Package 'updateObject'

October 18, 2024

Title Find/fix old serialized S4 instances

Description A set of tools built around updateObject() to work with old serialized S4 instances. The package is primarily useful to package maintainers who want to update the serialized S4 instances included in their package. This is still work-in-progress.

biocViews Infrastructure, DataRepresentation

URL https://bioconductor.org/packages/updateObject

BugReports https://github.com/Bioconductor/updateObject/issues

Version 1.9.1

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.2.0), methods, BiocGenerics (>= 0.51.1), S4Vectors

Imports utils, digest

Suggests GenomicRanges, SummarizedExperiment, InteractionSet, SingleCellExperiment, MultiAssayExperiment, BiSeq, testthat, knitr, rmarkdown, BiocStyle

SystemRequirements git

VignetteBuilder knitr

Collate capture_message.R updateSerializedObjects.R bump_pkg_version.R find_python.R git-utils.R updatePackageObjects.R updateBiocPackageRepoObjects.R

git_url https://git.bioconductor.org/packages/updateObject

git_branch devel

git_last_commit 2e8dfa1

git_last_commit_date 2024-09-02

Repository Bioconductor 3.20

Date/Publication 2024-10-17

Author Hervé Pagès [aut, cre]

Maintainer Hervé Pagès <hpages.on.github@gmail.com>

Contents

`bump_pkg_version` *Bump the version of a package*

Description

Use `bump_pkg_version()` to bump the version of a package.

Usage

```
bump_pkg_version(pkgpath=".", update.Date=FALSE)
```

Arguments

`pkgpath`	The path (as a single string) to the top-level directory of an R package source tree.
`update.Date`	`TRUE` or `FALSE`. If `TRUE` then the Date field (if present) gets updated to the current date.

Value

An invisible `NULL`.

See Also

- `updatePackageObjects` which uses `bump_pkg_version` internally when `bump.Version=TRUE`.
- `git_commit` for an example.

Examples

```
## Create dummy R package:
create_dummy_pkg <- function(desc, pkgpath) {
    dir.create(pkgpath)
    descpath <- file.path(pkgpath, "DESCRIPTION")
    write.dcf(rbind(desc), descpath)
    descpath
}

pkgname <- "Dummy"
desc <- c(
    Package=pkgname,
    Title="Not a real package",
    Description="I'm not real u know.",
    Version="3.0.9",
```

```
    Date="1969-07-20"
)
pkgpath <- file.path(tempdir(), pkgname)
descpath <- create_dummy_pkg(desc, pkgpath)

## Bump its Version:
bump_pkg_version(pkgpath)
cat(readLines(descpath), sep="\n")

## Bump its Version again and set Date to current date:
bump_pkg_version(pkgpath, update.Date=TRUE)
cat(readLines(descpath), sep="\n")

## Throw it away:
unlink(pkgpath, recursive=TRUE)
```

`git-utils` *Convenience Git-related utility functions used by updateBiocPackageRepoObjects()*

Description

`prepare_git_repo_for_work()` and `git_commit()` are used internally by `updateBiocPackageRepoObjects()` to perform Git operations.

Usage

```
prepare_git_repo_for_work(repopath=".", branch=NULL, git=NULL,
                          use.https=FALSE)

git_commit(repopath=".", commit_msg, push=FALSE,
           git=NULL, user_name=NULL, user_email=NULL)
```

Arguments

`repopath`	The path (as a single string) to the local Git repository of a Bioconductor package. If the specified path exists, `prepare_git_repo_for_work()` will check that it's a workable Git repo (i.e. contains no uncommitted changes). If that's the case then it will call `git pull` on it, otherwise it will return an error. If the specified path does not exist, `prepare_git_repo_for_work()` will try to infer the package name from `repopath` and clone it from `git.bioconductor.org`.
`branch`	The branch (as a single string) of the Git repository to work on. If `NULL`, then the current branch is used (if `repopath` already exists) or the default branch is used (if `repopath` does not exist and needs to be cloned).
`git`	The path (as a single string) to the git command if it's not in the PATH.
`use.https`	By default, `git clone git@git.bioconductor.org:packages/MyPackage` is used to clone a package repo from the Bioconductor Git server. Note that this works only for authorized maintainers of package **MyPackage**. By setting `use.https` to `TRUE`, the package will be cloned instead from `https://git.bioconductor.org:pack`

which should work for anybody, but then pushing back the changes to the package won't be possible.

`commit_msg` The Git commit message.

`push` Whether to push the changes or not. Changes are committed but not pushed by default. You need push access to the package Git repository at `git.bioconductor.org` in order to use `push=TRUE`.

`user_name, user_email`
Set the Git user name and/or email to use for the commit. This overrides the Git user name and/or email that the git command would otherwise use. See the COMMIT INFORMATION section in `system2("git", c("commit", "--help"))` for the details about where the git command normally takes this information from.

Value

`prepare_git_repo_for_work()` returns `FALSE` if the supplied path already exists, and `TRUE` if it didn't exist and needed to be cloned.

`git_commit()` returns an invisible `NULL`.

See Also

- `updateBiocPackageRepoObjects` which uses `prepare_git_repo_for_work` and `git_commit` internally.
- `bump_pkg_version`.

Examples

```
repopath <- file.path(tempdir(), "IdeoViz")

## We must use HTTPS access to clone the package because we are
## not maintainers of the IdeoViz package. A more realistic situation
## would be to use prepare_git_repo_for_work() on a package that we
## maintain, in which case 'use.https=TRUE' would not be needed:
prepare_git_repo_for_work(repopath, use.https=TRUE)

bump_pkg_version(repopath, update.Date=TRUE)

git_commit(repopath, commit_msg="version bump", push=FALSE)

unlink(repopath, recursive=TRUE)
```

---

updateBiocPackageRepoObjects

Update the serialized objects contained in a Bioconductor package Git repository or in a set of Bioconductor package Git repositories

---

Description

`updateBiocPackageRepoObjects()` and `updateAllBiocPackageRepoObjects()` are wrappers to `updatePackageObjects()` and `updateAllPackageObjects()` that take care of committing and pushing the changes made to the package(s).

Usage

```
updateBiocPackageRepoObjects(repopath=".", branch=NULL, filter=NULL,
        commit_msg=NULL, push=FALSE, remove.clone.on.success=FALSE,
        git=NULL, use.https=FALSE, user_name=NULL, user_email=NULL)

updateAllBiocPackageRepoObjects(all_repopaths=".", skipped_repos=NULL, ...)
```

Arguments

repopath	The path (as a single string) to the local Git repository of a Bioconductor package. See ?prepare_git_repo_for_work for more information.
branch	The branch (as a single string) of the Git repository to work on. See ?prepare_git_repo_for_work for more information.
filter	See ?updatePackageObjects.
commit_msg	The Git commit message. By default "Pass serialized S4 instances thru updateObject()" is used.
push	Whether to push the changes or not. Changes are committed but not pushed by default. You need push access to the package Git repository at git.bioconductor.org in order to use push=TRUE.
remove.clone.on.success	Whether to remove the Git clone on success or not. Only applies if repopath does not exist and needs to be cloned.
git, use.https	See ?prepare_git_repo_for_work.
user_name, user_email	See ?git_commit.
all_repopaths	Character vector of paths to local Git repositories of Bioconductor packages.
skipped_repos	Character vector of repository paths to ignore.
...	updateAllBiocPackageRepoObjects() walks over the all_repopaths vector and calls updateBiocPackageRepoObjects() on each repository path. All the arguments in ... are passed down to updateBiocPackageRepoObjects().

Value

updateBiocPackageRepoObjects() and updateAllBiocPackageRepoObjects() are wrappers to updatePackageObjects() and updateAllPackageObjects(), respectively, and return the same value.

See Also

- updatePackageObjects and updateAllPackageObjects.
- Utility functions prepare_git_repo_for_work and git_commit which are used internally by updateBiocPackageRepoObjects to perform the Git operations.

Examples

```
## ------------------------------------------------------------------
## updateBiocPackageRepoObjects()
## ------------------------------------------------------------------

## Typical use, assuming MyPackage is a Bioconductor package that you
```

```
## maintain:
## Not run:
  repopath <- file.path(tempdir(), "MyPackage")
  updateBiocPackageRepoObjects(repopath, push=TRUE)

## End(Not run)

## Note that by default `updateBiocPackageRepoObjects()` does NOT try
## to push the changes to git.bioconductor.org. Only the authorized
## maintainers of MyPackage can do that. In the examples below we
## must use HTTPS access to clone the package because we are not
## maintainers of the CellBench or BiocGenerics packages. Also we
## don't use 'push=TRUE' because we are not allowed to do that (it
## wouldn't work anyways).

## On a package with a mix of RDS and RDA files:
repopath <- file.path(tempdir(), "CellBench")
updateBiocPackageRepoObjects(repopath, branch="RELEASE_3_13",
                             remove.clone.on.success=TRUE,
                             use.https=TRUE)

## On a package with no serialized objects:
repopath <- file.path(tempdir(), "BiocGenerics")
updateBiocPackageRepoObjects(repopath, branch="RELEASE_3_13",
                             remove.clone.on.success=TRUE,
                             use.https=TRUE)

## Note that the RELEASE_3_13 branch of all Bioconductor packages got
## frozen in October 2021. The above examples are for illustrative
## purpose only. A more realistic situation would be to use
## updateBiocPackageRepoObjects() on the development version (i.e.
## the devel branch) of a package that you maintain, and to push the
## changes by calling the function with 'push=TRUE'.

## ------------------------------------------------------------------------
## updateAllBiocPackageRepoObjects()
## ------------------------------------------------------------------------

## Let's assume that the current directory is populated with the
## Git repositories of all Bioconductor software packages and that
## we have push access to them:

ALL_REPOS <- dir()  # get list of package repos to update

READ_RDS_FAILURE <- c(
    "BindingSiteFinder",
    "ChIPpeakAnno",
    "drugTargetInteractions"
)

LOAD_FAILURE <- c(
    "AlphaBeta",
    "CellaRepertorium",
    "CNVRanger",
    "gscreend",
    "HiLDA",
    "immunotation",
```

```
        "MAST",
        "midasHLA",
        "mixOmics",
        "oligoClasses",
        "TitanCNA",
        "Uniquorn"
    )

    UPDATEOBJECT_FAILURE <- c(
        "ACE",
        "AnnotationHubData",
        "arrayMvout",
        "Autotuner",
        "BASiCS",
        "bigmelon",
        "Biobase",
        "CAMERA",
        "categoryCompare",
        "cellHTS2",
        "cellmigRation",
        "CEMiTool",
        "CeTF",
        "cleanUpdTSeq",
        "CoGAPS",
        "CoreGx",
        "CrispRVariants",
        "crlmm",
        "decompTumor2Sig",
        "DIAlignR",
        "enhancerHomologSearch",
        "fcoex",
        "geNetClassifier",
        "GreyListChIP",
        "GSgalgoR",
        "hmdbQuery",
        "iCOBRA",
        "MassArray",
        "midasHLA",
        "MinimumDistance",
        "MSnbase",
        "msPurity",
        "multiHiCcompare",
        "musicatk",
        "MutationalPatterns",
        "openPrimeR",
        "OSAT",
        "PharmacoGx",
        "pipeFrame",
        "ProteoDisco",
        "puma",
        "qcmetrics",
        "QDNAseq",
        "r3Cseq",
        "RadioGx",
        "RTN",
        "sangeranalyseR",
        "synapter",
```

```
    "tigre",
    "topGO",
    "ToxicoGx",
    "VariantFiltering",
    "wateRmelon",
    "xcms"
)

## Contain files to push larger than 5 Mb.
PUSH_FAILURE <- c(
    "BiocSklearn",
    "BubbleTree",
    "CINdex",
    "erma",
    "ivygapSE",
    "SplicingGraphs",
    "vtpnet"
)

## Skipped for other reasons e.g. contain objects for which
## updateObject() takes forever or the package needs to be
## installed but cannot at the moment.
OTHER_SKIPPED_REPOS <- c(
    "BaalChIP", "BiGGR", "CytoTree", "gwascat",
    "mirIntegrator", "oposSOM", "PFP", "ROntoTools", "SLGI"
)

SKIPPED_REPOS <- c(
    READ_RDS_FAILURE,
    LOAD_FAILURE,
    UPDATEOBJECT_FAILURE,
    PUSH_FAILURE,
    OTHER_SKIPPED_REPOS
)

FILTER <- "\bDataFrame\b"

## Not run:
system.time(
    codes <- updateAllBiocPackageRepoObjects(ALL_REPOS,
                                             skipped_repos=SKIPPED_REPOS,
                                             branch="devel",
                                             filter=FILTER,
                                             push=TRUE)
)

## End(Not run)
```

`updatePackageObjects` *Update the serialized objects contained in a package or in a set of packages*

Description

Use `updatePackageObjects()` to update all the serialized objects contained in a package.

Use updateAllPackageObjects() to update all the serialized objects contained in a set of packages.

Usage

```
updatePackageObjects(pkgpath=".", filter=NULL,
                     dry.run=FALSE, bump.Version=FALSE)

updateAllPackageObjects(all_pkgpaths, skipped_pkgs=NULL, filter=NULL,
                     dry.run=FALSE, bump.Version=FALSE)
```

Arguments

pkgpath	The path (as a single string) to the top-level directory of an R package source tree.
filter, dry.run	These arguments are passed down to updateSerializedObjects(). See ?updateSerializedObjec for the details.
bump.Version	TRUE or FALSE. If TRUE and if some RDS or RDA files in the package actually get updated by updateSerializedObjects(), then the package version will get bumped, that is, the Version field in its DESCRIPTION file will get bumped from X.Y.Z to X.Y.(Z+1). For example, version 2.0.9 will become 2.0.10. Additionally, the Date field (if present) will get updated to the current date.
all_pkgpaths	Character vector of package paths.
skipped_pkgs	Character vector of package paths to ignore.

Value

updatePackageObjects() returns the value returned by its call to updateSerializedObjects(). See ?updateSerializedObjects for the details.

updateAllPackageObjects() returns a named integer vector *parallel* to all_pkgpaths.

See Also

- The updateSerializedObjects function which is the workhorse behind updatePackageObjects.
- updateBiocPackageRepoObjects and updateAllBiocPackageRepoObjects which are wrapper functions that also take care of committing and pushing the changes made to the packages.
- The bump_pkg_version function which is used internally by updatePackageObjects and updateAllPackageObjects when bump.Version=TRUE.

Examples

```
## ---------------------------------------------------------------------
## A SIMPLE updatePackageObjects() EXAMPLE
## ---------------------------------------------------------------------

## DemoPackage is a small demo package (contained in the updateObject
## package) with some old serialized GRanges objects in it.
pkgname <- "DemoPackage"
pkgpath0 <- system.file(pkgname, package="updateObject")

## Let's copy DemoPackage to a writable location.
pkgpath <- file.path(tempdir(), pkgname)
```

```
file.copy(pkgpath0, dirname(pkgpath), recursive=TRUE)

## Note that, in order to update the GRanges objects contained in
## DemoPackage, updatePackageObjects() will need to attach the
## GenomicRanges package. That's because this is where the GRanges
## class and updateObject() method for GRanges objects are both
## defined. See '?updateSerializedObjects' for more information.
## Also note that we don't need to perform two passes ("dry run" +
## "real run"), one pass is enough. Here we show the 2-pass procedure
## for illustrative purpose only.

## 1st pass: dry run
code <- updatePackageObjects(pkgpath, dry.run=TRUE)
code  # a non-negative code means everything went fine

## 2nd pass: do it for good!
updatePackageObjects(pkgpath, bump.Version=TRUE)

## An additional run would only confirm that there's nothing left
## to update.
code <- updatePackageObjects(pkgpath)
code  # 0 (no files to update)

unlink(pkgpath, recursive=TRUE)

## ---------------------------------------------------------------------
## FIND CANDIDATE PACKAGES IN CURRENT DIRECTORY
## ---------------------------------------------------------------------
## Not run:
## In this example we perform a "dry run" with updateAllPackageObjects()
## to find all the packages in a directory that contain old serialized
## objects.

## Let's assume that the current directory is populated with package
## git clones:
all_pkgs <- dir()  # get list of packages

## If we know that some packages are going to cause problems, we should
## skip them. Note that we could just do
##
##   all_pkgs <- setdiff(all_pkgs, SKIPPED_PKGS)
##
## for this. However, by using the 'skipped_pkgs' argument, all the
## packages in the original 'all_pkgs' will be represented in the
## returned vector, including the skipped packages:
SKIPPED_PKGS <- c(
    "BaalChIP", "BiGGR", "CytoTree", "gwascat",
    "mirIntegrator", "oposSOM", "PFP", "ROntoTools", "SLGI"
)

## --- Without a filter ---

## updateAllPackageObjects() will stop with an error if a package is
## required but not installed. The user is responsible for installing
## all the required packages (this is admittedly hard to know in advance).
codes <- updateAllPackageObjects(all_pkgs, skipped_pkgs=SKIPPED_PKGS,
                                 dry.run=TRUE)
```

```
sessionInfo()  # many packages
table(codes)

## The above code was successfully run in the MEAT0 folder on nebbiolo1
## (BioC 3.15, 2067 packages) on Nov 18, 2021:
## - took about 14 min
## - loaded 1190 packages (as reported by sessionInfo())
## - required about 9GB of RAM
##
## > table(codes)
## codes
## codes
##   -3   -2   -1    0    1    2    3    4    5    6    7    8    9   10
##    4   15   66 1549  240   90   46   28    7    5    4    3    3    1
##   13   18   20   23  125
##    2    1    1    1    1
##
## > sum(codes > 0) / length(codes)
## [1] 0.2094823
## 21

## --- With a filter ---

## We want to filter on the presence of the **word** "DataFrame" in
## the output of 'updateObject( , check=FALSE, verbose=TRUE)'. We can't
## just set 'filter' to '"DataFrame" for that as this would also produce
## matches in the presence of strings like "AnnotatedDataFrame":
filter <- "\bDataFrame\b"

codes <- updateAllPackageObjects(all_pkgs, skipped_pkgs=SKIPPED_PKGS,
                                 filter=filter,
                                 dry.run=TRUE)


## End(Not run)
```

`updateSerializedObjects` *Update the serialized objects contained in a directory*

Description

Use `updateSerializedObjects()` to find and update all the serialized objects contained in a directory. This is the workhorse behind higher-level functions `updatePackageObjects()` and family (`updateAllPackageObjects()`, `updateBiocPackageRepoObjects()`, and `updateAllBiocPackageRepoObjects()`).

`collect_rds_files()`, `collect_rda_files()`, `update_rds_file()`, and `update_rda_file()` are the low-level utilities used internally by `updateSerializedObjects()` to do the job.

Usage

```
updateSerializedObjects(dirpath=".", recursive=FALSE,
                        filter=NULL, dry.run=FALSE)
```

```
## Low-level utilities upon which updateSerializedObjects() is built:
collect_rds_files(dirpath=".", recursive=FALSE)
collect_rda_files(dirpath=".", recursive=FALSE)
update_rds_file(filepath, filter=NULL, dry.run=FALSE)
update_rda_file(filepath, filter=NULL, dry.run=FALSE)
```

Arguments

`dirpath`	The path (as a single string) to an arbitrary directory.
`recursive`	`TRUE` or `FALSE`. Should the directory be searched recursively to find the objects to update? By default the directory is *not* searched recursively.
`filter`	`NULL` (the default) or a single string containing a regular expression. When `filter` is set, only objects for which there is a match in the output of `updateObject(object, check=FALSE, verbose=TRUE)` actually get replaced with the object returned by the `updateObject` call. See Details section below for more on this. Note that the pattern matching is *case sensitive*.
`dry.run`	`TRUE` or `FALSE`. By default, updated objects are written back to their original file. Set `dry.run` to `TRUE` to perform a trial run with no changes made.
`filepath`	The path (as a single string) to a file containing serialized objects. This must be an RDS file (for `update_rds_file`) or RDA file (for `update_rda_file`).

Details

`update_rds_file()` and `update_rds_file()` use `updateObject()` internally to update individual R objects.

If no filter is specified (the default), each object is updated with `object <- updateObject(object, check=FALSE)`. If that turns out to be a no-op, then code 0 ("nothing to update") is returned. Otherwise 1 is returned.

If a filter is specified (via the `filter` argument) then `updateObject(object, check=FALSE, verbose=TRUE)` is called on each object and the output of the call is captured with `capture.output()`. Only if the output contains a match for `filter` is the object replaced with the object returned by the call. If this replacement turns out to be a no-op, or if the output contained no match for `filter`, then code 0 ("nothing to update") is returned. Otherwise 1 is returned.

The pattern matching is *case sensitive*.

Note that determining whether a call to `updateObject()` is a no-op or not is done by calling `digest::digest()` on the original object and object returned by `updateObject()`, and by comparing the 2 hash values. This is a LOT MORE reliable than using `identical()` which is notoriously unreliable!

Value

`updateSerializedObjects()` returns a single integer which is the number of updated files or a negative error code (-2 if loading an RDS or RDA file failed, -1 if `updateObject()` returned an error).

`collect_rds_files()` and `collect_rda_files()` return a character vector of (relative) file paths.

`update_rds_file()` and `update_rda_file()` return a single integer which is one of the following codes:

- -2 if loading the RDS or RDA file failed;

- -1 if `updateObject()` returned an error;
- 0 if there was nothing to update in the file;
- 1 if the file got updated.

See Also

- The `updatePackageObjects` function which is just a thin wrapper around `updateSerializedObjects`.
- The `updateObject` generic function in the **BiocGenerics** package.
- The `capture.output` function in the **utils** package.
- The `digest` function in the **digest** package.

Examples

```
dirpath <- system.file("extdata", package="updateObject")

## ---------------------------------------------------------------------
## WITHOUT A FILTER
## ---------------------------------------------------------------------

## updateSerializedObjects() prints one line per processed file:
updateSerializedObjects(dirpath, recursive=TRUE, dry.run=TRUE)

## Note that updateSerializedObjects() needs to attach/load the packages
## in which the classes of the objects to update are defined. These
## packages are: GenomicRanges for GRanges objects, SummarizedExperiment
## for SummarizedExperiment objects, and InteractionSet for GInteractions
## objects. This means that sessionInfo() will typically report more
## attached and loaded packages after a updateSerializedObjects() run
## than before:
sessionInfo()

## Also updateSerializedObjects() will raise an error if it fails to
## attach or load a package (typically because the package is missing).
## It will NOT try to install the package.

## ---------------------------------------------------------------------
## WITH A FILTER
## ---------------------------------------------------------------------

## We want to filter on the presence of the **word** "DataFrame" in
## the output of 'updateObject( , check=FALSE, verbose=TRUE)'. We can't
## just set 'filter' to '"DataFrame" for that as this would also produce
## matches in the presence of strings like "AnnotatedDataFrame":
filter <- "\bDataFrame\b"

updateSerializedObjects(dirpath, recursive=TRUE, filter=filter,
                        dry.run=TRUE)
```

Index