Package 'ReportingTools'

March 26, 2013

Title Tools for making reports in various formats

Version 1.0.0

Author
Jason A. Hackney, Melanie Huntley, Jessica L. Larson, Christina Chaivorapol, and Josh Kaminker

Maintainer Jason A. Hackney <hackney.jason@gene.com>

Depends methods

Imports
Biobase,hwriter,Category,GOstats,limma,lattice,AnnotationDbi,edgeR,annotate,PFAM.db, GSEABase, BiocGenerics(>= 0.1.6)

Suggests RUnit

Type Package

LazyLoad yes

License Artistic-2.0

Description The ReportingTools software package enables users to easily display reports of analysis results generated from sources such as microarray and sequencing data. The package allows users to create HTML pages that may be viewed on a web browser such as Safari, or in other formats readable by programs such as Excel. Users can generate tables with sortable and filterable columns, make and display plots, and link table entries to other data sources such as NCBI or larger plots within the HTML page. Using the package, users can also produce a table of contents page to link various reports together for a particular project that can be viewed in a web browser.

ByteCompile TRUE

biocViews Bioinformatics, Software, Visualization, Microarray, RNAseq,GO

R topics documented:

BaseReport-class *Class* "BaseReport"

Description

A ReportType defines a way of representing an R object in a different, ordered format. Several ReportTypes have been detailed in this package, including classes for HTML reports and R data packages.

Objects from the Class

Objects can be created by calls of the form `new("BaseReport", ...)`. This is an abstract class that is inherited by other ReportTypes, and should not be used directly.

Slots

`shortName`**:** A character string generally used as the filename for reports

`title`**:** A character string that gives a longer description of what the report shows.

`reportDirectory`**:** A file path that details where the report will be saved.

Methods

name `signature(object = "BaseReport")`: Get the shortName assigned to the report.

reportDirectory `signature(object = "BaseReport")`: Get the directory where the report will be generated.

reportDirectory<- `signature(object = "BaseReport", value = "character")`: Set the directory where the report will be generated.

show `signature(object = "BaseReport")`: ...

title `signature(main = "BaseReport")`: Return the title of the report

title<- `signature(object = "BaseReport", value = "character")`: Set the title of the report.

path `signature(object = "BaseReport")`: Get the filesystem location of the report.

See Also

HTMLReport DataPackage CSVFile

Examples

```
showClass("BaseReport")
```

CSVFile *Function for creating and initializing a CSVFile for publishing results*

Description

A CSVFile is a pointer to a comma separated value file on the file system. Publishing to a CSVFile overwrites the current contents of the given CSVFile.

Usage

```
CSVFile(shortName, title = "", reportDirectory = ".")
```

Arguments

shortName	A character string giving a short title for the report. Used as the base of the filename.
reportDirectory	A character string giving the location of the report.
title	A character string giving a longer description of the report.

Value

An object of class CSVFile, which will represents a file to be written to. There are no open filehandles, and calling publish multiple times will result in the file being overwritten.

See Also

CSVFile-class, HTMLReport-class

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
csv.file <- CSVFile(shortName = "my_csv_file",
   reportDirectory = "reportDirectory")
publish(my.df, csv.file)
```

CSVFile-class *Class* "CSVFile"

Description

A CSVFile is a pointer to a comma separated value file on the file system. Publishing to a CSVFile overwrites the current contents of the given CSVFile.

Objects from the Class

Objects from this class...

Slots

shortName**:** A character string used as the base of the filename. '.csv' is appended to the shortName to make the full file name.

title**:** The title of the report. This doesn't appear anywhere in the output, but can be used to track the report in an R session.

reportDirectory**:** A file path that details where the csv file will be saved.

Extends

Class "BaseReport", directly.

Methods

filename The name of the csv file, including extension

name<- Set the name of the CSVFile object

Examples

```
showClass("CSVFile")
```

DataPackage *Function for creating and initializing an object of class DataPackage.*

Description

This is a function for creating a DataPackage. Calls to publish an object to the package serialize the object to disk, adding a file to the data directory, and adding a stubbed .Rd documentation file for the dataset.

Usage

```
DataPackage(shortName, title = "", reportDirectory = ".", version = "0.0.1",
   dependencies = c("Biobase"), license = "", description = "",
   author = "nobody", maintainer = "nobody <nobody@nothing.net>")
```

Arguments

shortName	A character string giving the title of the DataPackage. This is used as both the title in the DESCRIPTION file and as the directory name for the package.
reportDirectory	Where the DataPackage directory will be created.
title	A character string giving the title of the package.
version	A character string giving the version of the DataPackage.
dependencies	A character vector listing what packages the DataPackage depends on.
license	A character string detailing the license the DataPackage published under.
description	A character string giving the description of the DataPackage.
author	A character string giving the author of the package.
maintainer	A character string giving the maintainer of the package.

Value

An object of class DataPackage. As a side effect, the directory structure of the data package will also be created in the location given by reportDirectory. On publication, objects are saved to the data directory of the package, and whatever dependencies the objects imply are added to the list of package dependencies.

See Also

DataPackage-class

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
data.package <- DataPackage('MyPackage', title = "My awesome package",
   author = "J.J. Nobody", maintainer = "J.J. Nobody <nobody@nowhere.net>")
publish('my.df', data.package)
```

DataPackage-class   *Class* "DataPackage"

Description

This is a pointer to an R data package. The result should be an installable R data package, with documentation for the data objects published therein.

Objects from the Class

Objects can be created by calls of the form `new("DataPackage", ...)`.

Slots

version: The current version of the generated data package

dependencies: A character vector of packages that the data package depends on.

author: The author(s) of the data package.

maintainer: The maintainer(s) of the data package. Defaults to the same person as the author.

license: What license to use when creating the data package.

description: The description of the data package.

package.Rd: Rd string for package-level help for the generated data package.

shortName: Name of the data package.

title: The title of the package

reportDirectory: The directory in which the data package is generated.

Extends

Class "BaseReport", directly.

Methods

dependencies Retrieve the list of dependencies for the data package.

dependencies<- Set the list of dependencies for the data package.

finish This is the final step in publishing objects to the data package. It recreates the DESCRIPTION file for the data package. The package name, title, version, author, maintainer, dependencies, license and description are set from the appropriate slots in the DataPackage object.

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
data.package <- DataPackage('MyPackage', title = "My awesome package",
   author = "J.J. Nobody", maintainer = "J.J. Nobody <nobody@nowhere.net>")
publish('my.df', data.package)

dependencies(data.package) # Returns "Biobase"
dependencies(data.package) <- c('Biobase','GSEABase')

finish(data.package)
```

---

filename-methods *Methods for getting the name of a file for a CSVFile or HTMLReport.*

Description

These methods return the filename for a report, for report types where a single file is produced.

Methods

signature(object = "CSVFile") Return the file name of the CSVFile. This is generated by adding '.csv' to the shortName of the report.

signature(object = "HTMLReport") Return the file name of the HTMLReport. This is generated by adding '.html' to the shortName of the report.

---

finish-methods *Finalizing reports after publishing results*

Description

This is a method for finalizing a report after results have been published. The exact nature of finalizing depends on the report type, as detailed below.

Methods

signature(publicationType = "DataPackage") Calling finish on a DataPackage object rewrites the DESCRIPTION file, making sure that all of the dependencies for the objects in the DataPackage are listed.

signature(publicationType = "HTMLReport") Calling finish on an HTMLReport calls the hwrite function closePage, which closes the body and html tags on the page and closes the connection to the file.

Examples

```
my.df <- data.frame()
## html.report <- HTMLReport(shortName = "my_html_file",
##     reportDirectory = "reportDirectory")
# publish(my.df, html.report)
## finish(html.report)
```

HTMLReport *Creating and initializing an HTMLReport object*

Description

This is the function one should use to instantiate an HTMLReport. It basically creates the report in the appropriate place, creating directories if needed. Special care should be taken if non-default javacsript or css files are used, as detailed below.

Usage

```
HTMLReport(shortName, title = NULL, reportDirectory = ".",
    baseUrl = "localhost", basePath = "", page = NULL,
    link.css = NULL, link.javascript = NULL, overwrite.js=TRUE)
```

Arguments

shortName	A short name for the report. It is used as the base of the HTML file.
title	The title for the page. It will be used in the title element in the page head, and will also be written at the top of the page.
reportDirectory	A character string giving the directory where the report will be written.
baseUrl	A character string giving the location of the page in terms of HTML locations.
basePath	A character string giving the location of the page in terms of filesystem locations.
page	The file handle for the html page, returned from the hwriter function openPage.
link.css	A character vector of file URLs detailing where to get style sheets. If unset, the default styling for ReportingTools is used.
link.javascript	A character vector of file URLs detailing where to get javascript. If unset, the default javascript libraries for ReportingTools are used.
overwrite.js	logical whether css and javascript files should be overwritten when copied into the report directory.

Details

Care should be used if link.javascript or link.css are set. If the javascript libraries for ReportingTools aren't linked, then it is likely that the resulting pages will not be as interactive as the default. The default css files are found in the extdata/csslib directory of the ReportingTools package, while the default javascript libraries are found in the extdata/jslib directory of the ReportingTools package. The default behavior is for the css and js files to be copied from the package directory into the reportDirectory of the HTMLReport. If these are copied to an alternate location, then the user can set the environment variables REPORTINGTOOLSCSSLIB and REPORTINGTOOLSJSLIB to point to the locations of the css and javascript files, respectively.

Value

An HTMLReport object with an open file handle to an HTML page.

See Also

HTMLReport-class, publish

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
html.report <- HTMLReport(shortName = "my_html_file",
   reportDirectory = "reportDirectory")
publish(my.df, html.report)
```

HTMLReport-class *Class* "HTMLReport"

Description

An HTMLReport is a pointer to an HTML page, built up using calls to publish or using hwrite calls on the page slot of the object.

Objects from the Class

Objects can be created by calls of the form new("HTMLReport", ...).

Slots

baseUrl: What is the base URL to which the HTMLReport's filename is appended.

basePath: What is the base file system URI to which the HTMLReport's filename is appended.

page: The connection to the html file. See openPage in the hwriter package.

shortName: A character string used as the base of the filename. '.html' is appended to the shortName to make the full file name.

title: The title of the HTMLReport. This is used as the page title in the header, and is also printed on the page.

reportDirectory: A file path that details where the report will be saved.

Extends

Class "BaseReport", directly.

Methods

basePath Return the basePath for the HTMLReport

baseUrl Return the baseUrl for the HTMLReport

filename Return the file name of the HTMLReport. This is generated by adding '.html' to the shortName of the report.

finish This is the final step in publishing results to an HTMLReport. It prints the closing tags for the html body, and then closes the connection to the html file. Further calls to publish will fail because the connection is closed.

name<- Set the short name for the html file.

page Returns the connection to the html file.

page<- Set the connection to the html file.

validConnection Is the connection to the html file still open? If finish has been called, this will return FALSE.

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
html.report <- HTMLReport(shortName = "my_html_file",
    title = "Report Title",
    reportDirectory = "reportDirectory")
publish(my.df, html.report)

basePath(html.report) # Returns "."

baseUrl(html.report) # Returns "localhost"

reportDirectory(html.report) # Returns "reportDirectory"

name(html.report) # Returns "my_html_file"

title(html.report) <- "A Better Title"
title(html.report) # Returns "A Better Title"

filename(html.report) # Returns "my_html_file.html"

path(html.report) # Returns "./reportDirectory/my_html_file.html"

url(html.report) # Returns "localhost/reportDirectory/my_html_file.html"

library(hwriter)
hwrite("Add some text here.", page(html.report))

finish(html.report)
```

---

mockRnaSeqData *A counts table of mock RNA-deq data in mouse.*

Description

Expression data from an RNA-seq experiment. Rows were randomly assigned mouse Entrez ids.

Usage

```
data(mockRnaSeqData)
```

Examples

```
data(mockRnaSeqData)
```

---

publish-methods *Methods for publishing a variety of data types in selected output formats*

Description

These are a series of methods for taking various data types and coercing them into selected output formats.

Methods

Data types can be output in several formats. Exactly what is done to coerce a data type into the given output format is described below.

data.frame:
`publish(object, htmlReport, tableTitle = NULL, ...)` The most basic object to publish to an HTMLReport is a data.frame. Most other methods involve coercing their objects to a data.frame and then calling publish on that data.frame. As such, this is where all styling for publishing tables can be centrally controlled. If tableTitle is specified, the title is printed above the table.

MArrayLM:
`publish(object, htmlReport, eSet, factor, n = 1000, pvalueCutoff = 0.01, lfc = 0, coef = NULL, adjust.m`
An MArrayLM object is coerced to a data.frame using a method similar to the topTable function from the limma package. The resulting table includes some selected feature data, and the log fold change and p-value from the linear model. A glyph showing expression levels of each gene are also plotted, based on the expression values from the ExpressionSet given in eSet, and the levels set in factor.

DGEExact:
`publish(object, htmlReport, countTable, conditions, annotation.db = "org.Hs.eg", pvalueCutoff = 0.01,`
Currently, only DGEExact objects returned from the exactTest function in the edgeR package are supported. DGEExact objects are coerced to a data.frame using a method similar to the topTags function from the edgeR package. The resulting table includes feature data, derived from the annotation package defined in annotation.db. For this to work, the feature names in the DGEExact object and the countTable have to be the primary identifier from the annotation packge. In most cases, the primary identifier will be Entrez Gene IDs.

HyperGResultBase, GOHyperGResult, PFAMHyperGResult:
`publish(object, htmlReport, selectedIDs, annotation.db, pvalueCutoff = 0.01,categorySize=10, makePl`
A HyperGResult object is coerced to a data.frame using a method similar to the summary function from the Category package. The resulting table includes for each classification the id, name, odds ratio of enrichment and p-value from the hypergeometric test. A glyph showing

the level of overlap of each classification with the selected genes is also plotted. The number of genes found in this classification is listed in the table and links to another page with a table of the corresponding genes, symbols and names. An additional page listing the overlap genes is also linked to the main output. For a GOHyperGResult object, a plot depicting the relationship between the significant ontologies and their parents is also plotted if makePlot is TRUE. The selectedIDs are the Entrez ids of the genes of interest (i.e. from the gene universe); annotation.db is the species of the ids.

GeneSetCollection:
publish(object, htmlReport, annotation.db, setStats=NULL, setPValues=NULL,...)
A GeneSetCollection object is coerced to a data.frame. The resulting table of set names includes links to additional pages containing the ids, names and symbols of each gene in the corresponding set. If GSEA statistics and/or p-values are provided, then they are also included in the gene set table. The genes within the GeneSetCollection must be Entrez ids and annotation.db is the species of the ids.

trellis:
publish(object, publicationType, figureTitle = NULL, filename = NULL, png.height = 480, png.width =
Trellis objects, as created by lattice functions are published to an HTMLReport by first printing a pdf and png version of the object, and then including the png image in the HTML page, linking it to the pdf version of the plot.

Special cases:
There are a few special types that can be published to a HTMLReport. Publishing an HTMLReport to another HTMLReport will create a link from one report to the other, using the title of the published report as the link text.

HTMLReportDataPackage There are two ways to publish R objects to a DataPackage: by using a character vector of object names, or by calling publish on the object directly, providing a name to use in saving the object. For most object types, basic documentation is created for the object in the data package, using the promptData function.

CSVFile data.frames can be published directly to csv files. Other data types, such as MArrayLM, DGEExact, GOHyperGResult, PFAMHyperGResult can be output as CSVFile, by means of coercion to a data.frame first.

Examples

```
my.df <- data.frame()
## html.report <- HTMLReport(shortName = "my_html_file",
##     reportDirectory = "reportDirectory")
# publish(my.df, html.report)
## finish(html.report)
```

---

reporting.theme *Color theme for use in ReportingTools*

Description

Getting some attractive colors for use in lattice graphics

Usage

```
reporting.theme()
```

Value

A list with slots as defined by the lattice function standard.theme

See Also

standard.theme

Examples

```
library(lattice)
theme <- reporting.theme()
lattice.options(default.theme = theme)
```

---

validConnection *Determine connection validity*

Description

Determine if a connection to an HTML page can be written to or not.

Usage

```
validConnection(htmlRep)
```

Arguments

htmlRep	An object of class HTMLReport.

Value

Returns TRUE if the page can be written to. If the file handle has been closed, then FALSE.

See Also

link{HTMLReport-class}

Examples

```
my.df <- data.frame(EGID = c("103", "104", "105", "106", "107"),
                    RPKM = c(4, 5, 3, 100, 75),
                    DE = c("Yes", "Yes", "No", "No", "No"))
html.report <- HTMLReport(shortName = "my_html_file",
    reportDirectory = "reportDirectory")
publish(my.df, html.report)
validConnection(html.report) # Returns TRUE

finish(html.report)
validConnection(html.report) # Returns FALSE
```

Index