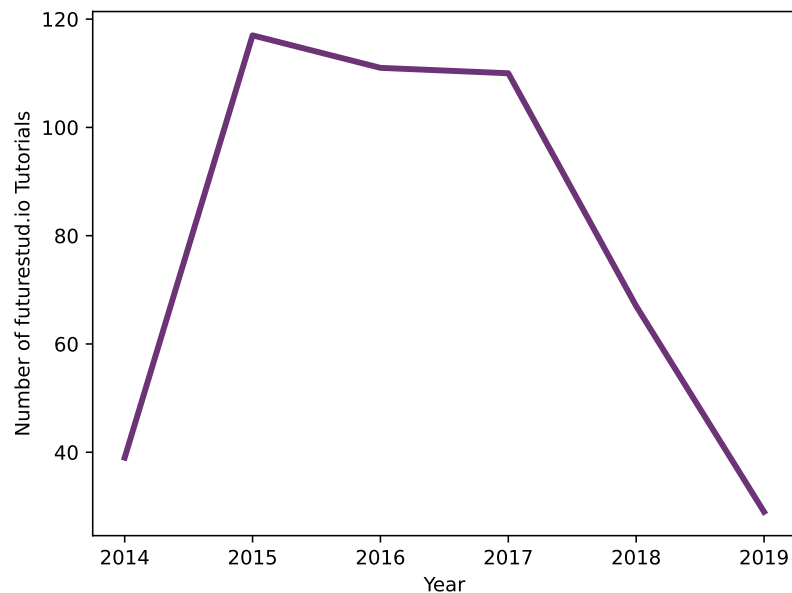
Figure 1: Image generated with python.

```
\begin{CacheMeCode}{python, set includegraphics options={width=.8\linewidth}}
import matplotlib.pyplot as plt
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{CacheMeCode}
```

will produce the image visible in fig. 1. **Importantly: you do not want to indent the content of CacheMeCode, or the space will also appear in the final code.**

Compute a value We also provide by default a number of helper functions. For instance, `write_to_out(text)` will write `text` to the `*-out.tex` file that is loaded automatically by LaTeX. This is useful to compute data that is not an image (note that `r"some string"` does not consider backslash as an escape string, which is handy to write LaTeX code in python):

For instance:

$\rightarrow$ The cosinus of 1 is 0.5403023058681398.

```
\begin{CacheMeCode}{python, do not include pdf}
import math
write_to_out(r"\gdef\cosComputedInPython{" + str(math.cos(1)) + r"}")
\end{CacheMeCode}

$\rightarrow$ The cosinus of 1 is \cosComputedInPython.
```

Improve an existing preset If you often use the same code (e.g. load matplotlib, save the file etc), you can directly modify the `__ROBEXT_MAIN_CONTENT__` placeholder to add the redundant information (or create a new template from scratch, see below). By default, it points to `__ROBEXT_MAIN_CONTENT_ORIG__` that contains directly the code typed by the user (this is true for all presets, as `CacheMe*` is in charge of setting this placeholder). When dealing with LaTeX

code, `__ROBEXT_MAIN_CONTENT__` should ideally contain a code that can be inserted as-it in the document in order to be compatible by default with `disable externalization`. So if you want to wrap the content of the user in an environment like `\begin{tikzpicture}...\end{tikzpicture}`, this is the placeholder to modify.
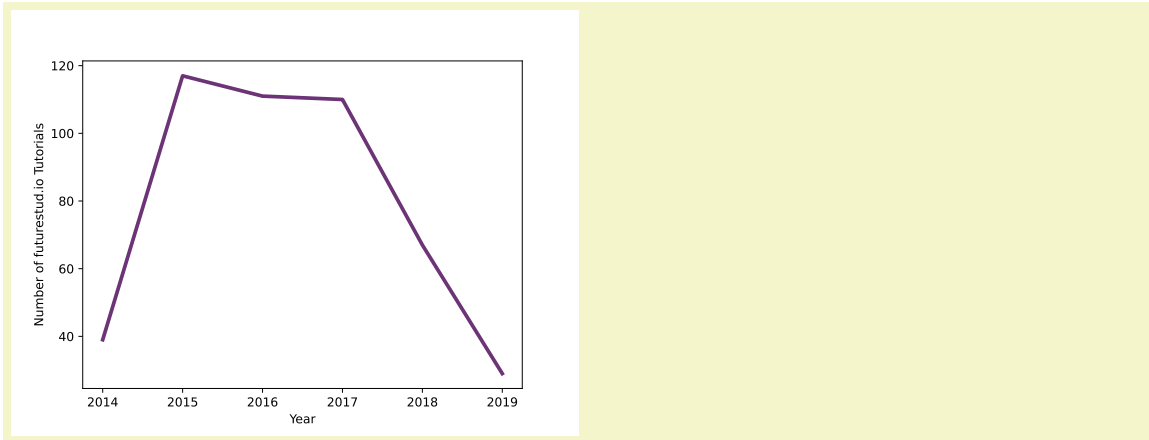

```
%% Create your style:
\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE_BEFORE__}
import matplotlib.pyplot as plt
import sys
\end{PlaceholderFromCode}

\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE_AFTER__}
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{PlaceholderFromCode}

\robExtConfigure{
  new preset={my matplotlib}{
    python,
    add before placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_MATPLOTLIB_TEMPLATE_BEFORE__},
    add to placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_MATPLOTLIB_TEMPLATE_AFTER__},
  },
}

%% Use your style:
%% See, you don't need to load matplotlib or save the file:
\begin{CacheMeCode}{my matplotlib, set includegraphics options={width=.5\linewidth}}
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
\end{CacheMeCode}
```

Custom parameters and placeholders   Let us say that you would like to define a default font size for your figure, but that you would like to allow the user to change this font size. Then, you should create a new placeholder with your default value, and use `set placeholder` to change this value later (see also the documentation of `CacheMeCode` to see how to create a new command to avoid typing `set placeholder`):

Default font size:
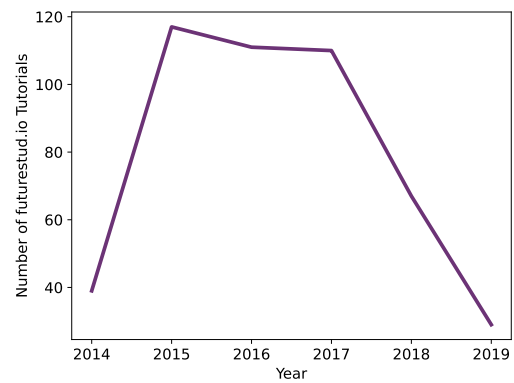
With font size 16:
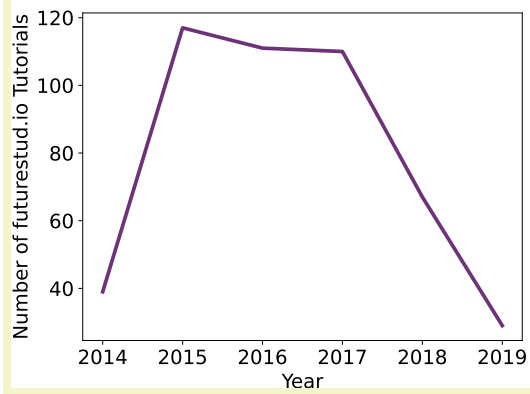
```
%% Create your style:

\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE_BEFORE__}
import matplotlib as mpl
import matplotlib.pyplot as plt
import sys
mpl.rcParams['font.size'] = __MY_MATPLOTLIB_FONT_SIZE__
\end{PlaceholderFromCode}

\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE_AFTER__}
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{PlaceholderFromCode}

\robExtConfigure{
  new preset={my matplotlib}{
    python,
    % We create a new placeholder (it is simple enough that you don't need to use PlaceholderFromCode)
    set placeholder={__MY_MATPLOTLIB_FONT_SIZE__}{12},
    add before placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_MATPLOTLIB_TEMPLATE_BEFORE__},
    add to placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_MATPLOTLIB_TEMPLATE_AFTER__},
  },
}

%% Use your style:
%% See, you don't need to load matplotlib or save the file:
Default font size: \begin{CacheMeCode}{my matplotlib, set includegraphics options={width=.5\linewidth}}
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
\end{CacheMeCode}

With font size 16:
\begin{CacheMeCode}{my matplotlib,
    set includegraphics options={width=.5\linewidth},
    set placeholder={__MY_MATPLOTLIB_FONT_SIZE__}{16}}
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
\end{CacheMeCode}
```

Note that if you manage to move all the code in the template and that the user can configure everything using the options and an empty content, you can use `CacheMeNoContent` that takes no argument and that consider its body as the options.

Custom include command There may be some cases where you do not want to include a picture. We already saw the option `do not include pdf` if you do not want to include anything. But you can customize the include function, using notably:
`custom include command={your include command}`

For instance, let us say that you would like to display both the source code used to obtain a given code, together with the output of this code. Then, you can write this style:

```
{
%% Create your style:
\begin{PlaceholderFromCode}{__MY_PRINT_BOTH_TEMPLATE_BEFORE__}
# File where print("bla") should be redirected
# get_filename_from_extension("-foo.txt") will give you the path of the file
# in the cache that looks like robExt-somehash-foo.txt
print_file = open(get_filename_from_extension("-print.txt"),  "w")
sys.stdout = print_file
# This code will read the current code, and extract the lines between
# that starts with "### CODESTARTSHERE" and "### CODESTOPSHERE", and will write
# it into the *-code.text (we do not want to print all these functions in
# the final code)
with open(get_filename_from_extension("-code.txt"), "w") as f:
    # The current script has extension .tex
    with open(get_current_script(), "r") as script:
        should_write = False
        for line in script:
            if line.startswith("### CODESTARTSHERE"):
                should_write = True
            elif line.startswith("### CODESTOPSHERE"):
                should_write = False
            elif "HIDEME" in line:
                pass
            else:
                if should_write:
                    f.write(line)
### CODESTARTSHERE
\end{PlaceholderFromCode}


\begin{PlaceholderFromCode}{__MY_PRINT_BOTH_TEMPLATE_AFTER__}
### CODESTOPSHERE
print_file.close()
\end{PlaceholderFromCode}

\robExtConfigure{
  new preset={my python print both}{
    python,
    add before placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_PRINT_BOTH_TEMPLATE_BEFORE__},
    add to placeholder no space={__ROBEXT_MAIN_CONTENT__}{__MY_PRINT_BOTH_TEMPLATE_AFTER__},
    set title/.style={
      set placeholder={__MY_TITLE__}{#1},
    },
    set title={Example},
    custom include command={
      % Useful to replace __MY_TITLE__:
      \evalPlaceholder{
        % \verbatiminput{\robExtAddCachePathAndName{\robExtFinalHash-code.txt}}
        \begin{tcolorbox}[colback=red!5!white,colframe=red!75!black,title=__MY_TITLE__]
          \lstinputlisting[frame=single,
            breakindent=.5\textwidth,
            frame=single,
            breaklines=true,
            style=mypython]{\robExtAddCachePathAndName{\robExtFinalHash-code.txt}}
          Output:
          \verbatiminput{\robExtAddCachePathAndName{\robExtFinalHash-print.txt}}
        \end{tcolorbox}
      }
    },
  },
}
```

Once the style is defined (actually we already defined in the library under the name `python print code and result`), you can just write:

```
\begin{CacheMeCode}{my python print both, set title={The for loop}}
for name in ["Alice", "Bob"]:
    print(f"Hello {name}")
\end{CacheMeCode}
```

to get:

> **The for loop**
>
> ```python
> 1  for name in ["Alice", "Bob"]:
> 2      print(f"Hello {name}")
> ```
>
> Output:
>
> ```
> Hello Alice
> Hello Bob
> ```

3.10.2   Other languages

We also provide support for other languages, notably `bash`, but it is relatively easy to add basic support for any new language. You only need to configure `set compilation command` to your command, `set template` to the file to compile (`__ROBEXT_MAIN_CONTENT__` contains the code typed by the user), and possibly a custom include command with `custom include command` if you do not want to do `\includegraphics` on the final pdf. For instance, to define a basic template for bash, you just need to use:

```
Linux 5.15.90 #1-NixOS SMP Tue Jan 24 06:22:49 UTC 2023
```

```latex
% Create your style
\begin{PlaceholderFromCode}{__MY_BASH_TEMPLATE__}
# Quit if there is an error
set -e
__ROBEXT_MAIN_CONTENT__
# Create the pdf file to certify that no compilation error occured
touch "__ROBEXT_OUTPUT_PDF__"
\end{PlaceholderFromCode}

\robExtConfigure{
  new preset={my bash}{
    set compilation command={bash "__ROBEXT_SOURCE_FILE__"},
    set template={__MY_BASH_TEMPLATE__},
    %%% Version 1:
    % verbatim output,
    %%% Version 2:
    custom include command={%
      \evalPlaceholder{%
        \verbatiminput{__ROBEXT_CACHE_FOLDER____ROBEXT_OUTPUT_PREFIX__-out.txt}%
      }%
    },
    % Ensure that the code does not break when externalization is disabled
    print verbatim if no externalization,
  }
}

% Use your style
\begin{CacheMeCode}{my bash}
# Write the system conf to a file *-out.txt
uname -srv > "__ROBEXT_OUTPUT_PREFIX__-out.txt"
\end{CacheMeCode}
```

Code inside a macro Due to fundamental LaTeX restrictions, it is impossible to use `CacheMeCode` inside a macro or some environments as LaTeX will strip all lines containing a percent character for instance. The solution here is to define our main content before, and then set it using `set main content` (that simply sets `__ROBEXT_MAIN_CONTENT_ORIG__`). In this example, we also show how `CacheMeNoContent` can be used when their is no content (the arguments to `CacheMe` are directly given in the body of `CacheMeNoContent`):
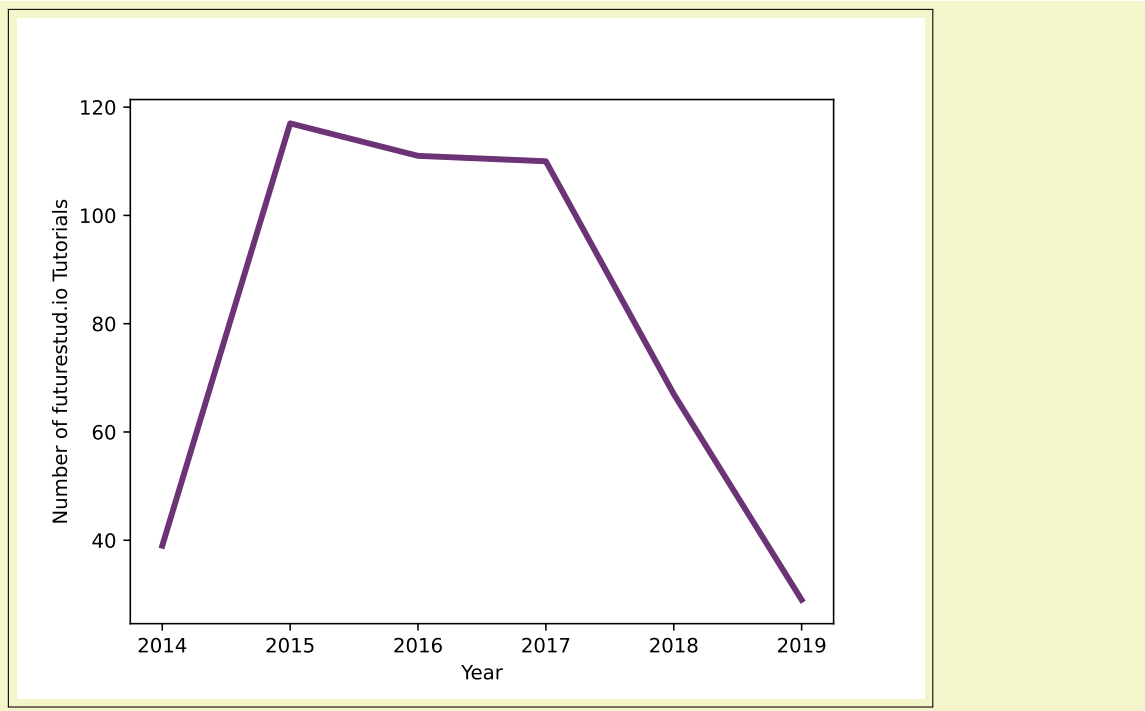
```latex
\begin{PlaceholderFromCode}{__TMP_MAIN_CONTENT__}
import matplotlib.pyplot as plt
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{PlaceholderFromCode}

\fbox{\begin{CacheMeNoContent}
  python,
  set includegraphics options={width=.8\linewidth},
  set main content={__TMP_MAIN_CONTENT__},
\end{CacheMeNoContent}}
```

4 Documentation

4.1 How it works

This library must be able to generate 3 elements for any cached content:

- a source file, that will be compiled, and is obtained by expanding the placeholder `__ROBEXT_TEMPLATE__` (see section 4.5),

- a compilation command obtained by expanding the placeholder `__ROBEXT_COMPILATION_COMMAND__`,

- a dependency file, that contains the hash of all the dependencies (see section 3.4 for details) and the compilation command,

- an inclusion command (this one is not used during the caching process, it is only used when including the compiled document in the main document), that you can set using `custom include command={your command}`.

The hash of all these elements is computed in order to obtain a reference hash, denoted `somehash` that looks like a unique random value (note that `__ROBEXT_OUTPUT_PDF__` and alike

are expanded after knowing the hash since they depend on the final hash value). This hash `somehash` will change whenever a dependency changes, or if the compilation command changes, ensuring purity. Then, the dependency file and the source file are written in the cache, by default in `robustExternalize/robExt-somehash.tex` and `robustExternalize/robExt-somehash.deps`. Then, the compilation command will be run from the cache folder. At the end, by default, we check if a file `robustExternalize/robExt-somehash.pdf` exists: if not we abort, otherwise we `\input` the file `robustExternalize/robExt-somehash-out.tex` and we run the include command (that includes the pdf by default). As we saw earlier, this command can be customized to use other files. **Importantly, all the files created during the compilation must be prefixed by** `robExt-somehash`, which can be obtained at runtime using `__ROBEXT_OUTPUT_PREFIX__`. This way, we can easily clean the cache while ensuring maximum purity.

In the following, we will denote by `*-foo.bar` the file in:
`robustExternalize/robExt-somehash-foo.bar`.

Note also that we usually define two names for each function, one normal and one prefixed with `robExt` (or `RobExt`) for environments. In this documentation, we only write the first form, but the second form is kept in case a conflicting package redefines some functions.

4.2 Placeholders

Placeholders are the main concept allowing this library to generate the content of a source file based on a template (a template will itself be a placeholder containing other placeholders). A placeholder is a special strings like `__COLOR_IMAGE__` inserted for instance in a string, that will be given a value later. This value will be used to replace (recursively) the placeholder in the template. For instance, if a placeholder `__LIKES__` contains `I like __FRUIT__ and __VEGETABLE__`, if the placeholder `__FRUIT__` contains `oranges` and if the placeholder `__VEGETABLE__` contains `salad`, then evaluating `__LIKES__` will output `I like oranges and salad`.

Note that the usage of underscore in only a convention, as any name can be used for the placeholder. There is however one rule to follow: the name of a placeholder should be made to avoid ambiguities when replacing the string, notably its name should not contain the name of another placeholder. For instance, if we define a placeholder called `NAME` containing `Foo` and a placeholder named `FULL_NAME` containing `Foo Bar`, then when evaluating the string `My name is FULL_NAME`, we have no way to know if the user wants to get `My name is FULL_Foo` or `My name is Foo Bar`. For this reason, **we advice users to start and end their placeholder names using two underscores** `__`, while using only upper case letters and single underscore `_` inside the placeholder name (this also improves readability). If you are worried about ambiguities like `__PLACEHOLDER1__PLACEHOLDER2__`, you can also use different separators for the beginning and the end like `__PLACEHOLDER1__`, but beware that `[` requires brackets around when used in pgf styles.

Placeholders are local variables (internally just some LaTeX 3 strings). You can therefore define a placeholder in a local group surrounded by brackets `{ ... }` if you want it to have a reduced scope.

4.2.1 Reading a placeholder

`\getPlaceholder[`⟨*new placeholder name*⟩`]{`⟨*name placeholder or string*⟩`}`
`\getPlaceholderInResult[`⟨*new placeholder name*⟩`]{`⟨*name placeholder or string*⟩`}`
`\getPlaceholderInResultFromList{`⟨*list,of,placeholders,to,replace*⟩`}[`⟨*new placeholder name*⟩`]{`⟨*name placeholder or string*⟩`}`

Get the value of a placeholder after replacing (recursively) all the inner placeholders. `\getPlaceholderInResult` puts the resulting string in a LaTeX 3 string `\l_robExt_result_str` and in `\robExtResult`, while `\getPlaceholder` directly outputs this string. You can also put inside the argument any arbitrary string, allowing you, for instance, to concatenate multiple placeholders, copy a placeholder etc. Note that you will get a string, but this string will not be evaluated by LaTeX (see `\evalPlaceholder` for that), for instance math will not be interpreted:

The placeholder evaluates to:
```
Hello Alice the great, I am a template $\delta _n$.
```
Combining placeholders produces:
```
In ``Hello Alice the great, I am a template $\delta _n$.'', the name is Alice
the great.
```

```
\placeholderFromContent{__MY_PLACEHOLDER__}{Hello __NAME__, I am a template $\delta_n$.}
\placeholderFromContent{__NAME__}{Alice __NICKNAME__}
\placeholderFromContent{__NICKNAME__}{the great}
The placeholder evaluates to:\\
\texttt{\getPlaceholder{__MY_PLACEHOLDER__}}\\
Combining placeholders produces:\\
\texttt{\getPlaceholder{In ``__MY_PLACEHOLDER__'', the name is __NAME__.}}
```

You can also specify the optional argument in order to additionally define a new placeholder containing the resulting string (but you might prefer to use its alias \setPlaceholderRec described below):

List of placeholders:
- Placeholder called __NEW_PLACEHOLDER__ contains:
```
In ``Hello Alice the great, I am a template $\delta _n$.'', the name is
Alice the great.
```
- Placeholder called __NICKNAME__ contains:
```
the great
```
- Placeholder called __NAME__ contains:
```
Alice __NICKNAME__
```
- Placeholder called __MY_PLACEHOLDER__ contains:
```
Hello __NAME__, I am a template $\delta _n$.
```

```
\placeholderFromContent{__MY_PLACEHOLDER__}{Hello __NAME__, I am a template $\delta_n$.}
\placeholderFromContent{__NAME__}{Alice __NICKNAME__}
\placeholderFromContent{__NICKNAME__}{the great}
\getPlaceholderInResult[__NEW_PLACEHOLDER__]{In ``__MY_PLACEHOLDER__'', the name is __NAME__.}
\printAllPlaceholdersExceptDefaults
```

The variation \getPlaceholderInResultFromList allows you to specify a set of placeholder to replace from:

List of placeholders:
- Placeholder called __NEW_PLACEHOLDER__ contains:
```
Here we go: Hello Alice, Bob and __NAME3__, I am a template $\delta _n$.
```
- Placeholder called __NAME3__ contains:
```
Charlie
```
- Placeholder called __NAME2__ contains:
```
Bob
```
- Placeholder called __NAME1__ contains:
```
Alice
```
- Placeholder called __MY_PLACEHOLDER__ contains:
```
Hello __NAME1__, __NAME2__ and __NAME3__, I am a template $\delta _n$.
```

```
\placeholderFromContent{__MY_PLACEHOLDER__}{Hello __NAME1__, __NAME2__ and __NAME3__, I am a template $\delta_n$.}
\placeholderFromContent{__NAME1__}{Alice}
\placeholderFromContent{__NAME2__}{Bob}
\placeholderFromContent{__NAME3__}{Charlie}
\getPlaceholderInResultReplaceFromList{__MY_PLACEHOLDER__,__NAME1__,__NAME2__}[__NEW_PLACEHOLDER__]{Here we go: __M
\printAllPlaceholdersExceptDefaults
```

\evalPlaceholder{⟨*name placeholder or string*⟩}

Evaluate the value of a placeholder after replacing (recursively) all the inner placeholders. You can also put inside any arbitrary string.

The placeholder evaluates to:
Hello Alice the great, I am a template δ_n.
Combining placeholders produces:
In "Hello Alice the great, I am a template δ_n.", the name is Alice the great.

```
\placeholderFromContent{__MY_PLACEHOLDER__}{Hello __NAME__, I am a template $\delta_n$.}
\placeholderFromContent{__NAME__}{Alice __NICKNAME__}
\placeholderFromContent{__NICKNAME__}{the great}
% The placeholder evaluates to \texttt{\getPlaceholder{__MY_PLACEHOLDER__}}.
The placeholder evaluates to:\\
\evalPlaceholder{__MY_PLACEHOLDER__}\\
Combining placeholders produces:\\
\evalPlaceholder{In ``__MY_PLACEHOLDER__'', the name is __NAME__.}
```

4.2.2   List and debug placeholders

It can sometimes be handy to list all placeholders, print their contents etc. We list here commands that are mostly useful for debugging purposes.

\printAllPlaceholdersExceptDefaults*

Prints the verbatim content of all defined placeholders (without performing any replacement of inner placeholders), except for the placeholders that are defined by default in this library (that we identify as they start with __ROBEXT_). The stared version does print the name of the placeholder defined in this library, but not their definition. This is mostly for debugging purposes.

List of placeholders:
- Placeholder called __NAME__ contains:

Alice

- Placeholder called __LIKES__ contains:

Hello __NAME__ I am a really basic template $\delta _n$.

```
\placeholderFromContent{__LIKES__}{Hello __NAME__ I am a really basic template $\delta_n$.}
\placeholderFromContent{__NAME__}{Alice}
\printAllPlaceholdersExceptDefaults
```

Compare with:

List of placeholders:

- Placeholder called `__NAME__` contains:

`Alice`

- Placeholder called `__LIKES__` contains:

`Hello __NAME__ I am a really basic template $\delta _n$.`

- Placeholder called `__ROBEXT_BASH_SHELL__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_BASH_TEMPLATE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_LSTINPUT_STYLE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_RESULT_MESSAGE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_CODE_MESSAGE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_TCOLORBOX_PROPS__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_PRINT_CODE_RESULT_TEMPLATE_AFTER__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_PRINT_CODE_RESULT_TEMPLATE_BEFORE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_EXEC__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_FINISHED_WITH_NO_ERROR__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_MAIN_CONTENT_WRAPPED__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON_IMPORT__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PYTHON__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_LATEX_ENGINE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_COMPILATION_COMMAND_OPTIONS__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_COMPILATION_COMMAND_LATEX__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_WRITE_DEPTH_TO_OUT_FILE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_CREATE_OUT_FILE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_MAIN_CONTENT_WRAPPED__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_LATEX__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_LATEX_TRIM_LENGTH__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_INCLUDEGRAPHICS_FILE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_INCLUDEGRAPHICS_OPTIONS__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PREAMBLE_AFTER_HYPERREF__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PREAMBLE_HYPERREF__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_PREAMBLE__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_DOCUMENT_CLASS__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_LATEX_OPTIONS__` defined by default (we hide the definition to save space)

- Placeholder called `__ROBEXT_MAIN_CONTENT__` defined by default (we hide the definition to

```
\placeholderFromContent{__LIKES__}{Hello __NAME__ I am a really basic template $\delta_n$.}
\placeholderFromContent{__NAME__}{Alice}
\printAllPlaceholdersExceptDefaults*
```

\printAllPlaceholders

Prints the verbatim content of all defined placeholders (without performing any replacement of inner placeholders), including the placeholders that are defined by default in this library. This is mostly for debugging purposes. Here is the result of \printAllPlaceholders:

List of placeholders:

- Placeholder called `__ROBEXT_BASH_SHELL__` contains:

`bash`

- Placeholder called `__ROBEXT_BASH_TEMPLATE__` contains:

`# Quit if there is an error`

`set -e`

`outputTxt="__ROBEXT_OUTPUT_PREFIX__-out.txt"`

`outputTex="__ROBEXT_OUTPUT_PREFIX__-out.tex"`

`outputPdf="__ROBEXT_OUTPUT_PDF__"`

`__ROBEXT_MAIN_CONTENT__`

`# Create the pdf file to certify that no compilation error occured`

`touch "${outputPdf}"`

- Placeholder called `__ROBEXT_PYTHON_LSTINPUT_STYLE__` contains:

`frame=single, breakindent=.5\textwidth , frame=single, breaklines=true, style=mypython`

- Placeholder called `__ROBEXT_PYTHON_RESULT_MESSAGE__` contains:

`Output:`

- Placeholder called `__ROBEXT_PYTHON_CODE_MESSAGE__` contains:

- Placeholder called `__ROBEXT_PYTHON_TCOLORBOX_PROPS__` contains:

`colback=red!5!white,colframe=red!75!black`

- Placeholder called `__ROBEXT_PYTHON_PRINT_CODE_RESULT_TEMPLATE_AFTER__` contains:

`### CODESTOPSHERE`

`print_file.close()`

- Placeholder called `__ROBEXT_PYTHON_PRINT_CODE_RESULT_TEMPLATE_BEFORE__` contains:

`# File where print("bla") should be redirected`

`# get_filename_from_extension("-foo.txt") will give you the path of the file`

`# in the cache that looks like robExt-somehash-foo.txt`

`print_file = open(get_filename_from_extension("-print.txt"), "w")`

`sys.stdout = print_file`

`# This code will read the current code, and extract the lines between`

`# that starts with "### CODESTARTSHERE" and "### CODESTOPSHERE", and will write`

`# it into the *-code.text (we do not want to print all these functions in`

`# the final code)`

`with open(get_filename_from_extension("-code.txt"), "w") as f:`

`    # The current script has extension .tex`

`    with open(get_current_script(), "r") as script:`

`        should_write = False`

```python
        for line in script:
            if line.startswith("### CODESTARTSHERE"):
                should_write = True
            elif line.startswith("### CODESTOPSHERE"):
                should_write = False
            elif "HIDEME" in line:
                pass
            else:
                if should_write:
                    f.write(line)
```
CODESTARTSHERE
- Placeholder called __ROBEXT_PYTHON_EXEC__ contains:
```
python
```
- Placeholder called __ROBEXT_PYTHON_FINISHED_WITH_NO_ERROR__ contains:
```
finished_with_no_error()
```
- Placeholder called __ROBEXT_PYTHON_MAIN_CONTENT_WRAPPED__ contains:
```python
# This file will be loaded in latex. Useful to pass data to the main document
f_out_write = open("__ROBEXT_OUTPUT_PREFIX__-out.tex", "w")
import os
import sys
def write_to_out(text):
    """Write to the -out.tex file that is loaded by default"""
    f_out_write.write(text)
def parse_args():
    args = {}
    if len(sys.argv) % 2 == 0:
        print("Error: the number of arguments must be even, as tuples of name
and value")
        exit(1)
    for i in range(0,len(sys.argv)-1,2):
        args[sys.argv[i+1]] = sys.argv[i+2]
    return args
def get_cache_folder():
    '''
    Path of the cache folder. Warning: this works only when the python script
    is located in this cache folder (that should be true when it's called
from LaTeX)
    '''
    return os.path.abspath(os.path.dirname(sys.argv[0]))
def get_file_base():
    '''
    Outputs the base of the files (i.e. something like robExt-somehash,
without any extension)
    '''
    return os.path.splitext(os.path.basename(sys.argv[0]))[0] # __file__ does
not work as it refers to the library
```

```python
def get_current_script():
    '''
    Outputs the path of the current script
    '''
    return os.path.abspath(sys.argv[0]) # __file__ does not work as it refers
to the library
def get_filename_from_extension(extension):
    '''
    If you want to create a file with extension 'extension' (with the
appropriate base name), this command
    is for you. For instance get_filename_from_extension(".mp4") would return
something like
    robExt-somehash.mp4
    the extension can also be like get_filename_from_extension("-out.tex")
etc.
    '''
    return os.path.join(get_cache_folder(), get_file_base() + extension)
def get_verbatim_output():
    '''Returns the path to -out.txt that is read by verbatim output'''
    return get_filename_from_extension("-out.txt")
def get_pdf_output():
    '''Returns the path to -out.txt that is read by verbatim output'''
    return get_filename_from_extension(".pdf")
def finished_with_no_error():
    '''
    Call this at the end of your script. This creates the path of the final
pdf file that should be
    created (otherwise robust-externalize will think that the compilation
failed)
    '''
    if not os.path.exists(get_filename_from_extension(".pdf")):
        # we create an empty path
        with open(get_filename_from_extension(".pdf"), 'w') as f:
            pass
### Starting main content
__ROBEXT_MAIN_CONTENT__
### Ending main content
__ROBEXT_PYTHON_FINISHED_WITH_NO_ERROR__
f_out_write.close()
```

- Placeholder called __ROBEXT_PYTHON_IMPORT__ contains:

- Placeholder called __ROBEXT_PYTHON__ contains:

```
__ROBEXT_PYTHON_IMPORT__
__ROBEXT_PYTHON_MAIN_CONTENT_WRAPPED__
```
- Placeholder called __ROBEXT_LATEX_ENGINE__ contains:

```
pdflatex
```

- Placeholder called `__ROBEXT_COMPILATION_COMMAND_OPTIONS__` contains:

`-halt-on-error`

- Placeholder called `__ROBEXT_COMPILATION_COMMAND_LATEX__` contains:

`__ROBEXT_LATEX_ENGINE__ __ROBEXT_COMPILATION_COMMAND_OPTIONS__`
`"__ROBEXT_SOURCE_FILE__"`

- Placeholder called `__ROBEXT_WRITE_DEPTH_TO_OUT_FILE__` contains:

```
\immediate\write\writeRobExt{%
  \string\def\string\robExtWidth{\the\wd\boxRobExt}%
  \string\def\string\robExtHeight{\the\ht\boxRobExt}%
  \string\def\string\robExtDepth{\the\dp\boxRobExt}%
}%
```

- Placeholder called `__ROBEXT_CREATE_OUT_FILE__` contains:

```
%% We save the height/depth of the content by using a savebox:
\newwrite\writeRobExt%
\immediate\openout\writeRobExt=\jobname-out.tex%
```

- Placeholder called `__ROBEXT_MAIN_CONTENT_WRAPPED__` contains:

```
__ROBEXT_CREATE_OUT_FILE__%
\newsavebox\boxRobExt%
\begin{lrbox}{\boxRobExt}%
  __ROBEXT_MAIN_CONTENT__%
\end{lrbox}%
\usebox{\boxRobExt}%
__ROBEXT_WRITE_DEPTH_TO_OUT_FILE__%
```

- Placeholder called `__ROBEXT_LATEX__` contains:

```
\documentclass[__ROBEXT_LATEX_OPTIONS__]{__ROBEXT_DOCUMENT_CLASS__}
__ROBEXT_PREAMBLE__
% most packages must be loaded before hyperref
% so we typically want to load hyperref here
__ROBEXT_PREAMBLE_HYPERREF__
% some packages must be loaded after hyperref
__ROBEXT_PREAMBLE_AFTER_HYPERREF__
\begin{document}%
__ROBEXT_MAIN_CONTENT_WRAPPED__
\end{document}
```

- Placeholder called `__ROBEXT_LATEX_TRIM_LENGTH__` contains:

`30cm`

- Placeholder called `__ROBEXT_INCLUDEGRAPHICS_FILE__` contains:

`\robExtAddCachePathAndName {\robExtFinalHash .pdf}`

- Placeholder called `__ROBEXT_INCLUDEGRAPHICS_OPTIONS__` contains:

- Placeholder called `__ROBEXT_PREAMBLE_AFTER_HYPERREF__` contains:

- Placeholder called `__ROBEXT_PREAMBLE_HYPERREF__` contains:

- Placeholder called `__ROBEXT_PREAMBLE__` contains:

- Placeholder called `__ROBEXT_DOCUMENT_CLASS__` contains:

`standalone`

- Placeholder called `__ROBEXT_LATEX_OPTIONS__` contains:

- Placeholder called `__ROBEXT_MAIN_CONTENT__` contains:

`__ROBEXT_MAIN_CONTENT_ORIG__`

- Placeholder called `__ROBEXT_UNDERSCORE__` contains:

`_`

- Placeholder called `__ROBEXT_PERCENT__` contains:

`%`

- Placeholder called `__ROBEXT_HASH__` contains:

`#`

- Placeholder called `__ROBEXT_BACKSLASH__` contains:

`\`

- Placeholder called `__ROBEXT_RIGHT_BRACE__` contains:

`}`

- Placeholder called `__ROBEXT_LEFT_BRACE__` contains:

`{`

- Placeholder called `__ROBEXT_VERBATIM_COMMAND__` contains:

`\verbatiminput`

\printPlaceholderNoReplacement{⟨*name placeholder*⟩}

Prints the verbatim content of a given placeholder, without evaluating it and **without replacing inner placeholders: it is used mostly for debugging purposes** and will be used in this documentation to display the content of the placeholder for educational purposes. The stared version prints it inline.

> The (unexpanded) template contains
> `Hello NAME I am a really basic template $\delta _n$.`
> .
> The (unexpanded) template contains `Hello NAME I am a really basic template $\delta _n$.`

```
\placeholderFromContent{__LIKES__}{Hello NAME I am a really basic template $\delta_n$.}
\placeholderFromContent{NAME}{Alice}
The (unexpanded) template contains \printPlaceholderNoReplacement{__LIKES__}.\\
The (unexpanded) template contains \printPlaceholderNoReplacement*{__LIKES__}
```

\printPlaceholder{⟨*name placeholder*⟩}

Like \printPlaceholderNoReplacement except that it first replaces the inner placeholders. The stared version prints it inline.

> The (unexpanded) template contains
> `Hello Alice I am a really basic template $\delta _n$.`
> .
> The (unexpanded) template contains `Hello Alice I am a really basic template $\delta _n$.`

```
\placeholderFromContent{__LIKES__}{Hello NAME I am a really basic template $\delta_n$.}
\placeholderFromContent{NAME}{Alice}
The (unexpanded) template contains \printPlaceholder{__LIKES__}.\\
The (unexpanded) template contains \printPlaceholder*{__LIKES__}
```

\evalPlaceholderNoReplacement{⟨*name placeholder*⟩}

Evaluates the content of a given placeholder as a LaTeX code, **without replacing the place-holders contained inside (mostly used for debugging purposes).**

> The (unexpanded) template evaluates to "Hello NAME I am a really basic template δ_n.".

```
\placeholderFromContent{__LIKES__}{Hello NAME I am a really basic template $\delta_n$.}
\placeholderFromContent{NAME}{Alice}
The (unexpanded) template evaluates to ``\evalPlaceholderNoReplacement{__LIKES__}''.
```

\getPlaceholderNoReplacement{⟨*name placeholder*⟩}

Like \evalPlaceholderNoReplacement except that it only outputs the string without evaluating the macros inside.

> The (unexpanded) template contains `Hello NAME I am a really basic template $\delta _n$.`

```
\placeholderFromContent{__LIKES__}{Hello NAME I am a really basic template $\delta_n$.}
\placeholderFromContent{NAME}{Alice}
The (unexpanded) template contains \texttt{\getPlaceholderNoReplacement{__LIKES__}}
```

4.2.3   Setting a value to a placeholder

\placeholderFromContent{⟨*name placeholder*⟩}{⟨*content placeholder*⟩}
\setPlaceholder{⟨*name placeholder*⟩}{⟨*content placeholder*⟩}
/robExt/set placeholder={⟨*name placeholder*⟩}{⟨*content placeholder*⟩}   (style, no default)
/robExt/set placeholder from content={⟨*name placeholder*⟩}{⟨*content placeholder*⟩} (style, no default)

\placeholderFromContent (and its alias \setPlaceholder and its equivalent pgf styles /robExt/set placeholder and /robExt/set placeholder from content) is useful to set a value to a given placeholder.

> The (unexpanded) template contains
>
> `Hello I am a basic template with math $\delta _n$ and macros \hello`
>
> and after evaluation and setting the value of hello, you get "Hello I am a basic template with math δ_n and macros Hello my friend!".

```
\placeholderFromContent{__LIKES__}{Hello I am a basic template with math $\delta_n$ and macros \hello}
The (unexpanded) template contains \printPlaceholderNoReplacement{__LIKES__} and %
after evaluation and setting the value of hello,%
\def\hello{Hello my friend!}%
you get ``\evalPlaceholder{__LIKES__}''.
```

As you can see, **the precise content is not exactly identical to the original string**: LaTeX comments are removed, spaces are added after macros, some newlines are removed etc. While this is usually not an issue when dealing with LaTeX code, it causes some troubles when dealing with non-LaTeX code. For this reason, we define **other commands** (see for instance PlaceholderFromCode below) that can accept verbatim content; the downside being that LaTeX forbids usage of these verbatim commands inside other macros, so you should always define them at the top level (this seems to be fundamental to how LaTeX works, as any input to a macro gets interpreted first as a LaTeX string, losing all comments for instance). Note

that this is not as restrictive as it may sound, as it is always possible to define the needed placeholders before any macro, while using them inside the macro, possibly combining them with other placeholders (defined either before or inside the macro).

But before seeing how to define placeholder containing arbitrary code, let us first see how we can define a placeholder recursively, by giving it a value based on its previous value (useful for instance in order to add stuff to it).

\setPlaceholderRec{⟨*new placeholder*⟩}{⟨*content with placeholder*⟩}
\setPlaceholderRecReplaceFromList{⟨*list,of,placeholder,to,replace*⟩}{⟨*new placeholder*⟩}{⟨*content with placeholder*⟩}
/robExt/set placeholder rec={⟨*name placeholder*⟩}{⟨*content placeholder*⟩} (style, no default)
/robExt/set placeholder rec replace from list={⟨*list,of,placeholder,to,replace*⟩}{⟨*name placeholder*⟩}{⟨*content placeholder*⟩} (style, no default)

\setPlaceholderRec{foo}{bar} is actually an alias for \getPlaceholderInResult[foo]{bar}. Note that contrary to \setPlaceholder, it recursively replaces all inner placeholders. This is particularly useful to add stuff to an existing (or not) placeholder:

List of placeholders:
- Placeholder called __MY_COMMAND__ contains:

pdflatex myfile

```
\setPlaceholderRec{__MY_COMMAND__}{pdflatex}
\setPlaceholderRec{__MY_COMMAND__}{__MY_COMMAND__ myfile}
\printAllPlaceholdersExceptDefaults
```

Note that the if the placeholder content contains at the end the placeholder name, we will automatically remove it to avoid infinite recursion at evaluation time. This has the benefit that you can add something to a placeholder even if this placeholder does not exists yet (in which case it will be understood as the empty string):

List of placeholders:
- Placeholder called __COMMAND_ARGS__ contains:

 -l -s

```
\setPlaceholderRec{__COMMAND_ARGS__}{__COMMAND_ARGS__ -l}
\setPlaceholderRec{__COMMAND_ARGS__}{__COMMAND_ARGS__ -s}
\printAllPlaceholdersExceptDefaults
```

The variation \setPlaceholderRecReplaceFromList allows us to specify a subset of placeholder that will be allowed to be expanded, and is an alias for \getPlaceholderInResultReplaceFromList (except that the optional argument is mandatory):

List of placeholders:
- Placeholder called __OTHER_PLACEHOLDER__ contains:

Here we go: Hello __NAME1__, Bob and Charlie, I am a template $\delta _n$.
- Placeholder called __NAME3__ contains:

Charlie
- Placeholder called __NAME2__ contains:

Bob
- Placeholder called __NAME1__ contains:

Alice
- Placeholder called __MY_PLACEHOLDER__ contains:

Hello __NAME1__, __NAME2__ and __NAME3__, I am a template $\delta _n$.

```
\placeholderFromContent{__MY_PLACEHOLDER__}{Hello __NAME1__, __NAME2__ and __NAME3__, I am a template $\delta_n$.}
\placeholderFromContent{__NAME1__}{Alice}
\placeholderFromContent{__NAME2__}{Bob}
\placeholderFromContent{__NAME3__}{Charlie}
\setPlaceholderRecReplaceFromList{__MY_PLACEHOLDER__,__NAME2__,__NAME3__}{__OTHER_PLACEHOLDER__}{Here we go: __MY_P
\printAllPlaceholdersExceptDefaults
```

Note that sometimes, you might not want to use \setPlaceholderRec to simply append some data to the placeholder as it will also evaluate the inner placeholders (meaning that you will not be able to redefine them later). For this reason, we also provide functions to add something to the placeholder without evaluating it first:

\addToPlaceholder*{⟨placeholder⟩}{⟨content to add⟩}
\addBeforePlaceholder*{⟨placeholder⟩}{⟨content to add⟩}
/robExt/add to placeholder={⟨name placeholder⟩}{⟨content to add⟩}   (style, no default)
/robExt/add to placeholder no space={⟨name placeholder⟩}{⟨content to add⟩}   (style, no default)
/robExt/add before placeholder={⟨name placeholder⟩}{⟨content to add⟩} (style, no default)
/robExt/add before placeholder no space={⟨name placeholder⟩}{⟨content to add⟩}   (style, no default)
/robExt/add before main content={⟨name placeholder⟩}{⟨content to add⟩}(style, no default)

　　\addToPlaceholder{foo}{bar} adds bar at the end of the placeholder foo (by default it also adds a space, unless you use the star version), creating it if it does not exist (the before variants add the content. . . before).

List of placeholders:
- Placeholder called __COMMAND__ contains:

time __ENGINE__ --option --other-option

- Placeholder called __ENGINE__ contains:

pdflatex

```
\setPlaceholder{__ENGINE__}{pdflatex}
\setPlaceholder{__COMMAND__}{__ENGINE__ --option}
\addToPlaceholder{__COMMAND__}{--other}
\addToPlaceholder*{__COMMAND__}{-option}
\addBeforePlaceholder{__COMMAND__}{time}
\printAllPlaceholdersExceptDefaults
```

　　add before main content is a particular case where the placeholder is __ROBEXT_MAIN_CONTENT__. It is practical if you want to define for instance a macro, but in a way that even if you disable externalization, the command should still compile (if you define the macro in the preamble, it will not be added when disabling externalization). For instance:

Here I am cached (Hello Alice.) and Here I am not (Hello Bob.).

```
\robExtConfigure{
  new preset={my preset}{latex, add before main content={\def\hello#1{Hello #1.}}},
}
\cacheMe[my preset]{Here I am cached (\hello{Alice})} and \cacheMe[my
preset, disable externalization]{Here I am not (\hello{Bob}).}
```

\placeholderFromString{⟨latex3 string⟩}
\setPlaceholderFromString{⟨latex3 string⟩}
　　(new in v1.1) This allows you to assign an existing LATEX3 string to a placeholder.

List of placeholders:
- Placeholder called `__my_percent_string__` contains:

```
%
```

```
\ExplSyntaxOn
\setPlaceholderFromString{__my_percent_string__}{\c_percent_str}
\printAllPlaceholdersExceptDefaults
\ExplSyntaxOff
```

We provide a list of placeholders that are useful to escape parts of the strings (but you should not really need them, if you need weird characters like percent, most of the time you want to use placeholderFromCode):

```
String containing {, }, \, #, _, %.
```

```
\printPlaceholder{
  String containing
  __ROBEXT_LEFT_BRACE__,
  __ROBEXT_RIGHT_BRACE__,
  __ROBEXT_BACKSLASH__,
  __ROBEXT_HASH__,
  __ROBEXT_UNDERSCORE__,
  __ROBEXT_PERCENT__.
}
```

\robExtPlaceholderReplaceInplace{⟨*placeholder*⟩}{⟨*from*⟩}{⟨*to*⟩}
\robExtPlaceholderReplaceInplaceEval{⟨*placeholder*⟩}{⟨*from*⟩}{⟨*to*⟩}
/robExt/placeholder replace in place={⟨*placeholder*⟩}{⟨*from*⟩}{⟨*to*⟩}   (style, no default)
/robExt/placeholder replace in place eval={⟨*placeholder*⟩}{⟨*from*⟩}{⟨*to*⟩}   (style, no default)

(new in v1.1) This allows you to replace a value in a placeholder. The `eval` variation first evaluates the string.

List of placeholders:
- Placeholder called `__NAMES__` contains:

```
Hello Charlie and Dylan.
```

```
\def\nameFrom{Bob}
\def\nameTo{Dylan}
\robExtConfigure{
  set placeholder={__NAMES__}{Hello Alice and Bob.},
  placeholder replace in place={__NAMES__}{Alice}{Charlie},
  placeholder replace in place eval={__NAMES__}{\nameFrom}{\nameTo},
}
\printAllPlaceholdersExceptDefaults
```

\placeholderHalveNumberHashesInplace{⟨*placeholder*⟩}
\placeholderDoubleNumberHashesInplace{⟨*placeholder*⟩}
/robExt/placeholder halve number hashes in place={⟨*placeholder*⟩}   (style, no default)
/robExt/placeholder double number hashes in place={⟨*placeholder*⟩}   (style, no default)

(new in v1.1) This allows you to either turn any `##` into `#` or the other way around (may be practical when dealing with arguments to functions).

List of placeholders:
- Placeholder called `__DEMO__` contains:

```
\def \hey #1{Hey #1.}
```

```
\robExtConfigure{
  set placeholder={__DEMO__}{\def\hey##1{Hey ##1.}},
  placeholder halve number hashes in place={__DEMO__},
}
\printAllPlaceholdersExceptDefaults
```

\removePlaceholder{⟨*placeholder*⟩}
/robExt/remove placeholder={⟨*placeholder*⟩}   (style, no default)
/robExt/remove placeholders={⟨*list,of,placeholder*⟩}   (style, no default)

   (new in v1.1) Remove placeholders.

```
List of placeholders:
- Placeholder called __DEMOB__ contains:
Bob
```

```
\robExtConfigure{
  set placeholder={__DEMOA__}{Alice},
  set placeholder={__DEMOB__}{Bob},
  set placeholder={__DEMOC__}{Charlie},
  remove placeholders={__DEMOA__,__DEMOC__},
}
\printAllPlaceholdersExceptDefaults
```

\evalPlaceholderInplace{⟨*name placeholder*⟩}
/robExt/eval placeholder inplace={⟨*name placeholder*⟩}   (style, no default)

   This command will update (inplace) the content of a macro by first replacing recursively the
   placeholders, and finally by expanding the LATEX macros.

```
List of placeholders:
- Placeholder called __MACRO_EVALUATED__ contains:
Initial value
- Placeholder called __MACRO_NOT_EVALUATED__ contains:
\mymacro
Compare Initial value and Final value.
```

```
\def\mymacro{Initial value}
\placeholderFromContent{__MACRO_NOT_EVALUATED__}{\mymacro}
\placeholderFromContent{__MACRO_EVALUATED__}{\mymacro}
\evalPlaceholderInplace{__MACRO_EVALUATED__}
\printAllPlaceholdersExceptDefaults
\def\mymacro{Final value}
Compare \evalPlaceholder{__MACRO_EVALUATED__} and \evalPlaceholder{__MACRO_NOT_EVALUATED__}.
```

/robExt/set placeholder eval={⟨*name placeholder*⟩}{⟨*content placeholder*⟩}   (style, no
   default)

   Alias for \setPlaceholderRec{#1}{#2}\evalPlaceholderInplace{#1}: set and evaluate
   recursively the placeholders and macros. This can be practical to pass the value of a coun-
   ter/macro to the template (of course, if this value is fixed, you can also directly load it from
   the preambule):

The current page is 34.

```
\begin{CacheMe}{tikzpicture, set placeholder eval={__thepage__}{\thepage}}
  \node[rounded corners, fill=red]{The current page is __thepage__.};
\end{CacheMe}
```

Note that this works well for commands that expand completely, but some more complex commands might not expand properly (like `cref`). I need to investigate how to solve this issue, meanwhile you can still disable externalization for these pictures.

`\begin{PlaceholderFromCode}{⟨name placeholder⟩}`
  ⟨*environment contents*⟩
`\end{PlaceholderFromCode}`
`\begin{setPlaceholderCode}{⟨name placeholder⟩}`
  ⟨*environment contents*⟩
`\end{setPlaceholderCode}`

These two (aliased) environments are useful to set a verbatim value to a given placeholder: the advantage is that you can put inside any code, including LaTeX comments, the downside is that you cannot use it inside macros and some environments (so you typically define it before the macros and call it inside).

```
List of placeholders:
- Placeholder called __PYTHON_CODE__ contains:
def my_function(b): # this is a python code
    c = {}
    d[42] = 0
    return b
```

```
\begin{PlaceholderFromCode}{__PYTHON_CODE__}
def my_function(b): # this is a python code
    c = {}
    d[42] = 0
    return b
\end{PlaceholderFromCode}
\printAllPlaceholdersExceptDefaults
```

Note that `PlaceholderFromCode` should not be used inside other macros or inside some environments (notably the ones that need to evaluate the body of the environment, e.g. using `+b` argument or `environ`) as verbatim content is parsed first by the macro, meaning that some characters might be changed or removed. For instance, any percent character would be considered as a comment, removing the rest of the line. However, this should not be be problem if you use it outside of any macro or environment, or if you load it from a file. For instance this code:

```
\begin{PlaceholderFromCode}{__PYTHON_CODE__}
def my_function(b): # this is a python code
    c = {}
    d[42] = 0
    return b % 2
\end{PlaceholderFromCode}
\printAllPlaceholdersExceptDefaults
```

would produce:

```
List of placeholders:
- Placeholder called __PYTHON_CODE__ contains:
def my_function(b): # this is a python code
    c = {}
    d[42] = 0
    return b % 2
```

```
\printAllPlaceholdersExceptDefaults
```

Note that of course, you can define a placeholder before a macro and call it inside (explaining how we can generate this documentation).

\placeholderPathFromFilename{⟨*name placeholder*⟩}{⟨*filename*⟩}
/robExt/set placeholder path from filename={⟨*name placeholder*⟩}{⟨*filename*⟩} (style, no default)

\placeholderPathFromFilename{__MYLIB__}{mylib.py} will copy mylib.py in the cache (setting its hash depending on its content), and set the content of the placeholder __MYLIB__ to the **path** of the library in the cache. Note that the path is relative to the cache folder (it is easier to use for instance if you want to call this library from a code already in the cache).

List of placeholders:
- Placeholder called __MYLIB__ contains:

robExt-F6666F86DB0ACE43E817A7EB3729FA56mylib.py

You can also get the path relative to the root folder:
robustExternalize/robExt-F6666F86DB0ACE43E817A7EB3729FA56mylib.py

```
\placeholderPathFromFilename{__MYLIB__}{mylib.py}
\printAllPlaceholdersExceptDefaults
You can also get the path relative to the root folder:\\
\robExtAddCachePath{\getPlaceholderNoReplacement{__MYLIB__}}
```

\placeholderFromFileContent{⟨*name placeholder*⟩}{⟨*filename*⟩}
/robExt/set placeholder from file content={⟨*name placeholder*⟩}{⟨*filename*⟩} (style, no default)

\placeholderFromFileContent{__MYLIB__}{mylib.py} will set the content of the placeholder __MYLIB__ to the content of mylib.py.

List of placeholders:
- Placeholder called __MYLIB__ contains:

```
def mylib():
    # Some comments
    a = b % 2
    return "Hello world"
```

```
\placeholderFromFileContent{__MYLIB__}{mylib.py}
\printAllPlaceholdersExceptDefaults
```

\placeholderPathFromContent{⟨*name placeholder*⟩}[⟨*suffix*⟩]{⟨*content*⟩}
/robExt/set placeholder path from content={⟨*name placeholder*⟩}{⟨*suffix*⟩}{⟨*content*⟩}
(style, no default)

\placeholderPathFromContent{__MYLIB__}{some content} will copy some content in a file in the cache (setting its hash depending on its content, the filename will end with suffix that defaults to .tex), and set the content of the placeholder __MYLIB__ to the **path** of the file in the cache. Note that the path is relative to the cache folder (it is easier to use for instance if you want to call this library from a code already in the cache).

List of placeholders:
- Placeholder called __MYLIB__ contains:

robExt-AC364A656060BFF5643DD21EAF3B64E6.py

You can also get the path relative to the root folder:
robustExternalize/robExt-AC364A656060BFF5643DD21EAF3B64E6.py
As a sanity check, this file contains

some contents b

```
\placeholderPathFromContent{__MYLIB__}[.py]{some contents b}
\printAllPlaceholdersExceptDefaults
You can also get the path relative to the root folder:\\
\robExtAddCachePath{\getPlaceholderNoReplacement{__MYLIB__}}\\
As a sanity check, this file contains
\verbatiminput{\robExtAddCachePath{\getPlaceholderNoReplacement{__MYLIB__}}}
```

\begin{PlaceholderPathFromCode}[⟨*suffix*⟩]{⟨*name placeholder*⟩}
  ⟨*environment contents*⟩
\end{PlaceholderPathFromCode}

This environment is similar to \placeholderPathFromContent except that it accepts verbatim code (therefore LaTeX comments, newlines etc. will not be removed). However, due to LaTeX limitations, this environment cannot be used inside macros or some environments, or this property will not be preserved. For instance, if you create your placeholder using:

```
\begin{PlaceholderPathFromCode}[.py]{__MYLIB__}
def my_function(b): # this is a python code
    c = {}
    d[42] = 0
    return b % 2
\end{PlaceholderPathFromCode}
```

You can then use it like:

> List of placeholders:
> - Placeholder called __MYLIB__ contains:
>
> `robExt-CDAE704490400F29B9C0C8DAE2CC48B7.py`
>
> You can also get the path relative to the root folder:
> robustExternalize/robExt-CDAE704490400F29B9C0C8DAE2CC48B7.py
> As a sanity check, this file contains
> ```
>     def my_function(b): # this is a python code
>         c = {}
>         d[42] = 0
>         return b % 2
> ```

```
\printAllPlaceholdersExceptDefaults
You can also get the path relative to the root folder:\\
\robExtAddCachePath{\getPlaceholderNoReplacement{__MYLIB__}}\\
As a sanity check, this file contains
\verbatiminput{\robExtAddCachePath{\getPlaceholderNoReplacement{__MYLIB__}}}
```

\copyPlaceholder{⟨*new placeholder*⟩}{⟨*old placeholder*⟩}
 /robExt/copy placeholder={⟨*new placeholder*⟩}{⟨*old placeholder*⟩}   (style, no default)

This creates a new placeholder with the content of old placeholder. Note that this is different from:
\setPlaceholder{new placeholder}{old placeholder}
because if we modify old placeholder, this will not affect new placeholder.

> List of placeholders:
> - Placeholder called __MY_CONTENT__ contains:
>
> `The content used to be __MY_OLD_CONTENT__`
>
> - Placeholder called __MY_OLD_CONTENT__ contains:
>
> `Some content`

```
\setPlaceholder{__MY_CONTENT__}{Some content}
\copyPlaceholder{__MY_OLD_CONTENT__}{__MY_CONTENT__}
\setPlaceholder{__MY_CONTENT__}{The content used to be __MY_OLD_CONTENT__}
\printAllPlaceholdersExceptDefaults
```

## 4.3 Caching a content

4.3.1 Basics

\cacheMe[⟨*preset style*⟩]{⟨*content to cache*⟩}
\begin{CacheMe}{⟨*preset style*⟩}
  ⟨*environment contents*⟩
\end{CacheMe}

This command (and its environment alias) is the main entry point if you want to cache the result of a file. The preset style is a pgfkeys-based style that is used to configure the template that is used, the compilation command, and more. You can either inline the style, or use some presets that configure the style automatically. After evaluating the style, the placeholders __ROBEXT_TEMPLATE__ (containing the content of the file) and __ROBEXT_COMPILATION_COMMAND__ (containing the compilation command run in the cache folder, that can use other placeholders internally like __ROBEXT_SOURCE_FILE__ to get the path to the source file) should be set. Note that we provide some basic styles that allow settings these placeholders easily. See section 4.5 for a list of existing placeholders and presets. The placeholder __ROBEXT_MAIN_CONTENT_ORIG__ will automatically be set by this command (or environment) so that it equals the content of the second argument (or the body of the environment). By default, __ROBEXT_MAIN_CONTENT__ will point to __ROBEXT_MAIN_CONTENT_ORIG__, possibly wrapping it inside an environment like \begin{tikzpicture} (most of the time, you want to modify and display __ROBEXT_MAIN_CONTENT__ rather than the _ORIG_ to easily recover the input of the user). This style can also configure the command to use to include the file and more. By default it will insert the compiled PDF, making sure that the depth is respected (internally, we read the depth from an aux file created by our LaTeX preset), but you can easily change it to anything you like.

For an educational purpose, we write here an example that does not exploit any preset. In practice, we recommend however to use our presets, or to define new presets based on our presets (see below for examples).
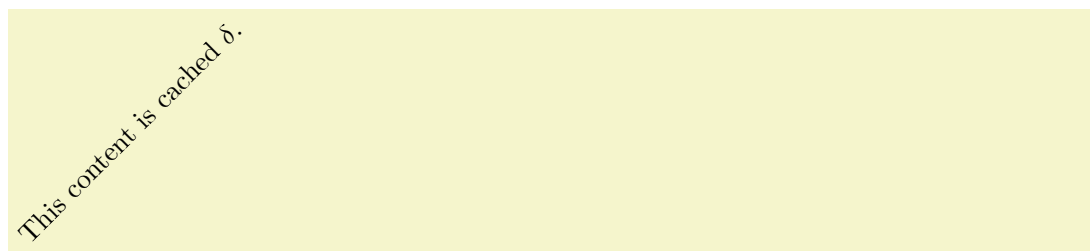

```
\begin{CacheMe}{set template={
    \documentclass{standalone}
    \begin{document}
    __ROBEXT_MAIN_CONTENT__
    \end{document}
  },
  set compilation command={pdflatex -shell-escape -halt-on-error "__ROBEXT_SOURCE_FILE__"},
  custom include command={%
    \includegraphics[width=4cm,angle=45]{\robExtAddCachePathAndName{\robExtFinalHash.pdf}}%
  },
}
This content is cached $\delta$.
\end{CacheMe}
```

\robExtConfigure{⟨*preset style*⟩}
/robExt/new preset={⟨*name preset*⟩}{⟨*preset options*⟩}   (style, no default)
/robExt/add to preset={⟨*name preset*⟩}{⟨*preset options*⟩}   (style, no default)

You can then create your own style (or preset) in `\robExtConfigure` (that is basically an alias for `\pgfkeys{/robExt/.cd,#1}`) containing your template, add your own placeholders and commands to configure them etc. We provide two helper functions since v1.1:

`new preset={your preset}{your configuration}`

and

`add to preset={your preset}{your configuration}`

in order to create/modify the presets. You can also use `my preset/.style` or

`my preset/.append style`

to configure them instead, but in that case make sure to double the number of hashes like in `\def\mymacro##1{hello ##1}`, as the #1 in `\def\mymacro#1{hello #1}`. would be understood as the (non-existent) argument of `my preset`.


```
%% Define your presets once:
\robExtConfigure{%
  new preset={my latex preset}{
    %% Create a default value for my new placeholders:
    set placeholder={__MY_COLOR__}{red},
    set placeholder={__MY_ANGLE__}{45},
    % We can also create custom commands to "hide" the notion of placeholder
    set my angle/.style={
      set placeholder={__MY_ANGLE__}{#1}
    },
    set template={
      \documentclass{standalone}
      \usepackage{xcolor}
      \begin{document}
      \color{__MY_COLOR__}__ROBEXT_MAIN_CONTENT__
      \end{document}
    },
    set compilation command={pdflatex -shell-escape -halt-on-error "__ROBEXT_SOURCE_FILE__"},
    custom include command={%
      % The include command is a regular LaTeX command, but using
      % \evalPlaceholder avoids the need to play with expandafter, getPlaceholder etc...
      \evalPlaceholder{%
        \includegraphics[width=4cm,angle=__MY_ANGLE__,origin=c]{%
          \robExtAddCachePathAndName{\robExtFinalHash.pdf}%
        }%
      }%
    },
  },
}

% Reuse them later...
\begin{CacheMe}{my latex preset}
This content is cached $\delta$.
\end{CacheMe}
% And configure them at will
\begin{CacheMe}{my latex preset, set placeholder={__MY_COLOR__}{green}, set my angle=-45}
This content is cached $\delta$.
\end{CacheMe}
```

`\begin{CacheMeCode}{`⟨*preset style*⟩`}`

⟨*environment contents*⟩
\end{CacheMeCode}

Like `CacheMe`, except that the code is read verbatim by LaTeX. This way, you can put non-LaTeX code inside safely, but you will not be able to use it inside a macro or some environments that read their body. Here is an example where we define an environment that automatically import matplotlib, save the figure, and insert it into a figure. Note that we define in this example new commands to type `set caption=foo` instead of `set placeholder={__FIG_CAPTION__}{foo}`.

```
%% Define the python code to use as a template
%% (impossible to define it in \robExtConfigure directly since
%% it is a verbatim environment)
\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE__}
import matplotlib.pyplot as plt
import sys
__ROBEXT_MAIN_CONTENT__
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{PlaceholderFromCode}

% Create a new preset called matplotlib
\robExtConfigure{
  new preset={matplotlib figure}{
    set template={__MY_MATPLOTLIB_TEMPLATE__},
    set compilation command={python "__ROBEXT_SOURCE_FILE__"},
    set caption/.style={
      set placeholder={__FIG_CAPTION__}{#1}
    },
    set label/.style={
      set placeholder={__FIG_LABEL__}{#1}
    },
    set includegraphics options/.style={
      set placeholder={__INCLUDEGRAPHICS_OPTIONS__}{#1}
    },
    set caption={},
    set label={},
    set includegraphics options={width=1cm},
    custom include command={%
      \evalPlaceholder{%
        \begin{figure}
          \centering
          \includegraphics[__INCLUDEGRAPHICS_OPTIONS__]{\robExtAddCachePathAndName{\robExtFinalHash.pdf}}%
          \caption{__FIG_CAPTION__a}
          \label{__FIG_LABEL__}
        \end{figure}%
      }%
    },
  },
}

%% Use it
\begin{CacheMeCode}{matplotlib figure, set includegraphics options={width=.8\linewidth}, set caption={Hello}}
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
\end{CacheMeCode}
```

Note that as we explained it before, due to LaTeX limitations, it is impossible to call `CacheMeCode` inside macros and inside some environments that evaluate their body. To avoid that issue, it is always possible to define the macro before and call it inside. We will exemplify this on the previous example, but note that **this example is only for educational purposes** since the environment `figure` does not evaluate its body, and `CacheMeCode` can therefore safely be used inside without using this trickery:
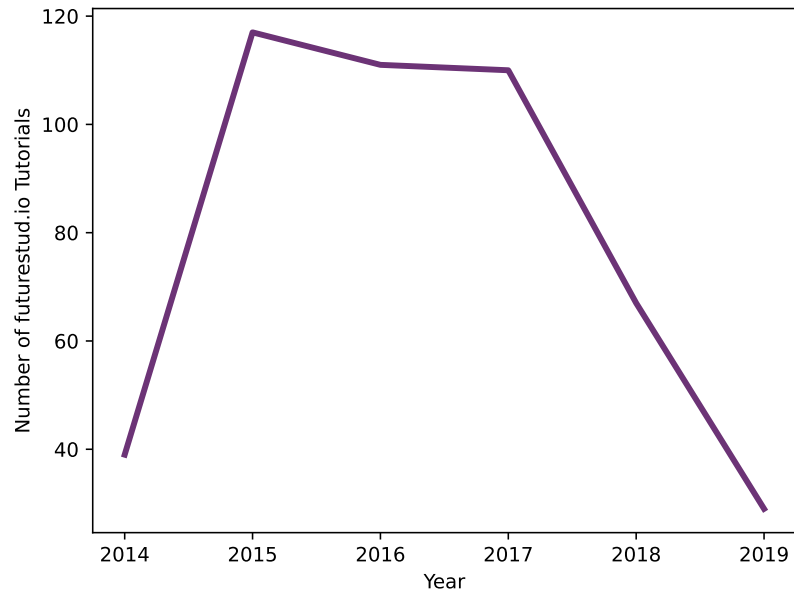
Figure 2: An example to show how matplotlib pictures can be inserted

```
%% Define the python code to use as a template
%% (impossible to define it in \robExtConfigure directly since
%% it is a verbatim environment)
\begin{PlaceholderFromCode}{__MY_MATPLOTLIB_TEMPLATE__}
import matplotlib.pyplot as plt
import sys
__ROBEXT_MAIN_CONTENT__
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{PlaceholderFromCode}

% Create a new preset called matplotlib
\robExtConfigure{
  new preset={matplotlib}{
    set template={__MY_MATPLOTLIB_TEMPLATE__},
    set compilation command={python "__ROBEXT_SOURCE_FILE__"},
    set includegraphics options/.style={
      set placeholder={__INCLUDEGRAPHICS_OPTIONS__}{#1}
    },
    set includegraphics options={width=1cm},
    custom include command={%
      \evalPlaceholder{%
        \includegraphics[__INCLUDEGRAPHICS_OPTIONS__]{\robExtAddCachePathAndName{\robExtFinalHash.pdf}}%
      }%
    },
  },
}


%% You cannot use CacheMeCode inside some macros or environments due to fundamental LaTeX limitations.
%% But you can always define them before, and call them inside:
\begin{SetPlaceholderCode}{__TMP__}
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
\end{SetPlaceholderCode}

\begin{figure}
  \centering
  \cacheMe[matplotlib, set includegraphics options={width=.8\linewidth}, set
caption={Hello}]{__TMP__}
  \caption{An example to show how code can be inserted into macros or environments that evaluate their contents (th
\end{figure}
```

4.3.2 Options to configure the template

/robExt/set template={⟨*content template*⟩}   (style, no default)

Style that alias to `set placeholder={__ROBEXT_TEMPLATE__}{#1}`, in order to define the placeholder that will hold the template of the final file.

4.3.3 Options to configure the compilation command

/robExt/set compilation command={⟨*compilation command*⟩}   (style, no default)

Style that alias to `set placeholder={__ROBEXT_COMPILATION_COMMAND__}{#1}`, in order to define the placeholder that will hold the compilation command.

/robExt/add argument to compilation command={⟨*argument*⟩}   (style, no default)
/robExt/add arguments to compilation command={⟨*argument*⟩}   (style, no default)

`add argument to compilation command` is a style that alias to:
`set placeholder={__ROBEXT_COMPILATION_COMMAND__}{__ROBEXT_COMPILATION_COMMAND__ "#1"}`
in order to add an argument to the compilation command. `add arguments to compilation command` (note the `s`) accepts multiple arguments separated by a comma.

/robExt/add key value argument to compilation command={⟨*key=value*⟩}   (style, no default)

Adds to the command line two arguments `key` and `value`. This is a way to quickly pass arguments to a script: the script just needs to loop over the arguments and consider the odd elements as keys and the next elements as the value. Another option is to insert some placeholders directly in the script.

/robExt/add key and file argument to compilation command={⟨*key=filename*⟩} (style, no default)

`filename` is the path to a file in the root folder. This adds, as:
`add key value argument to compilation command`
two arguments, where the first argument is the key, but this time the second argument is the path of `filename` relative to the cache folder (useful since scripts run from this folder). Moreover, it automatically ensures that when `filename` changes, the file gets recompiled. Note that contrary to some other commands, this does not copy the file in the cache, which is practical notably for large files like videos.

/robExt/force compilation   (style, no value)
/robExt/do not force compilation   (style, no value)

By default, we compile cached documents only if `-shell-escape` is enabled. However, if the user allowed `pdflatex` (needed to compile latex documents), `cd` (not needed when using `no cache folder`), and `mkdir` (not needed when using `no cache folder` or if the cache folder that defaults to `robustExternalize` is already created) to run in restricted mode (so without enabling `-shell-escape`), then there is no need for `-shell-escape`. In that case, set `force compilation` and this library will compile even if `-shell-escape` is disabled.

4.3.4 Options to configure the inclusion command

The inclusion command is the command that is run to include the cached file back in the pdf (e.g. based on `\includegraphics`). We describe now how to configure this command.

/robExt/custom include command advanced={⟨*include command*⟩}   (style, no default)

Sets the command to run to include the compiled file. You can use:
`\robExtAddCachePathAndName{\robExtFinalHash.pdf}`
in order to get the path of the compiled pdf file. Note that we recommend rather to use `custom include command` that automatically checks if the file compiled correctly and that load the `*-out.tex` file if it exists (useful to pass information back to the pdf).

/robExt/custom include command={⟨*include command*⟩}   (style, no default)

Sets the command to run to include the compiled file, after checking if the file has been correctly compiled and loading `*-out.tex` (useful to pass information back to the pdf).

/robExt/do not include pdf   (style, no value)

Do not include the pdf. Useful if you only want to compile the file but use it later (note that you should still generate a `.pdf` file, possibly empty, to indicate that the compilation runs smootly). Equivalent to:
`custom include command={}`

/robExt/enable manual mode   (style, no value)
/robExt/disable manual mode   (style, no value)
/robExt/enable fallback to manual mode   (style, no value)
/robExt/disable fallback to manual mode   (style, no value)

If you do or do not want to ask latex to run the compilation commands itself (for instance for security reasons), you can use these commands and run the command manually later:

The next picture must be manually compiled (see JOBNAME-robExt-compile-missing-figures.sh):

Draft mode: either compile with -shell-escape or compile:
`robustExternalize/robExt-9A58BBFD5C2B6C079891CF68E406829D.tex`
via
`cd robustExternalize/ && pdflatex -halt-on-error "robExt-9A58BBFD5C2B6C079891CF68E406829D.tex`
or call
`bash robust-externalize-robExt-compile-missing-figures.sh`
to compile all missing figures.

```
\robExtConfigure{
  enable manual mode
}

The next picture must be manually compiled %
(see JOBNAME-robExt-compile-missing-figures.sh):\\ %
\begin{tikzpictureC}[baseline=(A.base)]
  \node[fill=red, rounded corners](A){I must be manually compiled.};
  \node[fill=red, rounded corners, opacity=.3,overlay] at (A.north east){I am an overlay text};
\end{tikzpictureC}
```

The main difference between `manual mode` and `fallback to manual mode` is that `fallback to manual mode` will try to compile the file if `-shell-escape` is enabled, while `manual mode` will never run any command, even if `-shell-escape` is enabled. Note that `enable fallback to manual mode` is available starting from v1.1.

See section 4.8 for more details.

/robExt/include graphics args   (style, no value)

By default, the include commands runs `\includegraphics` on the pdf, and possibly raises it if needed. You can customize the arguments passed to `\includegraphics` here.

4.3.5   Configuration of the cache

If needed, you can configure the cache:

/robExt/set filename prefix={⟨*prefix*⟩}   (style, no default)

By default, the files in the cache starts with `robExt-`. If needed you can change this here, or by manually defining `\def\robExtPrefixFilename{yourPrefix-}`.

/robExt/set subfolder and way back={⟨*cache folder*⟩}{⟨*path to project from cache*⟩}   (style, no default)

/robExt/set cache folder and way back={⟨*cache folder*⟩}{⟨*path to project from cache*⟩}
(style, no default)

By default, the cache is located in `robustExternalize/`, using:
`set cache folder and way back={robustExternalize/}{../}`,
You can customize it the way you want, just be make sure that going to the second arguments after going to the first argument leads you back to the original position, and make sure to terminate the path with a `/` so that `__ROBEXT_WAY_BACK__common_inputs.tex` gives the path of the file `common_inputs.tex` in the root folder (do not write `__ROBEXT_WAY_BACK__/common_inputs.tex` as this would expand to the absolute path `/common_inputs.tex` if you disable the cache folder). Note that both:
`set cache folder and way back`
and
`set subfolder and way back`
are alias, but the first one was introduced in v1.1 as it is clearer, the second one being kept for backward compatibility.

/robExt/no cache folder   (style, no value)

Do not put the cache in a separate subfolder.

4.3.6 Customize or disable externalization

You might want (sometimes or always) to disable externalization, for instance to use `remember picture`, even if you used `\cacheTikz`:

`Point to me if you can`

/robExt/disable externalization   (style, no value)
/robExt/disable externalization now   (style, no value)
/robExt/enable externalization   (style, no value)

Enable or disable externalization.

This figure is not externalized. This way, it can use remember picture.

This figure is not externalized. This way, it can use remember picture.

This figure is not externalized. This way, it can use remember picture.

```
% In theory all pictures should be externalized (so remember picture should fail)
\cacheTikz
% But we can disable it temporarily
\begin{tikzpicture}<disable externalization>[remember picture]
  \node[rounded corners, fill=red](A){This figure is not externalized.
    This way, it can use remember picture.};
  \draw[->,overlay] (A) to[bend right] (pointtome);
\end{tikzpicture}\\

% You can also disable it globally/in a group:
{
  \robExtConfigure{disable externalization}

  \begin{tikzpicture}[remember picture]
    \node[rounded corners, fill=red](A){This figure is not externalized.
      This way, it can use remember picture.};
    \draw[->,overlay] (A.west) to[bend left] (pointtome);
  \end{tikzpicture}\\

  \begin{tikzpicture}[remember picture]
    \node[rounded corners, fill=red](A){This figure is not externalized.
      This way, it can use remember picture.};
    \draw[->,overlay] (A.east) to[bend right] (pointtome);
  \end{tikzpicture}
}
```

`disable externalization now` additionally redefines all automatically cached commands and environments to their default value right now (`disable externalization` would only do so when running the `command if no externalization` is run, which should be preferred if possible). This is mostly useful when an automatically cached command cannot be parsed (e.g. you specified the signature `O{}m` but in fact the command expects a more complicated parsing algorithm. For instance, in `tikz` we can omit the brackets, and it might confuse the system if we defined it as:

The short version of tikz can be confusing.

```
%% Not the recommended way to proceed, but this is for the example
\cacheCommand{tikz}[O{}m]{tikz}
{
  % Needed, or the parser would be confused by the next command
  \robExtConfigure{disable externalization now}
  \tikz\node[fill=green]{The short version of tikz can be confusing.};
}
```

/robExt/command if no externalization (style, no value)

You can easily change the command to run if externalization is disabled using by setting the **.code** of this key. By default, it is configured as:

```
command if no externalization/.code={%
  \robExtDisableTikzpictureOverwrite\evalPlaceholder{__ROBEXT_MAIN_CONTENT__}%
}
```

Unless you know what you are doing, you should include **\robExtDisableTikzpictureOverwrite** as it is often necessary to avoid infinite recursion when externalization is disabled and the original command has been replaced with a cached version (for instance done by **\cacheTikz**). Note that if you write your own style, try to modify **__ROBEXT_MAIN_CONTENT__** so that it can be included as-it in a document: this way you do not need to change this command.

/robExt/print verbatim if no externalization (style, no value)

Sets `command if no externalization` to print the verbatim content of **__ROBEXT_MAIN_CONTENT__** if externalization is disabled. Internally, it just sets it to:
`\printPlaceholder{__ROBEXT_MAIN_CONTENT__}`
This is mostly useful when typesetting **__ROBEXT_MAIN_CONTENT__** directly does not make sense (e.g. in python code). This style is used for instance in the `python` preset, allowing us to get:

```
with open("__ROBEXT_OUTPUT_PREFIX__-out.txt", "w") as f:
    for i in range(5):
        f.write(f"Hello {i}, we are on page 46\n")
```

```
\begin{CacheMeCode}{python,
    verbatim output,
    set placeholder eval={__thepage__}{\thepage},
    %% We disable externalization
    disable externalization}
with open("__ROBEXT_OUTPUT_PREFIX__-out.txt", "w") as f:
    for i in range(5):
        f.write(f"Hello {i}, we are on page __thepage__\n")
\end{CacheMeCode}
```

You can also disable the externalization on all elements that use a common preset, for instance you can disable externalization on all `bash` instances (useful if you are on Windows for instance):

```
# $outputTxt contains the path of the file that will be printed via
\verbatiminput
uname -srv > "${outputTxt}"
```

```
\robExtConfigure{
  % bash code will not be compiled (useful on windows for instance)
  add to preset={bash}{
    disable externalization
  },
}
\begin{CacheMeCode}{bash, verbatim output}
# $outputTxt contains the path of the file that will be printed via \verbatiminput
uname -srv > "${outputTxt}"
\end{CacheMeCode}
```

/robExt/execute after each externalization (style, no value)
/robExt/execute before each externalization (style, no value)

By doing `execute after each externalization={some code}`, you will run some code after the externalization. This might be practical for instance to update a counter (e.g. the number of pages. . .) based on the result of the compiled file.

4.3.7 Dependencies

In order to enforce reproducibility, you should tell what are the files that your code depends on, by adding this file as a dependency. This has the advantage that if this file is changed, your code is automatically recompiled. On the other hand, you might not want this behavior (e.g. if this file often changes in a non-important way): in that case, just don't add the file as a dependency (but keep that in mind as you might not be able to recompile your file if you clear the cache if you introduced breaking changes).

/robExt/dependencies={⟨*list,of,dependencies*⟩}   (style, no default)
/robExt/add dependencies={⟨*list,of,dependencies*⟩}   (style, no default)
/robExt/reset dependencies   (style, no value)

Set/add/reset the dependencies (you can put multiple files separated by commas). These files should be relative to the main compiled file. For instance, if you have a file `common_inputs.tex` that you want to input in both the main file and in the cached files, that contains, say:
`\def\myValueDefinedInCommonInputs{42}`
then you can add it as a dependency using:

The answer is 42.

```
\begin{CacheMe}{latex,
    add dependencies={common_inputs.tex},
    add to preamble={\input{__ROBEXT_WAY_BACK__common_inputs.tex}}}
  The answer is \myValueDefinedInCommonInputs.
\end{CacheMe}
```

Note that the placeholder `__ROBEXT_WAY_BACK__` contains the path from the cache folder (containing the `.tex` that will be cached) to the root folder.
This way, you can easily input files contained in the root folder.

4.3.8 Pass compiled file to another template

It can sometimes be handy to use the result of a previous cached file to cache another file, or to do anything else (e.g. it can also be practical to debug an issue). `name output` can be used to do that

/robExt/name output={⟨*macro name*⟩}   (style, no default)

`name output=foo` will create two global macros `\foo` and `\fooInCache`: `\foo` expands to the prefix of the files created in the template like `robExt-somehash`, and `\fooInCache` also adds the cache folder like `robustExternalize/robExt-somehash`. You can then use `set placeholder eval` to send it to another cached file. It is then your role to add the extension, usually `.tex` to get the source (even if the source is a python file), `.pdf` to get the pdf, `-out.tex` to get the file that is loaded before the import, `-out.txt` if you wanted to make it compatible with `verbatim output` (this list is not exhaustive as each script might decide to create a different output file). Here is a demo:

Hello World!

The prefix is `robExt-533423D6079B5BC1502646EBDF055F42` and with the cache folder it is in:
`robustExternalize/robExt-533423D6079B5BC1502646EBDF055F42`.
It this can be helpful for instance to debug, as you can inspect the source:

```
\documentclass[,margin=0cm]{standalone}
 \usepackage {tikz}
% most packages must be loaded before hyperref
% so we typically want to load hyperref here

% some packages must be loaded after hyperref

\begin{document}%
%% We save the height/depth of the content by using a savebox:
\newwrite\writeRobExt%
\immediate\openout\writeRobExt=\jobname-out.tex%
%
\newsavebox\boxRobExt%
\begin{lrbox}{\boxRobExt}%
  \begin {tikzpicture}[baseline=(A.base)] \node [draw,rounded corners,fill=pink!60](A){Hel
\end{lrbox}%
\usebox{\boxRobExt}%
\immediate\write\writeRobExt{%
  \string\def\string\robExtWidth{\the\wd\boxRobExt}%
  \string\def\string\robExtHeight{\the\ht\boxRobExt}%
  \string\def\string\robExtDepth{\the\dp\boxRobExt}%
}%
%

\end{document}
```

but it is also practical to define a template based on the previously cached files:

A cached file can use result from another cached file: Hello World! Hello World!

```
\begin{CacheMe}{tikzpicture, do not add margins, name output=mycode}[baseline=(A.base)]
  \node[draw,rounded corners,fill=pink!60](A){Hello World!};
\end{CacheMe}\\[3mm]

The prefix is \texttt{\mycode} and with the cache folder it is in:\\
\texttt{\mycodeInCache}.\\
It this can be helpful for instance to debug, as you can inspect the source:
\verbatiminput{\mycodeInCache.tex}
but it is also practical to define a template based on the previously cached files:\\

\begin{CacheMe}{tikzpicture, set placeholder eval={__previous__}{\mycode.pdf}}
  \node[rounded corners, fill=green!50]{A cached file can use result from another cached file:
    \includegraphics[width=2cm]{__previous__}\includegraphics[width=2cm]{__previous__}};
\end{CacheMe}
```

Note that if you do not want to display the first cached file, you can use `do not include pdf` to hide it.

4.4 Default presets

We provide by default some presets for famous languages (for now LATEX and python).

4.4.1 All languages

First, here are a few options that are available irrespective of the used language.

/robExt/set includegraphics options={⟨*options*⟩}   (style, no default)
/robExt/add to includegraphics options={⟨*options*⟩}   (style, no default)

  Set/add options to the `\includegraphics` run when inserting the pdf (by the default include
  command). By default it is empty, but the `latex` preset sets it to:
  `trim=__ROBEXT_LATEX_TRIM_LENGTH__ __ROBEXT_LATEX_TRIM_LENGTH__`
  `__ROBEXT_LATEX_TRIM_LENGTH__ __ROBEXT_LATEX_TRIM_LENGTH__`
  in order to remove the margin added in the standalone package options, which is needed to
  display overlay texts.

/robExt/verbatim output   (style, no value)

  Shortcut for:
  ```
  custom include command={%
    \evalPlaceholder{%
      __ROBEXT_VERBATIM_COMMAND__{%
        __ROBEXT_CACHE_FOLDER____ROBEXT_OUTPUT_PREFIX__-out.txt}%
    }%
  },
  ```
  i.e. instead of printing the pdf we print the content of the file `__ROBEXT_OUTPUT_PREFIX__-out.txt`
  using the command in `__ROBEXT_VERBATIM_COMMAND__`, that defaults to `\verbatiminput`:

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
```

```
\begin{CacheMeCode}{python, verbatim output}
with open("__ROBEXT_OUTPUT_PREFIX__-out.txt", "w") as f:
    for i in range(5):
        f.write(f"Hello {i}\n")
\end{CacheMeCode}
```

/robExt/debug   (style, no value)
/robExt/print command and source   (style, no value)

  (new in v1.1) These two (alias) commands allow you to print the compilation command, the
  dependency file (useful to be sure you tracked all dependencies), and the source file:

Command: (run in folder robustExternalize/)

```
pdflatex -halt-on-error "robExt-0A74AF1E310C7C9D61B0AF1E102DFE54.tex"
```

Dependencies:

```
command,pdflatex -halt-on-error "__ROBEXT_SOURCE_FILE__"
1708C4C849C592BA2C97C042FEE90430,
648E35E8624D2EFA2983FE62024FB1FD,common_inputs.tex
```

Source (in robustExternalize/robExt-0A74AF1E310C7C9D61B0AF1E102DFE54.tex):

```
\documentclass[,margin=30cm]{standalone}
 \input {../common_inputs.tex}
% most packages must be loaded before hyperref
% so we typically want to load hyperref here

% some packages must be loaded after hyperref

\begin{document}%
%% We save the height/depth of the content by using a savebox:
\newwrite\writeRobExt%
\immediate\openout\writeRobExt=\jobname-out.tex%
%
\newsavebox\boxRobExt%
\begin{lrbox}{\boxRobExt}%
  The answer is \myValueDefinedInCommonInputs .%
\end{lrbox}%
\usebox{\boxRobExt}%
\immediate\write\writeRobExt{%
  \string\def\string\robExtWidth{\the\wd\boxRobExt}%
  \string\def\string\robExtHeight{\the\ht\boxRobExt}%
  \string\def\string\robExtDepth{\the\dp\boxRobExt}%
}%
%

\end{document}
```

```
\begin{CacheMe}{latex,
    add dependencies={common_inputs.tex},
    add to preamble={\input{__ROBEXT_WAY_BACK__common_inputs.tex}},
    debug
  }
  The answer is \myValueDefinedInCommonInputs.
\end{CacheMe}
```

4.4.2 LaTeX and TikZ

The latex preset is used to cache any LaTeX content, like tikz pictures. Note that as of today, it supports overlay content out of the box (if the overlay is more than 30cm long, you might want to customize a placeholder), but not images that need to use remember picture.

/robExt/latex   (style, no value)

This style sets the template __ROBEXT_LATEX__ and the compilation command:
__ROBEXT_COMPILATION_COMMAND_LATEX__
(cf section 4.5 for details), and adds a number of styles described below, to easily configure the most common options. You can use it as follows:

The next picture is cached **My node that respects baseline.** I am an overlay text and you can see that overlay and depth works.

```
The next picture is cached %
\begin{CacheMe}{latex, add to preamble={\usepackage{tikz}}}
  \begin{tikzpicture}[baseline=(A.base)]
    \node[fill=red, rounded corners](A){My node that respects baseline.};
    \node[fill=red, rounded corners, opacity=.3,overlay] at (A.north east){I am an overlay text};
  \end{tikzpicture}
\end{CacheMe} and you can see that overlay and depth works.
```

To see how to create your own preset or automatically load a library, see section 4.6.

The next options can be used after calling the `latex` style:

/robExt/latex/use latexmk   (style, no value)
/robExt/latex/use lualatex   (style, no value)
/robExt/latex/use xelatex   (style, no value)

Use latexmk/lualatex/xelatex to compile. It is a shortcut for:
`set placeholder={__ROBEXT_LATEX_ENGINE__}{yourfavoriteengine}`

/robExt/latex/set latex options={⟨*latex options*⟩}   (style, no default)
/robExt/latex/add to latex options={⟨*latex options*⟩}   (style, no default)

Set/add elements to the set of latex options of the `\documentclass` (it will automatically add a comma before if you add an element). Internally it sets `__ROBEXT_LATEX_OPTIONS__`. By default, it sets:
`margin=__ROBEXT_LATEX_TRIM_LENGTH__` (where `__ROBEXT_LATEX_TRIM_LENGTH__` is defined as 30cm by default) in order to add a margin that will be trimmed later in the `\includegraphics`. This is useful not to cut stuff displayed outside of the bounding box (overlays).

/robExt/latex/set documentclass={⟨*documentclass*⟩}   (style, no default)

Set the documentclass of the document (defaults to `standalone`). Internally, it sets the placeholder `__ROBEXT_DOCUMENT_CLASS__`.

/robExt/latex/set preamble={⟨*code of preamble*⟩}   (style, no default)
/robExt/latex/add to preamble={⟨*code of preamble*⟩}   (style, no default)
/robExt/latex/set preamble hyperref={⟨*code of preamble*⟩}   (style, no default)
/robExt/latex/add to preamble hyperref={⟨*code of preamble*⟩}   (style, no default)
/robExt/latex/set preamble after hyperref={⟨*code of preamble*⟩} (style, no default)
/robExt/latex/add to preamble after hyperref={⟨*code of preamble*⟩} (style, no default)

Set/add element to the preamble (defaults to `standalone`). Internally, it sets the placeholder `__ROBEXT_PREAMBLE__`. The variations `hyperref` and `after hyperref` are used to put stuff after the preambule, as `hyperref` typically needs to be loaded last (or nearly): packages that must be loaded after `hyperref` like `cref` can be added via `add to preamble after hyperref`.

/robExt/latex/do not wrap code   (style, no value)

By default, the main content is wrapped into a box in order to measure its depth to properly set the baseline. If you do not want to do this wrapping, you can set this option. Internally, it is a shortcut for:
`set placeholder={__ROBEXT_MAIN_CONTENT_WRAPPED__}{__ROBEXT_MAIN_CONTENT__}` **IM-PORTANT**: note that it means that you might need to adapt your code to take into account the fact that they are inside a box (I don't know of any other solution to compute the depth, but it does not mean that there is none).

Sets `__ROBEXT_MAIN_CONTENT` to point to `__ROBEXT_MAIN_CONTENT_ORIG__` to remove any wrapping of the user input, for instance made by tikzpicture.

/robExt/tikz (style, no value)
/robExt/tikzpicture (style, no value)

tikz loads `latex` and then adds `tikz` to the preamble. `tikzpicture` first load `tikz` and wraps the main content within `\begin{tikzpicture}` and `\end{tikzpicture}` using:
`set placeholder={__ROBEXT_MAIN_CONTENT__}{\begin{tikzpicture}__ROBEXT_MAIN_CONTENT_ORIG__\end{`
so that the user does not need to type it. See for instance the introduction for examples of use.

4.4.3 Python

We provide support for python:

/robExt/python (style, no value)

Load the `python` preset (inspect `__ROBEXT_PYTHON__`) for details on the exact template, but note that this template might be subject to changes. We also provide a few helper functions:

- `write_to_out(text)` writes `text` to the `*-out.tex` file that will be loaded automatically before running the include function

- `parse_args()` is a function that returns a dictionary mapping some keys to values depending on the called arguments: for instance, if you call the python file with `python script key1 value1 key2 value2`, then the dictionary will map `key1` to `value1` and `key2` to `value2`. You might like this in conjunction with commands presented in section 4.3.3. Note that if you place placeholders in your code, you might not need this, but this is used if you plan to use your script outside of this library.

- `get_cache_folder()` outputs the cache folder.

- `get_file_base()` outputs the prefix of all files that should be created by this script, that looks like `robExt-somehash`.

- `get_current_script()` returns the current script.

- `get_filename_from_extension(extension)` outputs the prefix `robExt-somehash` concatenated with the extension. You often need this function to get the path of a file that your script is creating, for instance, `get_filename_from_extension("-out.txt")` is the path `*-out.txt` of the file that is read by `verbatim output`.

- `get_verbatim_output()` returns `get_filename_from_extension("-out.txt")`

- `finished_with_no_error()` creates the pdf file if it does not exists (to certify that the compilation ran without issues). The template automatically runs this function at the end.

We demonstrate its usage on a few examples:

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
```

```
\begin{CacheMeCode}{python, verbatim output}
with open(get_verbatim_output(), "w") as f:
    for i in range(5):
        f.write(f"Hello {i}\n")
\end{CacheMeCode}
```
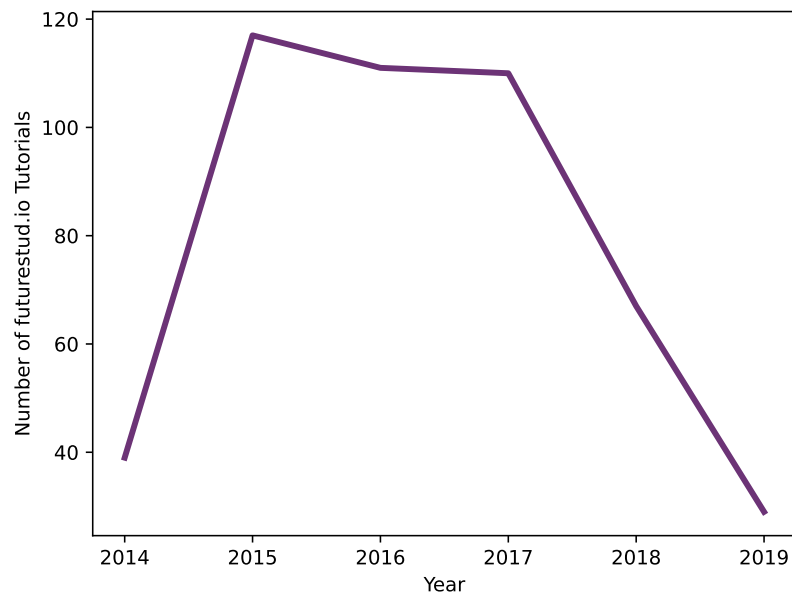
Figure 3: Image generated with python.

Importantly: you do not want to indent the whole content of CacheMeCode, or the spaces will also appear in the final code.

You can also generate some images. This code will produce the image in fig. 3:

```
\begin{CacheMeCode}{python, set includegraphics options={width=.8\linewidth}}
import matplotlib.pyplot as plt
year = [2014, 2015, 2016, 2017, 2018, 2019]
tutorial_count = [39, 117, 111, 110, 67, 29]
plt.plot(year, tutorial_count, color="#6c3376", linewidth=3)
plt.xlabel('Year')
plt.ylabel('Number of futurestud.io Tutorials')
plt.savefig("__ROBEXT_OUTPUT_PDF__")
\end{CacheMeCode}
```

Note that by default, the executable called `python` is run. It seems like on windows `python3` is not created and only `python` exists, while on linux the user can choose whether `python` should point to `python3` or `python2` (on NixOs, I directly have `python` pointing to `python3`, and in ubuntu, you might need to install `python-is-python3` or create a symlink, as explained here). In any case, you can customize the name of the executable by setting something like:

`\setPlaceholder{__ROBEXT_PYTHON_EXEC__}{python3}`

or using the style `force python3` that forces `python3`.

/robExt/python print code and result   (style, no value)

This is a demo style that can print a python code and its result.

> ### The for loop
>
> ```
> 1  for name in ["Alice", "Bob"]:
> 2      print(f"Hello {name}")
> ```
>
> Output:
> Hello Alice
> Hello Bob

53

```
\begin{CacheMeCode}{python print code and result, set title={The for loop}}
for name in ["Alice", "Bob"]:
    print(f"Hello {name}")
\end{CacheMeCode}
```

You can set `__ROBEXT_PYTHON_TCOLORBOX_PROPS__` the options of the tcolorbox, `__ROBEXT_PYTHON_CODE_MESSAGE__` and `__ROBEXT_PYTHON_RESULT_MESSAGE__` which are displayed before the corresponding block, `__ROBEXT_PYTHON_LSTINPUT_STYLE__` which contains the default lstinput style and `__MY_TITLE__` (cf `set title`) that contains the title of the box. Make sure to have the following packages to use the default styling:

```
\usepackage{pythonhighlight}
\usepackage{tcolorbox}
```

4.4.4 Bash

We provide a basic bash template, that sets:

```
set -e
outputTxt="__ROBEXT_OUTPUT_PREFIX__-out.txt"
outputTex="__ROBEXT_OUTPUT_PREFIX__-out.tex"
outputPdf="__ROBEXT_OUTPUT_PDF__"
```

in order to quit when an error occurs, and to define two variables containing the path to the pdf file and to the file that is read by the `verbatim output` setting (that just apply a `\verbatiminput` on that file). Finally, it also creates the file `outputPdf` with `touch` in order to notify that the compilation succeeded.

In practice:

```
Linux 5.15.90 #1-NixOS SMP Tue Jan 24 06:22:49 UTC 2023
```

```
\begin{CacheMeCode}{bash, verbatim output}
# $outputTxt contains the path of the file that will be printed via \verbatiminput
uname -srv > "${outputTxt}"
\end{CacheMeCode}
```

4.4.5 Verbatim text

Sometimes, it might be handy to write the text to a file and use it somehow. This is possible using `verbatim text`, that defaults to calling `\verbatiminput` on that file:

```
def some_verbatim_fct(a):
    # See this is a verbatim code where I can use the % symbol
    return a % b
```

```
\begin{CacheMeCode}{verbatim text}
def some_verbatim_fct(a):
    # See this is a verbatim code where I can use the % symbol
    return a % b
\end{CacheMeCode}
```

You can also call `verbatim text no include`: it will not include the text, but it sets a macro `\robExtPathToInput` containing the path to the input file. Use it the way you like! For instance, we define here a macro `codeAndResult` that prints the code and runs it (we use a pretty printer from pgf, so you need to load `\usepackage{tikz}\input{pgfmanual-en-macros.tex}` to use it). It is what we use right now in this documentation for verbatim blocks like here. You can obtain a simpler version using:

We will input the file robustExternalize/robExt-DDA097E3F2A45DB958F5A00BFAFF9B93.tex:

Demo % with percent

This file contains:

```
\NewDocumentCommand{\testVerbatim}{+v}{
\begin{flushleft}\ttfamily%
#1
\end{flushleft}}
\testVerbatim{Demo % with percent}
```

```
\begin{CacheMeCode}{verbatim text no include}
\NewDocumentCommand{\testVerbatim}{+v}{
\begin{flushleft}\ttfamily%
#1
\end{flushleft}}
\testVerbatim{Demo % with percent}
\end{CacheMeCode}
We will input the file \robExtPathToInput{}:
\input{\robExtPathToInput}
This file contains:
\verbatiminput{\robExtPathToInput}
```

4.5 List of special placeholders and presets

This library defines a number of pre-existing placeholders, or placeholders playing a special role. We list some of them in this section. All placeholders created by this library start with `__ROBEXT_`. Note that you can list all predefined placeholders (at least those globally defined) using `\printAllPlaceholdersExceptDefaults` (note that some other placeholders might be created directly in the style set right before the command, and may not appear in this list if you call it before setting the style).

4.5.1 Generic placeholders

We define two special placeholders that should be defined by the user (possibly indirectly, using presets offered by this library):

- `__ROBEXT_TEMPLATE__` is a placeholder that should contain the code of the file to compile.

- `__ROBEXT_MAIN_CONTENT_ORIG__` is a placeholder containing the text typed by the user, automatically set by `CacheMe`, `CacheMeCode` etc. You rarely need to deal with this placeholder directly since `__ROBEXT_MAIN_CONTENT__` will typically point to it and add some necessary wrapping.

- `__ROBEXT_MAIN_CONTENT__` is a placeholder that might be used inside `__ROBEXT_TEMPLATE__`, that points by default to `__ROBEXT_MAIN_CONTENT_ORIG__` and that contains the content that should be typed inside the document. For instance, this might be a tikz picture, a python function without the import etc. Note that it is often used to wrap the text of the user `__ROBEXT_MAIN_CONTENT_ORIG__`: for instance, the `tikzpicture` preset adds the `\begin{tikzpicture}` around the user code automatically: this way we do not need to edit the command to disable externalization.

- `__ROBEXT_COMPILATION_COMMAND__` contains the compilation command to run to compile the file (assuming we are in the cache folder).

We also provide a number of predefined placeholders in order to get the name of the source file etc... Note that most of these placeholders are defined (and/or expanded inplace) late during the compilation stage as one needs first to obtain the hash of the file, and therefore all dependencies, the content of the template etc.

- `__ROBEXT_SOURCE_FILE__` contains the path of the file to compile (containing the content of `__ROBEXT_TEMPLATE__`) like `robExt-somehash.tex`, relative to the cache folder (since we always go to this folder before doing any action, you most likely want to use this directly in the compilation command).

- `__ROBEXT_OUTPUT_PDF__` contains the path of the pdf file produced after the compilation command relative to the cache folder (like `robExt-somehash.pdf`). Even if you do not plan to output a pdf file, you should still create that file at the end of the compilation so that this library can know whether the compilation succeeded.

- `__ROBEXT_OUTPUT_PREFIX__` contains the prefix that all newly created file should follow, like `robExt-somehash`. If you want to create additional files (e.g. a picture, a video, a console output etc...) make sure to make it start with this string. It will not only help to ensure purity, but it also allows us to garbage collect useless files easily.

- `__ROBEXT_WAY_BACK__` contains the path to go back to the main project from the cache folder, like `../` (internally it is equals to the expanded value of `\robExtPrefixPathWayBack`).

- `__ROBEXT_CACHE_FOLDER__` contains the path to the cache folder. Since most commands are run from the cache folder, this should not be really useful to the user.

You can also use these placeholders to customize the default include function:

- `__ROBEXT_INCLUDEGRAPHICS_OPTIONS__` contains the options given to `\includegraphics` when loading the pdf

- `__ROBEXT_INCLUDEGRAPHICS_FILE__` contains the file loaded by `\includegraphics`, defaults to `\robExtAddCachePathAndName{\robExtFinalHash.pdf}`, that is itself equivalent to `__ROBEXT_CACHE_FOLDER____ROBEXT_OUTPUT_PDF__` or
`__ROBEXT_CACHE_FOLDER____ROBEXT_OUTPUT_PREFIX__.pdf`.

4.5.2   Placeholders related to LaTeX

Some placeholders are reserved only when dealing with LaTeX code:

- `__ROBEXT_LATEX__` is the main entrypoint, containing all the latex template. It internally calls other placeholders listed below.

- `__ROBEXT_LATEX_OPTIONS__`: contains the options to compile the document, like `a4paper`. Empty by default.

- `__ROBEXT_DOCUMENT_CLASS__`: contains the class of the document. Defaults to `standalone`.

- `__ROBEXT_PREAMBLE__`: contains the preamble. Is empty by default.

- `__ROBEXT_MAIN_CONTENT_WRAPPED__`: content inside the `document` environment. It will wrap the actual content typed by the user `__ROBEXT_MAIN_CONTENT__` around a box to compute its depth. If you do not want this behavior, you can set `__ROBEXT_MAIN_CONTENT_WRAPPED__` to be equal to `__ROBEXT_MAIN_CONTENT__`. It calls internally `__ROBEXT_CREATE_OUT_FILE__` and `__ROBEXT_WRITE_DEPTH_TO_OUT_FILE__` to do this computation.

- `__ROBEXT_CREATE_OUT_FILE__` creates a new file called `\jobname-out.tex` and open it in the handle called `\writeRobExt`

- `__ROBEXT_WRITE_DEPTH_TO_OUT_FILE__` writes the height, depth and width of the box `\boxRobExt` into the filed opened in `\writeRobExt`.

- `__ROBEXT_COMPILATION_COMMAND_LATEX__` is the command used to compile a LaTeX document. It uses internally other placeholders:

- `__ROBEXT_LATEX_ENGINE__` is the engine used to compile the document (defaults to `pdflatex`)

- `__ROBEXT_COMPILATION_COMMAND_OPTIONS__` contains the options used to compile the document (defaults to `-shell-escape -halt-on-error`)

4.5.3   Placeholders related to python

- `__ROBEXT_PYTHON_EXEC__` contains the python executable (defaults to `python`) used to compile

- `__ROBEXT_PYTHON__` contains the python template

- `__ROBEXT_PYTHON_IMPORT__` can contain import statements

- `__ROBEXT_PYTHON_MAIN_CONTENT_WRAPPED__` is used to add all the above functions. You can set it to `__ROBEXT_MAIN_CONTENT__` if you do not want them

- `__ROBEXT_PYTHON_FINISHED_WITH_NO_ERROR__` is called at the end to create the pdf file even if it is not created, you can set it to the empty string if you do not want to do that.

4.5.4   Placeholders related to bash

- `__ROBEXT_BASH_TEMPLATE__` contains the bash template. By default, it sets `set -e`, creates `outputTxt`, `outputTex` and `outputPdf` pointing to the corresponding files, and it created the pdf file at the end.

- `__ROBEXT_SHELL__` contains the shell (defaults to `bash`).

4.6   Customize presets and create your own style

Note that you can define your own presets simply by creating a new pgf style (please refer to tikz-pgf's documentation for more details). For instance, we defined the `tikz` and `tikzpicture` presets using:

```
\robExtConfigure{
  new preset={tikz}{
    latex,
    add to preamble={\usepackage{tikz}},
  },
  new preset={tikzpicture}{
    tikz,
    set placeholder={__ROBEXT_MAIN_CONTENT__}{\begin{tikzpicture}__ROBEXT_MAIN_CONTENT_ORIG__\end{tikzpicture}},
  },
}
```

in order to automatically load `tikz` and add the surrounding `tikzpicture` when needed (note that the style is always loaded **after** the definition of `__ROBEXT_MAIN_CONTENT_ORIG__`, so in theory you could also modify it directly even if it is not recommended). You can also customize an existing style by adding stuff to it using `add to preset` (or `.append style` but make sure to double the hashes). For instance, here, we add the `shadows` library to the `tikz` preset by default:

See, tikz's style now packs the `shadows` library by default:
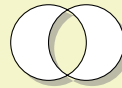
```
\robExtConfigure{
  add to preset={tikz}{
    add to preamble={\usetikzlibrary{shadows}},
  },
}
See, tikz's style now packs the |shadows| library by default: %
\begin{CacheMe}{tikzpicture}[even odd rule]
  \filldraw [drop shadow,fill=white] (0,0) circle (.5) (0.5,0) circle (.5);
\end{CacheMe}
```

4.7 Cache automatically a given environment

It might be handy to cache automatically a given environment: we already provide:
`\cacheTikz`
to cache all tikz pictures (unless externalization is disabled), but we also provide tools to handle arbitrary environments.

`\robExtExternalizeAllTikzpictures[⟨delimiters⟩]`
`\cacheTikz[⟨delimiters⟩]`

This will automatically cache all tikz pictures (both are alias, `\cacheTizk` is available from v1.1). Since v1.1, we added the delimiters options, and we parse `\tikz` as well. Note that we add an additional optional argument to the tikz picture via its first argument delimited by `delimiters` (defaults to `<>`) to specify preset options, which can for instance be practical to disable externalization on individual pictures. Cf. section 4.3.6 to see an example of use.

> See that this syntax can be safely used. (cached) Hello Alice! (cached) Foo (not cached)

```
\cacheTikz

\def\whereIAm{(not cached)}
\robExtConfigure{
  add to preset={tikz}{
    add to preamble={
      \def\whereIAm{(cached)}
    },
  },
}

\tikz \node[fill=pink, rounded corners]{See that this syntax can be safely used. \whereIAm};

\tikz<add to preamble={\def\sayHello#1{Hello #1!}}> \node[fill=green, rounded
corners]{\sayHello{Alice} \whereIAm};

\tikz<disable externalization> \node[fill=green, rounded corners]{Foo \whereIAm};
```

Here is an example to specify arguments via parenthesis:

> See that this syntax can be safely used. (cached) Hello Alice! (cached) Foo (not cached)

```
\cacheTikz[()]

\def\whereIAm{(not cached)}
\robExtConfigure{
  add to preset={tikz}{
    add to preamble={
      \def\whereIAm{(cached)}
    },
  },
}

\tikz \node[fill=pink, rounded corners]{See that this syntax can be safely used. \whereIAm};

\tikz(add to preamble={\def\sayHello#1{Hello #1!}}) \node[fill=green, rounded
corners]{\sayHello{Alice} \whereIAm};

\tikz(disable externalization) \node[fill=green, rounded corners]{Foo \whereIAm};
```

`\robExtDisableTikzpictureOverwrite`

This is useful to temporarily reset the current environment to their original value (since v1.1 it also resets other environments, not just `tikzpicture`). This must typically be called at the

beginning of `command if no externalization` to avoid infinite recursion if you redefine it, but expect for this case the user is not expected to use this option.

Let us say that you want to cache all elements of a given environment, like `minipage` or `zx-calculus` pictures (another package of mine):

\cacheEnvironment{⟨*delimiters*⟩}{⟨*name environment*⟩}{⟨*default preset options*⟩}

This will automatically cache the corresponding environment (but note that you still need to define it in the preamble of the cached files, for instance by loading the appropriate package):

A ZX figure and the same but aligned on the second line .
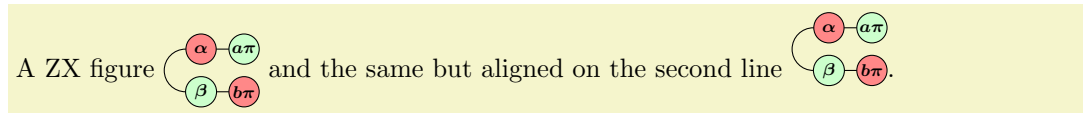
```
\cacheEnvironment{ZX}{latex, add to preamble={\usepackage{zx-calculus}}}
A ZX figure %
\begin{ZX}
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar          & \zxX*{b\pi}
\end{ZX} and the same but aligned on the second line %
\begin{ZX}[mbr=2]
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar          & \zxX*{b\pi}
\end{ZX}.
```

You can of course configure them using globally defined configuration, but you can also provide arguments to a single picture using the delimiters `delimiters` that default to `<...>` as the optional argument, like:
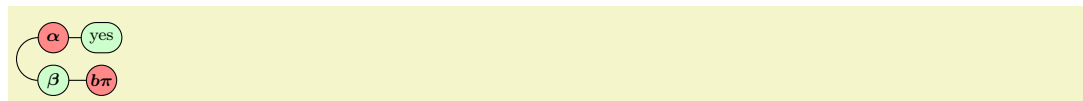

```
\cacheEnvironment{ZX}{latex, add to preamble={\usepackage{zx-calculus}}}
\begin{ZX}<add to preamble={\usepackage{amsmath}}> % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{yes}} \\
  \zxZ{\beta} \rar          & \zxX*{b\pi}
\end{ZX}
```

If you do not like `<>` or if your command already have this parameter, you can change it, for instance to get parens as delimiters, use:


```
\cacheEnvironment[()]{ZX}{latex, add to preamble={\usepackage{zx-calculus}}}
\begin{ZX}(add to preamble={\usepackage{amsmath}}) % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{yes}} \\
  \zxZ{\beta} \rar          & \zxX*{b\pi}
\end{ZX}
```

otherwise you must provide some presets arguments, possibly empty:
\begin{someweirdenv}<><args to someweirdenv>

You can also use this argument to disable externalisation on a per-picture basis:
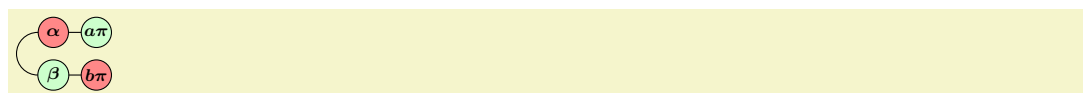
```
\cacheEnvironment{ZX}{latex, add to preamble={\usepackage{zx-calculus}}}
\begin{ZX}<disable externalization> % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar            & \zxX*{b\pi}
\end{ZX}
```

`\cacheCommand[⟨delimiter⟩]{⟨name command⟩}[⟨xparse signature⟩]{⟨default preset options⟩}`

This will automatically cache the corresponding command, in a similar way to environment. It work exactly the same, except that we (usually) need to specify the signature of the command or it is impossible to know where the command stops. If the command is defined using xparse-compatible commands like `\NewDocumentCommand`, this detection is automatically done. Otherwise, you need to specify it in the xparse format: long story short, `O{foo}` is an optional argument with default value foo, `m` is a mandatory argument.

First, let us see an example with a command defined using xparse:

A ZX figure and the same but aligned on the second line .
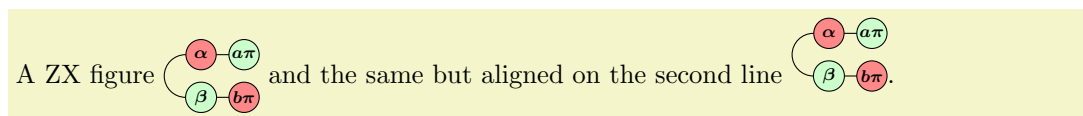
```
% We cache the command \zx that is defined with NewDocumentCommand: no need to specify the signature
\cacheCommand{zx}{latex, add to preamble={\usepackage{zx-calculus}}}
A ZX figure %
\zx{
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar            & \zxX*{b\pi}
} and the same but aligned on the second line %
\zx[mbr=2]{
  \zxX{\alpha} \rar \ar[d,C] & \zxZ*{a\pi} \\
  \zxZ{\beta} \rar            & \zxX*{b\pi}
}.
```

You can of course configure them using globally defined configuration, but you can also provide arguments to a single picture using the delimiters that default to `<>` as the optional argument, like:


```
\cacheCommand{zx}{latex, add to preamble={\usepackage{zx-calculus}\def\hello#1{Hello #1.}}}
\zx<add to preamble={\usepackage{amsmath}\def\bye#1{Bye #1.}}>[mbr=1]{ % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{\hello{Alice}}} \\
  \zxZ{\beta} \rar            & \zxZ{\text{\bye{Bob}}}
}
```

or with parens as delimiters:


```
\cacheCommand[()]{zx}{latex, add to preamble={\usepackage{zx-calculus}}}
\zx(add to preamble={\usepackage{amsmath}})[mbr=1]{ % amsmath provides \text
  \zxX{\alpha} \rar \ar[d,C] & \zxZ{\text{yes}} \\
  \zxZ{\beta} \rar            & \zxX*{b\pi}
}
```

Here is an example with a command not defined with `\NewDocumentCommand` (we also show how you can use the optional argument to disable externalisation on a per-picture basis):

```
\def\fromWhere{I am not cached}
\newcommand{\myMacroNotDefinedWithNewDocumentCommand}[3][default value]{
  Optional argument 1: #1,
  Argument 2: #2,
  Argument 3: #3 (\fromWhere).
}
\cacheCommand{myMacroNotDefinedWithNewDocumentCommand}[O{default value}mm]{
  latex,
  add to preamble={%
    \def\fromWhere{I am cached}
    \newcommand{\myMacroNotDefinedWithNewDocumentCommand}[3][default value]{
      Optional argument 1: #1,
      Argument 2: #2,
      Argument 3: #3 (\fromWhere).
    }
  }
}

\myMacroNotDefinedWithNewDocumentCommand{second arg}{last arg}\\
\myMacroNotDefinedWithNewDocumentCommand<disable externalization>{second arg}{last arg}
```

Defining its own macro without repeating its definition. Note that when you define yourself a macro function, the above structure might not be optimal as you need to define the macro in the main document (in case you disable the externalization) and in the template. One option to avoid repeated code is to write the definition in a common input file (see section 3.4), but you might prefer to keep everything in a single file. In that case, you can simply wrap your function into `cacheMe` yourself. Say you want your function to draw a circle like, `\tikz \draw[fill=#2] (0,0) circle [radius=#1];`, then wrap in inside `CacheMe`:

```
% D<>{} is an optional argument enclosed in <>, we use this to pass arguments to CacheMe
\NewDocumentCommand{\drawMyCircle}{D<>{}O{2mm}m}{
  \begin{CacheMe}{tikz, #1}
    \tikz \draw[fill=#3] (0,0) circle [radius=#2];
  \end{CacheMe}
}

You can externalize it \drawMyCircle{pink} %
or disable externalization \drawMyCircle<disable externalization>[4mm]{red}.
```

Manually wrap a command. In some more advanced cases, if you do not know the definition of the command and if `\cacheCommand` does not work (e.g. the number of mandatory argument depends on the first argument), you can wrap it manually. To do so, first copy first your command using `\DeclareCommandCopy{\robExtCommandOrigNAMEOFCOMMAND}{\NAMEOFCOMMAND}` and run `\robExtAddToCommandResetList{NAMEOFCOMMAND}` (this is needed to disable externalization), and override the command by wrapping it into a cache. Make sure to add `\def\robExtCommandOrigName{NAMEOFCOMMAND}` at the beginning of the function or `disable externalization` will not work:

```
Original command: \zx{\zxZ{\alpha} \rar & \zxX{\beta}}. %
\DeclareCommandCopy{\robExtCommandOrigzx}{\zx}% used to recover the original function
\robExtAddToCommandResetList{zx}% let us know that this function should be reset when externalization is disabled
\DeclareDocumentCommand{\zx}{D<>{}O{}m}{%
  \def\robExtCommandOrigName{zx}%
  \begin{CacheMe}{latex, add to preamble={\usepackage{zx-calculus}}, #1}%
    \zx[#2]{#3}%
  \end{CacheMe}%
}
This is cached \zx<>{\zxZ{\alpha} \rar & \zxX{\beta}}, you can add options like %
\zx<add to preamble={\usepackage{amsmath}}>{\zxZ{\alpha} \rar & \zxX{\text{Hey}}} %
and we can disable the externalization: %
\zx<disable externalization>{\zxZ{\alpha} \rar & \zxX{\text{Hey}}}.
```

You can do a similar trick for environment, by using instead \NewEnvironmentCopy{robExtEnvironmentOrigNAMEE
\robExtAddToEnvironmentResetList{NAMEENV} and \def\robExtEnvironmentOrigName{NAMEENV}.

4.8 Operations on the cache

Every time we compile a document, we create automatically a bunch of files:

- the cache is located by default in the robustExternalize folder. Feel free to remove this folder if you want to completely clear the cache (but then you need to recompile everything). See below if you want to clean it in a better way.

- \jobname-robExt-all-figures.txt contains the list of all figures contained in the document. Mostly useful to help the script that remove other figures.

- robExt-remove-old-figures.py is a python script that will remove all cached files that are not used anymore. Just run python robExt-remove-old-figures.py to clean it. You will then see the list of files that the script wants to remove: make sure it does not remove any important data, and press "y". Note that it will search for all files that look like *robExt-all-figures.txt to see the list of pictures that are still in use, and by default it will only remove the images in the robustExternalize folder that start with robExt-. If you change the path of the cache or the prefix, edit the script (should not be hard to do).

- \jobname-robExt-compile-missing-figures.sh contains a list of commands that you need to run to compile the images not yet compiled in the cache (this list will only be created if you enable the manual compilation mode).

- \jobname-robExt-tmp-file-you-can-remove.tmp is a temporary file. Feel free to remove it.

We go over some of these scripts.

4.8.1 Cleaning the cache

You might want to clean the cache. Of course you can remove all generated files, but if you want to keep the picture in use in the latest version of the document, we provide a python script (automatically generated in the root folder) to do this. Just install python3 and run:

```
python3 robExt-remove-old-figures.py
```

(on windows, the executable might be called python) You will then be prompted for a confirmation after providing the list of files that will be removed.

4.8.2 Listing all figures in use

After the compilation of the document, a file robExt-all-figures.txt is created with the list of the .tex file of all figures used in the current document.

4.8.3 Manually compiling the figures

When enabling the manual mode (useful if we don't want to enable `-shell-escape`):

```
\robExtConfigure{
  enable manual mode
}
```

or the fallback to manual mode (it will only enable the manual mode if `-shell-escape` is disabled):

```
\robExtConfigure{
  enable fallback to manual mode,
}
```

the library creates a file `JOBNAME-robExt-compile-missing-figures.sh` that contains the instructions to build the figures that are not yet in the cache (each line contains the compilation command to run). On Linux (or on Windows with bash/cygwin/... installed, it possibly even work out of the box without) you can easily execute them using:

```
bash JOBNAME-robExt-compile-missing-figures.sh
```

4.9   How to debug

If for some reasons you are unable to understand why a build fails, first check if you compiled your document with `-shell-escape` (not that this must appear **before** the filename). Then, you can look at the log file to get more advices: when a cached document is compiled, we always write the full compilation command before compiling the file in the log file. This way, you can easily check the content of the file and see why it fails to compile. The compilation errors are also displayed directly in the output.

You might often get an error `! Missing $ inserted.`: this is typically when a placeholder was left unreplaced (e.g. you forgot to define it, or you forgot to wrap a command in `\evalPlaceholder{}`): since LaTeX is asked to typeset `__something__`, it thinks that you are trying to write a subscript, and asks you to start first the math mode.

You can also use `debug` (or its alias `print command and source`) in order to disable the compilation and print instead the compilation command and the source file:

Command: (run in folder robustExternalize/)

```
pdflatex -halt-on-error "robExt-C258047503967D680D684293CEAD8CE4.tex"
```

Dependencies:

```
command,pdflatex -halt-on-error "__ROBEXT_SOURCE_FILE__"
50E2E35E8F3DC9D3A97DC41F843CF721,
```

Source (in robustExternalize/robExt-C258047503967D680D684293CEAD8CE4.tex):

```
\documentclass[,margin=30cm]{standalone}
 \usepackage {tikz}
% most packages must be loaded before hyperref
% so we typically want to load hyperref here

% some packages must be loaded after hyperref

\begin{document}%
%% We save the height/depth of the content by using a savebox:
\newwrite\writeRobExt%
\immediate\openout\writeRobExt=\jobname-out.tex%
%
\newsavebox\boxRobExt%
\begin{lrbox}{\boxRobExt}%
  \begin {tikzpicture}[baseline=(A.base)] \node [fill=red, rounded corners](A){My node that respe
\end{lrbox}%
\usebox{\boxRobExt}%
\immediate\write\writeRobExt{%
  \string\def\string\robExtWidth{\the\wd\boxRobExt}%
  \string\def\string\robExtHeight{\the\ht\boxRobExt}%
  \string\def\string\robExtDepth{\the\dp\boxRobExt}%
}%
%

\end{document}
```

```
\begin{tikzpictureC}<debug>[baseline=(A.base)]
  \node[fill=red, rounded corners](A){My node that respects baseline \ding{164}.};
  \node[fill=red, rounded corners, opacity=.3,overlay] at (A.north east){I am an overlay text};
\end{tikzpictureC}
```

5 TODO and known bugs:

- See how to deal with label, links and cite inside pictures (without disabling externalization). For label and links, I have a proof of concept, I "just" need to write it down. https://tex.stackexchange.com/questions/695277/clickable-includegraphics-for-cross-reference-data

- Deal with remember picture

6 Acknowledgments

I am deeply indebted to many users on tex.stackexchange.com that made the writing of this library possible. I can't list you all, but thank you so much! A big thanks also to the project

`https://github.com/sasozivanovic/memoize/` from which I borrowed most of the code to automatically wrap commands.

7 Changelog

- v1.1:

  - **not backward compatible**: the preset `tikzpicture` should be used instead of `tikz`: `tikz` does not wrap anymore its content into `\begin{tikzpicture}` in order to be usable inside the command version `\tikz` (but `tikzpicture` does load `tikz` first). Moreover, all arguments to `tikzpictureC` or `tikzpicture` (if you loaded `cacheTikz`) should be specified in the first argument, using `<your options>`. This is used to provide a more uniform interface with the new `\cacheEnvironment` command. Hopefully this should **not cause too much trouble** since the first version of the library has been published on CTAN only a few days ago.

  - You can now use ampersands etc freely thanks to lrbox.

  - You can now automatically cache environments using `\cacheEnvironment` and commands using `\cacheCommand`.

  - We provide `new preset` and `add to preset` to avoid doubling the number of hashes.

  - We provide new functions like `\setPlaceholderRecReplaceFromList` to only expand a subset of placeholders.

  - An error message is given if you forget to set a template.

  - Add `enable fallback to manual mode` to fallback to manual mode only if shell escape is missing.

  - Allow restricted mode

  - `debug` mode available

  - Rename `set subfolder and way back` to `set cache folder and way back` (in a backward compatible way)

Index

This index only contains automatically generated entries. A good index should also contain carefully selected keywords. This index is not a good index.