Package: projectData.monocle3 (via r-universe)

November 28, 2026

Title 'projectData' Adapter for Disk-Backed 'monocle3' Cell Data Sets

Version 0.1.0

Description A loader for the 'projectData' adapter framework. It restores a saved 'monocle3' cell_data_set whose counts are a 'BPCells' on-disk matrix from an installed data package, on first access, without attaching 'monocle3' or 'projectData' to the search path. Restored matrix files are placed in the working directory allocated by 'projectData'. For data package authors, it saves a cell_data_set into a package source tree, converting supported in-memory counts to a 'BPCells' on-disk matrix, and registers it in the package's 'projectData' manifest.

License MIT + file LICENSE

URL https://github.com/blaserlab/projectData.monocle3

BugReports https://github.com/blaserlab/projectData.monocle3/issues

Depends R (>= 4.1.0)

Imports methods, projectData (>= 0.1.0), stats, utils

Suggests BPCells, DelayedArray, knitr, Matrix, monocle3, pkgload, rmarkdown, S4Vectors, SingleCellExperiment, SummarizedExperiment, testthat (>= 3.2.0), withr (>= 2.5.0)

Remotes github::bnprks/BPCells/r, github::cole-trapnell-lab/monocle3

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Encoding UTF-8

Roxygen list(markdown = TRUE)

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

Date/Publication 2026-11-27 23:53:29 UTC

RemoteUrl https://github.com/blaserlab/projectData.monocle3

RemoteRef HEAD

RemoteSha 956ca7c9ff621ba2776fa2df5e62107ddce0562f

Contents

projectData_loaders 2
save_monocle_cds 3
write_monocle_manifest 7

Index **11**

`projectData_loaders` *Loader declaration for disk-backed Monocle 3 CDS objects*

Description

`projectData_loaders()` is the adapter export that projectData looks for when a manifest names `AdapterPackage: projectData.monocle3`. It returns one `projectData::data_loader()` with id `projectData.monocle3::cds`. projectData loads this package's namespace (it does not attach it) the first time a resource that needs the loader is accessed, so users never need `library(projectData.monocle3)`.

Usage

```
projectData_loaders()
```

Details

Each resource is one directory written by `monocle3::save_monocle_objects(cds, directory, archive_control = list(archive_type = "none"))` from a CDS whose counts are a BPCells on-disk matrix. The loader:

- `validate` checks the directory structure from `file_index.rds` and file existence only. It never restores the CDS or reads matrix data.
- `open` calls `monocle3::load_monocle_objects()` on the installed directory with `matrix_control$matrix_path` set to the unique `paths$work_dir` allocated by projectData, and returns a named list with the single CDS.

Value

A `projectData_loader` object.

Filesystem safety

monocle3 and BPCells copy the matrix directory by following whatever is inside it, so the loader refuses a resource before the native load when anything inside the saved directory is a symbolic link (internal, escaping or dangling), when an indexed path is absolute, uses `..`, or is of the wrong type, or when the BPCells directory is not flat or lacks the files BPCells writes. The directory itself, and the library it lives in, may be reached through symbolic links. The same check runs in `validate` and again in `open` (a listing of about ten entries; no matrix data is read), and `open` refuses a work directory that lies inside the installed resource. Resources written by `save_monocle_cds()` never contain links.

What the restored object references

Monocle copies the BPCells matrix directory (and a second, row-ordered copy it derives) into uniquely named subdirectories of `paths$work_dir`, so the counts do not depend on the installed package. The work directory lives in the session temporary directory and is removed by monocle3 when R exits. The UMAP nearest-neighbour index, however, is memory-mapped by Annoy directly from the installed file. On Linux and macOS the mapping survives deletion of the file; on Windows an installed copy cannot be removed while it is mapped. See `vignette("storage")` for details.

See Also

`projectData::data_loader()`, `projectData::data_loader_adapter()`

`save_monocle_cds` *Save a Monocle 3 CDS into a data package source tree*

Description

`save_monocle_cds()` is a **package-authoring** operation. With one call it turns a `cell_data_set` into everything a data package needs to ship it as a disk-backed resource that `projectData::data_attach()` opens through this package's loader:

Usage

```
save_monocle_cds(cds, name, package_dir = ".", overwrite = FALSE)
```

Arguments

`cds`	A Monocle 3 `cell_data_set`.
`name`	A single, syntactically valid R object name. It names the saved directory, the placeholder, the manifest resource and the object that `projectData::data_attach()` provides.
`package_dir`	Root of an existing R package *source* directory.
`overwrite`	Replace an existing saved dataset and its placeholder?

Details

```
my.datapkg/
|-- data/
|   `-- <name>.rda              placeholder (written here)
`-- inst/
    |-- extdata/
    |   `-- <name>/             native Monocle 3 serialization (written here)
    `-- projectData/
        `-- manifest.dcf        entry for <name> (created or updated here)
```

Nothing is written to the working directory, `R/data.R` is never edited, and the caller's `cds` is not modified.

Value

Invisibly, a list with the `name`, `package`, the paths of the saved `directory`, `placeholder` and `manifest`, `manifest_status` (`"added"`, `"updated"` or `"unchanged"`), whether existing files were `replaced`, and the names of any `removed_stale` manifest entries.

Serialization

The CDS is written with `monocle3::save_monocle_objects()` using `archive_control = list(archive_type = "none")` (no tar archive), so the directory is installed as is. The loader restores BPCells on-disk counts, so the counts of the saved CDS always are such a matrix; `save_monocle_objects()` copies the matrix directory into the saved directory. Everything else, including metadata, reductions, their nearest-neighbour indexes and any in-memory alternative assays and alternative experiments, is stored in `cds_object.rds` and the index files that Monocle writes.

BPCells conversion

Conversion is automatic; there is nothing to switch on. The `counts` assay is inspected first (with `methods::is()` against BPCells' `IterableMatrix` class and `BPCells::all_matrix_inputs()` to find the matrices underneath):

- **Counts that are already an on-disk BPCells matrix** (a `MatrixDir`, possibly lazily subset or transformed) are *not* converted again. They are saved as they are.
- **In-memory counts** (a `dgCMatrix` or `dgTMatrix`, or a BPCells matrix held in memory) are converted on a *separate working copy* of the CDS with `monocle3::convert_counts_matrix()` (`matrix_class = "BPCells"`, `matrix_mode = "dir"`). The intermediate matrix goes to a temporary directory of the session (never the working directory) that is removed once the CDS is saved, and the caller's `cds` is not modified. Cell and feature names must be unchanged by the conversion, which is checked. Monocle adds its internal, derived `counts_row_order` assay to the converted copy; it is a row-oriented copy of the counts, not a separate assay, and is regenerated whenever the CDS is loaded.
- **Any other representation** (for example a dense `matrix`, a `DelayedMatrix` or an HDF5-backed BPCells matrix) is an error raised before anything is written, naming the class found.

Only `counts` is converted; other assays are left alone. `save_monocle_objects()` itself writes the CDS as HDF5 instead of rds if any assay is HDF5-backed (such a save is rejected here because the loader does not support it).

Alternative assays

Assays other than `counts` (and the derived `counts_row_order`, which Monocle regenerates when loading) are not deliberately removed or converted. In-memory alternative assays are stored in `cds_object.rds` and have been verified to survive a save and a load through the adapter. An alternative assay that is itself a BPCells matrix is **not** supported: Monocle saves only a reference to its original directory, which will not exist for the people using the package. Saving is therefore **refused** with an error of class `projectData.monocle3_external_assay` (also `projectData.monocle3_save_error`) before anything is written. The error lists every offending assay with its location, such as `assay 'spliced'` or `altExp 'ADT' / altExp 'normalized' / assay 'counts'`. Convert such assays to an in-memory matrix, or remove them, before saving; this function never converts them because that could use a lot of memory.

The check is recursive: the assays of every alternative experiment (`SingleCellExperiment::altExp()`), at any depth, are inspected, and unlike at the top level there is no exception for `counts` or `counts_row_order` there. An alternative experiment with in-memory assays is supported. No matrix values are read; BPCells matrices are judged by the classes of the matrices underneath, and only `MatrixMem` (in-memory) ones are accepted; a BPCells wrapper around an R matrix (`Iterable_dgCMatrix_wrapper`) is refused too, as it is not known to be self-contained. `DelayedArray` assays are also refused, because whether they are file-backed is not examined and their serialization is untested. Other exotic representations are not checked and are not guaranteed to work.

Overwriting and failure

Before anything is written, the CDS, `name`, `package_dir`, installed dependencies, existing targets and the existing manifest are checked. Existing targets are replaced only with `overwrite = TRUE`, and only when this package can show it wrote them. A directory that merely contains files named like a Monocle save (`file_index.rds`, `cds_object.rds`, `bpcells_matrix_dir`) is not thereby owned.

An existing `inst/extdata/<name>` is replaceable when at least two of these three independent pieces of evidence are present:

- a `file_index.rds` that is a native Monocle index (written by `save_monocle_objects()`, with a well-formed file listing; the listed files need not all exist);
- a manifest entry for `<name>` with this package's loader, whose `Source` and `Objects` agree with it;
- a recognized placeholder `data/<name>.rda`, written by this package or the historical `"This is a placeholder object."`.

The directory must also hold nothing the evidence does not account for: only `file_index.rds` and the files it lists (or, when the index is unusable, `cds_object.rds` and `bpcells_matrix_dir`). The outcomes:

Existing `inst/extdata/<name>`
: none (or only a recognized placeholder)
: valid or damaged, two or more kinds of evidence
: native save, or marker files, with less evidence (for example a bare `bpcells_matrix_dir`, a malformed `file_index.rds`, or a
: contains other files, or is a file
: manifest entry of another loader
: empty directory

So a save made with this package whose manifest was deleted, or whose placeholder was removed, can still be replaced, and so can a damaged one (a deleted `cds_object.rds`) as long as two kinds of evidence remain. A manifest entry of this loader that disagrees with the directory is ambiguous. A `data/<name>.rda` that holds anything other than a recognized placeholder is never replaced (`projectData.monocle3_placeholder_not_recognized`). A name already used by a manifest entry of another loader, or declared as an object by another resource, is an error. All of these errors are raised before any file is staged or written, and none of them suggests `overwrite = TRUE` as a remedy when ownership is not established; there is no `force` argument. Replacing a damaged resource reports what was replaced and why it was allowed.

The manifest

The manifest is reconciled with the package exactly as by `write_monocle_manifest()`, after the CDS and its placeholder are in place: the new resource is added or updated, entries of this package whose saved directory or placeholder no longer exist are removed as stale (so deleting a saved directory by hand never blocks the next save), other valid saved resources that are not yet registered are added, and entries of other adapters are kept as they are. Nothing but the manifest entries is deleted; saved directories and placeholders are never removed by reconciliation. If an entry of another adapter is invalid, nothing is saved.

The new files are first created in a hidden staging directory in the package root (`.projectData.monocle3-stage-*`), where the saved directory is validated with the loader's own checks and the placeholder is read back. Only then are the previous files moved aside, the new ones renamed into place and the manifest written (atomically, by `projectData::data_manifest_write()`). If any of those steps fails, the new files are removed and the old ones are moved back. The staging directory is removed afterwards. This is **not** a filesystem transaction: renames are individually atomic, but a crash of the R process (or a failing rollback) can leave the staging directory behind, and the error then names where the previous files are. Saving itself can take a while for a large matrix; an interruption during that phase touches only the staging directory.

`file_index.rds`, written by Monocle, records the staging directory as the save location. The loader does not use it.

Filesystem safety

monocle3 and BPCells follow symbolic links when they copy a matrix directory and do not check where paths lead, so this function does. Before anything is staged, the directory of each on-disk BPCells matrix under `counts` is listed (no file is read, one flat directory per matrix) and refused, with an error of class `projectData.monocle3_unsafe_backing` (also `projectData.monocle3_save_error`), if it is missing, holds a nested directory, lacks the files BPCells writes (`version`, `storage_order`, `shape`, `row_names`, `col_names`, `idxptr` and `val` or `val_data`), or contains a symbolic link of any kind (internal, escaping or dangling): a link would put the bytes of some other file into your package. The directory itself may be reached through links, and so may the package directory and parents such as `inst/extdata`. An existing `inst/extdata/<name>` that is a link, or contains one, is never replaced (`projectData.monocle3_not_owned`), and a dangling link where a placeholder would go is refused too. The staging directory is created fresh (never through an existing path), is checked to lie inside the package, and is removed without following links. The saved tree is validated with the

loader's own checks before anything is moved into the package. Entry types are judged with base R, which cannot distinguish a FIFO or device file from a regular file; those are not detected.

Documentation

Document the dataset with the usual roxygen2 convention in `R/data.R`, a block followed by the quoted name, for example `"cds_main_adults"`. This function does not edit `R/data.R`; it only tells you when no such topic is found.

Using the saved data

After installing the package, `data("<name>", package = "<pkg>")` only loads the placeholder, which prints instructions. The actual CDS is obtained with `projectData::data_attach("<pkg>")` followed by `<name>`.

Conditions

Errors raised before anything is written have class `projectData.monocle3_save_error` plus one of `_exists` (needs `overwrite = TRUE`; also `_damaged` when the existing resource is damaged), `_not_owned`, `_ambiguous_ownership`, `_conflict`, `_placeholder_not_recognized`, `_external_assay`, `_unsafe_backing` or `_manifest_error`. A `package_dir` that is not a package source directory gives `projectData.monocle3_package_dir_error`. If the commit fails, the previous files are restored and the error says so.

See Also

`write_monocle_manifest()` to register directories saved by other means, `projectData_loaders()`; `vignette("authoring-monocle3-data")`

Examples

```
## Not run:
# In a data package source directory:
projectData.monocle3::save_monocle_cds(cds_main_adults, "cds_main_adults")

# Then document it in R/data.R, run devtools::document(), install, and:
projectData::data_attach("myDataPackage")
cds_main_adults

## End(Not run)
```

`write_monocle_manifest` *Register saved Monocle CDS directories in a data package's manifest*

Description

`write_monocle_manifest()` is a **package-authoring** operation for developers of data packages. It finds saved Monocle 3 `cell_data_set` directories in a package source tree and creates or updates `inst/projectData/manifest.dcf` so that projectData resolves each one through the `projectData.monocle3::cds` loader. Installed data packages never generate manifests at runtime; the manifest is written once, here, and shipped with the package.

Usage

```
write_monocle_manifest(package_dir = ".")
```

Arguments

`package_dir`	Path to the root of a data-package *source* directory (the one with `DESCRIPTION`). Defaults to the working directory.

Value

Invisibly, a data frame with one row per resource considered: `resource`, `status` (`"added"`, `"updated"`, `"unchanged"`, `"removed"`, `"skipped"` or `"conflict"`) and `detail`. Attributes `manifest` (the path) and `written` (whether the file was written) are attached. Skipped resources and conflicts also raise warnings of class `projectData.monocle3_skipped` and `projectData.monocle3_conflict`.

Expected layout

```
my.datapkg/
|-- DESCRIPTION
|-- data/
|   |-- human_cds.rda        placeholder for human_cds
|   `-- ordinary_data.rda    ordinary dataset, ignored here
`-- inst/
    |-- extdata/
    |   `-- human_cds/       from monocle3::save_monocle_objects()
    `-- projectData/
        `-- manifest.dcf     created or updated
```

Each directory directly below `inst/extdata/` that `monocle3::save_monocle_objects(archive_control = list(archive_type = "none"))` wrote is a candidate; its name is the resource name and the name of the object it provides. A directory is registered only if it passes the same structural checks the loader applies when the package is used (a `file_index.rds` written by `save_monocle_objects()`, one rds CDS, one BPCells matrix directory, every listed file present). Directories that are not Monocle saves are ignored; directories that look like one but fail the checks are skipped with the reason.

Placeholders

Every CDS needs a matching `data/<name>.rda` placeholder, so that the object is discoverable with `utils::data()` and can be documented in the usual way. The file is inspected without running anything: its serialized header is parsed to read the object name and, for a small character vector, its content. The historical string `"This is a placeholder object."` is recognized, and so is the classed placeholder that `save_monocle_cds()` writes (class `projectData_monocle3_placeholder`, whose

object_name and package attributes must match). A CDS without a placeholder is skipped with a warning (and, if it already had a manifest entry, that entry is removed as stale). A .rda that holds a real dataset, or that cannot be inspected, is never treated as a placeholder, and no registration is created for it. This function never creates or modifies placeholder files.

Reconciliation

The saved directories and placeholders currently in the package are the source of truth for the entries this package owns, that is, those whose Loader is projectData.monocle3::cds. The existing manifest is reconciled with them, so a directory deleted by hand during development never leaves a manifest that has to be repaired manually:

Existing entry	Action
Monocle resource with a valid directory and placeholder	kept, or updated where it differs
Monocle entry whose saved directory is gone	removed as stale
Monocle entry whose placeholder is missing	removed as stale
New valid directory with a placeholder	added
Entry of any other loader or adapter	kept unchanged, with all its fields
Resource name or object name already used by another adapter	**conflict**: reported, never taken over
Monocle entry whose directory exists but is invalid	kept; the call fails because the manifest would not be valid

Reconciliation only ever changes manifest entries. **It never deletes a saved CDS directory or a placeholder**, and removing an entry does not touch the files it described.

The existing manifest is first read with the structural checks needed to reconcile it safely (readable, known fields, ManifestVersion: 1, a Resource and Loader in every record, unique resources, well-formed Options), without requiring its sources to exist. A manifest that fails these checks stops the call and is left untouched. The complete proposed manifest is then validated by projectData::data_manifest_validate()'s rules (through projectData::data_manifest_write(), including this package's loader) and written only if that succeeds. An invalid entry of another adapter is never repaired or removed: the error reports it, and the manifest is left as it was.

An existing Monocle entry is updated only where it differs: the loader, source, object, adapter and recorded class are set to the values described below, a missing help topic is filled in, and a Title or Options you added by hand are kept. New entries are added at the end.

Each new entry has the fields Resource, Loader: projectData.monocle3::cds, Source: extdata/<name> (the installed layout, never an absolute path or inst/...), Objects, AdapterPackage: projectData.monocle3, RecordedClass: cell_data_set and HelpTopic. AdapterExport: projectData_loaders is the default, which projectData::data_manifest_write() leaves out of the file; it reads back the same either way.

The manifest is only rewritten when something was added, updated or removed, so running the function again changes nothing and prints nothing. When it is rewritten, the file is produced by projectData::data_manifest_write(), which validates every entry, checks that the text reads back unchanged, and replaces the manifest with a single rename. A failure leaves the previous manifest in place. As that writer emits every record in its standard field order, hand-chosen field order or spacing in entries you did not change is normalized when the manifest is rewritten. If no entry remains at all, the manifest file is removed (and its directory, if that is left empty), because a manifest without resources carries no information.

No CDS is loaded

Discovery reads only `file_index.rds`, directory listings and the header of each placeholder. No CDS is restored, no BPCells matrix is opened and no working directory is created. `monocle3` and `BPCells` need not be installed.

Conditions

A manifest that cannot be read or validated, or an invalid entry of another adapter, stops the call with an error of class `projectData.monocle3_manifest_error` and leaves the manifest untouched. A bad `package_dir` gives `projectData.monocle3_package_dir_error`. Skipped resources and conflicts raise warnings of class `projectData.monocle3_skipped` and `projectData.monocle3_conflict`.

See Also

`projectData_loaders()`, `projectData::data_manifest_write()`

Examples

```
## Not run:
# From the root of a data package that already contains saved CDS
# directories under inst/extdata/ and matching placeholders in data/:
projectData.monocle3::write_monocle_manifest(package_dir = ".")

## End(Not run)
```

Index

projectData::data_attach(), *3*
projectData::data_loader(), *2*, *3*
projectData::data_loader_adapter(), *3*
projectData::data_manifest_validate(), *9*
projectData::data_manifest_write(), *9*, *10*
projectData_loaders, 2
projectData_loaders(), *7*, *10*

save_monocle_cds, 3
save_monocle_cds(), *3*, *8*

utils::data(), *8*

write_monocle_manifest, 7
write_monocle_manifest(), *6*, *7*