Package: peeky (via r-universe)

July 3, 2026

Title Download and Extract 'shinylive' Applications

Version 0.1.0

Description Peek into published 'shinylive' applications by retrieving their source code. Supports downloading and extracting 'shiny' application source from 'quarto' documents and standalone 'shinylive' applications.

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

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

License AGPL (>= 3)

Imports jsonlite, httr, rvest, fs, cli

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

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder quarto

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libuv1-dev libxml2-dev libssl-dev

Repository https://coatless-rpkg.r-universe.dev

Date/Publication 2026-06-26 04:37:34 UTC

RemoteUrl https://github.com/coatless-rpkg/peeky

RemoteRef HEAD

RemoteSha 2b1b4e62b263d3460beb98d5c02ed60814a96861

Contents

`peek_quarto_shinylive_app` *Extract Shinylive Applications from Quarto Documents*

Description

Downloads a Quarto document and extracts all embedded Shinylive applications. Applications can be extracted either as separate directories (for individual use) or combined into a new Quarto document (for documentation). The function handles both R and Python Shinylive applications.

Usage

```
peek_quarto_shinylive_app(
  url,
  output_format = c("app-dir", "quarto"),
  output_path
)
```

Arguments

`url`	Character string. URL of the Quarto document containing Shinylive applications. The document should contain code blocks with class `'shinylive-r'` or `'shinylive-python'`.
`output_format`	Character string. Determines how the applications should be extracted. Must be one of: • `"app-dir"`: Creates separate directories for each application • `"quarto"`: Combines all applications into a single Quarto document
`output_path`	Character string. Where to write the extracted applications. A relative path is resolved against the current working directory; an absolute path is used as-is. • For `"app-dir"`: a directory that will hold the extracted applications • For `"quarto"`: the path of the `.qmd` file to create

Value

An object of class `"shinylive_commands"` that provides:

- Pretty-printed instructions via cli
- Commands to run each extracted application

- Information about output locations
- Setup instructions for Quarto documents (if applicable)

Output Formats

The two output formats serve different purposes:

- `"app-dir"`:
  - Creates numbered directories (app_1, app_2, etc.)
  - Each directory contains a complete, runnable application
  - Includes metadata about the original application
  - Best for running or modifying individual applications
- `"quarto"`:
  - Creates a single .qmd file containing all applications
  - Preserves original YAML options and file structure
  - Adds necessary Quarto configuration
  - Best for documentation or sharing multiple applications

Error Handling

The function will error with informative messages if:

- The URL cannot be accessed
- No Shinylive applications are found in the document
- The document structure is invalid

See Also

- `find_shinylive_code()` for the code block extraction
- `write_apps_to_dirs()` for directory output format
- `write_apps_to_quarto()` for Quarto document output format

Examples

```
# Extract as separate application directories under a temporary directory
result <- peek_quarto_shinylive_app(
  "https://quarto-ext.github.io/shinylive",
  output_format = "app-dir",
  output_path = file.path(tempdir(), "quarto_apps")
)

# Combine into a new Quarto document
result <- peek_quarto_shinylive_app(
  "https://quarto-ext.github.io/shinylive",
  output_format = "quarto",
  output_path = file.path(tempdir(), "my_apps.qmd")
)
```

```
# Print will show instructions for running the apps
print(result)
```

`peek_shinylive_app` *Download and Extract Shinylive Applications from URLs*

Description

Downloads and extracts Shinylive applications from various URL sources. The function can handle both standalone Shinylive applications and Quarto documents containing embedded Shinylive applications. It automatically detects the application type and extracts the necessary files.

Usage

```
peek_shinylive_app(url, output_dir)
```

Arguments

`url`	Character string. URL pointing to one of: • A standalone Shinylive application (containing `app.json`) • A directory containing `app.json` • A Quarto document with embedded Shinylive applications
`output_dir`	Character string. Directory where the application files should be extracted. A relative path is created under the current working directory; an absolute path is used as-is. The directory will be created if it doesn't exist.

Details

The function follows these steps:

1. Downloads and analyzes the content at the provided URL
2. Determines if the content is a Quarto document or standalone application
3. For Quarto documents:
   - Extracts all embedded Shinylive applications
   - Creates separate directories for each application
4. For standalone applications:
   - Locates and validates the `app.json` file
   - Extracts all application files to the specified directory

Value

An object containing the extracted application information:

- For standalone apps: Object of class `"standalone_shinylive_app"`
- For Quarto documents: Object of class `"quarto_shinylive_apps"`

Both object types implement custom print methods that display:

- Application type (R or Python)
- Commands to run the application
- List of extracted files
- Output directory location

URL Resolution

The function attempts several strategies to find app.json:

- Direct use of the provided URL
- Appending `"app.json"` to the URL
- Checking the parent directory

Error Handling

The function will error with informative messages if:

- The URL cannot be accessed
- No valid Shinylive application is found
- The `app.json` structure is invalid

See Also

- `peek_quarto_shinylive_app()` for handling Quarto documents specifically
- `peek_standalone_shinylive_app()` for handling standalone applications

Examples

```
# Write the extracted files to the session's temporary directory
url <- "https://tutorials.thecoatlessprofessor.com/convert-shiny-app-r-shinylive/"
app <- peek_shinylive_app(url, output_dir = file.path(tempdir(), "my_extracted_app"))

# Download from a Quarto document
apps <- peek_shinylive_app(
  "https://quarto-ext.github.io/shinylive/",
  output_dir = file.path(tempdir(), "my_extracted_apps")
)
```

`peek_standalone_shinylive_app` *Download and Extract a Standalone Shinylive Application*

Description

Downloads and extracts a standalone Shinylive application from a URL. The function locates the application's `app.json` file, validates its structure, and extracts all application files to a local directory. Works with both R and Python Shinylive applications.

Usage

```
peek_standalone_shinylive_app(url, output_dir)
```

Arguments

`url`	Character string. URL pointing to either: • A Shinylive app.json file directly • A directory containing app.json The function will automatically append `"app.json"` to directory URLs.
`output_dir`	Character string. Directory where the application files should be extracted. A relative path is created under the current working directory; an absolute path is used as-is. Will be created if it doesn't exist. If the directory already exists, files may be overwritten.

Value

An object of class `"standalone_shinylive_app"` containing:

- List of extracted files and their contents
- Source URL of the application
- Output directory location

The object has a custom print method that displays:

- Application type (R or Python)
- Command to run the application
- List of extracted files by type
- File locations

File Structure

A valid Shinylive application should have an app.json file containing:

- At least one application file (e.g., `app.R` or `app.py`)
- Optional supporting files (e.g., data files, `requirements.txt`)
- File metadata including name, content, and type

Error Handling

The function will error with informative messages if:

- No `app.json` file is found at the URL
- The `app.json` file has invalid structure
- The `app.json` file cannot be downloaded
- Required application files are missing

See Also

- `find_shinylive_app_json()` for `app.json` validation
- `write_standalone_shinylive_app()` for file extraction
- `peek_shinylive_app()` for a more general-purpose download function

Examples

```
# Download from a direct app.json URL into a temporary directory
app <- peek_standalone_shinylive_app(
  "https://tutorials.thecoatlessprofessor.com/convert-shiny-app-r-shinylive/app.json",
  output_dir = file.path(tempdir(), "standalone_app")
)

# Download from a directory URL (app.json will be appended)
app <- peek_standalone_shinylive_app(
  "https://tutorials.thecoatlessprofessor.com/convert-shiny-app-r-shinylive/",
  output_dir = file.path(tempdir(), "my_local_app")
)

# Print shows how to run the application
print(app)
```

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

Description

Print method for `quarto_shinylive_apps` objects

Usage

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

Arguments

x	Object of class "quarto_shinylive_apps"
...	Additional arguments passed to print

Value

Invisibly returns the original object of class quarto_shinylive_apps. Primarily called for its side effect of displaying formatted information about:

- Application types (R/Python)
- Commands to run each application
- File contents and organization
- Output locations

---

print.standalone_shinylive_app

Print method for standalone_shinylive_app *objects*

Description

Print method for standalone_shinylive_app objects

Usage

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

Arguments

x	Object of class "standalone_shinylive_app"
...	Additional arguments passed to print

Value

Invisibly returns the original object of class standalone_shinylive_app. Primarily called for its side effect of displaying formatted information about:

- Application type (R/Python)
- Command to run the application
- List of extracted files by type
- File locations

`write_apps_to_dirs` *Write Multiple Shinylive Applications to Separate Directories*

Description

Takes a list of parsed Shinylive applications and writes each to its own numbered subdirectory. Creates consistently numbered directories with proper padding (e.g., `app_01`, `app_02`) and preserves all application files and metadata.

Usage

```
write_apps_to_dirs(apps, base_dir)
```

Arguments

`apps`	List of parsed Shinylive applications. Each application should contain: • `engine`: Character string identifying the app type (`"r"` or `"python"`) • `options`: List of YAML-style options from the original code block • `files`: Named list of file definitions, each containing: – `name`: Character string of the file name – `content`: Character string of the file content – `type`: Character string indicating the file type
`base_dir`	Character string. Base directory where application subdirectories should be created. Will be created if it doesn't exist.

Details

The function performs these steps:

1. Creates the base directory if needed
2. Calculates proper padding for subdirectory numbers
3. For each application:
   - Creates a padded, numbered subdirectory (e.g., `app_01`, `app_02`)
   - Writes all application files, preserving directory structure
   - Creates a metadata JSON file with engine and options info

Value

No return value, called for its side effect of writing one application subdirectory per app (plus a metadata file) under `base_dir`.

Directory Structure

Creates a directory structure like:

```
base_dir/
|-- app_01/
|   |-- app.R
|   |-- data/
|   |   `-- example.csv
|   `-- shinylive_metadata.json
|-- app_02/
|   |-- app.py
|   `-- shinylive_metadata.json
`-- ...
```

Metadata File

Each directory includes a `shinylive_metadata.json` file containing:

```
{
  "engine": "r",
  "options": {
    "viewerHeight": 500,
    "...": "..."
  }
}
```

See Also

- `padding_width()` for directory number padding calculation
- `write_apps_to_quarto()` for alternative Quarto output format

Examples

```
# Example apps list structure
apps <- list(
  list(
    engine = "r",
    options = list(viewerHeight = 500),
    files = list(
      "app.R" = list(
        name = "app.R",
        content = "library(shiny)\n...",
        type = "text"
      )
    )
  ),
  list(
    engine = "python",
    options = list(),
    files = list(
```

```
      "app.py" = list(
        name = "app.py",
        content = "from shiny import App\n...",
        type = "text"
      )
    )
  )
)

write_apps_to_dirs(apps, file.path(tempdir(), "extracted_apps"))
```

`write_apps_to_quarto` *Write Shinylive Applications to a Quarto Document*

Description

Converts a list of parsed Shinylive applications into a single Quarto document. Creates a properly formatted .qmd file with YAML frontmatter, organized sections for each application, and correctly formatted code blocks with all necessary markers and options.

Usage

```
write_apps_to_quarto(apps, qmd_path)
```

Arguments

`apps`	List of parsed Shinylive applications. Each application should contain: • `engine`: Character string identifying the app type (`"r"` or `"python"`) • `options`: List of YAML-style options from the original code block • `files`: Named list of file definitions, each containing: – `name`: Character string of the file name – `content`: Character string of the file content – `type`: Character string indicating the file type
`qmd_path`	Character string. Path where the Quarto document should be written. Should end with `.qmd` extension. Parent directory will be created if it doesn't exist.

Details

The function performs these steps:

1. Creates YAML frontmatter with required Quarto settings
2. For each application:
   - Adds a section header with application number
   - Creates a code block with appropriate engine (`shinylive-r`/`shinylive-python`)
   - Converts and adds all application options
   - Adds file markers and content for each file
   - Properly closes the code block
3. Writes the complete document to the specified path

Value

No return value, called for its side effect of writing a Quarto document to `qmd_path`.

Document Structure

Creates a Quarto document with this structure:

````
---
title: Extracted Shinylive Applications
filters:
  - shinylive
---

# Shinylive Applications

## Application 1

```{shinylive-r}
#| viewerHeight: 500
## file: app.R
## type: text
library(shiny)
...
```

## Application 2
...
````

Option Formatting

Options are converted to YAML format based on their type:

- Logical: `#| option: true` or `#| option: false`
- Numeric: `#| option: 500`
- Character:
  - Single: `#| option: "value"`
  - Vector: `#| option: ["value1", "value2"]`

See Also

- `write_apps_to_dirs()` for alternative directory output format

Examples

```
# Example apps list structure
apps <- list(
  list(
    engine = "r",
```

```
    options = list(
      viewerHeight = 500,
      fullWidth = TRUE
    ),
    files = list(
      "app.R" = list(
        name = "app.R",
        content = "library(shiny)\n...",
        type = "text"
      )
    )
  )
)

write_apps_to_quarto(apps, file.path(tempdir(), "applications.qmd"))
```

`write_file_content` *Write File Content in a Shinylive App to Disk*

Description

Writes file content extracted from Shinylive applications to disk, handling both text and binary content appropriately. Creates any necessary parent directories and ensures proper encoding of content. For binary files, automatically decodes the base64-encoded content before writing.

Usage

```
write_file_content(content, file_path, type = "text")
```

Arguments

`content`	Character string containing the file content. For binary files, this should be base64-encoded content. For text files, this should be the raw text content.
`file_path`	Character string specifying the path where the file should be written. Parent directories will be created if they don't exist.
`type`	Character string specifying the file type, either "text" (default) or "binary". Binary files are assumed to be base64 encoded, as this is the standard format for binary content in Shinylive applications.

Details

The function handles two types of content:

- Text files (`type = "text"`):
  - Content is converted to UTF-8 encoding using `enc2utf8()`
  - Written using `writeLines()` with `useBytes = TRUE`
- Binary files (`type = "binary"`):

- Content is decoded from base64 using `jsonlite::base64_dec()`
- Written as raw binary data using `writeBin()`

Parent directories in the file path are automatically created if they don't exist using `fs::dir_create()` with `recurse = TRUE`.

Value

Invisible NULL, called for its side effect of writing a file to disk.

Examples

```
# Write a text file into a temporary directory
write_file_content(
  content = "library(shiny)\n\nui <- fluidPage()",
  file_path = file.path(tempdir(), "app", "app.R"),
  type = "text"
)

# Write a base64-encoded image
b64img <- paste0(
  "iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAYAAAAfFcSJAAAA",
  "DUlEQVR42mP8z8BQDwAEhQGAhKmMIQAAAABJRU5ErkJggg=="
)
write_file_content(b64img, file.path(tempdir(), "test.png"), type = "binary")
```

`write_standalone_shinylive_app`
Write Standalone Shinylive Application Files from JSON Data

Description

Extracts files from parsed Shinylive `app.json` data and writes them to a specified directory. Creates a standalone application object containing metadata and commands for running the application.

Usage

```
write_standalone_shinylive_app(json_data, source_url, output_dir)
```

Arguments

`json_data`	List. Parsed JSON data from a Shinylive `app.json` file. Each element should be a list containing: • `name`: Character string of the file name • `content`: Character string of the file content • `type`: Character string indicating the file type
`source_url`	Character string. The original URL from which the `app.json` was downloaded. Used for reference and provenance tracking in the returned object.

`output_dir` Character string. Directory where application files should be extracted. A relative path is created under the current working directory; an absolute path is used as-is. Will be created if it doesn't exist. Existing files in this directory may be overwritten.

Details

The function performs these steps:

1. Creates the output directory if it doesn't exist
2. Iterates through each file in the JSON data
3. Writes each file to the output directory, preserving names
4. Creates a standalone application object with metadata

File paths are created relative to the output directory. Parent directories in file paths will be created as needed.

Value

An object of class `"standalone_shinylive_app"` containing:

- `files`: List of extracted files and their metadata
- `output_dir`: Path to the directory containing extracted files
- `source_url`: Original URL of the application

File Structure

Expected JSON data structure:

```
[
  {
    "name": "app.R",
    "content": "library(shiny)\n...",
    "type": "text"
  },
  {
    "name": "data/example.csv",
    "content": "x,y\n1,2\n...",
    "type": "text"
  }
]
```

See Also

- `create_standalone_shinylive_app()` for object creation
- `validate_app_json()` for JSON data validation

Examples

```
# Example JSON data structure
json_data <- list(
  list(
    name = "app.R",
    content = "library(shiny)\nui <- fluidPage()\n...",
    type = "text"
  ),
  list(
    name = "data.csv",
    content = "col1,col2\n1,2\n3,4",
    type = "text"
  )
)

app <- write_standalone_shinylive_app(
  json_data,
  "https://example.com/app.json",
  file.path(tempdir(), "my_app")
)
```

Index