# Package 'shinyelectron'

August 7, 2026

**Title** Export 'Shiny' Applications as Desktop Apps using 'Electron'

**Version** 0.2.1

**Description** Provides tools to export 'Shiny' applications written in 'R' or 'Python' as standalone desktop applications using 'Electron'. The applications run as native, cross-platform programs. Depending on the runtime strategy chosen, end users do not need 'R' or 'Python' installed on their machine.

**License** AGPL (>= 3)

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

**BugReports** https://github.com/coatless-rpkg/shinyelectron/issues

**Depends** R (>= 4.4.0)

**Imports** cli (>= 3.6.6), fs (>= 2.1.0), jsonlite (>= 2.0.0), rappdirs (>= 0.3.4), whisker (>= 0.4.1), processx (>= 3.9.0), stats, tools, utils, yaml (>= 2.3.12)

**Suggests** testthat (>= 3.3.2), mockery (>= 0.4.5), quarto (>= 1.5.1), renv (>= 1.2.3), shinylive (>= 0.5.0), withr (>= 3.0.3)

**VignetteBuilder** quarto

**SystemRequirements** Node.js (>= 22.0.0), npm (>= 11.5.0)

**Encoding** UTF-8

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** James Joseph Balamuta [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-2826-8458>)

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

**Repository** CRAN

**Date/Publication** 2026-08-07 16:10:02 UTC

# Contents



---

`app_check` *Check Shiny Application Readiness for Export*

---

## Description

Validates that a Shiny application can be built as an Electron app. Checks app structure, configuration, runtime availability, dependencies, and signing credentials. Reports issues without aborting.

## Usage

```
app_check(
  appdir = ".",
  app_type = NULL,
  runtime_strategy = NULL,
  platform = NULL,
  sign = NULL,
```

```
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| appdir | Character string. Path to the app directory. Default ".". |
| app_type | Character string or NULL. App type override. If NULL, reads from config or autodetects from files in appdir. |
| runtime_strategy | Character string or NULL. Runtime strategy override. |
| platform | Character vector or NULL. Target platforms override. |
| sign | Logical or NULL. Signing override. |
| verbose | Logical. Whether to print the report. Default TRUE. |

## Value

Invisible list with:

| | |
|---|---|
| pass | Logical. TRUE if no errors found. |
| errors | Character vector of fatal issues. |
| warnings | Character vector of non-fatal issues. |
| info | Character vector of informational notes. |

## Examples

```
# Check a bundled example app
app_check(example_app("r"))

# Check with explicit overrides
app_check(example_app("r"), app_type = "r-shiny", runtime_strategy = "system")
```

---

# app_dependencies *Detect an app's package dependencies*

---

## Description

Scans a Shiny app's source for the R or Python packages it uses, with the same detection shinylive applies at build time. This is useful before a shinylive build, in CI especially, because `shinylive::export()` compiles the WebAssembly bundle from the packages installed in the current session, so an app's dependencies must be installed before conversion.

## Usage

```
app_dependencies(appdir, app_type = NULL)
```

### Arguments

| | |
|---|---|
| appdir | Character string. Path to the app or multi-app suite directory. |
| app_type | Character string or NULL. "r-shiny" or "py-shiny", or NULL to autodetect from appdir. |

### Value

Character vector of detected package names, excluding base R packages (for R) or the Python standard library (for Python).

### Examples

```
# Detect the packages a bundled example app uses
app_dependencies(example_app("r"))
```

---

## available_examples *List Available Examples*

---

### Description

Shows all bundled example applications with their descriptions.

### Usage

```
available_examples()
```

### Value

A data frame with columns: name (character ID), language ("r" or "python"), type (app type, "r-shiny" or "py-shiny"), and description (human-readable summary).

### Examples

```
available_examples()
```

build_electron_app *Build Electron Application*

## Description

Builds a distributable Electron application from a converted Shiny app. Creates platform-specific installers and executables.

## Usage

```
build_electron_app(
  app_dir,
  output_dir,
  app_name = NULL,
  app_type = "r-shiny",
  runtime_strategy = "shinylive",
  sign = FALSE,
  platform = NULL,
  arch = NULL,
  icon = NULL,
  config = NULL,
  overwrite = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| app_dir | Character string. Path to the converted Shiny/shinylive application. |
| output_dir | Character string. Path where the built Electron app will be saved. |
| app_name | Character string. Name of the application. If NULL, uses the base name of app_dir. |
| app_type | Character string. Language of the Shiny app: "r-shiny" (default) or "py-shiny". Unlike export(), this function does **not** autodetect the language from source files – the default "r-shiny" is used when app_type is not supplied. Supply "py-shiny" explicitly for Python Shiny applications. The legacy values "r-shinylive" / "py-shinylive" are accepted with a deprecation warning and translate to the canonical language plus runtime_strategy = "shinylive". |
| runtime_strategy | Character string. Runtime strategy: "shinylive", "bundled", "system", "auto-download", or "container". Default "shinylive". |
| sign | Logical. Whether to enable code signing for the built application. Default is FALSE. |
| platform | Character vector. Target platforms: "win", "mac", "linux". If NULL, builds for current platform. |

| | |
|---|---|
| arch | Character vector. Target architectures: "x64", "arm64". If NULL, uses current architecture. |
| icon | Character string. Path to application icon file. Platform-specific format required. |
| config | List. Configuration from _shinyelectron.yml file (optional). Used for template variables like window dimensions, port, and app version. |
| overwrite | Logical. Whether to overwrite existing output directory. Default is FALSE. |
| verbose | Logical. Whether to display detailed progress information. Default is TRUE. |

## Value

Character string. Path to the built Electron application directory.

## Details

This function creates a complete Electron application by:

- Setting up the Electron project structure
- Copying application files and templates
- Installing npm dependencies
- Building platform-specific distributables

## Examples

```
# Build Electron app for current platform
build_electron_app(
  app_dir = "path/to/shinylive/app",
  output_dir = "path/to/electron/build",
  app_name = "My Shiny App",
  app_type = "r-shiny"
)

# Build for multiple platforms
build_electron_app(
  app_dir = "path/to/app",
  output_dir = "path/to/build",
  app_name = "My App",
  app_type = "r-shiny",
  platform = c("win", "mac", "linux")
)
```

cache_clear *Clear the asset cache*

## Description

Removes cached R installations and/or npm packages from the cache directory.

## Usage

```
cache_clear(what = c("all", "r", "npm", "nodejs", "python"))
```

## Arguments

| | |
|---|---|
| what | Character string specifying what to clear. One of "all", "r", "npm", "nodejs", or "python". |

## Value

Invisibly returns NULL.

## Details

Use this function to free disk space or force re-downloading of assets:

- "r": Removes only cached R installations
- "npm": Removes only cached npm packages
- "nodejs": Removes only cached Node.js installations
- "python": Removes only cached Python installations
- "all": Removes all cached assets

If the cache directory doesn't exist, a message is shown and nothing is done.

## Examples

```
# Clear everything in the cache
cache_clear()

# Clear only R installations
cache_clear("r")

# Clear only npm packages
cache_clear("npm")

# Clear only Node.js installations
cache_clear("nodejs")

# Clear only Python installations
cache_clear("python")
```

cache_dir *Get or create the cache directory path*

## Description

Returns the path where shinyelectron stores downloaded runtimes (R, Python, Node.js) and other cached assets. By default, the directory is created if it doesn't already exist. Pass `create = FALSE` to query the path without side effects.

## Usage

```
cache_dir(create = TRUE)
```

## Arguments

| | |
|---|---|
| `create` | Logical. Whether to create the directory if it doesn't exist. Default is TRUE. |

## Value

Character string. Absolute path to the cache directory, typically `~/.cache/shinyelectron/assets` on Linux, `~/Library/Caches/shinyelectron/assets` on macOS, or `\%LOCALAPPDATA\%/shinyelectron/shinyelectro` on Windows.

## Cache Layout

Cached runtimes are organized by type, platform, architecture, and version:

```
assets/
|-- r/
|   |-- win/x64/4.5.3/
|   |-- mac/arm64/4.5.3/
|-- python/
|   |-- win/x64/3.14.6/
|   |-- mac/arm64/3.14.6/
|-- nodejs/
|   |-- v22.11.0/darwin-arm64/
|   |-- v22.11.0/win-x64/
```

Use `cache_info()` to see what's actually installed with disk usage.

## See Also

`cache_info()` to see what's cached, `cache_clear()` to remove cached assets, `cache_remove()` to remove a specific version.

## Examples

```
# Query the cache path without creating it
cache_dir(create = FALSE)

# Get or create the cache directory (writes to the user cache dir)
if (interactive()) {
  cache_dir()
}
```

---

`cache_info` *Show cached runtime information*

---

## Description

Lists all cached runtimes (R, Python, Node.js) with their versions, platforms, architectures, and disk usage. Modeled after `shinylive::assets_info()`.

## Usage

```
cache_info(quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `quiet` | Logical. If TRUE, suppresses console output and returns the results invisibly. Default is FALSE. |

## Value

A data frame (returned invisibly) with columns: `runtime` (character), `version` (character), `platform` (character), `arch` (character), `size` (character, human-readable), and `path` (character).

## See Also

`cache_clear()` to remove cached assets, `cache_dir()` for the cache location.

## Examples

```
# Programmatic access (safe to run -- just inspects the cache dir)
df <- cache_info(quiet = TRUE)
nrow(df)  # number of cached runtimes

# Pretty-print the cache contents
cache_info()
```

cache_remove *Remove a specific cached runtime version*

## Description

Removes a single cached runtime version instead of clearing the entire cache. Use `cache_info()` to see what's available.

## Usage

```
cache_remove(runtime, version, platform = NULL, arch = NULL)
```

## Arguments

| | |
|---|---|
| `runtime` | Character string. One of `"r"`, `"python"`, or `"nodejs"`. |
| `version` | Character string. Version to remove (e.g., `"4.5.3"`, `"3.14.6"`, `"v22.11.0"`). |
| `platform` | Character string. Platform (`"win"`, `"mac"`, or `"linux"`). Required for all runtimes including Node.js. Use the same canonical names that `cache_info()` reports in the `platform` column (e.g. `"mac"`, not `"darwin"`). |
| `arch` | Character string. Architecture (`"x64"` or `"arm64"`). Required for all runtimes including Node.js. |

## Value

Invisibly returns TRUE if removed, FALSE if not found.

## See Also

`cache_info()` to list cached versions, `cache_clear()` to remove all cached assets of a type.

## Examples

```
# Remove a specific R version
cache_remove("r", "4.4.0", "mac", "arm64")

# Remove a cached Python version
cache_remove("python", "3.14.6", "win", "x64")

# Remove one platform/arch slot of a Node.js version
cache_remove("nodejs", "v22.11.0", "mac", "arm64")
```

## check_auto_update_status *Check Auto-Update Status*

### Description

Reports the current auto-update configuration status.

### Usage

```
check_auto_update_status(appdir)
```

### Arguments

| | |
|---|---|
| appdir | Character path to app directory |

### Value

Invisibly returns the updates configuration list, which is always present because read_config() deep-merges defaults. Elements include enabled (logical, FALSE by default when auto-updates have never been enabled), provider (character), check_on_startup, auto_download, auto_install (logical), and, for the GitHub provider, github (a list with owner, repo, private).

### Examples

```
# Check update configuration on a temporary app
app <- file.path(tempdir(), "check-updates-demo")
dir.create(app, showWarnings = FALSE)
writeLines("library(shiny)", file.path(app, "app.R"))
enable_auto_updates(app, owner = "myusername", repo = "myapp", verbose = FALSE)
check_auto_update_status(app)
```

## convert_py_to_shinylive *Convert Python Shiny Application to Shinylive*

### Description

Converts a Python Shiny application directory into a shinylive application that can run entirely in the browser using Pyodide.

## Usage

```
convert_py_to_shinylive(
  appdir,
  output_dir,
  subdir = NULL,
  overwrite = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| appdir | Character string. Path to the directory containing the Python Shiny application. |
| output_dir | Character string. Path where the converted shinylive app will be saved. |
| subdir | Character or NULL. When set, the app is exported into a <subdir> subdirectory of output_dir as an additive shared-site export, preserving existing contents (including a shared shinylive/ asset tree). When NULL (default), a single-app export is performed and an existing output_dir is removed when overwrite = TRUE. |
| overwrite | Logical. Whether to overwrite existing output directory. Default is FALSE. |
| verbose | Logical. Whether to display detailed progress information. Default is TRUE. |

## Value

Character string. Path to the converted shinylive application directory.

## Details

This function converts a Python Shiny application to shinylive format using the Python shinylive package. The application will run entirely in the browser using Pyodide (Python compiled to WebAssembly).

**Requirements:**

- Python 3 must be available on the build machine
- The Python shinylive package must be installed (pip install shinylive)
- The app directory must contain an app.py file

## Examples

```
convert_py_to_shinylive(
  appdir = "path/to/python/shiny/app",
  output_dir = "path/to/shinylive/output"
)
```

convert_shiny_to_shinylive *Convert Shiny Application to Shinylive*

## Description

Converts a regular Shiny application directory into a shinylive application that can run entirely in the browser without requiring an R server.

## Usage

```
convert_shiny_to_shinylive(
  appdir,
  output_dir,
  subdir = NULL,
  overwrite = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| appdir | Character string. Path to the directory containing the Shiny application. |
| output_dir | Character string. Path where the converted shinylive app will be saved. |
| subdir | Character or NULL. When set, the app is exported into a <subdir> subdirectory of output_dir as an additive shared-site export: existing contents of output_dir (including a shared shinylive/ asset tree) are preserved. When NULL (default), a single-app export is performed and an existing output_dir is removed when overwrite = TRUE. |
| overwrite | Logical. Whether to overwrite existing output directory. Default is FALSE. |
| verbose | Logical. Whether to display detailed progress information. Default is TRUE. |

## Value

Character string. Path to the converted shinylive application directory.

## Details

This function converts a Shiny application to shinylive format, which allows the application to run entirely in the browser using WebR. The conversion process:

- Validates the input Shiny application structure
- Converts R code to be compatible with WebR
- Creates necessary shinylive configuration files
- Packages the application for browser execution

## Examples

```
# Convert a Shiny app to shinylive
convert_shiny_to_shinylive(
  appdir = "path/to/shiny/app",
  output_dir = "path/to/shinylive/output"
)
```

---

`disable_auto_updates` *Disable Auto-Updates*

---

## Description

Disables automatic update checking in the configuration file.

## Usage

```
disable_auto_updates(appdir, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `appdir` | Character path to app directory |
| `verbose` | Logical whether to show progress messages. Default `TRUE`. |

## Value

Invisibly returns the path to the updated config file.

## Examples

```
# Disable updates on a temporary app
app <- file.path(tempdir(), "disable-updates-demo")
dir.create(app, showWarnings = FALSE)
writeLines("library(shiny)", file.path(app, "app.R"))
enable_auto_updates(app, owner = "myusername", repo = "myapp", verbose = FALSE)
disable_auto_updates(app)
```

`enable_auto_updates` *Enable Auto-Updates*

## Description

Configures automatic update checking for your Electron application. GitHub Releases is the only provider supported today. S3 and Generic HTTP providers are planned and will be added in a future release.

## Usage

```
enable_auto_updates(
  appdir,
  provider = "github",
  owner = NULL,
  repo = NULL,
  check_on_startup = TRUE,
  auto_download = FALSE,
  auto_install = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `appdir` | Character path to app directory containing `_shinyelectron.yml` |
| `provider` | Character update provider. Currently only `"github"` is supported; `"s3"` and `"generic"` are not yet wired into the build and will be added in a future release. |
| `owner` | Character GitHub username or organization (required for github provider) |
| `repo` | Character GitHub repository name (required for github provider) |
| `check_on_startup` | Logical whether to check for updates when app starts. Default `TRUE`. |
| `auto_download` | Logical whether to download updates automatically. Default `FALSE`. |
| `auto_install` | Logical whether to install updates automatically on quit. Default `FALSE`. |
| `verbose` | Logical whether to show progress messages. Default `TRUE`. |

## Details

Auto-updates require:

1. A published application (e.g., to GitHub Releases)
2. Proper code signing for macOS and Windows (recommended)
3. The electron-updater package (automatically included in build)

**Update Providers:**
**GitHub Releases** (recommended for open source):

- Automatically detects new releases by comparing semver tags
- Requires `owner` and `repo` parameters
- Private repos require `GH_TOKEN` environment variable

**S3 Bucket** (planned, not yet supported):

- For self-hosted updates behind a CDN
- Configure bucket, region, and path in `_shinyelectron.yml`
- Bucket must allow public reads or use CloudFront signed URLs
- Required bucket structure: `/{path}/latest-mac.yml`, `latest-linux.yml`, `latest.yml`
- Note: `enable_auto_updates()` currently rejects `"s3"` as a provider; S3 support will be added in a future release.

**Generic HTTP Server** (planned, not yet supported):

- For any HTTP server hosting update files
- Configure base URL in `_shinyelectron.yml`
- Server must host `latest-mac.yml`, `latest-linux.yml`, `latest.yml` at the URL root
- Note: `enable_auto_updates()` currently rejects `"generic"` as a provider; Generic HTTP support will be added in a future release.

**Publishing Updates (GitHub Releases):**

After enabling auto-updates, follow this workflow to publish updates:

1. **Bump version** in `_shinyelectron.yml` (e.g., `version: "1.1.0"`)
2. **Rebuild** with `export(appdir, destdir, build = TRUE)`
3. **Create a GitHub Release** with a semver tag matching the version:
   - Tag: `v1.1.0` (the `v` prefix is required)
   - Upload the built artifacts from `destdir/electron-app/dist/`:
     - macOS: `.dmg` and `latest-mac.yml`
     - Windows: `.exe` installer and `latest.yml`
     - Linux: `.AppImage` and `latest-linux.yml`
4. The app checks for updates on startup (if `check_on_startup = TRUE`) and notifies the user when a new version is available

**Code Signing Requirement:**

macOS and Windows require code-signed builds for auto-updates to work. Unsigned apps will fail the update verification step. Set `signing: sign: true` in your config and provide credentials via environment variables (see `?validate_signing_config`).

## Value

Invisibly returns the path to the updated config file.

## See Also

`init_config()` for creating initial configuration

## Examples

```
# Enable GitHub-based updates on a temporary app
app <- file.path(tempdir(), "enable-updates-demo")
dir.create(app, showWarnings = FALSE)
writeLines("library(shiny)", file.path(app, "app.R"))
enable_auto_updates(app, owner = "myusername", repo = "myapp")

# Enable with automatic download
enable_auto_updates(app, owner = "myorg", repo = "dashboard",
                    auto_download = TRUE)
```

---

`example_app` *Get Path to an Example Application*

---

## Description

Returns the path to a bundled example application directory. Use this path as the `appdir` argument to `export()`.

## Usage

```
example_app(name)
```

## Arguments

| | |
|---|---|
| `name` | Character string. Name of the example (see `available_examples()`). |

## Value

Character string. Path to the example app directory.

## Examples

```
# Get the path to a bundled example
example_app("r")
example_app("python")

if (interactive()) {
  # Pass the path to export() to build a desktop app
  path <- example_app("r")
  export(path, "output", app_type = "r-shiny", runtime_strategy = "system")
}
```

export *Export Shiny Application as Electron Desktop Application*

## Description

Main entry point function that wraps the conversion, building, and optionally running of a Shiny application as an Electron desktop application.

## Usage

```
export(
  appdir,
  destdir,
  app_name = NULL,
  app_type = NULL,
  runtime_strategy = NULL,
  sign = FALSE,
  platform = NULL,
  arch = NULL,
  icon = NULL,
  overwrite = FALSE,
  build = TRUE,
  run_after = FALSE,
  open_after = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| appdir | Character string. Path to the directory containing the Shiny application. |
| destdir | Character string. Path to the destination directory where the Electron app will be created. |
| app_name | Character string. Name of the application. If NULL, uses the base name of appdir. |
| app_type | Character string or NULL. Language of the Shiny app: "r-shiny" or "py-shiny". If NULL (default), the type is autodetected from files in appdir. The legacy values "r-shinylive" and "py-shinylive" are accepted with a deprecation warning and translate to the canonical language plus runtime_strategy = "shinylive". |
| runtime_strategy | Character string or NULL. How R or Python reaches the end user: "shinylive", "bundled", "system", "auto-download", or "container". Default "shinylive" when neither argument nor config sets one. |
| sign | Logical. Whether to enable code signing for the built application. When TRUE, electron-builder will attempt to sign the app using credentials from environment variables or the config file. Default is FALSE. |

| | |
|---|---|
| platform | Character vector. Target platforms: "win", "mac", "linux". If NULL, builds for current platform. |
| arch | Character vector. Target architectures: "x64", "arm64". If NULL, uses current architecture. |
| icon | Character string. Path to application icon file. Platform-specific format required. |
| overwrite | Logical. Whether to overwrite existing output directory. Default is FALSE. |
| build | Logical. Whether to build distributable packages. Default is TRUE. |
| run_after | Logical. Whether to run the application in development mode after export. Default is FALSE. |
| open_after | Logical. Whether to open the generated project directory after export. Default is FALSE. |
| verbose | Logical. Whether to display detailed progress information. Default is TRUE. |

## Value

List containing paths to the converted app and built Electron app (if built).

## Details

This is the main function of the package that orchestrates the entire process:

- Validates the input Shiny application
- Converts the Shiny app to the specified format (shinylive by default)
- Sets up the Electron project structure
- Optionally builds distributable packages
- Optionally runs the application for testing

## Supported Combinations

Two languages, five delivery strategies.

- `r-shiny` or `py-shiny` plus `runtime_strategy = "shinylive"`: app compiles to WebAssembly and runs inside the Electron window with no runtime on disk.
- `r-shiny` or `py-shiny` plus `"auto-download"`, `"bundled"`, `"system"`, or `"container"`: app runs against a real R or Python process supplied by the chosen strategy.

## Examples

```
# Simplest call: app_type autodetects, runtime_strategy defaults to shinylive
export(
  appdir = "path/to/shiny/app",
  destdir = "path/to/electron/output"
)

# Run against a real R process instead of shinylive
export(
  appdir = "path/to/shiny/app",
```

```
  destdir = "path/to/output",
  runtime_strategy = "bundled"
)

# Pin language explicitly when autodetection is ambiguous
export(
  appdir = "path/to/shiny/app",
  destdir = "path/to/output",
  app_type = "r-shiny",
  runtime_strategy = "system"
)
```

---

`init_config` *Initialize configuration file*

---

## Description

Creates a template _shinyelectron.yml file in the specified directory.

## Usage

```
init_config(appdir, app_name = NULL, overwrite = FALSE, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `appdir` | Character path to app directory |
| `app_name` | Character application name. If NULL, derived from directory name. |
| `overwrite` | Logical whether to overwrite existing config. Default FALSE. |
| `verbose` | Logical whether to show progress. Default TRUE. |

## Value

Invisibly returns the path to the created config file.

## See Also

`wizard()` for an interactive configuration generator; `show_config()` to display the merged effective configuration.

## Examples

```
# Create a config for a temporary app
app <- file.path(tempdir(), "init-config-demo")
dir.create(app, showWarnings = FALSE)
writeLines("library(shiny)", file.path(app, "app.R"))
init_config(app, app_name = "My App")
```

`install_nodejs` *Install Node.js locally*

## Description

Downloads and installs Node.js to the shinyelectron cache directory. This allows using Node.js/npm without requiring system-wide installation.

## Usage

```
install_nodejs(
  version = NULL,
  platform = NULL,
  arch = NULL,
  force = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `version` | Character Node.js version to install. If NULL (default), automatically detects the latest LTS version. |
| `platform` | Character target platform ("win", "mac", "linux"). Default is current platform. |
| `arch` | Character target architecture ("x64", "arm64"). Default is current architecture. |
| `force` | Logical whether to reinstall if already exists. Default FALSE. |
| `verbose` | Logical whether to show progress. Default TRUE. |

## Value

Invisibly returns the path to the installed Node.js directory.

## See Also

`install_r_portable()`, `install_python_standalone()` for other runtime installers.

## Examples

```
# Install latest LTS version
install_nodejs()

# Install specific version
install_nodejs(version = "20.0.0")

# Force reinstall
install_nodejs(force = TRUE)
```

`install_python_standalone` *Install a portable Python distribution*

## Description

Downloads and caches a portable Python build from python-build-standalone.

## Usage

```
install_python_standalone(
  version = NULL,
  platform = NULL,
  arch = NULL,
  force = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `version` | Character string. Python version to install (e.g., `"3.14.6"`). If NULL, the maintained pin in `SHINYELECTRON_DEFAULTS$runtime_versions$python$version` is used. |
| `platform` | Character string. Target platform. |
| `arch` | Character string. Target architecture. |
| `force` | Logical. Whether to reinstall if already cached. |
| `verbose` | Logical. Whether to show progress. |

## Value

Character string. Path to the installed Python directory.

## See Also

`install_r_portable()`, `install_nodejs()` for other runtime installers; `python_executable()` to find the installed Python path.

## Examples

```
# Install default Python version
install_python_standalone()

# Install specific version
install_python_standalone(version = "3.12.0")
```

`install_r_portable` *Install a portable R distribution*

## Description

Downloads and caches a portable R build. Follows the same pattern as `install_nodejs()`.

## Usage

```
install_r_portable(
  version = NULL,
  platform = NULL,
  arch = NULL,
  force = FALSE,
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| `version` | Character string. R version to install. If NULL, installs latest. |
| `platform` | Character string. Target platform. |
| `arch` | Character string. Target architecture. |
| `force` | Logical. Whether to reinstall if already cached. |
| `verbose` | Logical. Whether to show progress. |

## Value

Character string. Path to the installed R directory.

## See Also

`install_python_standalone()`, `install_nodejs()` for other runtime installers; `r_executable()` to find the installed Rscript path.

## Examples

```
# Install latest R release
install_r_portable()

# Install specific version for a target platform
install_r_portable(version = "4.4.0", platform = "win", arch = "x64")
```

`run_electron_app` *Run Electron Application for Testing*

## Description

Launches a previously exported Electron application for testing and debugging without building distributable packages. Pass the `electron-app` directory from a prior `export()` call.

## Usage

```
run_electron_app(app_dir, port = 3000, open_devtools = TRUE, verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `app_dir` | Character string. Path to the Electron application directory (the `electron-app` subdirectory from `export()`). |
| `port` | Integer. Port number for the development server. Default is 3000. |
| `open_devtools` | Logical. Whether to open Chromium DevTools automatically. Default is TRUE. |
| `verbose` | Logical. Whether to display detailed progress information. Default is TRUE. |

## Value

Invisibly returns the completed `processx::run()` result list (with `status`, `stdout`, and `stderr`) after Electron exits, or `NULL` if the run is interrupted. Note that this call blocks until the Electron window is closed.

## Details

This function starts the Electron application for testing, which:

- Opens the application in an Electron window
- Optionally opens Chromium DevTools for debugging
- Does NOT build distributable packages (use `export(build = TRUE)` for that)

## Examples

```
# Run a previously exported Electron app in development mode
run_electron_app("path/to/electron/app")

# Run with custom port and no dev tools
run_electron_app(
  app_dir = "path/to/app",
  port = 8080,
  open_devtools = FALSE
)
```

## show_config *Show Effective Configuration*

### Description

Pretty-prints the merged effective configuration (params + config file + defaults) for a shinyelectron app directory. Useful for debugging and verifying settings.

### Usage

```
show_config(appdir = ".")
```

### Arguments

| | |
|---|---|
| appdir | Character path to the app directory. |

### Value

Invisibly returns the merged configuration list.

### Examples

```
# Show the merged configuration for a bundled example app
show_config(example_app("r"))
```

## sitrep_electron_build_tools *Build Tools Situation Report*

### Description

Checks platform-specific build tools required for creating Electron distributables.

### Usage

```
sitrep_electron_build_tools(verbose = TRUE)
```

### Arguments

| | |
|---|---|
| verbose | Logical. Whether to print detailed output. Default is TRUE. |

### Value

Invisibly returns a list with build tools information.

## Examples

```
# Check build tools
sitrep_electron_build_tools()
```

---

`sitrep_electron_dependencies`
*Dependencies Situation Report*

---

## Description

Checks R package dependencies required for shinyelectron functionality.

## Usage

```
sitrep_electron_dependencies(verbose = TRUE)
```

## Arguments

| | |
|---|---|
| `verbose` | Logical. Whether to print detailed output. Default is TRUE. |

## Value

Invisibly returns a list with dependency information.

## Examples

```
# Check R package dependencies (quiet = returns a list invisibly)
deps <- sitrep_electron_dependencies(verbose = FALSE)
length(deps$missing_required)


# Pretty-printed report
sitrep_electron_dependencies()
```

## sitrep_electron_project

*Project Situation Report*

### Description

Checks if the current directory contains a valid Electron project and diagnoses common project-related issues.

### Usage

```
sitrep_electron_project(project_dir = ".", verbose = TRUE)
```

### Arguments

| | |
|---|---|
| project_dir | Character. Path to the project directory. Default is current directory. |
| verbose | Logical. Whether to print detailed output. Default is TRUE. |

### Value

Invisibly returns a list with project diagnostic information.

### Examples

```
# Check the current directory for a shinyelectron project
sitrep_electron_project()

# Check a specific directory
sitrep_electron_project(tempdir())
```

## sitrep_electron_system

*System Requirements Situation Report*

### Description

Checks system requirements for shinyelectron including Node.js, npm, operating system, and architecture.

### Usage

```
sitrep_electron_system(verbose = TRUE)
```

### Arguments

| | |
|---|---|
| verbose | Logical. Whether to print detailed output. Default is TRUE. |

### Value

Invisibly returns a list with diagnostic information.

### Examples

```
# Check system requirements
sitrep_electron_system()

# Get diagnostic info without printing
info <- sitrep_electron_system(verbose = FALSE)
```

---

## sitrep_shinyelectron *Complete Situation Report*

---

### Description

Runs all diagnostic checks and provides a comprehensive report of your shinyelectron setup.

### Usage

```
sitrep_shinyelectron(project_dir = ".", verbose = TRUE)
```

### Arguments

| | |
|---|---|
| project_dir | Character. Path to the project directory to check. Default is current directory. |
| verbose | Logical. Whether to print detailed output. Default is TRUE. |

### Value

Invisibly returns a list with all diagnostic information.

### Examples

```
# Complete diagnostic check of the current setup
sitrep_shinyelectron()

# Get results as a list, without printing
results <- sitrep_shinyelectron(verbose = FALSE)

# Check a specific project directory
sitrep_shinyelectron(tempdir())
```

`wizard` *Interactive Configuration Wizard*

## Description

Walks through setup questions and generates a _shinyelectron.yml configuration file for your Shiny app.

## Usage

```
wizard(appdir)
```

## Arguments

| | |
|---|---|
| `appdir` | Character string. Path to the app directory. |

## Value

Invisible path to the generated config file.

## See Also

`init_config()` to create a template config file; `show_config()` to display the merged effective configuration.

## Examples

```
wizard(tempdir())
```

# Index