

# Package: carmar (via r-universe)

October 15, 2026

**Type** Package

**Title** Notebooks and Runnable Documents on Your Own Local R

**Version** 0.8.6

**Description** A single-file notebook for 'R' that runs on the 'R' installation already on the computer, and a 'Quarto' filter that makes the code chunks of a rendered web page runnable against each reader's own local 'R'. The notebook is a self-contained web page opened in the default browser; code is evaluated by a separate local 'R' process (the kernel) that the page reaches over a loopback socket, so closing the calling session does not close the notebook. Nothing is installed elsewhere on the system: on the user's explicit call the kernel process is started, its state is written under the directory returned by `tools::R_user_dir()`, and one small discovery record is written under `'~/carmar/run'` and removed on a clean stop.

**License** MIT + file LICENSE

**URL** <https://lacarm.com/carmar/>

**BugReports** <https://github.com/mohsaqr/CarmaR-releases/issues>

**Encoding** UTF-8

**Depends** R (>= 4.1)

**Imports** httpuv, jsonlite, knitr, evaluate (>= 1.0.0), processx, tools, utils

**Suggests** testthat (>= 3.0.0), rmarkdown

**SystemRequirements** Rscript (an R >= 4.1 installation reachable as Rscript); pandoc (optional, for citation styles); Quarto (>= 1.4, optional, to render publishable sites)

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**RoxygenNote** 7.3.3

**Config/pak/sysreqs** make zlib1g-dev

**Repository** https://mohsaqr.r-universe.dev  
**Date/Publication** 2026-09-15 14:55:10 UTC  
**RemoteUrl** https://github.com/mohsaqr/CarmaR-releases  
**RemoteRef** HEAD  
**RemoteSha** 80768eb8415548a9f7d3910f0dc487de16731b75

Contents

|                            |           |
|----------------------------|-----------|
| check_publishing . . . . . | 2         |
| check_upgrade . . . . .    | 3         |
| listen . . . . .           | 4         |
| make_runnable . . . . .    | 6         |
| run . . . . .              | 7         |
| run_published . . . . .    | 8         |
| serve_shared . . . . .     | 9         |
| sessions . . . . .         | 11        |
| stop_kernel . . . . .      | 11        |
| upgrade . . . . .          | 12        |
| use_publishing . . . . .   | 13        |
| <b>Index</b>               | <b>15</b> |

---

|                  |                                                               |
|------------------|---------------------------------------------------------------|
| check_publishing | <i>Check a rendered site for the CarmaR publishing pieces</i> |
|------------------|---------------------------------------------------------------|

---

Description

Reads every rendered ‘\*.html’ page under ‘site’ and reports, per page, whether the ‘carmar’ filter ran (the page carries the port and label meta tags) and whether the runtime assets it references - the ‘carmar-published-\*/carmar-publish.js’ script and its ‘.css’ - exist beside the page. A page with ‘port’ ‘NA’ was rendered without the filter; a page with ‘assets’ ‘FALSE’ will render but its Run controls will never appear, which is what happens when a site is published without its ‘site\_libs’ directory.

Usage

```
check_publishing(site = "_site")
```

Arguments

|      |                                                                                              |
|------|----------------------------------------------------------------------------------------------|
| site | Path to the rendered output directory (default “_site”; a ‘Quarto’ book renders to ‘_book’). |
|------|----------------------------------------------------------------------------------------------|

**Value**

A ‘data.frame’, one row per HTML page, paths ascending: ‘file’ (relative to ‘site’), ‘port’ (integer, ‘NA’ without the filter), ‘label’ (the Run button’s text, ‘NA’ without the filter), ‘assets’ (logical - both runtime files referenced by the page exist). Zero rows when the directory holds no HTML. An error of class ‘carmar\_no\_site’ when ‘site’ does not exist.

**Examples**

```
site <- tempfile("site")
dir.create(file.path(site, "site_libs", "carmar-published-0.1.0"), recursive = TRUE)
file.create(file.path(site, "site_libs", "carmar-published-0.1.0",
  c("carmar-publish.js", "carmar-publish.css")))
writeLines(c('<meta name="carmar-port" content="4747">',
  '<meta name="carmar-label" content="Run on my computer">',
  '<script src="site_libs/carmar-published-0.1.0/carmar-publish.js"></script>',
  '<link href="site_libs/carmar-published-0.1.0/carmar-publish.css" rel="stylesheet">'),
  file.path(site, "index.html"))
writeLines("<p>rendered without the filter</p>", file.path(site, "about.html"))
carmar::check_publishing(site)
unlink(site, recursive = TRUE)
```

check\_upgrade

*Check whether a newer CarmaR is available***Description**

Asks the release feed which version of CarmaR is current and compares it with the installed package. The answer is remembered in a small record under ‘tools::R\_user\_dir("carmar", "data")’, so the feed is consulted at most once every ‘CARMAR\_UPDATE\_EVERY’ seconds (a day by default) and a person is asked about the same version once, not at every open.

**Usage**

```
check_upgrade(force = FALSE)
```

**Arguments**

force                    ‘TRUE’ consults the feed even when the recorded answer is still fresh.

**Details**

This verb contacts the network, and only when called: nothing in the package does so at load, start or install. It asks ‘https://lacarm.com/carmar/carmar-version.json’ first and the GitHub release feed second, each with a ten-second timeout, and reads one JSON object whose fields are ‘version’ (such as “7.60”), ‘released’ (a date) and ‘install’ (the installer’s URL). A feed that returns anything else (a CDN error page arrives as HTTP 200 HTML) counts as not answering. When no feed answers, the record is still written, so an offline machine is not asked to retry at every open.

Settings, all environment variables:

**‘CARMAR\_NO\_UPDATE\_CHECK=1’** never consult the feed and write nothing; **‘source’** is **“off”**.

**‘CARMAR\_UPDATE\_EVERY’** seconds a recorded answer stays fresh (default 86400; **‘0’** consults the feed on every call).

**‘CARMAR\_UPDATE\_URL’** one URL to read instead of the ladder above (a mirror, or a **‘file://’** path); **‘source’** is then that URL’s host, or **“file”**.

### Value

A one-row **‘data.frame’** with columns, in this order: **‘installed’** (character, the installed package version), **‘available’** (character, the version the feed named; **‘NA’** when unknown), **‘newer’** (logical, **‘available’** is greater than **‘installed’**; **‘FALSE’** when unknown), **‘ask’** (logical, **‘TRUE’** only when **‘newer’** and this call consulted the feed rather than the record - so **‘TRUE’** at most once per **‘CARMAR\_UPDATE\_EVERY’** seconds), **‘checked’** (**‘POSIXct’**, when the feed was last consulted; **‘NA’** when never), **‘source’** (character: **“off”**, **“cache”**, **“lacarm.com”**, **“github”**, **“unreachable”**, or the host of **‘CARMAR\_UPDATE\_URL’**) and **‘note’** (character, one sentence for a person).

Raises **‘carmar\_bad\_setting’** when **‘CARMAR\_UPDATE\_EVERY’** is set to something that is not a non-negative number, and warns with class **‘carmar\_record\_unreadable’** (then consults the feed) when the record file cannot be read.

### See Also

**[upgrade()]** installs the version this verb names.

### Examples

```
## Not run:
check_upgrade()
check_upgrade(force = TRUE)

# The launchers ask from a shell with exactly this line. jsonlite writes
# the row as one JSON object, formats `checked` as "YYYY-MM-DD HH:MM:SS",
# and OMITs a field whose value is NA (so no `available` key means the
# version is unknown, and no `checked` key means never checked):
# [{"installed":"7.17","available":"7.60","newer":true,"ask":true,
#   "checked":"2026-09-09 10:00:00","source":"lacarm.com","note":"..."}]
# Rscript -e "cat(jsonlite::toJSON(carmar::check_upgrade(), auto_unbox = TRUE, dataframe = 'rows'))"

## End(Not run)
```

---

listen

*Keep a kernel listening for published pages*

---

### Description

Starts a headless CarmaR kernel on a FIXED port for the runnable R chunks of a published web page (a **‘Quarto’** site rendered with the **‘carmar’** filter, see **[use\_publishing()]**). Nothing opens: the reader keeps reading the page, presses Run, and the page talks to this kernel.

**Usage**

```
listen(site = NULL, port = 4747, open = FALSE)
```

```
## S3 method for class 'carmar_listener'
print(x, ...)
```

**Arguments**

|      |                                                                                                                                                                                                                      |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| site | The published page's origin, e.g. <code>"https://book.example"</code> or <code>"http://localhost:8080"</code> ; scheme, host, optional port, no path. <code>'NULL'</code> (the default) approves nothing in advance. |
| port | The fixed loopback port the published page expects (default 4747, which is what the filter writes unless the author changed it).                                                                                     |
| open | Also open the full local notebook against this kernel? Default <code>'FALSE'</code> ; a published page normally needs only the kernel.                                                                               |
| x    | A <code>'carmar_listener'</code> , as returned by <code>[listen()]</code> .                                                                                                                                          |
| ...  | Ignored.                                                                                                                                                                                                             |

**Details**

The port is a promise, not a preference: a published page carries the port it will look for, so `'listen()'` never moves to another one. A CarmaR kernel already answering on `'port'` is reused; a port held by anything else is an error of class `'carmar_port_taken'`.

A page is not trusted because it asked. Pass `'site'` - the page's exact origin, scheme and host and nothing more - to approve that one site for this kernel's lifetime. When the kernel is started here the approval rides in its environment; when a kernel already answers, the approval is sent to its loopback `'published/authorize'` endpoint, which accepts only requests from a local, non-browser process such as this one. Without `'site'`, the page's first Run opens a local consent page and the reader approves it there, once per kernel.

**Value**

Invisibly, a one-row `'data.frame'` of class `'carmar_listener'`: `'port'` (integer), `'url'` (the kernel's loopback address), `'site'` (the origin approved, `'NA'` when none was given), `'approved'` (logical - was `'site'` approved for this kernel), `'started'` (logical - `'TRUE'` when this call started the kernel, `'FALSE'` when one was already listening). Its print method says where the kernel listens and for which site.

`'print()'` returns `'x'` invisibly.

**What is written where**

`'run()'`, `[listen()]` and `[serve_shared()]` create the package's state directory, `'tools::R_user_dir("carmar", "data")'`, and write there the session record `'url-<port>'`, the kernel log `'kernel-<port>.log'` and a one-line trampoline page the browser is handed. The kernel itself writes its discovery record `'~/carmar/run/kernel-<port>.json'` (mode 0600, removed on a clean stop; `'CARMAR_RUNTIME_DIR'` relocates it). Nothing else on the system is touched, and no network request leaves the machine.

## Examples

```
# Not run: starts an R process that outlives this session.
## Not run:
carmar::listen("https://book.example")
carmar::listen()                # approve sites from the page itself

## End(Not run)
# The site must be an exact origin - a path is refused before anything starts:
try(carmar::listen("https://book.example/chapter-1.html"))
```

---

make\_runnable

*Make already rendered HTML pages runnable*


---

## Description

A page that was rendered long ago - an R Markdown report, a bookdown ‘\_book/’, a pkgdown article, any HTML that shows R code in ‘<pre class="r">’ or ‘<code class="sourceCode r">’ blocks - becomes runnable without being rendered again: one ‘<script>’ tag before ‘</head>’ loads the CarmaR runtime, which gives every R block a Run button and a strip at the top of the page that connects to the reader’s own local R (see ‘listen()’).

## Usage

```
make_runnable(
  path,
  assets = runtime_url(),
  port = 4747,
  label = "Run on my computer"
)

## S3 method for class 'carmar_runnable'
print(x, ...)
```

## Arguments

|        |                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| path   | an ‘.html’ file, or a directory whose ‘.html’ files (searched recursively) are all stamped - a bookdown ‘_book/’, say.                                                                                                                                                                                                                                                                                                   |
| assets | where the page loads the runtime from: a URL prefix ending in ‘/’ (default: CarmaR’s copy on the jsDelivr CDN, served from the ‘CarmaR-releases’ repository; “https://lacarm.com/carmar/publish/” is the self-hosted mirror), or “local” to copy the two runtime files from this package into a ‘carmar/’ folder beside the pages and reference them relatively - for a site that must not load anything from elsewhere. |
| port   | the loopback port the page dials (the fixed 4747 unless the reader is told otherwise); written as ‘<meta name="carmar-port">’.                                                                                                                                                                                                                                                                                           |
| label  | the strip’s button; written as ‘<meta name="carmar-label">’.                                                                                                                                                                                                                                                                                                                                                             |
| x      | a ‘carmar_runnable’ result.                                                                                                                                                                                                                                                                                                                                                                                              |
| ...    | ignored.                                                                                                                                                                                                                                                                                                                                                                                                                 |

## Details

The tag is inserted into each file in place, once: a file that already loads ‘carmar-publish.js’ is left as it is. The runtime links its own stylesheet, so the tag is all a page needs.

## Value

Invisibly, a ‘data.frame’ of class ‘carmar\_runnable’, one row per HTML file, paths ascending: ‘file’ (character), ‘stamped’ (logical: ‘TRUE’ when the tag was inserted now, ‘FALSE’ when it was already there), ‘assets’ (character: the ‘src’ the page loads). Raises a classed error ‘carmar\_no\_html’ when ‘path’ names no HTML file, and ‘carmar\_no\_extension’ when ‘assets = "local"' and this installation carries no runtime.

‘print()’ returns ‘x’ invisibly.

## Examples

```
dir <- file.path(tempdir(), "report")
dir.create(dir, showWarnings = FALSE)
writeLines(c("<html><head><title>t</title></head>",
            "<body><pre class=\"r\"><code>1 + 1</code></pre></body></html>"),
            file.path(dir, "index.html"))
make_runnable(dir)
make_runnable(dir)           # a second call changes nothing
```

---

run

*Launch CarmaR*


---

## Description

Starts an independent local CarmaR kernel and opens the notebook in the default browser. ‘run()’ never asks permission to do what it was told: you typed the command, so the session starts. The kernel is a separate R process; closing the R session that called ‘run()’ does not close the notebook.

## Usage

```
run(port = 4747, open = TRUE, new = TRUE, file = NULL)
```

## Arguments

|      |                                                                                                                                                                                                                                                                                                                                       |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| port | Port for the kernel. The default 4747 is deliberate and fixed: browsers keep a page’s saved notebooks per address, so a kernel that moved ports would greet the student with an empty notebook every time. When ‘port’ is busy the next free port is used and said out loud.                                                          |
| open | Open the browser once the kernel answers? Default ‘TRUE’.                                                                                                                                                                                                                                                                             |
| new  | Start an independent session? Default ‘TRUE’, and that default is the rule: a session is a process you can see, each ‘run()’ is a new one on the next free port, and nothing is rejoined. ‘FALSE’ is the one exception, an explicit opt-in that reopens the notebook of a session already live on ‘port’ instead of starting another. |

**file** Optional document path delivered to the notebook. With `'new = FALSE'` a file association hands the document to the session already on screen (the kernel's `'/open'` door) and starts one only when none is live.

### Value

Invisibly, a single string: the versioned notebook `'file://'` URL with its kernel selected in the fragment (`'#kernel=<port>'`), the launch capability the kernel announced (`'&pair='`), and the kernel's session stamp (`'&session=<started>'`, its `'/health'` boot time) so the page can tell a fresh R from a reload of the same one.

### What is written where

`'run()'`, `[listen()]` and `[serve_shared()]` create the package's state directory, `'tools::R_user_dir("carmar", "data")'`, and write there the session record `'url-<port>'`, the kernel log `'kernel-<port>.log'` and a one-line trampoline page the browser is handed. The kernel itself writes its discovery record `'~/carmar/run/kernel-<port>.json'` (mode 0600, removed on a clean stop; `'CARMAR_RUNTIME_DIR'` relocates it). Nothing else on the system is touched, and no network request leaves the machine.

### Examples

```
# Not run: starts an R process that outlives this session and opens a
# browser tab.
## Not run:
carmar::run()
carmar::run(new = FALSE) # the exception: reopen a session live on 4747

## End(Not run)
```

---

run\_published

*Start CarmaR for a published page*

---

### Description

A thin alias of `[listen()]`, kept for the readers whose page told them to type `'carmar::run_published()'`. Every argument is passed through unchanged.

### Usage

```
run_published(site = NULL, port = 4747, open = FALSE)
```

### Arguments

|      |                                                                                                                                                                                                                      |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| site | The published page's origin, e.g. <code>"https://book.example"</code> or <code>"http://localhost:8080"</code> : scheme, host, optional port, no path. <code>'NULL'</code> (the default) approves nothing in advance. |
| port | The fixed loopback port the published page expects (default 4747, which is what the filter writes unless the author changed it).                                                                                     |
| open | Also open the full local notebook against this kernel? Default <code>'FALSE'</code> ; a published page normally needs only the kernel.                                                                               |

**Value**

The ‘carmar\_listener’ data frame [listen()] returns, invisibly.

**Examples**

```
# Not run: starts an R process that outlives this session.
## Not run:
carmar::run_published("https://book.example")

## End(Not run)
```

---

|              |                                                    |
|--------------|----------------------------------------------------|
| serve_shared | <i>Serve CarmaR to other people on the network</i> |
|--------------|----------------------------------------------------|

---

**Description**

The classroom deployment: one CarmaR per person, each started under that person’s own operating-system account on its own port, with TLS and login handled entirely by a reverse proxy in front. CarmaR itself never sees a password - it trusts an identity header, and only when the request came from the proxy’s own address.

**Usage**

```
serve_shared(
  hosts,
  port = 4747,
  bind = "0.0.0.0",
  user_header = "X-Forwarded-User",
  trusted_proxy = "127.0.0.1",
  origins = character(),
  allow_unauthenticated = FALSE
)
```

**Arguments**

|               |                                                                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hosts         | The hostnames readers will type, e.g. “stats.example.edu”. Required: off loop-back the ‘Host’ allow-list is the only thing standing between this kernel and a DNS-rebinding attack, so an empty one is refused rather than defaulted. |
| port          | Port for this person’s kernel (default 4747). One port per person; the proxy maps each account to its own.                                                                                                                            |
| bind          | Address to listen on (default “0.0.0.0”). Use the proxy-facing interface address when the machine has more than one.                                                                                                                  |
| user_header   | The header the proxy sets to the authenticated user’s name (default “X-Forwarded-User”; Caddy’s ‘forward_auth’ writes ‘Remote-User’).                                                                                                 |
| trusted_proxy | Address(es) the proxy connects from (default “127.0.0.1”). The identity header is read from these and ignored from anywhere else.                                                                                                     |

**origins** Extra exact browser origins allowed to open a socket, e.g. `"https://stats.example.edu"`. The kernel's own address is always allowed.

**allow\_unauthenticated** Serve with no proxy identity at all? Default `'FALSE'`. `'TRUE'` means everyone who can reach this port can run R as the account this kernel runs under. There is no safe network on which that is a shortcut.

## Details

Binding off loopback INVERTS one of CarmaR's trust rules, and this verb exists so that inversion is stated rather than stumbled into. On `'127.0.0.1'`, a client that sends no `'Origin'` header is allowed outright: no Origin means it is not a browser, so it is a process belonging to this same user, who could have run `'Rscript'` anyway. Reachable from the network, the identical header means an unauthenticated stranger - so here it is refused. The kernel will not start at all without `'hosts'`, and will not start unauthenticated unless you say `'allow_unauthenticated = TRUE'` out loud.

## Value

Invisibly, a one-row `'data.frame'` with `'port'` (integer), `'bind'`, `'url'` (the kernel's own address), `'hosts'` (comma-separated), `'authenticated'` (logical) and `'log'` (path to the kernel log).

## The deployment

The shape is `'reader -> TLS reverse proxy -> 127.0.0.1:<port>'`, one kernel per person, each running as that person's own account so that file permissions, quotas and `'~'` keep meaning what they already mean on the machine. The proxy authenticates the reader (OIDC, SAML, LDAP - whatever the institution already has) and maps the authenticated name to that person's port, setting `'user_header'` on every forwarded request. The kernel reads that header only from `'trusted_proxy'`, so it is an identity for exactly as long as nobody but the proxy can reach the port. With Caddy, a `'forward_auth'` block in front of `'reverse_proxy localhost:<port>'` does both; Caddy's `'forward_auth'` writes `'Remote-User'`, hence that value for `'user_header'`. A misconfigured kernel - off loopback with no `'hosts'`, or with no authentication in front - does not come up degraded: it refuses to start and says which part of the posture was unsafe.

## What is written where

`'run()'`, `[listen()]` and `[serve_shared()]` create the package's state directory, `'tools::R_user_dir("carmar", "data")'`, and write there the session record `'url-<port>'`, the kernel log `'kernel-<port>.log'` and a one-line trampoline page the browser is handed. The kernel itself writes its discovery record `'~/carmar/run/kernel-<port>.json'` (mode 0600, removed on a clean stop; `'CARMAR_RUNTIME_DIR'` relocates it). Nothing else on the system is touched, and no network request leaves the machine.

## Examples

```
# Not run: binds a network port and starts an R process that outlives
# this session.
## Not run:
carmar::serve_shared(hosts = "stats.example.edu")
carmar::serve_shared(hosts = "stats.example.edu", port = 4801,
                      user_header = "Remote-User")
```

```
## End(Not run)
```

---

sessions

*List CarmaR sessions*

---

## Description

Every CarmaR kernel recorded on this machine - package launches (the ‘url-<port>’ state records) and every other door (the shared ‘~/carmar/run/kernel-<port>.json’ runtime registry) - one row per port, each health-checked at call time over loopback. Read-only: dead records are reported, not removed (‘run()’ prunes them on its next launch). Once attached, kernels retain R across page disconnections by default. Use explicit Quit to stop them. Positive ‘CARMAR\_LINGER’ values opt into automatic idle shutdown; zero disables it.

## Usage

```
sessions()
```

## Value

A ‘data.frame’, one row per recorded session, ports ascending: ‘port’ (integer), ‘alive’ (logical), ‘title’ (what the notebook attached to the session calls itself; ‘NA’ until a page names one), ‘listen’ (logical - a headless kernel started by [listen()] and not yet claimed by a notebook), ‘page’ (the notebook file URL that reopens the session; ‘NA’ when it is not running), ‘source’ (“package” for [run()] launches, “runtime” for other doors). Zero rows when nothing is recorded.

## Examples

```
carmar::sessions()
```

---

stop\_kernel

*Stop a CarmaR kernel*

---

## Description

Asks the kernel on ‘port’ to shut down through its loopback-only ‘/shutdown’ endpoint - the clean path, the same one the notebook’s Quit uses. Pass “all” to stop every live session [sessions()] can see. A stopping kernel removes its own ‘~/carmar/run/kernel-<port>.json’; this function removes the package’s ‘url-<port>’ record.

## Usage

```
stop_kernel(port = 4747)
```

**Arguments**

port                      The port [run()] or [listen()] used (default 4747), or "all".

**Value**

For one port, invisibly 'TRUE' if a kernel was asked to stop and 'FALSE' if none was running there. For "all", invisibly a 'data.frame' with one row per live session that was asked: 'port' (integer) and 'stopped' (logical); zero rows when nothing was running.

**Examples**

```
# A port with no kernel behind it: nothing to stop, FALSE, no error.
carmar::stop_kernel(65000)
```

---

upgrade

*Upgrade CarmaR to the current release*

---

**Description**

Downloads the public installer, 'Install-CarmaR.R' - the same file the install line on the website sources - and runs it. This is the one verb in the package that downloads and runs code, and it does so only when called. The installer replaces the installed 'carmar' package with the current release and reinstalls the CarmaR application beside it, and this verb records the result so that [check\_upgrade()] reports the new version as current.

**Usage**

```
upgrade(run = FALSE)
```

**Arguments**

run                      'TRUE' lets the installer start CarmaR when it finishes, as the install line does; 'FALSE' (the default) installs and returns. The choice reaches the installer as 'CARMAR\_INSTALL\_START' ("1" or "0"), set for the duration of the call and restored afterwards.

**Details**

The installer is fetched from 'https://lacarm.com/carmar/Install-CarmaR.R' first and from the GitHub release second, to a temporary file, with a ten-second timeout each. A file is refused unless it parses as R and its first three lines carry the marker 'Install CarmaR', so a CDN error page (HTTP 200 HTML) or a truncated download is never sourced. 'CARMAR\_INSTALL\_URL' names one source to use instead (a mirror, or a 'file://' path); 'source' in the result is then that URL's host, or "file".

After this verb the current R session should not use carmar until it is restarted. The installer unloads the 'carmar' namespace before replacing the package on disk (a package replaced while loaded leaves the session bound to the old lazy-load database, which is the "carmar.rdb is corrupt" error). This verb itself reads the new version from the reinstalled package's 'DESCRIPTION' on disk and touches nothing else in the namespace.

**Value**

Invisibly, a one-row ‘data.frame’ with columns ‘before’ and ‘after’ (character, the package version before and after), ‘upgraded’ (logical, ‘after’ is greater than ‘before’) and ‘source’ (character: “lacarm.com”, “github”, or the host of ‘CARMAR\_INSTALL\_URL’).

Raises ‘carmar\_installer\_unavailable’ when no source answered, and ‘carmar\_installer\_invalid’ when what answered is not the installer (it does not parse, or its first three lines lack the marker). Whatever the installer itself raises propagates unchanged.

**See Also**

[check\_upgrade()] says whether there is anything to upgrade to.

**Examples**

```
## Not run:
upgrade()
upgrade(run = TRUE)

## End(Not run)
```

---

use\_publishing

---

*Make a Quarto project’s R chunks runnable with CarmaR*


---

**Description**

Copies the CarmaR ‘Quarto’ extension into ‘project/\_extensions/carmar’, enables the ‘carmar’ filter in the project’s ‘\_quarto.yml’, and writes the filter’s options block (‘carmar: port:’ and ‘label:’) when the file has none, so the two things an author may want to change are visible. After the next ‘quarto render’, every visible R chunk on the published pages gains a Run control against each reader’s own local R (the reader starts it with [listen()]).

**Usage**

```
use_publishing(
  project = ".",
  extension_source = system.file("quarto", "_extensions", "carmar", package = "carmar")
)

## S3 method for class 'carmar_publishing'
print(x, ...)
```

**Arguments**

**project** Path to the ‘Quarto’ project (the directory holding ‘\_quarto.yml’, or where one should be created). Default: the working directory.

|                  |                                                                                                                                                                                   |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| extension_source | Directory holding the extension files. The default is the copy shipped inside the installed carmar package; a development checkout can pass its own ‘_extensions/carmar’ instead. |
| x                | A ‘carmar_publishing’, as returned by [use_publishing()].                                                                                                                         |
| ...              | Ignored.                                                                                                                                                                          |

### Value

Invisibly, a one-row ‘data.frame’ of class ‘carmar\_publishing’: ‘project’, ‘extension’ (the destination directory), ‘config’ (the ‘\_quarto.yml’ path), ‘enabled’ (logical - is the filter active in that file), ‘options’ (logical - does the file carry a ‘carmar:’ options block). Its print method says what happened and what to do next.

‘print()’ returns ‘x’ invisibly.

### How a published page reaches R

The filter rewrites nothing: the page still renders identically for a reader without R. It adds ‘<meta name="carmar-port">’ and ‘<meta name="carmar-label">’ to the page head and a script and stylesheet pair under ‘site\_libs/'. On the reader’s machine the script finds the page’s R chunks and dials a CarmaR kernel on the page’s port over a loopback WebSocket. The kernel does not trust a site because it asked: the reader approves the exact origin once per kernel, either in advance (‘listen("https://book.example")’) or on a local consent page the first Run opens. Code that should be shown but never offered to run goes in a fenced div with the class ‘carmar-no-run’. A site with its own Content Security Policy must allow ‘connect-src ws://127.0.0.1:4747’ (or the port chosen). This is execution, not a sandbox: what a reader runs has the same access as their own R console, so keep the code visible.

Idempotent: running it again refreshes the extension files (JS and CSS together - the pair must never drift apart) and leaves an already-enabled ‘\_quarto.yml’ untouched. A ‘\_quarto.yml’ it cannot edit safely (an inline ‘filters: [...]’ list, or several top-level ‘filters:’ keys) is left exactly as it was, and the lines to add are printed instead.

### Examples

```
project <- tempfile("book")
dir.create(project)
writeLines(c("project:", " type: book"), file.path(project, "_quarto.yml"))
carmar::use_publishing(project)
cat(readLines(file.path(project, "_quarto.yml")), sep = "\n")
unlink(project, recursive = TRUE)
```

# Index

check\_publishing, [2](#)  
check\_upgrade, [3](#)  
  
listen, [4](#)  
  
make\_runnable, [6](#)  
  
print.carmar\_listener (listen), [4](#)  
print.carmar\_publishing  
    (use\_publishing), [13](#)  
print.carmar\_runnable (make\_runnable), [6](#)  
  
run, [7](#)  
run\_published, [8](#)  
  
serve\_shared, [9](#)  
sessions, [11](#)  
stop\_kernel, [11](#)  
  
upgrade, [12](#)  
use\_publishing, [13](#)