Package 'livelink'

August 1, 2026

Title Create Shareable Links for 'webR' and 'Shinylive' Environments

Version 0.1.1

Description Creates shareable links for 'R' code in 'WebAssembly' (WASM) Read-Eval-Print Loop (REPL) environments like 'webR' <https://webr.r-wasm.org/> and for 'Shiny' applications using 'Shinylive' <https://shinylive.io/>. Supports single scripts, multi-file projects, exercise and solution pairs, and batch processing. Includes encoding, decoding, and previewing of links for both 'R' and 'Python' environments.

License AGPL (>= 3)

URL https://r-pkg.thecoatlessprofessor.com/livelink/, https://github.com/coatless-rpkg/livelink

BugReports https://github.com/coatless-rpkg/livelink/issues

Depends R (>= 4.1.0)

Imports base64enc, cli, clipr, jsonlite, lzstring, RcppMsgPack, stats, tools, utils

Suggests knitr, quarto, testthat (>= 3.0.0), withr

Encoding UTF-8

SystemRequirements Quarto command line tools (https://github.com/quarto-dev/quarto-cli).

Config/Needs/website pkgdown

Config/testthat/edition 3

Collate 'as-data-frame.R' 'build-webr-repl-link.R' 'classes.R' 'decode-shinylive-link.R' 'utils.R' 'decode-webr-repl-link.R' 'extract-link.R' 'format-methods.R' 'knit-print.R' 'knitr-engine.R' 'options.R' 'process-input.R' 'shinylive-link.R' 'validators.R' 'webr-repl-directory-links.R' 'webr-repl-links.R'

VignetteBuilder quarto

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author James Joseph Balamuta [aut, cre, cph],
reprex authors [cph] (stringify_expression() and
trim_common_leading_ws(), adapted in R/process-input.R; see
inst/COPYRIGHTS)

Maintainer James Joseph Balamuta <james.balamuta@gmail.com>

Repository CRAN

Date/Publication 2026-07-31 22:10:02 UTC

Contents

`as.character.webr_link` *Extract URLs as character vector*

Description

Pulls the shareable URL(s) out of a livelink object.

Usage

```
## S3 method for class 'webr_link'
as.character(x, ...)

## S3 method for class 'shinylive_link'
as.character(x, ...)

## S3 method for class 'webr_project'
as.character(x, ...)

## S3 method for class 'webr_exercise'
as.character(x, ...)

## S3 method for class 'webr_directory'
as.character(x, ...)

## S3 method for class 'webr_decoded'
as.character(x, ...)

## S3 method for class 'webr_decoded_batch'
as.character(x, ...)

## S3 method for class 'webr_preview'
as.character(x, ...)

## S3 method for class 'shinylive_project'
as.character(x, ...)

## S3 method for class 'shinylive_directory'
as.character(x, ...)

## S3 method for class 'shinylive_decoded'
as.character(x, ...)

## S3 method for class 'shinylive_decoded_batch'
as.character(x, ...)

## S3 method for class 'shinylive_preview'
```

```
as.character(x, ...)
```

Arguments

x	Link object
...	Additional arguments

Value

A character vector of URLs.

- Most objects give a single URL.
- An exercise gives two, named `exercise` and `solution`.
- A directory or a batch gives one URL for each file.

See Also

`repl_urls()` for the same result from a function you can call by name.

---

as.data.frame.livelink

Turn a livelink container into a data frame

Description

The container classes convert to a tidy data frame.

Usage

```
## S3 method for class 'webr_directory'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'shinylive_directory'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'webr_decoded_batch'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'shinylive_decoded_batch'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A `webr_directory`, `shinylive_directory`, `webr_decoded_batch`, or `shinylive_decoded_batch` object.
row.names	A character vector of row names, or `NULL`.
optional	Ignored. It is here so the arguments match those of `as.data.frame()`.
...	Ignored.

Details

A folder of links or a batch of decoded results can be tabulated, filtered, joined, or written to CSV with the tools you already use. For a tibble, wrap the result with `tibble::as_tibble(as.data.frame(x))`.

Value

A data frame.

- A directory gives one row for each generated link, with the columns `filename` and `url`.
- A decoded batch gives one row for each URL that decoded successfully, with the columns `name`, `url`, `total_files`, `total_size`, and `output_dir`. A URL that failed to decode carries no result and is left out. The counts are held in the object's `total_urls` and `successful_urls` fields.

See Also

`webr_repl_directory()` for the webR directory objects tabulated here.

`shinylive_directory()` for the Shinylive directory objects tabulated here.

`decode_webr_link()` and `decode_shinylive_link()` for the decoded batch objects tabulated here.

Examples

```
dir <- tempfile()
dir.create(dir)
writeLines("plot(1:10)",       file.path(dir, "one.R"))
writeLines("hist(rnorm(100))", file.path(dir, "two.R"))

links <- webr_repl_directory(dir)
as.data.frame(links)
```

---

decode_shinylive_link *Decode Shinylive link(s) to extract files to local directory*

Description

Decodes Shinylive sharelinks to extract the embedded files and save them to a local directory.

Usage

```
decode_shinylive_link(
  url,
  output_dir = file.path(tempdir(), "shinylive_files"),
  overwrite = FALSE,
  create_subdir = TRUE,
  name_dirs = TRUE
)
```

Arguments

url	Character string or vector containing Shinylive URL(s)
output_dir	Character string specifying the output directory path. Defaults to a shinylive_files directory inside the session temporary directory. Pass an explicit path to extract somewhere permanent.
overwrite	Logical. Whether to overwrite existing files. Defaults to FALSE.
create_subdir	Logical. If TRUE (default), each decoded link is extracted into its own subdirectory under output_dir rather than directly into it. For a single URL the subdirectory is named shinylive_<hash>, where <hash> is a short fingerprint of the URL. For multiple URLs, see name_dirs. Set FALSE to extract straight into output_dir. With multiple URLs that means identically named files (such as app.R) collide, so a later app overwrites an earlier one when overwrite = TRUE, or is skipped when overwrite = FALSE.
name_dirs	Logical. For multiple URLs, this controls how the per-link subdirectories are named. TRUE (default) numbers them app_01, app_02, and so on. FALSE names each one shinylive_<hash> from the URL fingerprint. Ignored for a single URL, and ignored when create_subdir = FALSE.

Details

This is the reverse operation of creating Shinylive links. Handles both single URLs and multiple URLs automatically.

Value

The decoded contents, in one of two shapes.

For a single URL, a shinylive_decoded object, which is a list with these entries.

- files_info, a data frame with one row per file written, with the columns filename, path, type, and size_bytes.
- output_dir, the directory the files were written to.
- url, the link that was decoded.
- engine, "r" or "python", read off the link.
- mode, "editor" when the link opens the editor, "app" when it opens the running app on its own.
- total_files, how many files were written.
- total_size, the size of those files added up, in bytes.

For several URLs, a shinylive_decoded_batch object, which is a list with these entries.

- results, a named list of the shinylive_decoded objects described above, one per link that decoded, each named after the subdirectory it went into.
- base_dir, the directory those per-link subdirectories sit in.
- urls, the links you passed in.
- total_urls, how many links you passed in.

- `successful_urls`, how many of them decoded.
- `total_files`, how many files were written across all of them.
- `total_size`, the size of those files added up, in bytes.

See Also

`preview_shinylive_link()` to inspect a link without writing files.

`shinylive_r_link()` and `shinylive_py_link()` to create links.

Examples

```
# Round-trip: build a link, then decode it back to files
url <- as.character(shinylive_r_link("library(shiny)"))

result <- decode_shinylive_link(url)
print(result)

# Extract to a directory of your choosing
out <- file.path(tempdir(), "my_app")
decode_shinylive_link(url, output_dir = out, create_subdir = FALSE, overwrite = TRUE)
list.files(out)

# Both engines at once
urls <- c(url, as.character(shinylive_py_link("from shiny import App")))
decode_shinylive_link(urls, output_dir = file.path(tempdir(), "my_apps"))
```

---

decode_webr_link *Decode webR REPL link(s) to extract files to local directory*

Description

Decodes webR REPL sharelinks to extract the embedded files and save them to a local directory.

Usage

```
decode_webr_link(
  url,
  output_dir = file.path(tempdir(), "webr_files"),
  overwrite = FALSE,
  create_subdir = TRUE,
  name_dirs = TRUE
)
```

Arguments

url	Character string or vector containing webR REPL URL(s)
output_dir	Character string specifying the output directory path. Defaults to a webr_files directory inside the session temporary directory. Pass an explicit path to extract somewhere permanent.
overwrite	Logical. Whether to overwrite existing files. Defaults to FALSE.
create_subdir	Logical. If TRUE (default), each decoded link is extracted into its own subdirectory under output_dir rather than directly into it. For a single URL the subdirectory is named webr_<hash>, where <hash> is a short fingerprint of the URL. For multiple URLs, see name_dirs. Set FALSE to extract straight into output_dir.
name_dirs	Logical. For multiple URLs, controls how the per-link subdirectories are named. • TRUE (default) numbers them script_01, script_02, and so on. • FALSE names each one webr_<hash> from the URL fingerprint. Ignored for a single URL, and ignored when create_subdir = FALSE (all files then extract into output_dir).

Details

This is the reverse operation of creating webR links. Handles both single URLs and multiple URLs automatically.

Value

What comes back depends on how many URLs you pass.

For one URL, a webr_decoded object, which is a list with these entries.

- files_info, a data frame of the files written, one row per file, with the columns filename, path, autorun, and size_bytes.
- output_dir, the directory the files were written to.
- url, the link that was decoded.
- mode, the panels the link asks for, or NULL for all of them.
- version, the webR version the link points at.
- flags, the encoding flags read off the link.
- total_files, how many files were written.
- total_size, the size of those files in bytes.

For several URLs, a webr_decoded_batch object, which is a list with these entries.

- results, a list of webr_decoded objects, one for each URL that decoded.
- base_dir, the directory the per-link subdirectories sit under.
- urls, the URLs you passed in, one per entry.
- total_urls, how many URLs were handed in.
- successful_urls, how many of them decoded.
- total_files, how many files were written across every URL.
- total_size, the size of those files in bytes.

See Also

preview_webr_link() to inspect a link without writing files.

webr_repl_link(), webr_repl_project(), and webr_repl_exercise() create the links this function decodes.

Examples

```
# Round-trip: build a link, then decode it back to files
url <- as.character(webr_repl_link("plot(1:10)"))

result <- decode_webr_link(url)
print(result)

# Extract to a directory of your choosing
out <- file.path(tempdir(), "my_code")
decode_webr_link(url, output_dir = out, create_subdir = FALSE, overwrite = TRUE)
list.files(out)

# Several links at once
urls <- c(url, as.character(webr_repl_link("hist(rnorm(100))")))
decode_webr_link(urls, output_dir = file.path(tempdir(), "my_scripts"))
```

---

format.livelink *Format a livelink object as a character vector*

Description

Returns an object's printed representation as a character vector, one element per line.

Usage

```
## S3 method for class 'webr_link'
format(x, ...)

## S3 method for class 'webr_project'
format(x, ...)

## S3 method for class 'webr_exercise'
format(x, ...)

## S3 method for class 'webr_directory'
format(x, ...)

## S3 method for class 'webr_decoded'
format(x, ...)

## S3 method for class 'webr_decoded_batch'
```

```
format(x, ...)

## S3 method for class 'webr_preview'
format(x, ...)

## S3 method for class 'shinylive_link'
format(x, ...)

## S3 method for class 'shinylive_project'
format(x, ...)

## S3 method for class 'shinylive_directory'
format(x, ...)

## S3 method for class 'shinylive_decoded'
format(x, ...)

## S3 method for class 'shinylive_decoded_batch'
format(x, ...)

## S3 method for class 'shinylive_preview'
format(x, ...)
```

Arguments

x	A livelink object (a link, project, exercise, directory, decoded result, batch, or preview).
...	Passed to the object's `print()` method, so preview-specific options such as `show_content` work here too.

Details

Having the printed form as a character vector means it can be captured, logged, pasted into a report, or otherwise reused. `print()` renders the very same content to the console. `format()` hands it back to you instead.

Value

A character vector of formatted lines.

See Also

`as.character.webr_link()` for the shareable URL alone, without the surrounding printed report.

as.data.frame.livelink for turning directory and batch objects into a data frame.

Examples

```
link <- webr_repl_link("plot(1:10)")
format(link)
```

```
writeLines(format(link))
```

`knit_print.livelink` *Render livelink objects as links in knitted documents*

Description

Emit a clickable Markdown link when a link object is the visible result of a 'knitr' chunk.

Usage

```
## S3 method for class 'webr_link'
knit_print(x, ...)

## S3 method for class 'webr_project'
knit_print(x, ...)

## S3 method for class 'webr_exercise'
knit_print(x, ...)

## S3 method for class 'webr_directory'
knit_print(x, ...)

## S3 method for class 'shinylive_link'
knit_print(x, ...)

## S3 method for class 'shinylive_project'
knit_print(x, ...)

## S3 method for class 'shinylive_directory'
knit_print(x, ...)
```

Arguments

`x`	A livelink object (a link, project, exercise, or directory).
`...`	Ignored.

Details

'knitr' calls `knit_print()` on the last value of a chunk. Without these methods a link object would fall back to `print()`, dumping cli console output (a header, box glyphs, metadata) into the rendered page. These methods instead emit a clickable Markdown link, which is almost always what you want when a link object is the visible result of a chunk.

These methods apply only inside 'knitr'. Call `print()` explicitly for the full cli description, or `as.character()` for the bare URL.

Value

A `knit_asis` object (via `knitr::asis_output()`).

Shape of the rendered link

A single link becomes `[Open in webR](url)`, or the Shinylive equivalent, and a project renders the same way. A directory or an exercise carries several named URLs, so it becomes a bulleted list with one titled link per entry.

See Also

livelink-knitr for the chunk hook and engine.

`format.livelink()` and `as.character.webr_link()` for other renderings.

`vignette("links-in-documents", package = "livelink")`.

`livelink-knitr` *Turn document chunks into shareable links*

Description

livelink plugs into 'knitr' two ways, and which you want depends on one question. **Should the code also run in your document?**

Usage

```
use_livelink_hook()

use_livelink_engine()
```

Details

Reach for either of these rather than expression input (`webr_repl_link({ ... })`) inside a knitted document. 'knitr' evaluates chunks through `evaluate::evaluate()`, which discards the source R kept, and comments live in that. Comments inside a `{ }` expression are therefore silently dropped from the link when the document renders, and no `keep.source` setting brings them back. Both the hook and the engine are handed the chunk's verbatim source, so nothing is lost.

Both are registered automatically when livelink is loaded, provided 'knitr' is installed. Call these yourself only if you have reset `knitr::knit_hooks` or `knitr::knit_engines`.

Value

Called for their side effect. The value is returned invisibly.

- `TRUE` if registration happened.
- `FALSE` if 'knitr' is not installed.

A chunk hook

Set on an ordinary `r` chunk. The chunk runs as usual (its output, plots and all, appear in the rendered page) and a link is added underneath. Use this for code you want your reader to *see the result of* and *also* be able to open and play with.

````
```{r}
#| livelink: true
#| autorun: true
# Load the data
data(mtcars)
plot(mtcars$mpg, mtcars$wt)
```
````

An engine

Written as ```` ```{livelink} ````. The chunk is displayed but **not** run, so only the link is produced. Use this for code your session cannot or should not execute, such as a Shiny app, something needing a package you have not installed, or anything slow.

````
```{livelink}
#| engine.target: shinylive-r
library(shiny)
shinyApp(fluidPage(), function(input, output) {})
```
````

There is deliberately no `{shinylive-r}` or `{shinylive-py}` engine. 'knitr' will not accept a chunk whose engine name contains a hyphen (its chunk syntax forbids it), and in Quarto such a cell is handed to the Shinylive extension rather than to 'knitr'. Name Shinylive through `engine.target` instead.

Chunk options

`livelink` Hook only. Use `true` for a webR link, or name the target directly with `"webr"`, `"shinylive-r"`, or `"shinylive-py"`.

`engine.target` Engine only. `"webr"` (default), `"shinylive-r"`, or `"shinylive-py"`.

`autorun` Logical. Run the code as soon as the link opens. webR only.

`panels` Character vector of webR panels, e.g. `c("editor", "plot")`.

`mode` Shinylive only. Display mode, `"editor"` (default) or `"app"`.

`filename` webR only. Name for the script file webR creates in the browser (default `"script.R"`). It must end in `.R` for `autorun` to work.

`link.text` Text for the hyperlink. Defaults to `"Open in webR"` or `"Open in Shinylive"`.

`link.only` Engine only. If `TRUE`, show the link without the source.

Setting options once

These are ordinary 'knitr' chunk options, so `opts_chunk` sets them for a whole document, and a single chunk opts out with `livelink: false`:

```
knitr::opts_chunk$set(livelink = TRUE, autorun = TRUE)
```

`echo` does not gate the link

It is natural to assume the code must be visible for a link to be made. It need not be. `echo` controls whether the **source is shown in your page**. The link is built from the chunk's source, which 'knitr' hands over either way. So `echo: false` gives a working link whose code the reader simply cannot see.

`eval: false` is the other half. The chunk is displayed but not run, which makes an `r` chunk behave rather like the engine.

See Also

`vignette("links-in-documents", package = "livelink")` for the whole picture.

`webr_repl_link()` for why a braced expression loses comments in a knitted document.

Examples

```
# Both are registered on load. Call directly only after resetting knitr's hooks.
use_livelink_hook()
use_livelink_engine()
```

`preview_shinylive_link` *Preview Shinylive link contents without writing files to disk*

Description

Decodes a Shinylive URL and returns information about the embedded files without actually saving them to disk.

Usage

```
preview_shinylive_link(url)
```

Arguments

`url`	Character string containing the Shinylive URL

Details

Nothing is written to disk, which makes this the safe way to look inside a link someone else sent you before extracting it.

Value

A `shinylive_preview` object, which is a list with these entries.

- `url`, the link that was read.
- `engine`, `"r"` or `"python"`, read off the link.
- `mode`, `"editor"` when the link opens the editor, `"app"` when it opens the running app on its own.
- `files_data`, a list with one entry per file, each holding `name`, `content`, and `type`.
- `total_files`, how many files the link carries.
- `total_size`, the size of those files added up, in bytes.
- `file_types`, the distinct file types found, such as `"text"` and `"binary"`.

See Also

`decode_shinylive_link()` to extract files to disk.

`shinylive_r_link()` and `shinylive_py_link()` to create links.

Examples

```
url <- as.character(shinylive_r_link("library(shiny)"))

# Inspect a link without writing anything to disk
preview <- preview_shinylive_link(url)

# Default print (no content)
print(preview)

# Show file contents
print(preview, show_content = TRUE)

# Show file contents with custom length limit
print(preview, show_content = TRUE, max_content_length = 200)

# Access the preview data
preview$files_data
preview$total_files
```

`preview_webr_link` *Preview webR REPL link contents without writing files to disk*

Description

Decodes a webR URL and returns information about the embedded files without actually saving them to disk.

Usage

```
preview_webr_link(url)
```

Arguments

url	Character string containing the webR URL

Details

Use print method options to control the display. The returned object's `print()` method accepts `show_content` and `max_content_length` to display file bodies.

Value

A `webr_preview` object, which is a list with these entries.

- `url`, the link that was read.
- `mode`, the panels the link asks for, or `NULL` for all of them.
- `version`, the webR version the link points at.
- `flags`, the encoding flags read off the link.
- `files_data`, a list with one entry per embedded file, each holding the file's `name`, its `path` inside webR, and its `text`.
- `total_files`, how many files the link carries.
- `total_size`, the size of those files in bytes.
- `autorun_files`, the names of the files that run as soon as the link opens.

Nothing is written to disk.

See Also

`decode_webr_link()` to extract the embedded files to disk once you are happy with the preview.

Examples

```
url <- as.character(webr_repl_link("plot(1:10)"))

# Inspect a link without writing anything to disk
preview <- preview_webr_link(url)

# Default print (no content)
print(preview)

# Show file contents
print(preview, show_content = TRUE)

# Show file contents with custom length limit
print(preview, show_content = TRUE, max_content_length = 200)

# Access the preview data
```

```
preview$files_data
preview$total_files
```

`print.shinylive_decoded` *Print method for shinylive_decoded objects*

Description

Displays the files recovered from a Shinylive link and where they were written.

Usage

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

Arguments

`x`	shinylive_decoded object
`...`	Additional arguments (ignored)

Value

The `shinylive_decoded` object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the link, the output folder, each file with its size, the total size, and the mode. See `decode_shinylive_link()` for the entries the object holds.

`print.shinylive_decoded_batch` *Print method for shinylive_decoded_batch objects*

Description

Displays a summary of the Shinylive links decoded in a batch.

Usage

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

Arguments

`x`	shinylive_decoded_batch object
`...`	Additional arguments (ignored)

Value

The `shinylive_decoded_batch` object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the base folder, the output folder and file count for each link that decoded, and the totals across all of them. See `decode_shinylive_link()` for the entries the object holds.

`print.shinylive_directory` *Print method for shinylive_directory objects*

Description

Displays every app sharelink generated from a source directory.

Usage

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

Arguments

`x`	shinylive_directory object
`...`	Additional arguments (ignored)

Value

The `shinylive_directory` object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the source folder, one sharelink per app, the engine, and the mode. See `shinylive_directory()` for the entries the object holds.

`print.shinylive_link` *Print method for shinylive_link objects*

Description

Displays the app sharelink and the files it carries.

Usage

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

Arguments

`x`	shinylive_link object
`...`	Additional arguments (ignored)

Value

The `shinylive_link` object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the URL, the files the app is made of, the engine, and the mode. See `shinylive_r_link()` for the entries the object holds.

`print.shinylive_preview`

Print method for shinylive_preview objects

Description

Displays what a Shinylive link contains without writing anything to disk.

Usage

```
## S3 method for class 'shinylive_preview'
print(x, show_content = FALSE, max_content_length = 500, ...)
```

Arguments

`x`	shinylive_preview object
`show_content`	Logical. Whether to print the contents of each file. Defaults to `FALSE`.
`max_content_length`	Maximum number of characters of content to show per file. Defaults to 500.
`...`	Additional arguments (ignored)

Value

The `shinylive_preview` object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the link, the file count and total size, the engine, the mode, and each file with its kind and size. See `preview_shinylive_link()` for the entries the object holds.

`print.shinylive_project`

Print method for shinylive_project objects

Description

Displays the project sharelink and the files it carries.

Usage

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

Arguments

x	shinylive_project object
...	Additional arguments (ignored)

Value

The shinylive_project object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the URL, the files the project carries, the engine, and the mode. See shinylive_project() for the entries the object holds.

---

print.webr_decoded *Print method for webr_decoded objects*

Description

Displays the files recovered from a webR link and where they were written.

Usage

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

Arguments

x	webr_decoded object
...	Additional arguments (ignored)

Value

The webr_decoded object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the link, the output folder, each file with its size, and any file that was skipped along with the reason. See decode_webr_link() for the entries the object holds.

---

print.webr_decoded_batch *Print method for webr_decoded_batch objects*

Description

Displays a summary of the webR links decoded in a batch.

Usage

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

Arguments

x	webr_decoded_batch object
...	Additional arguments (ignored)

Value

The webr_decoded_batch object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the base folder, each link that decoded with its file count and output folder, and a count of the links that failed. See decode_webr_link() for the entries the object holds.

---

print.webr_directory *Print method for webr_directory objects*

Description

Displays every sharelink generated from a source directory.

Usage

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

Arguments

x	webr_directory object
...	Additional arguments (ignored)

Value

The webr_directory object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the source folder, one sharelink per file, the interface, and the version. See webr_repl_directory() for the entries the object holds.

print.webr_exercise *Print method for webr_exercise objects*

Description

Displays the exercise sharelink and the matching solution sharelink.

Usage

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

Arguments

x	webr_exercise object
...	Additional arguments (ignored)

Value

The webr_exercise object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers both the exercise sharelink and the solution sharelink. See webr_repl_exercise() for the entries the object holds.

print.webr_link *Print method for webr_link objects*

Description

Displays the sharelink and its details in the console.

Usage

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

Arguments

x	webr_link object
...	Additional arguments (ignored)

Value

The webr_link object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the URL, the file and where it lands, the interface, the version, and autorun. See webr_repl_link() for the entries the object holds.

print.webr_preview *Print method for webr_preview objects*

Description

Displays what a webR link contains without writing anything to disk.

Usage

```
## S3 method for class 'webr_preview'
print(x, show_content = FALSE, max_content_length = 500, ...)
```

Arguments

x	webr_preview object
show_content	Logical. Whether to print the contents of each file. Defaults to FALSE.
max_content_length	Maximum number of characters of content to show per file. Defaults to 500.
...	Additional arguments (ignored)

Value

The webr_preview object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the link, the file count and total size, the interface, the version, the encoding flags, and each file with its size. See preview_webr_link() for the entries the object holds.

print.webr_project *Print method for webr_project objects*

Description

Displays the project sharelink and the files it carries.

Usage

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

Arguments

x	webr_project object
...	Additional arguments (ignored)

Value

The webr_project object it was handed, returned invisibly, so it can be passed straight on. Called for the summary it prints, which covers the URL, every file and where it lands, which files run on open, the interface, and the version. See webr_repl_project() for the entries the object holds.

---

repl_urls *Extract shareable URLs from livelink objects*

Description

Extracts the shareable URL(s) from any livelink object, covering both webR REPL and Shinylive results. Provides a clear way to get just the URLs for sharing or further work.

Usage

```
repl_urls(x, ...)

## Default S3 method:
repl_urls(x, ...)
```

Arguments

x	A livelink object. Supported classes are webr_link, webr_project, webr_exercise, webr_directory, webr_decoded, webr_decoded_batch, webr_preview, shinylive_link, shinylive_project, shinylive_directory, shinylive_decoded, shinylive_decoded_batch, and shinylive_preview.
...	Additional arguments passed to methods

Value

A character vector of URLs.

- Most objects give a single URL.
- An exercise gives two, named exercise and solution.
- A directory or a batch gives one URL for each file.

See Also

The as.character() methods (for example as.character.webr_link()), which repl_urls() calls to do the work.

Examples

```
# Single link
link <- webr_repl_link("plot(1:10)")
repl_urls(link)

# Exercise (returns named vector)
exercise <- webr_repl_exercise("# TODO", "plot(1:10)", "test")
repl_urls(exercise)

# Shinylive links work the same way
repl_urls(shinylive_r_link("library(shiny)"))

# Decoded files (returns the original URL)
decoded <- decode_webr_link(as.character(link))
repl_urls(decoded)
```

`set_webr_base_url` *Set global base URL for webR links*

Description

Overrides the base URL used when building webR REPL links, for instance to point at a webR site you host yourself or at a fixed webR version.

Usage

```
set_webr_base_url(base_url = NULL)
```

Arguments

`base_url`	Custom base URL to use for all webR links

Details

The value is stored in the `livelink.base_url` option and applies to webR links only. Shinylive links are unaffected. Once set, a custom base URL takes precedence over the `version` argument given to the functions that build links.

Value

Invisibly returns the value supplied to `base_url`.

- The custom URL when you set one.
- `NULL` when you reset to the default.

See Also

`webr_repl_link()` for single-script links that honor this option.

`webr_repl_project()` for multi-file projects that honor this option.

Examples

```
# Remember the current setting so it can be restored afterwards
old <- getOption("livelink.base_url")

# Set custom base URL
set_webr_base_url("https://my-custom-webr.com/")

# Reset to default (removes custom setting)
set_webr_base_url(NULL)

# Restore the previous setting
set_webr_base_url(old)
```

---

shinylive_directory *Create Shinylive sharelinks from a directory of Shiny apps*

Description

Batch processes directories containing Shiny applications to create individual Shinylive links. Each subdirectory is treated as a separate Shiny app project.

Usage

```
shinylive_directory(
  directory_path,
  engine,
  mode = "editor",
  header = TRUE,
  app_file = NULL,
  base_url = NULL
)
```

Arguments

directory_path	Character string specifying the path to the directory containing Shiny app directories
engine	Engine to use, either "r" for R Shiny or "python" for Python Shiny
mode	Shinylive display mode (default "editor"). "editor" shows an editable code panel beside the running app. "app" shows only the running app.
header	Logical, whether to show the Shinylive header bar. It applies only when mode = "app" and is ignored in the default "editor" mode. Defaults to TRUE.
app_file	Main app filename to look for (defaults to "app.R" for R and "app.py" for Python)
base_url	Custom Shinylive base URL. If NULL (default), links point at https://shinylive.io.

Details

Only text files with extensions .R, .py, .txt, .md, .csv, .json, .yaml, or .yml are embedded in a link. Other files (for example images or binary data) are skipped with a warning.

Value

A `shinylive_directory` object, which is a list with these entries.

- `urls`, one sharelink per app, as a named character vector whose names are the app subdirectory names. It is empty when no app was found.
- `engine`, the Shiny flavor the links run, `"r"` or `"python"`.
- `mode`, the Shinylive display mode, `"editor"` or `"app"`.
- `source_directory`, the directory that was read.

Use `repl_urls()` on the object to get the URLs on their own.

See Also

`shinylive_project()` for a single multi-file app.

`shinylive_r_link()` and `shinylive_py_link()` for single apps.

Examples

```
# Each app lives in its own subdirectory:
#   shiny_apps/
#     app1/app.R
#     app2/app.R
shiny_apps <- tempfile()
dir.create(file.path(shiny_apps, "app1"), recursive = TRUE)
dir.create(file.path(shiny_apps, "app2"), recursive = TRUE)
writeLines("library(shiny)", file.path(shiny_apps, "app1", "app.R"))
writeLines("library(shiny)", file.path(shiny_apps, "app2", "app.R"))

links <- shinylive_directory(shiny_apps, engine = "r", mode = "editor")
print(links)

# Extract just the URLs
repl_urls(links)
```

---

shinylive_project *Create a Shinylive sharelink for multi-file projects*

Description

Creates Shinylive projects for either R or Python from named lists or file path vectors.

Usage

```
shinylive_project(
  input,
  engine,
  mode = "editor",
  header = TRUE,
  base_url = NULL
)
```

Arguments

input	Input for multiple files. Can be: • Named list: `list("app.R" = code1, "utils.R" = code2)` • Vector of file paths: `c("app.R", "utils.R", "data.csv")`
engine	Engine to use, either "r" for R Shiny or "python" for Python Shiny
mode	Shinylive display mode (default `"editor"`). `"editor"` shows an editable code panel beside the running app. `"app"` shows only the running app.
header	Logical, whether to show the Shinylive header bar. It applies only when `mode = "app"` and is ignored in the default `"editor"` mode. Defaults to `TRUE`.
base_url	Custom Shinylive base URL. If NULL (default), links point at https://shinylive.io.

Value

A `shinylive_project` object, which is a list with these entries.

- `url`, the one sharelink that carries every file, as a single string.
- `files`, the names of the files carried in the link, as a character vector.
- `engine`, the Shiny flavor the link runs, `"r"` or `"python"`.
- `mode`, the Shinylive display mode, `"editor"` or `"app"`.

Use `as.character()` on the object to get the URL on its own.

See Also

`shinylive_r_link()` and `shinylive_py_link()` for single apps.

`decode_shinylive_link()` and `preview_shinylive_link()` to read a link back.

Examples

```
# Named list input
files <- list(
  "app.R" = "library(shiny)\nshinyApp(fluidPage(), function(i, o) {})",
  "utils.R" = "# Utility functions"
)
shinylive_project(files, engine = "r", mode = "editor")

# File paths input
```

```
project_dir <- tempfile()
dir.create(project_dir)
app <- file.path(project_dir, "app.R")
utils <- file.path(project_dir, "utils.R")
writeLines("library(shiny)", app)
writeLines("# utils", utils)
shinylive_project(c(app, utils), engine = "r")
```

`shinylive_py_link` *Create a Shinylive sharelink for Python Shiny apps*

Description

Generates a shareable URL for Python Shiny applications that can run in the browser using Shinylive.

Usage

```
shinylive_py_link(
  input = NULL,
  mode = "editor",
  header = TRUE,
  base_url = NULL
)
```

Arguments

`input`	App input. Can be: • Character string: Python code for the app • File path: Path to app.py file • Vector of file paths: Multiple files for the app • Named list: `list("app.py" = code1, "utils.py" = code2)` • NULL: Read from clipboard
`mode`	Shinylive display mode (default `"editor"`). `"editor"` shows an editable code panel beside the running app. `"app"` shows only the running app.
`header`	Logical, whether to show the Shinylive header bar. It applies only when `mode = "app"` and is ignored in the default `"editor"` mode. Defaults to `TRUE`.
`base_url`	Custom Shinylive base URL. If NULL (default), links point at https://shinylive.io.

Details

The whole app travels inside the URL, so opening the link runs it in the reader's browser with no server behind it. A single string becomes `app.R`. Pass a named list to ship several files.

Value

A `shinylive_link` object, which is a list with these entries.

- `url`, the sharelink itself, as a single string.
- `files`, the names of the files carried in the link, as a character vector.
- `engine`, the Shiny flavor the link runs, `"python"` here.
- `mode`, the Shinylive display mode, `"editor"` or `"app"`.

Use `as.character()` on the object to get the URL on its own.

See Also

`shinylive_project()` for multi-file apps.

`decode_shinylive_link()` and `preview_shinylive_link()` to read a link back.

Examples

```
# String input
app_code <- "
from shiny import App, render, ui
app_ui = ui.page_fluid(ui.h2('Hello World'))
def server(input, output, session): pass
app = App(app_ui, server)
"
shinylive_py_link(app_code)

# Multiple files as a named list
shinylive_py_link(list(
  "app.py" = app_code,
  "utils.py" = "def helper(): return 42"
))

# File path input
app_dir <- tempfile()
dir.create(app_dir)
app_path <- file.path(app_dir, "app.py")
writeLines("from shiny import App, ui", app_path)
shinylive_py_link(app_path)

# Read the app from the clipboard
if (interactive()) {
  shinylive_py_link()
}
```

`shinylive_r_link` *Create a Shinylive sharelink for R Shiny apps*

Description

Generates a shareable URL for R Shiny applications that can run in the browser using Shinylive.

Usage

```
shinylive_r_link(input = NULL, mode = "editor", header = TRUE, base_url = NULL)
```

Arguments

`input`	App input. Can be: • R expression (no quotes needed): `shinylive_r_link({ shinyApp(ui, server) })` • Character string: R code for the app • File path: Path to app.R file • Vector of file paths: Multiple files for the app • Named list: `list("app.R" = code1, "utils.R" = code2)` • NULL: Read from clipboard
`mode`	Shinylive display mode (default `"editor"`). `"editor"` shows an editable code panel beside the running app. `"app"` shows only the running app.
`header`	Logical, whether to show the Shinylive header bar. It applies only when `mode = "app"` and is ignored in the default `"editor"` mode. Defaults to `TRUE`.
`base_url`	Custom Shinylive base URL. If NULL (default), links point at https://shinylive.io.

Details

The whole app travels inside the URL, so opening the link runs it in the reader's browser with no server behind it. A single string becomes `app.py`. Pass a named list to ship several files.

Value

A `shinylive_link` object, which is a list with these entries.

- `url`, the sharelink itself, as a single string.
- `files`, the names of the files carried in the link, as a character vector.
- `engine`, the Shiny flavor the link runs, `"r"` here.
- `mode`, the Shinylive display mode, `"editor"` or `"app"`.

Use `as.character()` on the object to get the URL on its own.

Comments in expression input

Comments inside a `{ }` expression are recovered from the source R keeps, so they survive interactively but are dropped inside a knitted 'Quarto' or 'R Markdown' document. Pass a string or a file path, or use the `livelink` chunk engine, if you need them preserved. See `webr_repl_link()` for the details.

See Also

`shinylive_project()` for multi-file apps.

`decode_shinylive_link()` and `preview_shinylive_link()` to read a link back.

livelink-knitr to give a document chunk its own link.

`vignette("webr-and-shinylive", package = "livelink")` for the guide.

Examples

```
# Expression input (no quotes needed!)
shinylive_r_link({
  ui <- fluidPage(titlePanel("Hello World"))
  server <- function(input, output) {}
  shinyApp(ui, server)
})

# Multiple files as a named list
shinylive_r_link(list(
  "app.R" = "library(shiny)\nshinyApp(fluidPage(), function(i, o) {})",
  "utils.R" = "helper <- function() 42"
))

# File path input
app_dir <- tempfile()
dir.create(app_dir)
app_path <- file.path(app_dir, "app.R")
writeLines("library(shiny)", app_path)
shinylive_r_link(app_path)

# Read the app from the clipboard
if (interactive()) {
  shinylive_r_link()
}
```

---

webr_repl_directory *Create webR REPL sharelinks from a directory of R files*

Description

Batch processes all R files in a directory into webR sharelinks.

Usage

```
webr_repl_directory(
  directory_path,
  autorun = FALSE,
  single_link = FALSE,
  pattern = "\\.R$",
  base_path = "/home/web_user/",
  panels = NULL,
  version = "latest",
  base_url = NULL
)
```

Arguments

directory_path	Character string specifying the path to the directory containing R files
autorun	Logical. Whether to enable autorun for all generated links. Defaults to FALSE. With single_link = TRUE, this runs every R file in the bundle on arrival.
single_link	Logical. If FALSE (default), each matched file becomes its own link and the result is a webr_directory. If TRUE, all matched files are packed into one link and the result is a single webr_project.
pattern	Regular expression matched against file names in directory_path. Defaults to "\\.R$", i.e. files ending in .R.
base_path	Base directory path for files in webR. Defaults to "/home/web_user/".
panels	Character vector or string specifying which webR interface panels to show. The valid panels are "plot", "files", "terminal", and "editor". Can be c("plot", "files") or "plot-files". If NULL (default), shows all panels.
version	webR version to use ("latest" or specific version >= "v0.5.4")
base_url	webR application URL. If NULL, uses global option or builds from version

Details

By default each file becomes its own webR sharelink. With single_link = TRUE the whole directory is bundled into one link instead, exactly as webr_repl_project() would. Useful for converting collections of scripts, examples, or course materials.

Value

By default, a webr_directory object, which is a list with these entries.

- urls, the sharelinks, as a named character vector with one entry per matched file, named by file name.
- base_path, where the files are placed inside webR.
- mode, the panels the links ask for, or NULL for all of them.
- version, the webR version the links point at.
- source_directory, the directory the files were read from.

With `single_link = TRUE`, a `webr_project` object instead, which is a list with these entries.

- `url`, the one sharelink that carries every matched file, as a single string.
- `files`, the file contents that went into the link, as a named list keyed by file name.
- `base_path`, where the files are placed inside webR.
- `mode`, the panels the link asks for, or `NULL` for all of them.
- `version`, the webR version the link points at.
- `autorun_files`, the files that run as soon as the link opens.

Use `as.character()` on either object to get the URLs on their own.

See Also

`webr_repl_project()`, which bundles a named list or a vector of file paths into one link.

Examples

```
# A directory of R scripts
examples <- tempfile()
dir.create(examples)
writeLines("plot(1:10)", file.path(examples, "plot.R"))
writeLines("hist(rnorm(100))", file.path(examples, "hist.R"))

links <- webr_repl_directory(examples, autorun = TRUE)
print(links)

# Bundle the whole directory into one link instead
webr_repl_directory(examples, single_link = TRUE, panels = c("editor", "plot"))

# Show only the editor and terminal panels
webr_repl_directory(examples, panels = c("editor", "terminal"))

# Match a subset of files
webr_repl_directory(examples, pattern = "^plot")

# The URLs, named by file
repl_urls(links)
```

---

webr_repl_exercise *Create paired exercise and solution webR REPL links*

Description

Generates a pair of webR links for teaching. One link holds the exercise for the student and the other holds the solution.

Usage

```
webr_repl_exercise(
  exercise_text,
  solution_text,
  exercise_name,
  base_path = "/home/web_user/",
  version = "latest",
  base_url = NULL
)
```

Arguments

exercise_text	Character string containing the exercise code with placeholders or TODOs
solution_text	Character string containing the complete solution code
exercise_name	Base name for the exercise (will create "name_exercise.R" and "name_solution.R")
base_path	Base directory path for files (default: "/home/web_user/")
version	webR version to use ("latest" or specific version >= "v0.5.4")
base_url	webR application URL. If NULL, uses global option or builds from version

Details

The exercise link is built without autorun, so the student works through it, while the solution link is built with autorun enabled.

Value

A webr_exercise object, which is a list with these entries.

- exercise, the student's link, a webr_link object that does not autorun.
- solution, the answer link, a webr_link object that autoruns on opening.

Each of the two is itself a list holding the entries described in webr_repl_link(), so links$solution$url is the solution URL on its own.

See Also

webr_repl_link(), which this builds on.

vignette("teaching", package = "livelink") for using links in a course.

Examples

```
exercise_code <- "
# Exercise: Calculate mean of mtcars$mpg
# TODO: Complete the line below
mean_mpg <- 0
print(mean_mpg)
"
```

```
solution_code <- "
# Solution: Calculate mean of mtcars$mpg
mean_mpg <- mean(mtcars$mpg)
print(mean_mpg)
"

links <- webr_repl_exercise(exercise_code, solution_code, "basic_stats")
links$exercise
links$solution

# Custom path and version
webr_repl_exercise(exercise_code, solution_code, "stats",
                   base_path = "/exercises/", version = "v0.5.4")
```

`webr_repl_link` *Create a webR REPL sharelink from R code*

Description

Turns a single R script into a URL that runs it in the webR REPL.

Usage

```
webr_repl_link(
  input = NULL,
  filename = "script.R",
  path = NULL,
  autorun = FALSE,
  panels = NULL,
  version = "latest",
  base_url = NULL
)
```

Arguments

`input`	Code input. Can be: • R expression (no quotes needed): `webr_repl_link({ plot(1:10) })` • Character string: R code to execute • File path: Path to R file to read • NULL: Read from clipboard (requires clipr package)
`filename`	Name for the file (default: `"script.R"`)
`path`	Full path where the file will be placed in webR. If NULL (default), the file is placed at `"/home/web_user/{filename}"`.
`autorun`	Logical. Whether to auto-execute the code when link is opened (default: `FALSE`). Only R files (`.R`) can be auto-executed.

panels	Character vector or string specifying which webR interface panels to show. The panels are "plot", "files", "terminal", and "editor". Can be c("plot", "files") or "plot-files". If NULL (default), shows all panels.
version	webR version to use ("latest" or specific version >= "v0.5.4")
base_url	webR application URL. If NULL, uses global option or builds from version

Details

The code travels inside the URL, in the fragment after the #, which browsers keep local and never send to a server. Opening the link boots webR in the reader's own tab, so nothing is installed and nothing runs on your side.

input is deliberately permissive. It takes an expression in braces, a string, a path to a file, or the clipboard when nothing is passed at all. One script goes in one link. For several files, see webr_repl_project().

Value

A webr_link object, which is a list with these entries.

- url, the sharelink itself, as a single string.
- filename, the name the script is given inside webR.
- path, where that file is placed inside webR.
- mode, the panels the link asks for, or NULL for all of them.
- version, the webR version the link points at.
- autorun, TRUE when the code runs as soon as the link opens.

Use as.character() on the object to get the URL on its own.

Comments in expression input

Expression input recovers your code from the source R kept when it read the expression. R only keeps that source when keep.source is enabled, so comments survive in an interactive session but are dropped when the calling code is read without it. That is what happens inside a knitted 'Quarto' or 'R Markdown' document, because 'knitr' evaluates chunks through evaluate::evaluate(), which throws the source away. No keep.source setting recovers them there.

If you need comments preserved, pass the code as a string or a file path, or write it as a chunk in the document. See livelink-knitr and vignette("links-in-documents", package = "livelink").

See Also

webr_repl_project() for multi-file projects.

webr_repl_exercise() for exercise and solution pairs.

livelink-knitr to give a document chunk its own link.

vignette("getting-started", package = "livelink") for an introduction.

Examples

```
# Expression input (no quotes needed!)
webr_repl_link({
  plot(1:10)
  summary(mtcars)
})

# Traditional string input
webr_repl_link("plot(1:10)")

# Choose which panels the REPL shows
webr_repl_link({ hist(rnorm(100)) }, panels = c("plot", "editor"))

# Run the code as soon as the link opens
webr_repl_link("plot(1:10)", autorun = TRUE)

# File path input
script <- tempfile(fileext = ".R")
writeLines("plot(1:10)", script)
webr_repl_link(script)

# Read the code from the clipboard
if (interactive()) {
  webr_repl_link()
}
```

`webr_repl_project` *Create webR REPL sharelink for multiple files*

Description

Creates a webR sharelink for projects with multiple R files, data files, or other resources.

Usage

```
webr_repl_project(
  input,
  autorun_files = character(0),
  base_path = "/home/web_user/",
  panels = NULL,
  version = "latest",
  base_url = NULL
)
```

Arguments

`input`	Input for multiple files. Can be:

- Named list of braced expressions, so each file is written as R rather than as a string full of escaped newlines: `list("main.R" = { plot(1:10) }, "utils.R" = { f <- function() 42 })`
- Named list of strings: `list("main.R" = code1, "utils.R" = code2)`
- Vector of file paths: `c("main.R", "utils.R", "data.csv")`

The two list forms mix freely, which is what you want for a project holding both code and, say, a `README.md`.

`autorun_files` Character vector of filenames to auto-execute when project loads, or "all" to autorun all R files (default: none)

`base_path` Base directory path for all files (default: `"/home/web_user/"`)

`panels` Character vector or string specifying which webR interface panels to show. The panels are `"plot"`, `"files"`, `"terminal"`, and `"editor"`. Can be `c("plot", "files")` or `"plot-files"`. If NULL (default), shows all panels.

`version` webR version to use (`"latest"` or specific version >= "v0.5.4")

`base_url` webR application URL. If NULL, uses global option or builds from version

Details

Every file is placed under `base_path` inside webR, so a `source("utils.R")` in one file finds its sibling. Names are used verbatim and may carry a subdirectory, as in `"R/helpers.R"`.

Value

A `webr_project` object, which is a list with these entries.

- `url`, the sharelink itself, as a single string.
- `files`, the names of the files carried in the link.
- `base_path`, the folder the files are placed in inside webR.
- `mode`, the panels the link asks for, or `NULL` for all of them.
- `version`, the webR version the link points at.
- `autorun_files`, the names you asked to run on opening.

Writing a project as code

A file's contents can be given as a `{ ... }` block instead of a string, which spares you escaping every newline and quote:

```
webr_repl_project(list(
  "main.R"    = { source("utils.R"); summarise(mtcars) },
  "utils.R"   = { summarise <- function(d) summary(d) },
  "README.md" = "# Analysis"
))
```

The blocks are **never evaluated**. They are source to ship, not code to run, so an assignment inside one leaves nothing behind in your session.

Two things to know. Comments inside `{ }` survive in an interactive session but not in a knitted document (see `webr_repl_link()`). And a `library()` call inside a block is visible to `R CMD check`, which will report the package as an undeclared dependency of *yours*. In a vignette or an example, use a string for code that loads packages.

See Also

`webr_repl_link()` for the single-file case.

Examples

```
# Each file written as R, rather than as an escaped string
webr_repl_project(list(
  "main.R"  = { source("utils.R"); summarise(mtcars) },
  "utils.R" = { summarise <- function(d) summary(d) }
))

# Strings still work, and the two forms mix
files <- list(
  "main.R" = "source('utils.R')\nresult <- analyze_data(mtcars)",
  "utils.R" = "analyze_data <- function(data) { summary(data) }",
  "README.md" = "# My Analysis\nThis project analyzes the mtcars dataset."
)
webr_repl_project(files, autorun_files = "main.R")

# Autorun every R file in the project
webr_repl_project(files, autorun_files = "all")

# File paths input
project_dir <- tempfile()
dir.create(project_dir)
main <- file.path(project_dir, "main.R")
utils <- file.path(project_dir, "utils.R")
writeLines("source('utils.R')", main)
writeLines("# utils", utils)
webr_repl_project(c(main, utils))
```

Index