

Package ‘RDesk’

September 3, 2026

Title Native Desktop Application Framework

Version 1.0.7

Description Build native desktop applications for Windows (with future support planned for macOS and Linux) using 'R' and embedded 'webviews'. Provides a robust 'R6'-based event loop, asynchronous background task management via 'mirai' and 'callr', and a native message bridge for seamless 'R'-to-user-interface communication without listening ports or network overhead. Allows 'R' developers to create professional, standalone desktop tools with modern web-based user interfaces while maintaining a pure 'R' backend.

OS_type windows

License MIT + file LICENSE

Encoding UTF-8

Language en-US

URL <https://github.com/Janakiraman-311/RDesk>,
<https://janakiraman-311.github.io/RDesk/>

BugReports <https://github.com/Janakiraman-311/RDesk/issues>

SystemRequirements Windows 10 or later, Microsoft 'WebView2' Runtime
(<https://developer.microsoft.com/microsoft-edge/webview2/>)

RoxygenNote 7.3.3

Imports R6, jsonlite, digest, processx, callr, mirai (>= 1.0.0),
base64enc, zip, stats, utils, parallel

Suggests testthat (>= 3.0.0), withr, knitr, rmarkdown, ggplot2, dplyr,
broom, devtools, pkgbuild, rstudioapi, renv

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Janakiraman G [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-6050-2318>>),
Serge Zaitsev [cph] (Author of webview library (src/webview/webview.h)),

Steffen André Langnes [cph] (Contributor to webview library
(src/webview/webview.h)),
Björn Hoehrmann [cph] (Copyright holder credited in nlohmann/json),
Florian Loitsch [cph] (Copyright holder credited in nlohmann/json),
Niels Lohmann [cph] (Author of nlohmann/json
(inst/include/nlohmann/json.hpp)),
Evan Nemerson [cph] (Copyright holder credited in nlohmann/json),
The Abseil Authors [cph] (Copyright holder credited in nlohmann/json),
Microsoft Corporation [cph] (WebView2 SDK (src/webview2_sdk/))

Maintainer Janakiraman G <janakiraman.bt@gmail.com>

Repository CRAN

Date/Publication 2026-09-03 14:00:02 UTC

Contents

App	2
async	11
async_progress	12
build_app	13
RDeskStorage	15
rdesk_async	16
rdesk_auto_update	17
rdesk_cancel_job	18
rdesk_create_app	19
rdesk_df_to_list	20
rdesk_error_plot	21
rdesk_is_bundle	21
rdesk_jobs_list	22
rdesk_jobs_pending	22
rdesk_message	23
rdesk_parse_message	24
rdesk_plot_to_base64	25
rdesk_service	26
rdesk_storage	26
rdesk_watch	28

Index	29
--------------	-----------

App	Create and launch a native desktop application window from R.
-----	---

Description

Provides bidirectional native pipe communication between R and the UI.

Public fields

prefs Preferences storage manager (roaming user data)
recent Recent items storage manager (local user data)
shared Shared data storage manager (machine-wide data)

Methods**Public methods:**

- App\$new()
- App\$on_ready()
- App\$on_close()
- App\$check_update()
- App\$register_hotkey()
- App\$set_tray_menu()
- App\$clipboard_write()
- App\$clipboard_read()
- App\$on_message()
- App\$send()
- App\$load_ui()
- App\$set_size()
- App\$set_position()
- App\$set_title()
- App\$minimize()
- App\$maximize()
- App\$restore()
- App\$fullscreen()
- App\$always_on_top()
- App\$set_menu()
- App\$dialog_open()
- App\$dialog_save()
- App\$dialog_folder()
- App\$message_box()
- App\$dialog_color()
- App\$notify()
- App\$loading_start()
- App\$loading_progress()
- App\$loading_done()
- App\$toast()
- App\$set_tray()
- App\$remove_tray()
- App\$service()
- App\$watch()

- `App$quit()`
- `App$get_dir()`
- `App$run()`
- `App$clone()`

Method `new()`: Create a new RDesk application

Usage:

```
App$new(title, width = 1200L, height = 800L, www = NULL, icon = NULL)
```

Arguments:

`title` Window title string

`width` Window width in pixels (default 1200)

`height` Window height in pixels (default 800)

`www` Directory containing HTML/CSS/JS assets (default: built-in template)

`icon` Path to window icon file

Returns: A new App instance

Method `on_ready()`: Register a callback to fire when the window is ready

Usage:

```
App$on_ready(fn)
```

Arguments:

`fn` A zero-argument function called after the server starts and window opens

Returns: The App instance (invisible)

Method `on_close()`: Register a callback to fire when the user attempts to close the window

Usage:

```
App$on_close(fn)
```

Arguments:

`fn` A zero-argument function. Should return TRUE to allow closing, FALSE to cancel.

Returns: The App instance (invisible)

Method `check_update()`: Check for application updates from a remote URL

Usage:

```
App$check_update(version_url, current_version = NULL)
```

Arguments:

`version_url` URL to a JSON metadata file (e.g. `{"version": "1.1.0", "url": "http://..."}`)

`current_version` Optional version string to compare against. Defaults to app description version.

Returns: A list with update status and metadata

Method `register_hotkey()`: Register a global keyboard shortcut (hotkey)

Usage:

```
App$register_hotkey(keys, fn)
```

Arguments:

keys Character string representing the key combination (e.g., "Ctrl+Shift+A")

fn A zero-argument function to be called when the hotkey is pressed

Returns: The App instance (invisible)

Method `set_tray_menu()`: Set the native system tray menu

Usage:

`App$set_tray_menu(items)`

Arguments:

items A named list of lists defining the menu structure

Returns: The App instance (invisible)

Method `clipboard_write()`: Write text to the system clipboard

Usage:

`App$clipboard_write(text)`

Arguments:

text Character string to copy

Returns: The App instance (invisible)

Method `clipboard_read()`: Read text from the system clipboard

Usage:

`App$clipboard_read()`

Returns: Character string from clipboard or NULL

Method `on_message()`: Register a handler for a UI -> R message type

Usage:

`App$on_message(type, fn)`

Arguments:

type Unique message identifier string

fn A function(payload) called when this message type arrives

Returns: The App instance (invisible)

Method `send()`: Send a message from R to the UI

Usage:

`App$send(type, payload = list())`

Arguments:

type Character string message type (received by `rdesk.on()` in JS)

payload A list or data.frame to serialise as JSON payload

Returns: The App instance (invisible)

Method `load_ui()`: Load an HTML file into the window

Usage:

```
App$load_ui(path = "index.html")
```

Arguments:

path Path relative to the www directory (e.g. "index.html")

Returns: The App instance (invisible)

Method `set_size()`: Set the window size dynamically

Usage:

```
App$set_size(width, height)
```

Arguments:

width New width (pixels)

height New height (pixels)

Returns: The App instance (invisible)

Method `set_position()`: Set the window position dynamically

Usage:

```
App$set_position(x, y)
```

Arguments:

x Horizontal position from left (pixels)

y Vertical position from top (pixels)

Returns: The App instance (invisible)

Method `set_title()`: Set the window title dynamically

Usage:

```
App$set_title(title)
```

Arguments:

title New title

Returns: The App instance (invisible)

Method `minimize()`: Minimize the window to the taskbar

Usage:

```
App$minimize()
```

Returns: The App instance (invisible)

Method `maximize()`: Maximize the window to fill the screen

Usage:

```
App$maximize()
```

Returns: The App instance (invisible)

Method `restore()`: Restore the window from minimize/maximize

Usage:

```
App$restore()
```

Returns: The App instance (invisible)

Method fullscreen(): Toggle fullscreen mode

Usage:

App\$fullscreen(enabled = TRUE)

Arguments:

enabled If TRUE, enters fullscreen. If FALSE, exits.

Returns: The App instance (invisible)

Method always_on_top(): Set the window to stay always on top of others

Usage:

App\$always_on_top(enabled = TRUE)

Arguments:

enabled If TRUE, always on top.

Returns: The App instance (invisible)

Method set_menu(): Set the native window menu

Usage:

App\$set_menu(items)

Arguments:

items A named list of lists defining the menu structure

Returns: The App instance (invisible)

Method dialog_open(): Open a native file-open dialog

Usage:

App\$dialog_open(title = "Open File", filters = NULL)

Arguments:

title Dialog title

filters List of file filters, e.g. list("CSV files" = "*.csv")

Returns: Selected file path (character) or NULL if cancelled

Method dialog_save(): Open a native file-save dialog

Usage:

App\$dialog_save(title = "Save File", default_name = "", filters = NULL)

Arguments:

title Dialog title

default_name Initial filename

filters List of file filters

Returns: Selected file path (character) or NULL if cancelled

Method dialog_folder(): Open a native folder selection dialog

Usage:

App\$dialog_folder(title = "Select Folder")

Arguments:

title Dialog title

Returns: Selected directory path (character) or NULL if cancelled

Method `message_box()`: Show a native message box / alert

Usage:

```
App$message_box(message, title = "RDesk", type = "ok", icon = "info")
```

Arguments:

message The message text

title The dialog title

type One of "ok", "okcancel", "yesno", "yesnocancel"

icon One of "info", "warning", "error", "question"

Returns: The button pressed (character: "ok", "cancel", "yes", "no")

Method `dialog_color()`: Open a native color selection dialog

Usage:

```
App$dialog_color(initial_color = "#FFFFFF")
```

Arguments:

initial_color Optional hex color to start with (e.g. "#FF0000")

Returns: Selected hex color code or NULL if cancelled

Method `notify()`: Send a native desktop notification

Usage:

```
App$notify(title, body = "")
```

Arguments:

title Notification title

body Notification body text

Returns: The App instance (invisible)

Method `loading_start()`: Show a loading state in the UI

Usage:

```
App$loading_start(
  message = "Loading...",
  progress = NULL,
  cancellable = FALSE,
  job_id = NULL
)
```

Arguments:

message Text shown under the spinner

progress Optional numeric 0-100 for a progress bar

cancellable If TRUE, shows a cancel button in the UI

job_id Optional job_id from `rdesk_async()` to wire cancel button

Method loading_progress(): Update progress on an active loading state

Usage:

App\$loading_progress(value, message = NULL)

Arguments:

value Numeric 0-100

message Optional updated message

Method loading_done(): Hide the loading state in the UI

Usage:

App\$loading_done()

Method toast(): Show a non-blocking toast notification in the UI

Usage:

App\$toast(message, type = "info", duration_ms = 3000L)

Arguments:

message Text to show

type One of "info", "success", "warning", "error"

duration_ms How long to show it (default 3000ms)

Method set_tray(): Set or update the system tray icon

Usage:

App\$set_tray(label = "RDesk App", icon = NULL, on_click = NULL)

Arguments:

label Tooltip text for the tray icon

icon Path to .ico file (optional)

on_click Character "left" or "right" or callback function(button)

Returns: The App instance (invisible)

Method remove_tray(): Remove the system tray icon

Usage:

App\$remove_tray()

Returns: The App instance (invisible)

Method service(): Service this app's pending native events

Usage:

App\$service()

Returns: The App instance (invisible)

Method watch(): Enable or disable live hot reloading for this application

Usage:

App\$watch(enabled = TRUE)

Arguments:

enabled Logical. If TRUE, enables hot reload.

Returns: The App instance (invisible)

Method `quit()`: Close the window and stop the app's event loop.

Usage:

`App$quit()`

Returns: The App instance (invisible)

Method `get_dir()`: Get the application root directory (where `www/` and `R/` are located).

Usage:

`App$get_dir()`

Returns: Character string path.

Method `run()`: Start the application - opens the window

Usage:

`App$run(block = TRUE)`

Arguments:

`block` If TRUE (default), blocks with an event loop until the window is closed.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`App$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Safe logical check (unwrapped)
app_dir <- system.file("templates/hello", package = "RDesk")
if (nzchar(app_dir)) {
  message("Built-in app directory: ", app_dir)
}

if (interactive()) {
  app <- App$new(title = "Car Visualizer", width = 1200, height = 800)

  app$on_ready(function() {
    message("App is ready!")
  })

  # Handle messages from UI
  app$on_message("get_data", function(payload) {
    list(cars = mtcars[1:5, ])
  })

  # Start the app
  app$run()
}
```

async	<i>Wrap a message handler to run asynchronously with zero configuration</i>
-------	---

Description

`async()` is the simplest way to make an RDesk message handler non-blocking. Wrap any handler function with `async()` and RDesk automatically handles background execution, loading states, error toasts, and result routing.

`async()` transforms a standard RDesk message handler into a background task. The UI remains responsive while the task runs. When finished, a result message (e.g., `get_data_result`) is automatically sent back to the UI.

Usage

```
async(  
  fn,  
  app = NULL,  
  loading_message = "Working...",  
  cancellable = TRUE,  
  error_message = "Error: "  
)
```

Arguments

<code>fn</code>	The handler function, taking a payload argument.
<code>app</code>	The RDesk App instance. If <code>NULL</code> , tries to resolve from the global registry.
<code>loading_message</code>	Message to display in the UI overlay while working.
<code>cancellable</code>	Whether the UI should show a 'Cancel' button.
<code>error_message</code>	Prefix for toast notifications if the task fails.

Details

To ensure the background worker has access to all application logic, RDesk automatically sources every `.R` file in the application's `R/` directory before executing the task. It also snapshots currently loaded packages (excluding system packages) to recreate the environment.

Value

A wrapped handler function suitable for `app$on_message()`.

Examples

```
if (interactive()) {
  app$on_message("filter_cars", async(function(payload) {
    mtcars[mtcars$cyl == payload$cylinders, ]
  }, app = app))
}
```

 async_progress

Update progress of a background async task

Description

async_progress allows a long-running background task to send progress updates back to the main application thread. The main thread will automatically capture these updates and refresh the UI loading overlay.

Usage

```
async_progress(value, message = NULL)
```

Arguments

value	Numeric value from 0 to 100 representing the progress percentage.
message	Optional character string describing the current step/state. Shown beneath the progress bar in the loading overlay.

Details

Call this function from inside the function passed to [async\(\)](#) or [rdesk_async\(\)](#). It writes a JSON progress record to a temporary file that the main event loop polls every iteration. The file is cleaned up automatically when the job completes.

The progress value should increase monotonically from 0 to 100. The loading overlay in the front-end updates with each new value received.

Value

Invisible TRUE if progress was written successfully, FALSE if the progress file path is not set (e.g. outside a background task).

See Also

[async\(\)](#), [rdesk_async\(\)](#)

Examples

```

if (interactive()) {
  app$on_message("heavy_task", async(function(payload) {
    for (i in 1:10) {
      Sys.sleep(0.5)
      async_progress(i * 10, paste("Step", i, "of 10"))
    }
    "done"
  }, app = app))
}

```

build_app

*Build a self-contained distributable from an RDesk application***Description**

Build a self-contained distributable from an RDesk application

Usage

```

build_app(
  app_dir = ".",
  out_dir = file.path(tempdir(), "dist"),
  app_name = NULL,
  version = NULL,
  r_version = NULL,
  include_packages = character(0),
  portable_r_method = c("extract_only", "installer"),
  runtime_dir = NULL,
  overwrite = FALSE,
  build_installer = FALSE,
  publisher = "RDesk User",
  website = "https://github.com/Janakiraman-311/RDesk",
  license_file = NULL,
  icon_file = NULL,
  prune_runtime = TRUE,
  dry_run = FALSE
)

```

Arguments

app_dir	Path to the app directory (must contain app.R and www/)
out_dir	Output directory for the built artifact (created if needed)
app_name	Name of the application. Defaults to name in DESCRIPTION or "MyRDeskApp".
version	Version string. Defaults to version in DESCRIPTION or "1.0.0".
r_version	R version to bundle e.g. "4.4.2". Defaults to current R version.

include_packages	Character vector of extra CRAN packages to bundle. RDesk's own dependencies are always included automatically.
portable_r_method	How to provision the bundled R runtime when runtime_dir = "download" is used explicitly on Windows. "extract_only" requires standalone 7-Zip and never launches the R installer. "installer" allows the legacy silent installer path explicitly.
runtime_dir	Controls which R runtime is bundled with the app. • NULL (default) – copies the developer's currently running R installation. Guarantees the bundled packages and the runtime are the same R version, eliminating renv version-mismatch crashes. • A filesystem path – copies that R installation root directly. • "download" – downloads a portable R installer from CRAN on Windows only (legacy behaviour; requires network; risks version mismatch with renv).
overwrite	If TRUE, overwrite existing output. Default FALSE.
build_installer	If TRUE, also build a platform installer when supported: a Windows .exe via InnoSetup or a macOS .dmg. Linux currently produces only the bundle and .tar.gz archive.
publisher	Documentation for the application publisher (used in installer).
website	URL for the application website (used in installer).
license_file	Path to a license file (.txt or .rtf) to include in the installer.
icon_file	Path to an .ico file for the installer and application shortcut.
prune_runtime	If TRUE, remove unnecessary files (Tcl/Tk, docs, tests) from the bundled R runtime to reduce size (~15-20MB saving). Default TRUE.
dry_run	If TRUE, performs a quick validation of the app structure and environment without performing the full build. Default FALSE.

Value

The built artifact path or bundle metadata, invisibly.

Examples

```
# Prepare an app directory (following scaffold example)
app_path <- file.path(tempdir(), "MyApp")
rdesk_create_app("MyApp", path = tempdir())

# Perform a dry-run build (fast, no external binaries downloaded)
build_app(app_path, out_dir = tempdir(), dry_run = TRUE)

# Clean up
unlink(app_path, recursive = TRUE)
```

*RDeskStorage**RDesk Storage Manager*

Description

Provides a lightweight, JSON-backed key-value store for application state, preferences, and settings. Handles multi-user data isolation automatically.

Methods

Public methods:

- `RDeskStorage$new()`
- `RDeskStorage$set()`
- `RDeskStorage$get()`
- `RDeskStorage$remove()`
- `RDeskStorage$clear()`
- `RDeskStorage$keys()`
- `RDeskStorage$path()`
- `RDeskStorage$clone()`

Method `new()`: Initialize a new RDeskStorage manager

Usage:

```
RDeskStorage$new(app_name, storage_type = "local")
```

Arguments:

`app_name` Application name

`storage_type` Storage type: "roaming", "local", or "shared"

Method `set()`: Set a key-value pair in storage

Usage:

```
RDeskStorage$set(key, value)
```

Arguments:

`key` Character string key

`value` Value to store (must be JSON serializable)

Returns: The RDeskStorage instance (invisible)

Method `get()`: Retrieve a value from storage

Usage:

```
RDeskStorage$get(key, default = NULL)
```

Arguments:

`key` Character string key

`default` Default value to return if the key is not found

Returns: The stored value, or the default value

Method `remove()`: Remove a key from storage

Usage:

`RDeskStorage$remove(key)`

Arguments:

`key` Character string key

Returns: The `RDeskStorage` instance (invisible)

Method `clear()`: Clear all data in this storage domain

Usage:

`RDeskStorage$clear()`

Returns: The `RDeskStorage` instance (invisible)

Method `keys()`: List all keys currently in storage

Usage:

`RDeskStorage$keys()`

Returns: Character vector of keys

Method `path()`: Get the directory path containing the storage file

Usage:

`RDeskStorage$path()`

Returns: Character string directory path

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`RDeskStorage$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

rdesk_async

Run a task in the background

Description

Automatically switches between 'mirai' (persistent daemons) and 'callr' (on-demand processes).

Usage

```
rdesk_async(
  task,
  args = list(),
  on_done = NULL,
  on_error = NULL,
  timeout_sec = NULL,
  app_id = NULL
)
```

Arguments

task	A function to run in the background.
args	A list of arguments to pass to the task.
on_done	Callback function(result) called when the task finishes successfully.
on_error	Callback function(error) called if the task fails.
timeout_sec	Optional timeout in seconds. If exceeded, the job is cancelled and on_error() receives a timeout error.
app_id	Optional App ID used to associate a job with a specific app.

Value

Invisible job ID.

Examples

```
# Fast, non-interactive task check (safe to unwrap)
rdesk_jobs_pending()

if (interactive()) {
  # Run a long-running computation in the background
  rdesk_async(
    task = function(n) { Sys.sleep(2); sum(runif(n)) },
    args = list(n = 1e6),
    on_done = function(res) message("Task finished: ", res),
    on_error = function(err) message("Task failed: ", err$message)
  )
}
```

rdesk_auto_update	<i>Automatically check for and install app updates</i>
-------------------	--

Description

rdesk_auto_update is a high-level function designed for bundled (standalone) applications. It checks a remote version string, compares it with the current version, and if a newer version is found, it downloads and executes the installer silently before quitting the current application.

Usage

```
rdesk_auto_update(
  version_url,
  download_url,
  current_version,
  silent = FALSE,
  app = NULL
)
```

Arguments

version_url	URL to a plain text file containing the latest version string (e.g., "1.1.0")
download_url	URL to the latest installer .exe
current_version	Current app version string e.g. "1.0.0"
silent	If TRUE, downloads and installs without prompting. Default FALSE.
app	Optional App instance for showing toast notifications.

Value

Invisible TRUE if update was applied, FALSE otherwise.

rdesk_cancel_job	<i>Cancel a running background job</i>
------------------	--

Description

Attempts to stop a background job and remove it from the job registry. The behaviour differs between the two async backends:

callr The subprocess is killed immediately via `process$kill()`.

mirai The task inside the persistent daemon cannot be interrupted. The job is removed from the registry only, so its `on_done` and `on_error` callbacks are suppressed. The daemon itself keeps running.

Usage

```
rdesk_cancel_job(job_id)
```

Arguments

job_id	Character string. The job ID returned by rdesk_async() .
--------	--

Value

`invisible(TRUE)` if the job was found and cancelled, `invisible(FALSE)` if no job with that ID exists.

See Also

[rdesk_async\(\)](#), [rdesk_jobs_list\(\)](#)

rdesk_create_app	<i>Create a new RDesk application</i>
------------------	---------------------------------------

Description

Scaffolds a professional RDesk application with a modern dashboard layout. The app includes a sidebar for filters, KPI cards, and an asynchronous ggplot2 charting engine fueled by mtcars (default).

Usage

```
rdesk_create_app(
  name,
  path = tempdir(),
  data_source = NULL,
  viz_type = NULL,
  use_async = NULL,
  theme = "light",
  open = TRUE
)
```

Arguments

name	App name. Must be a valid directory name.
path	Directory to create the app in. Default is current directory.
data_source	Internal use. Defaults to "builtin".
viz_type	Internal use. Defaults to "mixed".
use_async	Internal use. Defaults to TRUE.
theme	One of "light", "dark", "system". Default "system".
open	Logical. If TRUE and in RStudio, opens the new project in a new session.

Value

Path to the created app directory, invisibly.

Examples

```
if (interactive()) {
  # Create the Professional Hero Dashboard in a temporary directory
  rdesk_create_app("MyDashboard", path = tempdir())
}

# The following demonstrates just the return value without opening a window
# (Fast and safe - no \dontrun needed for this specific logical check)
path <- file.path(tempdir(), "TestLogic")
if (!dir.exists(path)) {
```

```
# This is just a placeholder example of how to call the function safely
message("Scaffold path will be: ", path)
}
```

rdesk_df_to_list

Convert a data frame to a list suitable for JSON serialization

Description

Converts a data frame into the two-part list structure that `rdesk.js` expects: a `rows` component (list of per-row named lists) and a `cols` component (character vector of column names). Empty or NULL data frames return empty containers rather than an error.

Usage

```
rdesk_df_to_list(df)
```

Arguments

df A data frame to convert. NULL or zero-row data frames are handled gracefully.

Value

A list with two elements:

rows A list of named lists, one per row.

cols A character vector of column names.

See Also

[rdesk_plot_to_base64\(\)](#) for converting plot output.

Examples

```
result <- rdesk_df_to_list(head(mtcars, 3))
stopifnot(length(result$rows) == 3)
stopifnot("mpg" %in% result$cols)

# NULL input returns empty containers
empty <- rdesk_df_to_list(NULL)
stopifnot(length(empty$rows) == 0)
```

rdesk_error_plot	<i>Generate a base64-encoded error plot</i>
------------------	---

Description

Renders a minimal ggplot2 plot containing the supplied error message text and returns it as a Base64 data URI. Used as a safe fallback by [rdesk_plot_to_base64\(\)](#) when plot generation fails. Returns NULL silently if ggplot2 is not installed.

Usage

```
rdesk_error_plot(message = "Error generating plot")
```

Arguments

message	Character string to display on the error plot. Defaults to "Error generating plot".
---------	---

Value

A single-element character string (data URI) or NULL if ggplot2 is not available.

See Also

[rdesk_plot_to_base64\(\)](#)

rdesk_is_bundle	<i>Check if the app is running in a bundled (standalone) environment</i>
-----------------	--

Description

Returns TRUE when the current R process was launched by the RDesk stub binary as part of a compiled standalone application. The stub sets the R_BUNDLE_APP environment variable to "1" before starting R. Use this predicate to switch between development-time and runtime paths (e.g. logging directories, asset resolution).

Usage

```
rdesk_is_bundle()
```

Value

TRUE if running inside a bundled .exe or .app, FALSE otherwise.

Examples

```
# Returns FALSE in a normal interactive R session
rdesk_is_bundle()
```

rdesk_jobs_list	<i>List currently pending background jobs</i>
-----------------	---

Description

Returns a data frame snapshot of all jobs currently registered in the RDesk job registry. Useful for debugging and monitoring from the R console while an application is running.

Usage

```
rdesk_jobs_list()
```

Value

A data.frame with the following columns:

job_id Character. Unique job identifier.

started POSIXct. Time the job was submitted.

backend Character. Either "mirai" or "callr".

app_id Character. ID of the App instance that owns the job, or NA.

Returns a zero-row data frame when no jobs are pending.

See Also

[rdesk_jobs_pending\(\)](#), [rdesk_cancel_job\(\)](#)

rdesk_jobs_pending	<i>Check if any background jobs are pending</i>
--------------------	---

Description

Returns the number of background jobs currently registered in the RDesk job registry. A value greater than zero means at least one task is still running or has been submitted but not yet polled for completion.

This function is safe to call from the event loop or from tests without opening a window.

Usage

```
rdesk_jobs_pending()
```

Value

Integer. Number of pending jobs (0 means all tasks are complete).

See Also

[rdesk_jobs_list\(\)](#) for more detail about each pending job.

Examples

```
# Fast, non-interactive task check (safe to run unconditionally)
rdesk_jobs_pending()
```

rdesk_message

Construct a standard RDesk IPC message envelope

Description

Builds the JSON envelope that both R and the JavaScript frontend use to exchange typed messages. Each envelope carries a unique ID, a message type string, a contract version, an arbitrary payload, and a millisecond-precision Unix timestamp.

Usage

```
rdesk_message(
  type,
  payload = list(),
  version = getOption("rdesk.ipc_version", "1.0")
)
```

Arguments

type	Character string. The message type / action name (e.g. "get_data").
payload	A named list representing the message data. Defaults to an empty list.
version	IPC contract version string. Defaults to the <code>rdesk.ipc_version</code> option (currently "1.0").

Details

The envelope schema is:

```
{
  "id":      "msg_<epoch_ms>_<random>",
  "type":    "<action_name>",
  "version": "1.0",
  "payload": { ... },
  "timestamp": <unix_seconds>
}
```

Large payloads (above 1 MB) trigger a warning so developers notice performance-sensitive serialization early.

Value

A named list representing the standard JSON envelope. Pass this directly to `jsonlite::toJSON()` or to `app$send()`.

See Also

[rdesk_parse_message\(\)](#) for the corresponding deserializer.

Examples

```
# Construct a typed message with a simple payload
msg <- rdesk_message("get_data", list(filter = "cyl == 6"))
stopifnot(msg$type == "get_data")
stopifnot(msg$version == "1.0")
stopifnot(!is.null(msg$id))
```

rdesk_parse_message	<i>Parse and validate an incoming RDesk IPC message</i>
---------------------	---

Description

Deserializes a raw JSON string received from the JavaScript frontend (or from a bundled launcher via stdin) into a validated R list. Performs lightweight structural validation: both type and payload fields must be present. Launcher/native events (which carry an event field instead) bypass validation and are returned as-is.

Usage

```
rdesk_parse_message(raw_json)
```

Arguments

raw_json	A single-element character string containing the JSON to parse.
----------	---

Value

A named list containing the validated message components, or NULL if the JSON is malformed or the required fields are absent.

See Also

[rdesk_message\(\)](#) for constructing outgoing messages.

Examples

```
# Round-trip: construct then parse
msg <- rdesk_message("ping", list(ts = 1234))
raw <- jsonlite::toJSON(msg, auto_unbox = TRUE)
parsed <- rdesk_parse_message(raw)
stopifnot(parsed$type == "ping")

# Malformed JSON returns NULL
stopifnot(is.null(rdesk_parse_message("not_json")))
```

rdesk_plot_to_base64 *Convert a ggplot2 object to a base64-encoded PNG string*

Description

Renders a ggplot2 object to a temporary PNG file and returns the result as a Base64-encoded data URI string (data:image/png;base64,...). This format can be assigned directly to an tag src attribute in the JavaScript frontend via app\$send().

If ggplot2 is not installed, the function stops with a helpful message. If rendering fails, a fallback error plot is returned instead of NULL.

Usage

```
rdesk_plot_to_base64(plot, width = 6, height = 4, dpi = 96)
```

Arguments

plot	A ggplot2 object to render.
width	Width of the rendered image in inches. Default 6.
height	Height of the rendered image in inches. Default 4.
dpi	Dots per inch resolution. Default 96.

Value

A single-element character string containing the data URI, or the output of [rdesk_error_plot\(\)](#) if rendering fails.

See Also

[rdesk_error_plot\(\)](#) for the fallback error plot.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    ggplot2::geom_point()
  b64 <- rdesk_plot_to_base64(p)
  stopifnot(grepl("^data:image/png;base64,", b64))
}
```

rdesk_service	<i>Service all active RDesk applications</i>
---------------	--

Description

Processes native OS events for all open windows and polls all pending background jobs for completion. This is the single function you need to call periodically when running apps in non-blocking mode (`block = FALSE`).

In blocking mode (`block = TRUE`, the default) `app$run()` calls this function automatically inside its event loop and you do not need to call it manually.

Usage

```
rdesk_service()
```

Value

```
invisible(NULL)
```

See Also

[App](#) for the full application API.

Examples

```
if (interactive()) {  
  app <- App$new(title = "My App")  
  app$run(block = FALSE) # non-blocking  
  # ... do other work here ...  
  rdesk_service()        # poll for events  
}
```

rdesk_storage	<i>Create a multi-user storage manager</i>
---------------	--

Description

`rdesk_storage` provides file-based persistent storage for desktop applications. On Windows, it handles multi-user data isolation by mapping key-value stores to the correct system folder depending on the requested type.

Usage

```
rdesk_storage(app_name, type = c("local", "roaming", "shared"))
```

Arguments

app_name	Character string. The application name, used as the subfolder name.
type	Storage type: one of "roaming" (user roaming folder, APPDATA), "local" (user local folder, LOCALAPPDATA), or "shared" (machine-wide folder, PROGRAMDATA).

Details

The three storage types correspond to different Windows user profile folders:

roaming User roaming folder (APPDATA). Suitable for per-user preferences that should follow the user across machines when roaming profiles are enabled.

local User local folder (LOCALAPPDATA). Suitable for cache, history, or data that is specific to one machine.

shared Machine-wide folder (PROGRAMDATA). Suitable for configuration shared by all users on the same computer.

Outside a bundled application (e.g. during development or R CMD check), all storage types fall back to a subdirectory of `tempdir()` to comply with CRAN policies on persistent file writes.

Value

An `RDeskStorage` R6 instance with `get()`, `set()`, `remove()`, `clear()`, `keys()`, and `path()` methods.

See Also

[RDeskStorage](#) for the full method reference.

Examples

```
# Create a local storage manager for an app called "MyApp"
s <- rdesk_storage("MyApp", "local")

# Store and retrieve a value
s$set("last_filter", "cyl == 6")
stopifnot(s$get("last_filter") == "cyl == 6")

# List all keys
s$keys()

# Remove a specific key
s$remove("last_filter")
```

rdesk_watch

*Enable live hot reloading for an RDesk application***Description**

rdesk_watch enables live monitoring of R source files and UI asset files (HTML, CSS, JS). When a UI file changes, the application automatically reloads the page. When an R script changes, the framework sources the modified module and automatically re-binds application event handlers.

Usage

```
rdesk_watch(app, enabled = TRUE)
```

Arguments

app	The RDesk App instance to monitor.
enabled	Logical. If TRUE (default), enables live monitoring. Set to FALSE to disable.

Details

RDesk hot reload works by polling file modification times once per event loop iteration (roughly every 10 ms). When a change is detected:

R files The modified file is `source()`d in the global environment. If a function named `init_handlers` exists in the global environment, it is called with the App instance so that message handlers are re-registered.

HTML/CSS/JS files A `__reload_ui__` message is sent to the frontend, triggering a full page reload in the WebView.

Hot reloading is designed for development only. It should not be enabled in bundled production builds.

Value

The App instance (invisible), to allow method chaining.

See Also

[App](#)

Examples

```
if (interactive()) {
  app <- App$new(title = "My App")
  rdesk_watch(app) # or equivalently: app$watch(TRUE)
  app$run()
}
```

Index

App, [2](#), [26](#), [28](#)

async, [11](#)

async(), [12](#)

async_progress, [12](#)

build_app, [13](#)

rdesk_async, [16](#)

rdesk_async(), [12](#), [18](#)

rdesk_auto_update, [17](#)

rdesk_cancel_job, [18](#)

rdesk_cancel_job(), [22](#)

rdesk_create_app, [19](#)

rdesk_df_to_list, [20](#)

rdesk_error_plot, [21](#)

rdesk_error_plot(), [25](#)

rdesk_is_bundle, [21](#)

rdesk_jobs_list, [22](#)

rdesk_jobs_list(), [18](#), [23](#)

rdesk_jobs_pending, [22](#)

rdesk_jobs_pending(), [22](#)

rdesk_message, [23](#)

rdesk_message(), [24](#)

rdesk_parse_message, [24](#)

rdesk_parse_message(), [24](#)

rdesk_plot_to_base64, [25](#)

rdesk_plot_to_base64(), [20](#), [21](#)

rdesk_service, [26](#)

rdesk_storage, [26](#)

rdesk_watch, [28](#)

RDeskStorage, [15](#), [27](#)