

The runcode package

Haim Bar and HaiYing Wang
haim.bar@uconn.edu, haiying.wang@uconn.edu

v2.6.2, 2026

Abstract

`runcode` is a $\text{\LaTeX}$ package that executes programming source codes (including all command-line tools) from $\text{\LaTeX}$ and embeds the results in the resulting PDF file. Many programming languages can be used and any command-line executable can be invoked when preparing the PDF from a `.tex` file. `runcode` is also available on [CTAN](#).

It is recommended to use this package in server mode together with the Python `talk2stat` package. Currently, the server mode supports [Julia](#), [MatLab](#), [Python](#), and [R](#). More languages will be added.

For more details, usage examples, and troubleshooting, refer to the package's GitHub repository at <https://github.com/Ossifragus/runcode>.

Contents

1	Installation	2
2	Usage	2
2.1	Loading the package	2
2.2	Output box style	3
2.3	Code display style	3
2.4	Basic commands	3
2.5	Extended commands	4
2.6	Language-specific shortcuts	4
2.6.1	Python batch mode	5
3	Auxiliary tools	5
3.1	<code>wait_for_server.py</code>	5
3.2	<code>runcode-Makefile.sample</code>	5
3.3	<code>consolidate.py</code>	6
4	Revisions	6
5	Contributing	8

1 Installation

You can simply put the `runcode.sty` file in your L^AT_EX project folder.

The server mode requires the [talk2stat](#) package. Some chunk commands (e.g., `\showChunk`) also require the [advance-touch](#) package. Install them from the command line with:

```
pip3 install talk2stat advance-touch
```

Note: `runcode` requires the `shell-escape` option when compiling a L^AT_EX document:

```
pdflatex --shell-escape yourfile.tex
```

When using AUCTeX or a similar IDE, you can avoid setting `shell-escape` manually by adding the following magic comment at the top of your `.tex` file:

```
% !TEX program = pdflatex --shell-escape
```

Replace `pdflatex` with `xelatex` or `lualatex` as appropriate.

2 Usage

2.1 Loading the package

```
\usepackage[options]{runcode}
```

Available options:

`cache` Use cached results (do not re-run code).

`fvextra` Use the [fvextra](#) package to display code.

`julia` Start a server for [Julia](#) (requires `talk2stat`).

`listings` Use the [listings](#) package to display code.

`matlab` Start a server for [MatLab](#) (requires `talk2stat`).

`minted` Use the [minted](#) package to display code (requires [Pygments](#)). This is the default.

`nominted` Use [fvextra](#) instead of `minted` (no syntax highlighting; no `Pygments` required).

`nohup` Use `nohup` when starting a server. Some editors (e.g. Emacs with AUCTeX) terminate child processes after compilation; this option keeps the server alive. **Must be declared before any language option**, e.g. `[nohup,R]` works but `[R,nohup]` does not. See Section [3](#) for tools that smooth the `nohup` workflow.

`python` Start a server for [Python](#) (requires `talk2stat`).

`R` Start a server for [R](#) (requires `talk2stat`).

`reducedspace` Reduce the vertical space around output boxes.

`run` Force source code to run (override cache).

`stopserver` Stop the server(s) when PDF compilation finishes.

2.2 Output box style

Output boxes are displayed using `colorbox` and can be customised with `\tcbset`, e.g.:

```
\tcbset{breakable, colback=red!5!white, colframe=red!75!black}
```

2.3 Code display style

The style of *source code* blocks (shown by `\showCode`, `\showChunk`, and the `\runLANGChunk` commands) depends on the package option used to display code:

- **minted** (default): use `\setminted` to control language-specific highlighting, e.g.:

```
\setminted[r]{linenos, frame=single, bgcolor=bg, breaklines=true}
```

- **fvextra** or **nominted**: use `\fvset` to set options globally, e.g.:

```
\fvset{frame=single, numbers=left, breaklines=true}
```

- **listings**: use `\lstset` to configure the style, e.g.:

```
\lstset{basicstyle=\ttfamily\small, frame=single, numbers=left}
```

2.4 Basic commands

`\runExtCode` `\runExtCode{<prog>}{<src>}{<label>}[<flag>]` runs an external code file.

- `<prog>`: the executable program.
- `<src>`: the source file name.
- `<label>`: the output file name (empty $\Rightarrow$ uses counter `codeOutput`).
- `[<flag>]`: optional run control — omit or empty to use the global `runcode` Boolean; `run` to force execution; `cache` (or any other value) to use cached results.

Checksum-based cache invalidation (v2.5): when neither `cache` nor `run` is active, `runcode` computes an MD5 checksum of `<src>` and compares it with the checksum stored from the previous run. The script is re-executed only if the source has changed or if the output file is missing; otherwise the cached output is reused. This avoids unnecessary reruns without requiring the user to manage cache flags manually.

`\showCode` `\showCode{<lang>}{<src>}[<first>][<last>]` displays source code using `minted`, `fvextra`, or `listings`.

- $\langle lang \rangle$: programming language.
 - $\langle src \rangle$: source file name.
 - $[\langle first \rangle]/[\langle last \rangle]$: first and last line to display (optional; defaults to the whole file).
- `\includeOutput` `\includeOutput{\langle label \rangle}[\langle type \rangle]` embeds the output of executed code.
- $\langle label \rangle$: output file name (must match $\langle label \rangle$ in `\runExtCode`; empty $\Rightarrow$ uses counter).
 - $[\langle type \rangle]$: `vbox` (default) = verbatim in a box; `tex` = raw L^AT_EX; `inline` = inline text.
- `\inln` `\inln{\langle prog \rangle}{\langle code \rangle}[\langle label \rangle][\langle type \rangle]` runs a short piece of code and displays the result inline.
- $\langle prog \rangle$: executable or language server.
 - $\langle code \rangle$: source code.
 - $[\langle label \rangle]$: output file name (optional).
 - $[\langle type \rangle]$: controls display format and caching. Values: `inline` (default), `vbox`, `tex`; append `.cache` to any of these (e.g. `vbox.cache`) to reuse a cached result, re-running only if the output file is absent.
- `\showChunk` `\showChunk{\langle lang \rangle}{\langle src \rangle}{\langle id \rangle}[\langle begin \rangle][\langle end \rangle]` displays a labelled chunk of a source file. Chunks are delimited by comment lines containing `label==\langle id \rangle` (start) and `==end` (end) by default. These delimiters can be overridden with $[\langle begin \rangle]$ and $[\langle end \rangle]$.
- `\writeChunk` `\writeChunk{\langle lang \rangle}{\langle src \rangle}{\langle id \rangle}[\langle begin \rangle][\langle end \rangle]` extracts the chunk identified by $\langle id \rangle$ from $\langle src \rangle$ and writes it to `generated/\langle src \rangle-\langle id \rangle` on disk. This is called internally by `\showChunk` and `\runLANGChunk`, but can also be used standalone when only the extraction step is needed. In normal (non-cache) mode, extraction is skipped and the cached file reused whenever $\langle src \rangle$'s MD5 checksum matches the one recorded the last time it was extracted, so repeated `\showChunk` calls against an unchanged source are effectively free; editing $\langle src \rangle$ automatically invalidates every chunk cached from it, so no manual cache-clearing step is needed. In `cache` mode, the checksum check itself is skipped: an existing cached file is trusted outright with no subprocess call at all, matching `\runExtCode`'s and `\inln`'s behaviour in that mode (see Section 2).

2.5 Extended commands

- `\runCodeIncOut` `\runCodeIncOut{\langle prog \rangle}{\langle src \rangle}[\langle flag \rangle][\langle label \rangle][\langle type \rangle]` combines `\runExtCode` and `\includeOutput` in one call.

2.6 Language-specific shortcuts

Replace `LANG` with `Julia`, `MatLab`, `Python`, or `R` in the following commands.

- `\runLANG` `\runLANG[\langle prog \rangle]{\langle src \rangle}{\langle label \rangle}[\langle flag \rangle]` runs an external `LANG` code file. $[\langle prog \rangle]$ is optional and defaults to the `talk2stat` `LANG` server.

- `\runLANGIncOut` `\runLANGIncOut[⟨prog⟩]{⟨src⟩}[⟨flag⟩][⟨label⟩][⟨type⟩]` runs a LANG code file and embeds the output.
- `\inlnLANG` `\inlnLANG[⟨prog⟩]{⟨code⟩}[⟨label⟩][⟨type⟩]` runs LANG source code and displays the result inline. If `⟨code⟩` is wrapped in triple backticks (`“code”`), it is sent directly to the server; otherwise it is written to a file first.
- `\runLANGChunk` `\runLANGChunk[⟨prog⟩]{⟨src⟩}{⟨id⟩}[⟨flag⟩][⟨label⟩][⟨type⟩]` extracts the chunk `⟨id⟩` from `⟨src⟩`, runs it, and embeds the output.
- `[⟨label⟩]`: output file name. When omitted, the label is derived automatically as `⟨src⟩-⟨id⟩` (e.g. `code/analysis.R-section2`).
 - `[⟨type⟩]`: output display type (`vbox`, `tex`, `inline`, or their `.cache` variants).

For example:

```
\runR{code/analysis.R}{result1}
\runRIncOut{code/analysis.R}[] [result1]
\runRChunk{code/analysis.R}{section2}
\inlnR{“mean(c(1,2,3,4,5))”}
```

2.6.1 Python batch mode

- `\runPythonBatch` `\runPythonBatch[⟨src⟩][⟨label⟩]` runs a Python file in batch mode (no server). Requires the [dill](#) module (`pip3 install dill`), which provides session save/restore so that variables persist across calls.

3 Auxiliary tools

The following scripts are distributed with the package and address common workflow needs. Copy them into your project directory alongside your `.tex` file.

3.1 wait_for_server.py

Polls a `talk2stat` server until it is ready to accept connections. This eliminates the server-startup race condition that can occur in `nohup` mode, where the first `\runR` call may arrive before the server socket is open.

```
python3 wait_for_server.py [LANG [DIR [TIMEOUT]]]
```

- `LANG`: language server to wait for — `R`, `python`, `julia`, or `matlab` (default: `R`).
- `DIR`: working directory of the server (default: `./`).
- `TIMEOUT`: seconds before giving up (default: 120).

3.2 runcode-Makefile.sample

A sample `Makefile` for documents compiled in `nohup` server mode. Copy it to your project as `Makefile` and set the `MAIN` and `LANGS` variables. The `make all` target encodes three best practices:

1. Quit any existing server before starting a fresh one, so the new server picks up the current `.config` settings (including any updated `PIPETIMEOUT`).
2. Call `wait_for_rserver.py` to block until the server is ready before the first $\text{\LaTeX}$ pass.
3. After the draft pass (`--no-pdf / -draftmode`), send a language-appropriate no-op command as a sync barrier (`invisible(NULL)` for R, `pass` for Python, etc.) to ensure all queued scripts have finished writing their output files before the final passes begin.

PIPETIMEOUT note: the default timeout in the generated `.config` is 60 seconds. For long-running scripts, raise it by editing the `.config` file (e.g. `PIPETIMEOUT = 3600`) and restarting the server (`make stopserver`).

3.3 consolidate.py

Produces a self-contained copy of a `runcode` project in which all cached outputs are inlined into the `.tex` source, so the result compiles without `talk2stat` or any language runtime. This is useful for submission to Overleaf, a journal, or any recipient who does not have `runcode` installed.

```
python3 consolidate.py [--out DIR] [--engine CMD] [--no-compile]
                      [--exclude GLOB] MAIN.tex
```

The script copies the project to `standalone/` (configurable via `--out`), transforms all `.tex` files by replacing `\includeOutput`, `\inlnR/\inlnPython/...`, `\runRIncOut`, `\runRChunk`, etc. with the cached content from `generated/`, and substitutes `\usepackage{runcode}` with a minimal `tcolorbox + listings` shim. Run-only commands (`\runR`, `\runPython`, ...) are silently removed. A `make consolidate` target is provided in `runcode-Makefile.sample`.

4 Revisions

- **v2.6.2, August 2026:** The v2.6.1 MD5 checksum check fixed `\writeChunk`'s missing cache, but the checksum subprocess itself ran unconditionally — including under the `cache` package option, where it defeats the point: on Overleaf (which `make_overleaf.py`-style workflows put into `cache` mode) this meant one `python3` spawn per `\showChunk` call, on every compile, even with a warm cache. `\writeChunk` now checks `\ifruncode` first, matching `\runExtCode` and `\inln`: in `cache` mode it trusts an existing cached file outright with zero subprocess calls; the checksum check only runs in normal mode. Verified against the full factorial design (`\ifruncode` true/false $\times$ missing/fresh/stale cached file) in the new `examples/WriteChunkCacheTest/` regression test.
- **v2.6.1, August 2026:** `\writeChunk` re-scanned its whole source file, line by line, via a slow $\text{\TeX}$ -native loop on every call, with no caching — unlike every other command in the package. On documents with many `\showChunk` calls against large source files, across the multiple $\text{\LaTeX}$ passes a typical build requires, this dominated compile time (in one real-world document, a

single pass went from 3m33s to 11s once `\showChunk` calls were disabled). `\writeChunk` now applies the same MD5 checksum-based staleness check used elsewhere in the package (see the v2.6 entry below): the scan is skipped and the cached `generated/⟨src⟩-⟨id⟩` file reused whenever `⟨src⟩` is unchanged since it was last extracted.

- **v2.6, June 2026:**

1. Checksum-based cache invalidation: source files are fingerprinted with MD5; a script is re-executed only when its source has changed or when the output file is missing, avoiding unnecessary reruns.
2. New `wait_for_rserver.py`: eliminates the server-startup race condition in `nohup` mode (see Section 3).
3. New `runcode-Makefile.sample`: sample `Makefile` with server lifecycle management, startup synchronisation, and sync barrier (see Section 3).
4. New `consolidate.py`: produces standalone L^AT_EX projects with all cached outputs inlined (see Section 3).

- **v2.4.1, June 10, 2026:** Bug fixes in `\writeChunk`: (1) replaced `mkdir -p` with Python `os.makedirs` for cross-platform subdirectory creation; (2) fixed stream exhaustion when more than 16 `\showChunk` calls appear in one document (streams are now pre-allocated at package load time); (3) removed erroneous `\write18{ad ...}` call in `\showChunk`. Added AUCTeX magic comment to example files.

- **v2.4, January 17, 2025:** (1) Put file names within `$$` for more robust warnings; (2) use the Python `advance-touch` package to create sub-folders for chunk-related commands.

- **v2.3, January 12, 2024:** Two bug fixes: (1) removed extra space after `\inlnX`; (2) fixed compilation error when an underscore appeared inside R code.

- **v2.2, September 8, 2023:** Added `\showChunk` basic command and `\runLANGChunk` commands for multiple languages.

- **v2.1, June 30, 2023:** Detokenize code passed to `\inln`. This prevents special L^AT_EX characters (e.g. backslash) from being escaped before reaching the interpreter. Thanks to [kiryph](#) for the report.

- **v2.0, June 23, 2023:** Added `\runCodeIncOut`; updated `\runExtCode` to override cache when output is missing; updated `\inln` to accept `cache` in `Arg4`. Thanks to [kiryph](#) for the suggestions.

- **v1.9, June 13, 2023:** Updated `\inln`: optional `Arg3` is the output file name, optional `Arg4` is the output type.

- **v1.8, January 18, 2023:** Added support for the listings package.

- **v1.7, August 20, 2022:** Renamed the `tmp/` folder to `generated/` (CTAN requirement); renamed the troubleshooting file.

- **v1.6, August 10, 2022:** Stop only configured/running servers; new `reducedspace` option; default server timeout changed to 60 s; expanded troubleshooting document.
- **v1.5, July 23, 2022:** Removed `utf8x` option from `inputenc` (conflict with `hyperref`).
- **v1.4, July 18, 2022:** Fixed a bug in cache mode.
- **v1.3, May 14, 2022:** Removed hard-coded `minted` options.
- **v1.2, May 3, 2022:** Added Python server and batch-mode options.
- **v1.1, April 17, 2021:** Added `nohup` option; improved error handling for missing code files and zero-byte outputs.

5 Contributing

We welcome contributions by opening issues or pull requests on [GitHub](#). Additional example documents written using `runcode` are especially appreciated.

Citing `runcode`: Haim Bar and HaiYing Wang (2021). [Reproducible Science with L^AT_EX](#). *Journal of Data Science* 19(1), 111–125. DOI: 10.6339/21-JDS998.