Package 'recount3'

September 21, 2021

Title Explore and download data from the recount3 project

Version 1.2.5

Date 2021-09-14

Description The recount3 package enables access to a large amount of uniformly processed RNA-seq data from human and mouse. You can download RangedSummarizedExperiment objects at the gene, exon or exon-exon junctions level with sample metadata and QC statistics. In addition we provide access to sample coverage BigWig files.

License Artistic-2.0

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

URL https://github.com/LieberInstitute/recount3

BugReports https://github.com/LieberInstitute/recount3/issues

biocViews Coverage, DifferentialExpression, GeneExpression, RNASeq, Sequencing, Software, DataImport

Suggests BiocStyle, covr, knitcitations, knitr, RefManageR, rmarkdown, testthat, pryr, interactiveDisplayBase, recount

VignetteBuilder knitr

Depends SummarizedExperiment

Imports BiocFileCache, methods, rtracklayer, S4Vectors, utils, RCurl, data.table, R.utils, Matrix, GenomicRanges, sessioninfo, tools

git_url https://git.bioconductor.org/packages/recount3

git_branch RELEASE_3_13

git_last_commit 5338a44

git_last_commit_date 2021-09-14

Date/Publication 2021-09-21

Author Leonardo Collado-Torres [aut, cre] (<https://orcid.org/0000-0003-2140-308X>)

Maintainer Leonardo Collado-Torres <lcolladotor@gmail.com>

R topics documented:

`annotation_ext` *Obtain the file extension for a given organism and annotation*

Description

Given an `organism` and an `annotation`, this function returns the corresponding file extension used in the `recount3` files.

Usage

```
annotation_ext(
  organism = c("human", "mouse"),
  annotation = annotation_options(organism)
)
```

Arguments

`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.
`annotation`	A `character(1)` specifying which annotation you want to use.

Value

A `character(1)` with the annotation file extension to be used.

See Also

Other internal functions for accessing the recount3 data: `create_rse_manual()`, `file_retrieve()`, `locate_url_ann()`, `locate_url()`, `project_homes()`, `read_counts()`, `read_metadata()`

Examples

```
annotation_ext("human")
annotation_ext("human", "fantom6_cat")
annotation_ext("human", "refseq")
annotation_ext("mouse")
```

---

annotation_options *List available annotation options for a given organism*

Description

This function will return the available annotation options for a given `organism`.

Usage

```
annotation_options(organism = c("human", "mouse"))
```

Arguments

`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.

Value

A `character()` vector with the supported annotation options for the given `organism`.

Examples

```
annotation_options("human")
annotation_options("mouse")
```

`available_projects` *List available projects in recount3*

Description

List available projects in recount3

Usage

```
available_projects(
  organism = c("human", "mouse"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3"),
  bfc = recount3_cache(),
 available_homes = project_homes(organism = organism, recount3_url = recount3_url)
)
```

Arguments

`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.
`recount3_url`	A `character(1)` specifying the home URL for `recount3` or a local directory where you have mirrored `recount3`.
`bfc`	A BiocFileCache-class object where the files will be cached to, typically created by `recount3_cache()`.
`available_homes`	A `character()` vector with the available project homes for the given `recount3_url`. If you use a non-standard `recount3_url`, you will likely need to specify manually the valid values for `available_homes`.

Value

A `data.frame()` with the project ID (`project`), the `organism`, the `file_source` from where the data was accessed, the `recount3` project home location (`project_home`), the project `project_type` that differentiates between `data_sources` and `compilations`, the `n_samples` with the number of samples in the given project.

Examples

```
## Find all the human projects
human_projects <- available_projects()

## Explore the results
dim(human_projects)
head(human_projects)

## How many are from a data source vs a compilation?
table(human_projects$project_type, useNA = "ifany")
```

```
## What are the unique file sources?
table(
    human_projects$file_source[human_projects$project_type == "data_sources"]
)

## Note that big projects are broken up to make them easier to access
## For example, GTEx and TCGA are broken up by tissue
head(subset(human_projects, file_source == "gtex"))
head(subset(human_projects, file_source == "tcga"))

## Find all the mouse projects
mouse_projects <- available_projects(organism = "mouse")

## Explore the results
dim(mouse_projects)
head(mouse_projects)

## How many are from a data source vs a compilation?
table(mouse_projects$project_type, useNA = "ifany")

## What are the unique file sources?
table(
    mouse_projects$file_source[mouse_projects$project_type == "data_sources"]
)

## Use with a custom recount3_url:
available_projects(
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3test",
    available_homes = "data_sources/sra"
)

## You can also rely on project_homes() if the custom URL has a text file
## that can be read with readLines() at:
## <recount3_url>/<organism>/homes_index
available_projects(
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3test"
)
```

`available_samples` *List available samples in recount3*

Description

This function returns a `data.frame()` with the samples that are available from `recount3`. Note that a specific sample might be available from a given `data_source` and none or many `collections`.

Usage

```
available_samples(
```

```
  organism = c("human", "mouse"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3"),
  bfc = recount3_cache(),
  verbose = getOption("recount3_verbose", TRUE),
 available_homes = project_homes(organism = organism, recount3_url = recount3_url)
)
```

Arguments

organism	A character(1) specifying which organism you want to download data from. Supported options are "human" or "mouse".
recount3_url	A character(1) specifying the home URL for recount3 or a local directory where you have mirrored recount3.
bfc	A BiocFileCache-class object where the files will be cached to, typically created by recount3_cache().
verbose	A logical(1) indicating whether to show messages with updates.
available_homes	A character() vector with the available project homes for the given recount3_url. If you use a non-standard recount3_url, you will likely need to specify manually the valid values for available_homes.

Value

A data.frame() with the sample ID used by the original source of the data (external_id), the project ID (project), the organism, the file_source from where the data was accessed, the date the sample was processed (date_processed) in YYYY-MM-DD format, the recount3 project home location (project_home), and the project project_type that differentiates between data_sources and compilations.

Examples

```
## Find all the human samples available from recount3
human_samples <- available_samples()
dim(human_samples)
head(human_samples)

## How many are from a data source vs a compilation?
table(human_samples$project_type, useNA = "ifany")

## What are the unique file sources?
table(
    human_samples$file_source[human_samples$project_type == "data_sources"]
)

## Find all the mouse samples available from recount3
mouse_samples <- available_samples("mouse")
dim(mouse_samples)
head(mouse_samples)

## How many are from a data source vs a compilation?
```

```
table(mouse_samples$project_type, useNA = "ifany")
```

`compute_read_counts` *Compute read counts*

Description

As described in the recount workflow, the counts provided by the recount2 project are base-pair counts. You can scale them using `transform_counts()` or compute the read counts using the area under coverage information (AUC).

Usage

```
compute_read_counts(
  rse,
  round = TRUE,
  avg_mapped_read_length = "recount_qc.star.average_mapped_length"
)
```

Arguments

`rse`	A RangedSummarizedExperiment-class created by `create_rse()`.
`round`	A `logical(1)` specifying whether to round the transformed counts or not.
`avg_mapped_read_length`	A `character(1)` specifying the metdata column name that contains the average fragment length after aligning. This is typically twice the average read length for paired-end reads.

Details

This function is similar to `recount::read_counts(use_paired_end = TRUE, round = TRUE)` but more general and with a different name to avoid NAMESPACE conflicts. Note that the default value of `round` is different than in `recount::read_counts()`. This was done to match the default value of `round` in `transform_counts()`.

Value

A `matrix()` with the read counts. By default this function uses the average read length to the QC annotation.

References

Collado-Torres L, Nellore A and Jaffe AE. recount workflow: Accessing over 70,000 human RNA-seq samples with Bioconductor version 1; referees: 1 approved, 2 approved with reservations. F1000Research 2017, 6:1558 doi: 10.12688/f1000research.12223.1.

See Also

Other count transformation functions: compute_scale_factors(), is_paired_end(), transform_counts()

Examples

```
## Create a RSE object at the gene level
rse_gene_SRP009615 <- create_rse_manual("SRP009615")
colSums(compute_read_counts(rse_gene_SRP009615)) / 1e6

## Create a RSE object at the gene level
rse_gene_DRP000499 <- create_rse_manual("DRP000499")
colSums(compute_read_counts(rse_gene_DRP000499)) / 1e6

## You can compare the read counts against those from recount::read_counts()
## from the recount2 project which used a different RNA-seq aligner
## If needed, install recount, the R/Bioconductor package for recount2:
# BiocManager::install("recount")
recount2_readsums <- colSums(assay(recount::read_counts(
    recount::rse_gene_SRP009615
), "counts")) / 1e6
recount3_readsums <- colSums(compute_read_counts(rse_gene_SRP009615)) / 1e6
recount_readsums <- data.frame(
    recount2 = recount2_readsums[order(names(recount2_readsums))],
    recount3 = recount3_readsums[order(names(recount3_readsums))]
)
plot(recount2 ~ recount3, data = recount_readsums)
abline(a = 0, b = 1, col = "purple", lwd = 2, lty = 2)

## Repeat for DRP000499, a paired-end study
recount::download_study("DRP000499", outdir = tempdir())
load(file.path(tempdir(), "rse_gene.Rdata"), verbose = TRUE)

recount2_readsums <- colSums(assay(recount::read_counts(
    rse_gene
), "counts")) / 1e6
recount3_readsums <- colSums(compute_read_counts(rse_gene_DRP000499)) / 1e6
recount_readsums <- data.frame(
    recount2 = recount2_readsums[order(names(recount2_readsums))],
    recount3 = recount3_readsums[order(names(recount3_readsums))]
)
plot(recount2 ~ recount3, data = recount_readsums)
abline(a = 0, b = 1, col = "purple", lwd = 2, lty = 2)
```

---

compute_scale_factors *Compute count scaling factors*

Description

This function computes the count scaling factors used by `transform_counts()`. This function is similar to `recount::scale_counts(factor_only = TRUE)`, but it is more general.

Usage

```
compute_scale_factors(
  x,
  by = c("auc", "mapped_reads"),
  targetSize = 4e+07,
  L = 100,
  auc = "recount_qc.bc_auc.all_reads_all_bases",
  avg_mapped_read_length = "recount_qc.star.average_mapped_length",
  mapped_reads = "recount_qc.star.all_mapped_reads",
  paired_end = is_paired_end(x, avg_mapped_read_length)
)
```

Arguments

x	Either a RangedSummarizedExperiment-class created by `create_rse()` or the sample metadata created by `read_metadata()`.
by	Either `auc` or `mapped_reads`. If set to `auc` it will compute the scaling factor by the total coverage of the sample. That is, the area under the curve (AUC) of the coverage. If set to `mapped_reads` it will scale the counts by the number of mapped reads (in the QC annotation), whether the library was paired-end or not, and the desired read length (`L`).
targetSize	A `numeric(1)` specifying the target library size in number of single end reads.
L	A `integer(1)` specifying the target read length. It is only used when `by = 'mapped_reads'` since it cancels out in the calculation when using `by = 'auc'`.
auc	A `character(1)` specifying the metadata column name that contains the area under the coverage (AUC). Note that there are several possible AUC columns provided in the sample metadata generated by `create_rse()`.
avg_mapped_read_length	A `character(1)` specifying the metdata column name that contains the average fragment length after aligning. This is typically twice the average read length for paired-end reads.
mapped_reads	A `character(1)` specifying the metadata column name that contains the number of mapped reads.
paired_end	A `logical()` vector specifying whether each sample is paired-end or not.

Value

A `numeric()` with the sample scale factors that are used by `transform_counts()`.

See Also

Other count transformation functions: `compute_read_counts()`, `is_paired_end()`, `transform_counts()`

Examples

```
## Download the metadata for SRP009615, a single-end study
SRP009615_meta <- read_metadata(
```

```
    metadata_files = file_retrieve(
        locate_url(
            "SRP009615",
            "data_sources/sra",
        )
    )
)

## Compute the scaling factors
compute_scale_factors(SRP009615_meta, by = "auc")
compute_scale_factors(SRP009615_meta, by = "mapped_reads")

## Download the metadata for DRP000499, a paired-end study
DRP000499_meta <- read_metadata(
    metadata_files = file_retrieve(
        locate_url(
            "DRP000499",
            "data_sources/sra",
        )
    )
)

## Compute the scaling factors
compute_scale_factors(DRP000499_meta, by = "auc")
compute_scale_factors(DRP000499_meta, by = "mapped_reads")

## You can compare the factors against those from recount::scale_counts()
## from the recount2 project which used a different RNA-seq aligner
## If needed, install recount, the R/Bioconductor package for recount2:
# BiocManager::install("recount")
recount2_factors <- recount::scale_counts(
    recount::rse_gene_SRP009615,
    by = "auc", factor_only = TRUE
)
recount3_factors <- compute_scale_factors(SRP009615_meta, by = "auc")
recount_factors <- data.frame(
    recount2 = recount2_factors[order(names(recount2_factors))],
    recount3 = recount3_factors[order(names(recount3_factors))]
)
plot(recount2 ~ recount3, data = recount_factors)
abline(a = 0, b = 1, col = "purple", lwd = 2, lty = 2)
```

`create_rse` *Create a recount3 RangedSummarizedExperiment gene or exon object*

Description

Once you have identified a project you want to work with, you can use this function to construct a `recount3` RangedSummarizedExperiment-class (RSE) object at the gene or exon expression feature level. This function will retrieve the data, cache it, then assemble the RSE object.

Usage

```
create_rse(
  project_info,
  type = c("gene", "exon", "jxn"),
  annotation = annotation_options(project_info$organism),
  bfc = recount3_cache(),
  jxn_format = c("ALL", "UNIQUE"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3"),
  verbose = getOption("recount3_verbose", TRUE)
)
```

Arguments

project_info	A data.frame() with one row that contains the information for the project you are interested in. You can find which project to work on using available_projects().
type	A character(1) specifying whether you want to access gene, exon, or exon-exon junction counts.
annotation	A character(1) specifying which annotation you want to download. Only used when type is either gene or exon.
bfc	A BiocFileCache-class object where the files will be cached to, typically created by recount3_cache().
jxn_format	A character(1) specifying whether the exon-exon junction files are derived from all the reads (ALL) or only the uniquely mapping read counts (UNIQUE). Note that UNIQUE is only available for some projects: GTEx and TCGA for human.
recount3_url	A character(1) specifying the home URL for recount3 or a local directory where you have mirrored recount3.
verbose	A logical(1) indicating whether to show messages with updates.

Value

A RangedSummarizedExperiment-class object.

Examples

```
## Find all available human projects
human_projects <- available_projects()

## Find the project you are interested in
proj_info <- subset(
    human_projects,
    project == "SRP009615" & project_type == "data_sources"
)

## Create a RSE object at the gene level
rse_gene_SRP009615 <- create_rse(proj_info)

## Explore the resulting RSE gene object
```

```
rse_gene_SRP009615

## Information about how this RSE object was made
metadata(rse_gene_SRP009615)

## Number of genes by number of samples
dim(rse_gene_SRP009615)

## Information about the genes
rowRanges(rse_gene_SRP009615)

## Sample metadata
colnames(colData(rse_gene_SRP009615))

## Check how much memory this RSE object uses
pryr::object_size(rse_gene_SRP009615)

## Create an RSE object using gencode_v29 instead of gencode_v26
rse_gene_SRP009615_gencode_v29 <- create_rse(
    proj_info,
    annotation = "gencode_v29",
    verbose = FALSE
)
rowRanges(rse_gene_SRP009615_gencode_v29)

## Create an RSE object using FANTOM6_CAT instead of gencode_v26
rse_gene_SRP009615_fantom6_cat <- create_rse(
    proj_info,
    annotation = "fantom6_cat"
)
rowRanges(rse_gene_SRP009615_fantom6_cat)

## Create an RSE object using RefSeq instead of gencode_v26
rse_gene_SRP009615_refseq <- create_rse(
    proj_info,
    annotation = "refseq"
)
rowRanges(rse_gene_SRP009615_refseq)

## Create an RSE object using ERCC instead of gencode_v26
rse_gene_SRP009615_ercc <- create_rse(
    proj_info,
    annotation = "ercc"
)
rowRanges(rse_gene_SRP009615_ercc)

## Create an RSE object using SIRV instead of gencode_v26
rse_gene_SRP009615_sirv <- create_rse(
    proj_info,
    annotation = "sirv"
)
rowRanges(rse_gene_SRP009615_sirv)
```

```
## Obtain a list of RSE objects for all gene annotations
rses_gene <- lapply(annotation_options(), function(x) {
    create_rse(proj_info, type = "gene", annotation = x)
})
names(rses_gene) <- annotation_options()
rses_gene

## Create a RSE object at the exon level
rse_exon_SRP009615 <- create_rse(
    proj_info,
    type = "exon"
)

## Explore the resulting RSE exon object
rse_exon_SRP009615

dim(rse_exon_SRP009615)
rowRanges(rse_exon_SRP009615)
pryr::object_size(rse_exon_SRP009615)

## Create a RSE object at the exon-exon junction level
rse_jxn_SRP009615 <- create_rse(
    proj_info,
    type = "jxn"
)

## Explore the resulting RSE exon-exon junctions object
rse_jxn_SRP009615

dim(rse_jxn_SRP009615)
rowRanges(rse_jxn_SRP009615)
pryr::object_size(rse_jxn_SRP009615)

## Obtain a list of RSE objects for all exon annotations
## Not run:
rses_exon <- lapply(annotation_options(), function(x) {
    create_rse(proj_info, type = "exon", annotation = x, verbose = FALSE)
})
names(rses_exon) <- annotation_options()

## End(Not run)
```

`create_rse_manual` *Internal function for creating a recount3 RangedSummarizedExperiment object*

Description

This function is used internally by `create_rse()` to construct a `recount3` RangedSummarizedExperiment-class object that contains the base-pair coverage counts at the gene or `exon` feature level for a given annotation.

Usage

```
create_rse_manual(
  project,
  project_home = project_homes(organism = organism, recount3_url = recount3_url),
  type = c("gene", "exon", "jxn"),
  organism = c("human", "mouse"),
  annotation = annotation_options(organism),
  bfc = recount3_cache(),
  jxn_format = c("ALL", "UNIQUE"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3"),
  verbose = getOption("recount3_verbose", TRUE)
)
```

Arguments

project	A character(1) with the ID for a given study.
project_home	A character(1) with the home directory for the project. You can find these using project_homes().
type	A character(1) specifying whether you want to access gene, exon, or exon-exon junction counts.
organism	A character(1) specifying which organism you want to download data from. Supported options are "human" or "mouse".
annotation	A character(1) specifying which annotation you want to download. Only used when type is either gene or exon.
bfc	A BiocFileCache-class object where the files will be cached to, typically created by recount3_cache().
jxn_format	A character(1) specifying whether the exon-exon junction files are derived from all the reads (ALL) or only the uniquely mapping read counts (UNIQUE). Note that UNIQUE is only available for some projects: GTEx and TCGA for human.
recount3_url	A character(1) specifying the home URL for recount3 or a local directory where you have mirrored recount3.
verbose	A logical(1) indicating whether to show messages with updates.

Value

A RangedSummarizedExperiment-class object.

References

https://doi.org/10.12688/f1000research.12223.1 for details on the base-pair coverage counts used in recount2 and recount3.

See Also

Other internal functions for accessing the recount3 data: annotation_ext(), file_retrieve(), locate_url_ann(), locate_url(), project_homes(), read_counts(), read_metadata()

Examples

```
## Unlike create_rse(), here we create an RSE object by
## fully specifying all the arguments for locating this study
rse_gene_SRP009615_manual <- create_rse_manual(
    "SRP009615",
    "data_sources/sra"
)
rse_gene_SRP009615_manual

## Check how much memory this RSE object uses
pryr::object_size(rse_gene_SRP009615_manual)

## Test with a collection that has a single sample
## NOTE: this requires loading the full data for this study when
## creating the RSE object
rse_gene_ERP110066_collection_manual <- create_rse_manual(
    "ERP110066",
    "collections/geuvadis_smartseq",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
)
rse_gene_ERP110066_collection_manual

## Check how much memory this RSE object uses
pryr::object_size(rse_gene_ERP110066_collection_manual)

## Mouse example
rse_gene_DRP002367_manual <- create_rse_manual(
    "DRP002367",
    "data_sources/sra",
    organism = "mouse"
)
rse_gene_DRP002367_manual

## Information about how this RSE was made
metadata(rse_gene_DRP002367_manual)

## Test with a collection that has one sample, at the exon level
## NOTE: this requires loading the full data for this study (nearly 6GB!)
## Not run:
rse_exon_ERP110066_collection_manual <- create_rse_manual(
    "ERP110066",
    "collections/geuvadis_smartseq",
    type = "exon",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
)
rse_exon_ERP110066_collection_manual


## Check how much memory this RSE object uses
pryr::object_size(rse_exon_ERP110066_collection_manual)
# 409 MB
```

```
## Test with a collection that has one sample, at the junction level
## NOTE: this requires loading the full data for this study
system.time(rse_jxn_ERP110066_collection_manual <- create_rse_manual(
    "ERP110066",
    "collections/geuvadis_smartseq",
    type = "jxn",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
))
rse_jxn_ERP110066_collection_manual

## Check how much memory this RSE object uses
## NOTE: this doesn't run since 2 files are missing on the test site!
pryr::object_size(rse_jxn_ERP110066_collection_manual)

## End(Not run)

## Not run:
## For testing and debugging
project <- "ERP110066"
project_home <- "collections/geuvadis_smartseq"

project <- "SRP009615"
project_home <- "data_sources/sra"
type <- "gene"
organism <- "human"
annotation <- "gencode_v26"
jxn_format <- "ALL"
bfc <- recount3_cache()
recount3_url <- "http://idies.jhu.edu/recount3/data"
verbose <- TRUE

## End(Not run)
```

`expand_sra_attributes` *Expand SRA attributes*

Description

This function expands the SRA attributes stored in `sra.sample_attributes` variable in the `colData()` slot of a RangedSummarizedExperiment-class produced by `create_rse()`.

Usage

```
expand_sra_attributes(rse)
```

Arguments

`rse`	A RangedSummarizedExperiment-class object created by `create_rse()` or `create_rse_manual()`.

Details

Note that this function will work on projects available from SRA only. Furthermore, SRA attributes are project-specific. Thus, if you use this function in more than one RSE object, you won't be able to combine them easily with `cbind()` and will need to manually merge the `colData()` slots from your set of RSE files before being able to run `cbind()`.

Value

A RangedSummarizedExperiment-class object with expanded metadata columns.

Author(s)

Andrew E Jaffe modified by Leonardo Collado-Torres.

Examples

```
## Find all available human projects
human_projects <- available_projects()

## Find the project you are interested in
proj_info <- subset(
    human_projects,
    project == "SRP009615" & project_type == "data_sources"
)

## Create a RSE object at the gene level
rse_gene_SRP009615 <- create_rse(proj_info)

## Expand the SRA attributes (see details for more information)
rse_gene_SRP009615 <- expand_sra_attributes(rse_gene_SRP009615)
```

---

file_retrieve *Download a remote file and cache it to re-use later*

Description

Download a remote file and cache it to re-use later

Usage

```
file_retrieve(
  url,
  bfc = recount3_cache(),
  verbose = getOption("recount3_verbose", TRUE)
)
```

Arguments

url	A character(1) with the file URL or the actual local path in which case, it won't be cached. If length(url) > 1 , this function will be used recursively.
bfc	A BiocFileCache-class object where the files will be cached to, typically created by recount3_cache().
verbose	A logical(1) indicating whether to show messages with updates.

Value

A character(1) with the path to the cached file.

See Also

Other internal functions for accessing the recount3 data: annotation_ext(), create_rse_manual(), locate_url_ann(), locate_url(), project_homes(), read_counts(), read_metadata()

Examples

```
## Download the metadata file for project SRP009615
url_SRP009615_meta <- locate_url(
    "SRP009615",
    "data_sources/sra"
)
local_SRP009615_meta <- file_retrieve(
    url = url_SRP009615_meta
)
local_SRP009615_meta

## Download the gene counts file for project SRP009615
url_SRP009615_gene <- locate_url(
    "SRP009615",
    "data_sources/sra",
    type = "gene"
)
local_SRP009615_gene <- file_retrieve(
    url = url_SRP009615_gene
)
local_SRP009615_gene
```

---

is_paired_end *Guess whether the samples are paired end*

Description

Based on two alignment metrics, this function guesses the samples are paired end or not.

Usage

```
is_paired_end(
  x,
  avg_mapped_read_length = "recount_qc.star.average_mapped_length",
  avg_read_length = "recount_seq_qc.avg_len"
)
```

Arguments

x	Either a RangedSummarizedExperiment-class created by `create_rse()` or the sample metadata created by `read_metadata()`.
avg_mapped_read_length	A `character(1)` specifying the metdata column name that contains the average fragment length after aligning. This is typically twice the average read length for paired-end reads.
avg_read_length	A `character(1)` specifying the metadata column name that contains the average read length prior to aligning.

Value

A `logical()` vector specifying whether each sample was likely paired-end or not.

See Also

Other count transformation functions: `compute_read_counts()`, `compute_scale_factors()`, `transform_counts()`

Examples

```
## Download the metadata for SRP009615, a single-end study
SRP009615_meta <- read_metadata(
    metadata_files = file_retrieve(
        locate_url(
            "SRP009615",
            "data_sources/sra",
        )
    )
)

## Are the samples paired end?
is_paired_end(SRP009615_meta)

## Download the metadata for DRP000499, a paired-end study
DRP000499_meta <- read_metadata(
    metadata_files = file_retrieve(
        locate_url(
            "DRP000499",
            "data_sources/sra",
        )
    )
```

```
)
is_paired_end(DRP000499_meta)
```

`locate_url` *Construct the URL to access a particular* `recount3` *file*

Description

Given an organism of interest, this function constructs the URL for accessing one of the output files from the `recount3` project. You can then download the file using `file_retrieve()`.

Usage

```
locate_url(
  project,
  project_home = project_homes(organism = organism, recount3_url = recount3_url),
  type = c("metadata", "gene", "exon", "jxn", "bw"),
  organism = c("human", "mouse"),
  sample = NULL,
  annotation = annotation_options(organism),
  jxn_format = c("ALL", "UNIQUE"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3")
)
```

Arguments

`project`	A `character(1)` with the ID for a given study.
`project_home`	A `character(1)` with the home directory for the `project`. You can find these using `project_homes()`.
`type`	A `character(1)` specifying whether you want to access gene counts, exon counts, exon-exon junctions or base-pair BigWig coverage files (one per `sample`).
`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.
`sample`	A `character()` vector with the sample ID(s) you want to download.
`annotation`	A `character(1)` specifying which annotation you want to download. Only used when `type` is either `gene` or `exon`.
`jxn_format`	A `character(1)` specifying whether the exon-exon junction files are derived from all the reads (`ALL`) or only the uniquely mapping read counts (`UNIQUE`). Note that `UNIQUE` is only available for some projects: GTEx and TCGA for human.
`recount3_url`	A `character(1)` specifying the home URL for `recount3` or a local directory where you have mirrored `recount3`.

Value

A `character()` with the URL(s) for the file(s) of interest.

See Also

Other internal functions for accessing the recount3 data: annotation_ext(), create_rse_manual(), file_retrieve(), locate_url_ann(), project_homes(), read_counts(), read_metadata()

Examples

```
## Example for metadata files from a project from SRA
locate_url(
    "SRP009615",
    "data_sources/sra"
)

## Example for metadata files from a project that is part of a collection
locate_url(
    "ERP110066",
    "collections/geuvadis_smartseq",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
)

## Example for a BigWig file
locate_url(
    "SRP009615",
    "data_sources/sra",
    "bw",
    "human",
    "SRR387777"
)

## Locate example gene count files
locate_url(
    "SRP009615",
    "data_sources/sra",
    "gene"
)
locate_url(
    "SRP009615",
    "data_sources/sra",
    "gene",
    annotation = "refseq"
)

## Example for a gene count file from a project that is part of a collection
locate_url(
    "ERP110066",
    "collections/geuvadis_smartseq",
    "gene",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
)

## Locate example junction files
locate_url(
    "SRP009615",
```

```
    "data_sources/sra",
    "jxn"
)

## Example for metadata files from a project from SRA
locate_url(
    "ERP001942",
    "data_sources/sra"
)
```

`locate_url_ann` *Construct the URL to a recount3 annotation file*

Description

Given a expression feature `type`, `organism` and `annotation`, this function constructs the URL (or file path) to access a recount3 annotation file. This function is used by `create_rse_manual()`.

Usage

```
locate_url_ann(
  type = c("gene", "exon"),
  organism = c("human", "mouse"),
  annotation = annotation_options(organism),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3")
)
```

Arguments

`type`	A `character(1)` specifying whether you want to access gene counts or exon data.
`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.
`annotation`	A `character(1)` specifying which annotation you want to download. Only used when `type` is either `gene` or `exon`.
`recount3_url`	A `character(1)` specifying the home URL for `recount3` or a local directory where you have mirrored `recount3`.

Value

A `character(1)` with the URL (or file path) to access the recount3 annotation file.

See Also

Other internal functions for accessing the recount3 data: `annotation_ext()`, `create_rse_manual()`, `file_retrieve()`, `locate_url()`, `project_homes()`, `read_counts()`, `read_metadata()`

Examples

```
locate_url_ann()
locate_url_ann(organism = "mouse")
```

`project_homes` *Find available project home options*

Description

This function finds the home for a given project (study) of interest based on the `organism` and the `home_type`.

Usage

```
project_homes(
  organism = c("human", "mouse"),
  recount3_url = getOption("recount3_url", "http://duffel.rail.bio/recount3")
)
```

Arguments

`organism`	A `character(1)` specifying which organism you want to download data from. Supported options are `"human"` or `"mouse"`.
`recount3_url`	A `character(1)` specifying the home URL for `recount3` or a local directory where you have mirrored `recount3`.

Details

By default it reads a small text file from `recount3_url/organism/homes_index` using `readLines()`. This text file should contain each possible project home per line. See http://duffel.rail.bio/recount3/human/homes_index for an example.

Value

A `character()` vector with the available `project_home` options.

See Also

Other internal functions for accessing the recount3 data: `annotation_ext()`, `create_rse_manual()`, `file_retrieve()`, `locate_url_ann()`, `locate_url()`, `read_counts()`, `read_metadata()`

Examples

```
## List the different available `project_home` options for the default
## arguments
project_homes("human")
project_homes("mouse")

## Test files
project_homes("human",
    recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
)
```

`read_counts` *Read a counts file*

Description

This function reads in a `recount3` gene or gexon counts file into R. You can first locate the file using `locate_url()` then download it to your computer using `file_retrieve()`.

Usage

```
read_counts(counts_file, samples = NULL)
```

Arguments

`counts_file`	A `character(1)` with the local path to a `recount3` counts file.
`samples`	A `character()` with `external_id` sample IDs to read in. When `NULL` (default), all samples will be read in. This argument is used by `create_rse_manual()`.

Value

A `data.frame()` with sample IDs as the column names.

References

https://doi.org/10.12688/f1000research.12223.1 for details on the base-pair coverage counts used in recount2 and recount3.

See Also

Other internal functions for accessing the recount3 data: `annotation_ext()`, `create_rse_manual()`, `file_retrieve()`, `locate_url_ann()`, `locate_url()`, `project_homes()`, `read_metadata()`

Examples

```
## Download the gene counts file for project SRP009615
url_SRP009615_gene <- locate_url(
    "SRP009615",
    "data_sources/sra",
    type = "gene"
)
local_SRP009615_gene <- file_retrieve(url = url_SRP009615_gene)

## Read the gene counts, take about 3 seconds
system.time(SRP009615_gene_counts <- read_counts(local_SRP009615_gene))
dim(SRP009615_gene_counts)

## Explore the top left corner
SRP009615_gene_counts[seq_len(6), seq_len(6)]

## Explore the first 6 samples.
summary(SRP009615_gene_counts[, seq_len(6)])

## Note that the count units are in
## base-pair coverage counts just like in the recount2 project.
## See https://doi.org/10.12688/f1000research.12223.1 for more details
## about this type of counts.
## They can be converted to reads per 40 million reads, RPKM and other
## counts. This is more easily done once assembled into a
## RangedSummarizedExperiment object.

## Locate and retrieve an exon counts file
local_SRP009615_exon <- file_retrieve(
    locate_url(
        "SRP009615",
        "data_sources/sra",
        type = "exon"
    )
)
local_SRP009615_exon

## Read the exon counts, takes about 50-60 seconds
system.time(
    SRP009615_exon_counts <- read_counts(
        local_SRP009615_exon
    )
)
dim(SRP009615_exon_counts)
pryr::object_size(SRP009615_exon_counts)

## Explore the top left corner
SRP009615_exon_counts[seq_len(6), seq_len(6)]

## Explore the first 6 samples.
summary(SRP009615_exon_counts[, seq_len(6)])
```

read_metadata *Read the metadata files*

Description

This function reads in the recount3 metadata files into R. You can first locate the files using locate_url() then download it to your computer using file_retrieve().

Usage

```
read_metadata(metadata_files)
```

Arguments

metadata_files	A character() with the local path to recount3 metadata files.

Value

A data.frame() with all lower case column names for the sample metadata.

See Also

Other internal functions for accessing the recount3 data: annotation_ext(), create_rse_manual(), file_retrieve(), locate_url_ann(), locate_url(), project_homes(), read_counts()

Examples

```
## Download the metadata files for project ERP110066
url_ERP110066_meta <- locate_url(
    "ERP110066",
    "data_sources/sra"
)
local_ERP110066_meta <- file_retrieve(
    url = url_ERP110066_meta
)

## Read the metadata
ERP110066_meta <- read_metadata(local_ERP110066_meta)
dim(ERP110066_meta)
colnames(ERP110066_meta)

## Read the metadata files for a project in a collection
## Note: using the test files since I can't access collections right now
## for this collection
ERP110066_collection_meta <- read_metadata(
    metadata_files = file_retrieve(
        locate_url(
            "ERP110066",
            "collections/geuvadis_smartseq",
```

```
            recount3_url = "http://snaptron.cs.jhu.edu/data/temp/recount3"
        )
    )
)
dim(ERP110066_collection_meta)
## New columns for this collection
colnames(ERP110066_collection_meta)[!colnames(ERP110066_collection_meta) %in% colnames(ERP110066_meta)]

## Read the metadata for a mouse project
DRP002367_meta <- read_metadata(
    metadata_files = file_retrieve(
        locate_url("DRP002367", "data_sources/sra", organism = "mouse")
    )
)
dim(DRP002367_meta)

## Locate and read the GTEx bladder metadata
gtex_bladder_meta <- read_metadata(
    file_retrieve(
        locate_url("BLADDER", "data_sources/gtex")
    )
)

dim(gtex_bladder_meta)
colnames(gtex_bladder_meta)
```

`recount3_cache` *Specify where to cache the recount3 files*

Description

This function allows you to specify where the recount3 files will be cached to. It is powered by BiocFileCache-class.

Usage

```
recount3_cache(cache_dir = getOption("recount3_cache", NULL))
```

Arguments

`cache_dir`	A `character(1)` specifying the directory that will be used for caching the data. If `NULL` a sensible default location will be used.

Value

A BiocFileCache-class object where the recount3 files will be cached to.

See Also

Other recount3 cache functions: `recount3_cache_files()`, `recount3_cache_rm()`

Examples

```
## Locate the recount3 cache default directory
recount3_cache()
```

`recount3_cache_files` *Locate recount3 cached files*

Description

This function returns the list of URLs of the recount3 files you have stored in your `recount3_cache()`.

Usage

```
recount3_cache_files(bfc = recount3_cache())
```

Arguments

`bfc`	A BiocFileCache-class object where the files will be cached to, typically created by `recount3_cache()`.

Value

A `character()` with the URLs of the recount3 files you have downloaded.

See Also

Other recount3 cache functions: `recount3_cache_rm()`, `recount3_cache()`

Examples

```
## List the URLs you have downloaded
recount3_cache_files()
```

`recount3_cache_rm` *Remove recount3 cached files*

Description

This function removes the recount3 files you have stored in your `recount3_cache()`.

Usage

```
recount3_cache_rm(bfc = recount3_cache())
```

Arguments

bfc	A BiocFileCache-class object where the files will be cached to, typically created by recount3_cache().

Value

A character(0) if the removal of files was successful.

See Also

Other recount3 cache functions: recount3_cache_files(), recount3_cache()

Examples

```
## List the URLs you have downloaded
recount3_cache_files()
## Not run:
## Now delete the cached files
recount3_cache_rm()

## List againt your recount3 files (should be empty)
recount3_cache_files()

## End(Not run)
```

---

transform_counts *Transform the raw counts provided by the recount3 project*

Description

In preparation for a differential expression analysis, you will have to choose how to scale the raw counts provided by the recount3 project. These raw counts are similar to those provided by the recount2 project, except that they were generated with a different aligner and a modified counting approach. The raw coverage counts for recount2 are described with illustrative figures at https://doi.org/10.12688/f1000research.12223.1. Note that the raw counts are the sum of the base level coverage so you have to take into account the total base-pair coverage for the given sample (default option) by using the area under the coverage (AUC), or alternatively use the mapped read lengths. You might want to do some further scaling to take into account the gene or exon lengths. If you prefer to calculate read counts without scaling check the function compute_read_counts().

Usage

```
transform_counts(
  rse,
  by = c("auc", "mapped_reads"),
  targetSize = 4e+07,
  L = 100,
  round = TRUE,
```

```
  ...
)
```

Arguments

rse	A RangedSummarizedExperiment-class created by create_rse().
by	Either auc or mapped_reads. If set to auc it will compute the scaling factor by the total coverage of the sample. That is, the area under the curve (AUC) of the coverage. If set to mapped_reads it will scale the counts by the number of mapped reads (in the QC annotation), whether the library was paired-end or not, and the desired read length (L).
targetSize	A numeric(1) specifying the target library size in number of single end reads.
L	A integer(1) specifying the target read length. It is only used when by = 'mapped_reads' since it cancels out in the calculation when using by = 'auc'.
round	A logical(1) specifying whether to round the transformed counts or not.
...	Further arguments passed to compute_scale_factors().

Details

This function is similar to recount::scale_counts() but more general and with a different name to avoid NAMESPACE conflicts.

Value

A matrix() with the transformed (scaled) counts.

See Also

Other count transformation functions: compute_read_counts(), compute_scale_factors(), is_paired_end()

Examples

```
## Create a RSE object at the gene level
rse_gene_SRP009615 <- create_rse_manual("SRP009615")

## Scale the counts using the AUC
assays(rse_gene_SRP009615)$counts <- transform_counts(rse_gene_SRP009615)

## See that now we have two assayNames()
rse_gene_SRP009615
assayNames(rse_gene_SRP009615)

## You can compare the scaled counts against those from
## recount::scale_counts() from the recount2 project
## which used a different RNA-seq aligner
## If needed, install recount, the R/Bioconductor package for recount2:
# BiocManager::install("recount")
recount2_sizes <- colSums(assay(recount::scale_counts(
    recount::rse_gene_SRP009615,
    by = "auc"
```

```
), "counts")) / 1e6
recount3_sizes <- colSums(assay(rse_gene_SRP009615, "counts")) / 1e6
recount_sizes <- data.frame(
    recount2 = recount2_sizes[order(names(recount2_sizes))],
    recount3 = recount3_sizes[order(names(recount3_sizes))]
)
plot(recount2 ~ recount3, data = recount_sizes)
abline(a = 0, b = 1, col = "purple", lwd = 2, lty = 2)

## Compute RPKMs
assays(rse_gene_SRP009615)$RPKM <- recount::getRPKM(rse_gene_SRP009615)
colSums(assay(rse_gene_SRP009615, "RPKM"))

## Compute TPMs
assays(rse_gene_SRP009615)$TPM <- recount::getTPM(rse_gene_SRP009615)
colSums(assay(rse_gene_SRP009615, "TPM")) / 1e6 ## Should all be equal to 1
```

Index