Package: multiverse.internals (via r-universe)

January 9, 2026

Title Internal Infrastructure for R-multiverse

Description R-multiverse requires this internal infrastructure package to automate contribution reviews and populate universes.

Version 1.1.10

License MIT + file LICENSE

URL https://r-multiverse.org/multiverse.internals/, https://github.com/r-multiverse/multiverse.internals

BugReports https://github.com/r-multiverse/multiverse.internals/issues

Depends R (>= 4.5.0)

Imports cli, desc, gh, igraph, jsonlite, nanonext, pkgsearch, R.utils, rversions, stats, tools, utils, vctrs, yaml

Suggests gert, testthat (>= 3.0.0)

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/pak/sysreqs git libglpk-dev libxml2-dev libssl-dev

Repository https://test.r-universe.dev

Date/Publication 2026-01-06 21:31:59 UTC

RemoteUrl https://github.com/r-multiverse/multiverse.internals

RemoteRef 1.1.10

RemoteSha 0b6af6eb06221df195c2735dedf83052360a1bc9

Contents

`filter_meta` *Filter* PACKAGES *and* PACKAGES.gz *metadata files.*

Description

Filter the PACKAGES and PACKAGES.gz files to only list certain packages.

Usage

```
filter_meta(path_meta, path_staging)
```

Arguments

`path_meta`	Directory path where PACKAGES and PACKAGES.gz files reside locally.
`path_staging`	Path to a GitHub clone of the Staging universe.

See Also

Other staging: `freeze_dependencies()`, `rclone_includes()`, `stage_candidates()`

Examples

```
## Not run:
path_meta <- tempfile()
dir.create(path_meta)
mock <- system.file(
  file.path("mock", "meta"),
  package = "multiverse.internals",
  mustWork = TRUE
)
file.copy(mock, path_meta, recursive = TRUE)
path_staging <- tempfile()
url_staging <- "https://github.com/r-multiverse/staging"
gert::git_clone(url = url_staging, path = path_staging)
filter_meta(path_meta, path_staging)

## End(Not run)
```

---

freeze_dependencies *Freeze dependencies*

Description

Freeze the targeted versions of base R and CRAN packages.

Usage

```
freeze_dependencies(path_staging, path_community)
```

Arguments

path_staging	Character string, directory path to the source files of the Staging universe.
path_community	Character string, local directory path to the clone of the Community universe GitHub repository.

Details

freeze_dependencies() runs during the month-long dependency freeze phase of Staging in which base R and CRAN packages are locked in the Staging universe until after the next Production snapshot. This establishes checks in the Staging universe using the exact set of dependencies that will be used in the candidate freeze (see stage_candidates()).

freeze_dependencies() copies the Community repository packages.json into the Staging repository to reset the Staging process. It also writes a config.json file with the date of the targeted CRAN snapshot.

Value

NULL (invisibly)

See Also

Other staging: `filter_meta()`, `rclone_includes()`, `stage_candidates()`

Examples

```
## Not run:
url_staging = "https://github.com/r-multiverse/staging"
url_community = "https://github.com/r-multiverse/community"
path_staging <- tempfile()
path_community <- tempfile()
gert::git_clone(url = url_staging, path = path_staging)
gert::git_clone(url = url_community, path = path_community)
freeze_dependencies(
  path_staging = path_staging,
  path_community = path_community
)

## End(Not run)
```

---

interpret_status *Interpret the status of a package*

Description

Summarize the status of a package in human-readable text.

Usage

```
interpret_status(package, status)
```

Arguments

package	Character string, name of the package.
status	A list with one status entry per package. Obtained by reading the results of `record_status()`.

Value

A character string summarizing the status of a package in prose.

See Also

Other status: `update_status()`

issues_advisories *Report package advisories*

Description

Report packages whose current versions have advisories in the R Consortium Advisory Database: https://github.com/RConsortium/r-advisory-database

Usage

```
issues_advisories(meta = meta_packages())
```

Arguments

meta	Package metadata from meta_packages().

Value

A data frame with one row for each problematic package and columns with details.

See Also

Other issues: issues_dependencies(), issues_licenses(), issues_r_cmd_check(), issues_remotes(), issues_synchronization(), issues_version_conflicts(), issues_versions()

Examples

```
## Not run:
issues_advisories()

## End(Not run)
```

issues_dependencies *Report package dependency issues*

Description

Flag packages which have issues in their strong dependencies (Imports:, Depends:, and LinkingTo: in the DESCRIPTION.) These include indirect/upstream dependencies, as well, not just the explicit mentions in the DESCRIPTION file.

Usage

```
issues_dependencies(packages, meta = meta_packages(), verbose = FALSE)
```

Arguments

packages	Character vector of names of packages with other issues.
meta	Package metadata from meta_packages().
verbose	TRUE to print progress while checking dependency status, FALSE otherwise.

Value

A data frame with one row for each package impacted by upstream dependencies. Each element of the dependencies column is a nested list describing the problems upstream.

To illustrate the structure of this list, suppose Package tarchetypes depends on package targets, and packages jagstargets and stantargets depend on tarchetypes. In addition, package targets has a problem in R CMD check which might cause problems in tarchetypes and packages downstream.

status_dependencies() represents this information in the following list:

```
list(
  jagstargets = list(targets = "tarchetypes"),
  tarchetypes = list(targets = character(0)),
  stantargets = list(targets = "tarchetypes")
)
```

In general, the returned list is of the form:

```
list(
  impacted_reverse_dependency = list(
    upstream_culprit = c("direct_dependency_1", "direct_dependency_2")
  )
)
```

where upstream_culprit causes problems in impacted_reverse_dependency through direct dependencies direct_dependency_1 and direct_dependency_2.

See Also

Other issues: issues_advisories(), issues_licenses(), issues_r_cmd_check(), issues_remotes(), issues_synchronization(), issues_version_conflicts(), issues_versions()

Examples

```
## Not run:
issues_dependencies(packages = "targets")

## End(Not run)
```

issues_licenses *Report package license issues*

Description

Report packages without standard free and open-source licenses.

Usage

```
issues_licenses(meta = meta_packages())
```

Arguments

meta	Package metadata from meta_packages().

See Also

Other issues: issues_advisories(), issues_dependencies(), issues_r_cmd_check(), issues_remotes(), issues_synchronization(), issues_version_conflicts(), issues_versions()

Examples

```
## Not run:
issues_licenses()

## End(Not run)
```

issues_remotes *Report packages with* Remotes: *fields.*

Description

Report packages with Remotes: fields in the DESCRIPTION file.

Usage

```
issues_remotes(meta = meta_packages())
```

Arguments

meta	Package metadata from meta_packages().

See Also

Other issues: issues_advisories(), issues_dependencies(), issues_licenses(), issues_r_cmd_check(), issues_synchronization(), issues_version_conflicts(), issues_versions()

Examples

```
## Not run:
issues_remotes()

## End(Not run)
```

`issues_r_cmd_check` *R-universe package* `R CMD check` *issues.*

Description

Report issues from `R CMD check` on R-universe.

Usage

```
issues_r_cmd_check(meta = meta_packages())
```

Arguments

`meta`	Package metadata from `meta_packages()`.

Details

`issues_r_cmd_check()` reads output from the R-universe `R CMD check` results API to scan all R-multiverse packages for status that may have happened during building and testing.

Value

A data frame with one row for each problematic package and columns for the package names and `R CMD check` issues.

Package status

Functions like `issues_versions()` and `issues_r_cmd_check()` perform health checks for all packages in R-multiverse. For a complete list of checks, see the `issues_*()` functions listed at https://r-multiverse.org/multiverse.internals/reference/index.html. `record_versions()` updates the version number history of releases in R-multiverse, and `record_status()` gathers together all the status about R-multiverse packages.

See Also

Other issues: `issues_advisories()`, `issues_dependencies()`, `issues_licenses()`, `issues_remotes()`, `issues_synchronization()`, `issues_version_conflicts()`, `issues_versions()`

Examples

```
## Not run:
issues_r_cmd_check()

## End(Not run)
```

issues_synchronization *Report package check synchronization issues.*

Description

Ensure the reported R-universe checks are synchronized. Report the packages whose checks have not been synchronized.

Usage

```
issues_synchronization(meta = meta_packages(), verbose = FALSE)
```

Arguments

meta	Package metadata from meta_packages().
verbose	TRUE to print progress messages, FALSE otherwise.

Details

R-universe automatically rechecks downstream packages if an upstream dependency increments its version number. R-multiverse needs to wait for these downstream checks to finish before it makes decisions about accepting packages into Production. issues_synchronization() scrapes the GitHub Actions API to find out if any R-universe checks are still running for a package. In addition, to give rechecks enough time to post on GitHub Actions, it flags packages published within the last 5 minutes.

Value

A tibble with one row per package and the following columns:

- package: Name of the package.
- synchronization: Synchronization status: "success" if the checks are synchronized, "incomplete" if checks are still running on R-universe GitHub Actions, and "recent" if the package was last published so recently that downstream checks may not have started yet.

See Also

Other issues: issues_advisories(), issues_dependencies(), issues_licenses(), issues_r_cmd_check(), issues_remotes(), issues_version_conflicts(), issues_versions()

Examples

```
## Not run:
meta <- meta_packages(repo = "https://wlandau.r-universe.dev")
issues_synchronization(meta)

## End(Not run)
```

`issues_versions` *Package version issues.*

Description

Check package version number history for compliance.

Usage

```
issues_versions(versions)
```

Arguments

`versions`	Character of length 1, file path to a JSON manifest tracking the history of released versions of packages.

Details

This function checks the version number history of packages in R-multiverse and reports any packages with issues. The current released version of a given package must be unique, and it must be greater than all the versions of all the previous package releases.

Value

A data frame with one row for each problematic package and columns with the package names and version issues.

See Also

Other issues: `issues_advisories()`, `issues_dependencies()`, `issues_licenses()`, `issues_r_cmd_check()`, `issues_remotes()`, `issues_synchronization()`, `issues_version_conflicts()`

Examples

```
lines <- c(
  "[",
  " {",
  " \"package\": \"package_unmodified\",",
  " \"version_current\": \"1.0.0\",",
  " \"hash_current\": \"hash_1.0.0\",",
  " \"version_highest\": \"1.0.0\",",
  " \"hash_highest\": \"hash_1.0.0\"",
  " },",
  " {",
  " \"package\": \"version_decremented\",",
  " \"version_current\": \"0.0.1\",",
  " \"hash_current\": \"hash_0.0.1\",",
  " \"version_highest\": \"1.0.0\",",
  " \"hash_highest\": \"hash_1.0.0\"",
```

```
  " },",
  " {",
  " \"package\": \"version_incremented\",",
  " \"version_current\": \"2.0.0\",",
  " \"hash_current\": \"hash_2.0.0\",",
  " \"version_highest\": \"2.0.0\",",
  " \"hash_highest\": \"hash_2.0.0\"",
  " },",
  " {",
  " \"package\": \"version_unmodified\",",
  " \"version_current\": \"1.0.0\",",
  " \"hash_current\": \"hash_1.0.0-modified\",",
  " \"version_highest\": \"1.0.0\",",
  " \"hash_highest\": \"hash_1.0.0\"",
  " }",
  "]"
)
versions <- tempfile()
writeLines(lines, versions)
out <- issues_versions(versions)
str(out)
```

`issues_version_conflicts` *Report packages with version conflicts in another repository.*

Description

Report packages with higher versions in different repositories. A higher version in a different repository could cause that repository to override R-multiverse in `install.packages()`.

Usage

```
issues_version_conflicts(meta = meta_packages(), repo = "cran")
```

Arguments

`meta`	Package metadata from `meta_packages()`.
`repo`	Character string naming the repository to compare versions.

Value

A data frame with one row for each problematic package and columns with details.

See Also

Other issues: `issues_advisories()`, `issues_dependencies()`, `issues_licenses()`, `issues_r_cmd_check()`, `issues_remotes()`, `issues_synchronization()`, `issues_versions()`

Examples

```
## Not run:
issues_version_conflicts()

## End(Not run)
```

`meta_packages` *List package metadata*

Description

List package metadata in an R universe.

Usage

```
meta_packages(repo = "https://community.r-multiverse.org")
```

Arguments

`repo`	URL of the repository to query.

Value

A data frame with one row per package and columns with package metadata.

See Also

Other meta: `meta_snapshot()`

Examples

```
## Not run:
meta_packages()

## End(Not run)
```

meta_snapshot *Snapshot metadata*

Description

Show the metadata for the current targeted Production snapshot.

Usage

```
meta_snapshot(today = Sys.Date())
```

Arguments

today	An object that as.Date() can convert to class "Date".

Value

A data frame with one row and columns with metadata about the targeted snapshot.

See Also

Other meta: meta_packages()

Examples

```
meta_snapshot(today = Sys.Date())
```

rclone_includes *Write Rclone includes.*

Description

Write text files to pass to the --include-from flag in Rclone when uploading snapshots.

Usage

```
rclone_includes(path_staging)
```

Arguments

path_staging	Character string, directory path to the source files of the Staging universe.

See Also

Other staging: filter_meta(), freeze_dependencies(), stage_candidates()

Examples

```
## Not run:
url_staging = "https://github.com/r-multiverse/staging"
path_staging <- tempfile()
path_community <- tempfile()
gert::git_clone(url = url_staging, path = path_staging)
stage_candidates(path_staging = path_staging)
rclone_includes(path_staging)

## End(Not run)
```

`record_nonstandard_licenses` *Record nonstandard licenses*

Description

R-multiverse packages must have valid free and open-source (FOSS) licenses to protect the intellectual property rights of the package owners (c.f. https://en.wikipedia.org/wiki/Free_and_open-source_software). `record_nonstandard_licenses()` records packages with nonstandard licenses.

Usage

```
record_nonstandard_licenses(
  path_status = "status.json",
  path_nonstandard_licenses = "nonstandard_licenses.json"
)
```

Arguments

`path_status`	Character string, local path to the `status.json` file of the repository.
`path_nonstandard_licenses`	Character string, output path to write JSON data with the names and licenses of packages with non-standard licenses.

Value

`NULL` (invisibly). Called for its side effects.

record_status *Record package status.*

Description

Record R-multiverse package status in package-specific JSON files.

Usage

```
record_status(
  repo = "https://community.r-multiverse.org",
  versions = "versions.json",
  output = "status.json",
  staging = NULL,
  mock = NULL,
  verbose = FALSE
)
```

Arguments

repo	URL of the repository to query.
versions	Character of length 1, file path to a JSON manifest tracking the history of released versions of packages.
output	Character string, file path to the JSON file to record new package status. Each call to `record_status()` overwrites the contents of the file.
staging	Character string, file path to the JSON manifest of package versions in the Staging universe. Used to identify staged packages. Set to `NULL` (default) to ignore when processing the Community universe.
mock	For testing purposes only, a named list of data frames for inputs to various intermediate functions.
verbose	`TRUE` to print progress while checking dependency status, `FALSE` otherwise.

Value

`NULL` (invisibly).

Package status

Functions like `issues_versions()` and `issues_r_cmd_check()` perform health checks for all packages in R-multiverse. For a complete list of checks, see the `issues_*()` functions listed at https://r-multiverse.org/multiverse.internals/reference/index.html. `record_versions()` updates the version number history of releases in R-multiverse, and `record_status()` gathers together all the status about R-multiverse packages.

Examples

```
## Not run:
output <- tempfile()
versions <- tempfile()
repo <- "https://community.r-multiverse.org"
record_versions(versions = versions, repo = repo)
record_status(repo = repo, versions = versions, output = output)
writeLines(readLines(output))

## End(Not run)
```

`record_versions` *Record the manifest of package versions.*

Description

Record the manifest of versions of packages and their hashes.

Usage

```
record_versions(
  versions = "versions.json",
  repo = "https://community.r-multiverse.org",
  current = multiverse.internals::get_current_versions(repo = repo)
)
```

Arguments

`versions`	Character of length 1, file path to a JSON manifest tracking the history of released versions of packages.
`repo`	URL of the repository to query.
`current`	A data frame of current versions and hashes of packages in `repo`. This argument is exposed for testing only.

Details

This function tracks a manifest containing the current version, the current hash, the highest version ever released, and the hash of the highest version ever released. `issues_versions()` uses this information to determine whether the package complies with best practices for version numbers.

Value

`NULL` (invisibly). Writes version information to a JSON file.

Package status

Functions like issues_versions() and issues_r_cmd_check() perform health checks for all packages in R-multiverse. For a complete list of checks, see the issues_*() functions listed at https://r-multiverse.org/multiverse.internals/reference/index.html. record_versions() updates the version number history of releases in R-multiverse, and record_status() gathers together all the status about R-multiverse packages.

Examples

```
## Not run:
output <- tempfile()
versions <- tempfile()
# First snapshot:
record_versions(
  versions = versions,
  repo = repo
)
readLines(versions)
# In subsequent snapshots, we have historical information about versions.
record_versions(
  versions = versions,
  repo = repo
)
readLines(versions)

## End(Not run)
```

---

review_license *Review a package license.*

Description

Review a package license string from the DESCRIPTION file to make sure the package has a valid free open-source (FOSS) license. This is vital to make sure R-multiverse has legal permission to distribute the package source code.

Usage

```
review_license(license)
```

Arguments

license	Character string with the "License:" field of the DESCRIPTION file of the package in question.

Value

Invisibly returns TRUE if the package has a valid free open-source (FOSS) license according to tools::analyze_license(). FALSE otherwise. review_license() also prints an R console message to the communicate the result. Licenses for which review_license() returns FALSE are prohibited in R-multiverse.

See Also

Other Manual package reviews: review_package()

Examples

```
review_license("MIT + file LICENSE")
review_license("just file LICENSE")
```

---

review_package *Review a package.*

Description

Review a package for registration in R-multiverse.

Usage

```
review_package(name, url, advisories = NULL)
```

Arguments

name	Character string, name of the package to check.
url	Either a character string with the package URL or a custom JSON string with a package entry.
advisories	Character vector of names of packages with advisories in the R Consortium Advisory Database. If NULL, then review_package_text() downloads the advisory database and checks if the package has a vulnerability listed there. The advisory database is cached internally for performance.

Details

review_package() runs all the checks from https://r-multiverse.org/review.html#automatic-acceptance that can be done using the package name and source code repository URL.

Value

Invisibly returns TRUE if there is a problem with the package entry, otherwise FALSE if there are no issues. In either case, review_package() prints an R console message with the result.

For security reasons, review_package() might only print the first finding it encounters. If that happens, there will be an informative note at the end of the console message, and compliance with R-multiverse policies will need to be checked manually. In particular, please use review_license() to check the "License:" field in the package DESCRIPTION file.

See Also

Other Manual package reviews: review_license()

Examples

```
review_package(
  name = "webchem",
  url = "https://github.com/ropensci/webchem"
)
review_package(
  name = "polars",
  url = "https://github.com/pola-rs/r-polars"
)
```

---

review_pull_request *Review an R-multiverse contribution pull request.*

Description

Review a pull request to add packages to R-multiverse.

Usage

```
review_pull_request(
  owner = "r-multiverse",
  repo = "contributions",
  number,
  advisories = NULL,
  organizations = NULL
)
```

Arguments

owner	Character of length 1, name of the package repository owner.
repo	URL of the repository to query.
number	Positive integer of length 1, index of the pull request in the repo.
advisories	Character vector of names of packages with advisories in the R Consortium Advisory Database. If NULL, the function reads the database.
organizations	Character vector of names of GitHub organizations. Pull requests from authors who are not members of at least one of these organizations will be flagged for manual review. If NULL, the function reads the list of trusted organizations.

Value

NULL (invisibly).

See Also

Other Automated package reviews: `review_pull_requests()`

`review_pull_requests` *Review R-multiverse contribution pull requests.*

Description

Review pull requests which add packages to `packages.json`.

Usage

```
review_pull_requests(owner = "r-multiverse", repo = "contributions")
```

Arguments

`owner`	Character of length 1, name of the package repository owner.
`repo`	URL of the repository to query.

Value

`NULL` (invisibly).

See Also

Other Automated package reviews: `review_pull_request()`

`stage_candidates` *Stage release candidates*

Description

Stage release candidates for the targeted Production snapshot.

Usage

```
stage_candidates(path_staging)
```

Arguments

`path_staging`	Character string, directory path to the source files of the Staging universe.

Details

stage_candidates() implements the candidate freeze during the month-long period prior to the Production snapshot. Packages that pass R-multiverse checks are frozen (not allowed to update further) and staged for Production. Packages with at least one failing not staged for Production, and maintainers can update them with new source code releases.

stage_candidates() writes packages.json to control contents of the Staging universe.

Value

NULL (invisibly)

See Also

Other staging: filter_meta(), freeze_dependencies(), rclone_includes()

Examples

```
## Not run:
url_staging <- "https://github.com/r-multiverse/staging"
path_staging <- tempfile()
gert::git_clone(url = url_staging, path = path_staging)
stage_candidates(path_staging = path_staging)

## End(Not run)
```

---

update_status *Update the package status repository*

Description

Update the repository which reports the status on individual packages.

Usage

```
update_status(path_status, path_staging, path_community)
```

Arguments

path_status	Character string, directory path to the source files of the package status repository.
path_staging	Character string, local directory path to the clone of the Staging universe GitHub repository.
path_community	Character string, local directory path to the clone of the Community universe GitHub repository.

See Also

Other status: interpret_status()

Examples

```
## Not run:
url_staging <- "https://github.com/r-multiverse/staging"
url_community <- "https://github.com/r-multiverse/community"
url_status <- "https://github.com/r-multiverse/status"
path_status <- tempfile()
path_staging <- tempfile()
path_community <- tempfile()
gert::git_clone(url = url_status, path = path_status)
gert::git_clone(url = url_staging, path = path_staging)
gert::git_clone(url = url_community, path = path_community)
update_status(
  path_status = path_status,
  path_staging = path_staging,
  path_community = path_community
)
writeLines(
  readLines(
    file.path(path_status, "community", "multiverse.internals.html")
  )
)
writeLines(
  readLines(
    file.path(path_status, "community", "multiverse.internals.xml")
  )
)

## End(Not run)
```

---

update_topics *Update topics*

Description

Update the list of packages for each R-multiverse topic.

Usage

```
update_topics(path, repo = "https://community.r-multiverse.org", mock = NULL)
```

Arguments

path	Character string, local file path to the topics repository source code.
repo	Character string, URL of the Community universe.
mock	List of named objects for testing purposes only.

Value

NULL (invisibly). Called for its side effects.

Examples

```
## Not run:
path <- tempfile()
gert::git_clone("https://github.com/r-multiverse/topics", path = path)
update_topics(path = path, repo = "https://community.r-multiverse.org")

## End(Not run)
```

Index