

Package: rapbase (via r-universe)

September 29, 2026

Type Package

Title Base Functions and Resources for Rapporteket

Version 3.10.2

Maintainer Arnfinn Hykkerud Steindal <arnfinn.steindal@gmail.com>

Description Provide common functions and resources for registry specific R-packages at Rapporteket
<https://rapporteket.github.io/rapporteket/articles/short_introduction.html>.
This package is relevant for developers of packages/registries at Rapporteket.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports bookdown, bslib, DBI, digest, dplyr, DT, jsonlite, kableExtra, knitr, logger, readr, rlang, RMariaDB, rmarkdown, rpivotTable, sendmailR, shiny, shinyalert, sship (>= 0.9.0), utils, yaml

URL <https://github.com/Rapporteket/rapbase>

BugReports <https://github.com/Rapporteket/rapbase/issues>

Suggests blob, httptest, lifecycle, odbc, ps, RSQLite, rvest, testthat, withr

Config/roxygen2/version 8.1.0

Config/pak/sysreqs
cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev make libharfbuzz-dev libicu-dev libmysqlclient-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev zlib1g-dev

Repository <https://areedv.r-universe.dev>

Date/Publication 2026-09-29 12:55:40 UTC

RemoteUrl <https://github.com/rapporteket/rapbase>

RemoteRef HEAD

RemoteSha a0bacfbb3faebb0b86f73498edec7f00da4b4d27

Contents

.getFun	3
.testAutoReport	3
appLog	4
appNavbarUserWidget	4
autoReportFormatInput	6
autoReportFormatServer	6
autoReportInput	7
autoReportOrgInput	7
autoReportOrgServer	8
autoReportServer	8
autoReportServer2	10
autoReportUI	11
createAutoReport	12
deleteAutoReport	13
export	14
exportApp	15
exportGuide	16
filterAutoRep	17
findNextRunDate	17
getConfig	18
getGithub	19
halloRapporteket	20
howWeDealWithPersonalData	20
isRapContext	21
loadRegData	21
logger	22
loggerSetup	25
logShinyInputChanges	26
makeAutoReportTab	26
makeRunDayOfYearSequence	27
makeStandardTable	28
memoryDisplay	29
memoryTitle	30
navbarWidget	31
noOptOutOk	32
rapbase	33
rapCloseDbConnection	33
rapOpenDbConnection	33
rapTheme	34
readAutoReportData	34
readLog	35
regTitle	35
renderRmd	36
repLogger2	37
runAutoReport	38
runBulletin	39

<code>.getFun</code>	3
sanitizeLog	39
sendEmail	40
stagingData	40
stats	42
statsGuide	43
testdata	44
userAttribute	45
userInfo	46
writeAutoReportData	47
Index	48

<code>.getFun</code>	<i>Provide explicit reference to function for do.call</i>
----------------------	---

Description

Provide explicit reference to function for do.call

Usage

`.getFun(x)`

Arguments

x string with explicit reference, i.e. 'package::function'

Value

value of the exported 'function' in 'package'

<code>.testAutoReport</code>	<i>Simple test of automated report</i>
------------------------------	--

Description

Simple test of automated reporting from definitions provided in a yaml config file

Usage

`.testAutoReport(aNum = 1, aChar = "a", anExp = Sys.Date(), bulletin = 0)`

Arguments

aNum	a number
aChar	a character
anExp	an expression
bulletin	Integer defining if report is of type bulletin (1) or not (0). Set to 0 by default

Value

A simple message listing the contents of the arguments

Examples

```
.testAutoReport()
```

appLog

App log test dataset.

Description

A dataset containing test entries for the application log.

Usage

```
appLog
```

Format

A data frame with 20 rows and 8 variables:

id character id

time character timestamp

user user name

name user full name

group users group/registry

role users role

resh_id users organization

message log message

appNavbarUserWidget

Create widget for registry apps at Rapporteket

Description

Provides a widget-like information and utility block to be applied to all registry apps at Rapporteket. Contains the user name, organization and logout/exit as hyperlinked text.

Usage

```
appNavbarUserWidget(
  user = "Undefined person",
  organization = "Undefined organization",
  addUserInfo = FALSE,
  selectOrganization = FALSE,
  namespace = NULL
)
```

Arguments

user	String providing the name of the user
organization	String providing the organization of the user
addUserInfo	Logical defining whether a user data pop-up is to be part of the widget (TRUE) or not (FALSE, default)
selectOrganization	Logical if organization can be selected.
namespace	Character string providing the namespace to use, if any. Defaults is NULL in which case no namespace will be applied.

Details

Normally, user information will be provided through the session parameter and hence this will have to be provided from the server. The "rendering" of this info must hence be done within a layout element at the client such as a tabPanel. Selecting any one of them should be fine... At the client, both uiOutput and textOutput will be fine "rendering the information provided by the server.

Example of use in shiny (pseudo code):

```
server <- function(input, output, session) {
  ...
  output$appUserName <- renderText(getUserName(session))
  output$appUserOrg <- renderText(getUserReshId(session))
  ...
}

ui <- tagList(
  navbarPage(
    ...,
    tabPanel(...,
      appNavbarUserWidget(user = uiOutput(appUserName),
        organization = textOutput(appUserOrg))
    ),
    ...
  )
)
```

Value

Ready made html script

Examples

```
appNavbarUserWidget()
```

autoReportFormatInput *Report Format Selection UI*

Description

User interface module for selecting report output format.

Usage

```
autoReportFormatInput(id)
```

Arguments

id Character string providing the Shiny module id.

Value

A Shiny UI object.

autoReportFormatServer
Report Format Selection Server

Description

Server module for selecting report output format.

Usage

```
autoReportFormatServer(id)
```

Arguments

id Character string providing the Shiny module id.

Value

Reactive value containing the selected report format.

autoReportInput	<i>Auto Report Configuration UI</i>
-----------------	-------------------------------------

Description

User interface module for configuring and creating auto reports.

Usage

```
autoReportInput(id)
```

Arguments

id	Character string providing the Shiny module id.
----	---

Value

A Shiny UI object.

autoReportOrgInput	<i>Organization Selection UI</i>
--------------------	----------------------------------

Description

User interface module for selecting an organization used as data source for dispatchments and bulletins.

Usage

```
autoReportOrgInput(id)
```

Arguments

id	Character string providing the Shiny module id.
----	---

Value

A Shiny UI object.

autoReportOrgServer	<i>Organization Selection Server</i>
---------------------	--------------------------------------

Description

Server module for organization selection.

Usage

```
autoReportOrgServer(id, orgs)
```

Arguments

id	Character string providing the Shiny module id.
orgs	Named list containing organization names and ids.

Value

A list containing:

name Reactive organization name.

value Reactive organization id.

autoReportServer	<i>Auto Report Server Module</i>
------------------	----------------------------------

Description

Creates and manages automated report subscriptions, dispatchments, and bulletins.

Usage

```
autoReportServer(  
  id,  
  registryName,  
  type,  
  org = NULL,  
  paramNames = shiny::reactiveVal(c("")),  
  paramValues = shiny::reactiveVal(c("")),  
  reports = NULL,  
  orgs = NULL,  
  eligible = shiny::reactiveVal(TRUE),  
  freq = "month",  
  user,  
  runAutoReportButton = FALSE  
)
```

Arguments

<code>id</code>	Character string providing the shiny module id.
<code>registryName</code>	Character string with registry package name.
<code>type</code>	One of "subscription", "dispatchment" or "bulletin".
<code>org</code>	Reactive organization id.
<code>paramNames</code>	Reactive vector of parameter names.
<code>paramValues</code>	Reactive vector of parameter values.
<code>reports</code>	Report metadata list.
<code>orgs</code>	Named list of organizations.
<code>eligible</code>	Reactive logical indicating module availability.
<code>freq</code>	Default report frequency.
<code>user</code>	User metadata reactives.
<code>runAutoReportButton</code>	Logical indicating if testing button should be shown.

Details

The 'reports' argument must be a list where each entry represents one report configuration:

synopsis Description of the report.

fun Exported report function name.

paramNames Function argument names.

paramValues Corresponding argument values.

Value

A Shiny server module.

Examples

```
## make a list for report metadata
reports <- list(
  FirstReport = list(
    synopsis = "First example report",
    fun = "fun1",
    paramNames = c("organization", "topic", "outputFormat"),
    paramValues = c(111111, "work", "html")
  ),
  SecondReport = list(
    synopsis = "Second example report",
    fun = "fun2",
    paramNames = c("organization", "topic", "outputFormat"),
    paramValues = c(111111, "leisure", "pdf")
  )
)
```

```

## make a list of organization names and numbers
orgs <- list(
  OrgOne = 111111,
  OrgTwo = 222222
)

## client user interface function
ui <- shiny::fluidPage(
  shiny::sidebarLayout(
    shiny::sidebarPanel(
      autoReportFormatInput("test"),
      autoReportOrgInput("test"),
      autoReportInput("test")
    ),
    shiny::mainPanel(
      autoReportUI("test")
    )
  )
)

## server function
server <- function(input, output, session) {
  org <- autoReportOrgServer("test", orgs)
  format <- autoReportFormatServer("test")

  # set reactive parameters overriding those in the reports list
  paramNames <- shiny::reactive(c("organization", "outputFormat"))
  paramValues <- shiny::reactive(c(org$value(), format()))

  autoReportServer(
    id = "test",
    registryName = "rapbase",
    type = "dispatchment",
    org = org$value,
    paramNames = paramNames,
    paramValues = paramValues,
    reports = reports,
    orgs = orgs,
    eligible = shiny::reactiveVal(TRUE),
    freq = "month",
    user = user
  )
}

# run the shiny app in an interactive environment
if (interactive()) {
  shiny::shinyApp(ui, server)
}

```

Description

Backward compatible wrapper around [autoReportServer()].

Usage

```
autoReportServer2(...)
```

Arguments

... Arguments passed directly to [autoReportServer()].

Value

Same result as [autoReportServer()].

autoReportUI	<i>Auto Report Main UI</i>
--------------	----------------------------

Description

User interface module for displaying existing auto reports and related information.

Usage

```
autoReportUI(id)
```

Arguments

id Character string providing the Shiny module id.

Value

A Shiny UI object.

createAutoReport	<i>Create and add report to config</i>
------------------	--

Description

Adds an entry to the system configuration of reports to run at given intervals. After generating the configuration from the new entry the function load the current system configuration, adds the new entry and saves the updated system configuration.

Usage

```
createAutoReport(
  synopsis,
  package,
  type = "subscription",
  fun,
  paramNames,
  paramValues,
  owner,
  ownerName = "",
  email,
  organization,
  runDayOfYear,
  startDate = as.character(Sys.Date()),
  terminateDate = NULL,
  interval = "",
  intervalName = "",
  dryRun = FALSE
)
```

Arguments

synopsis	String with description of the report and to be used in subject field of email distributed reports
package	String with package name also corresponding to registry
type	Character string defining type of auto report. Currently, one of 'subscription' (default) or 'dispatchment'
fun	String providing name of function to be called for generating report
paramNames	String vector where each element corresponds to the input parameter to be used in the above function
paramValues	String vector with corresponding values to paramNames
owner	String providing the owner of the report. Usually a user name
ownerName	String providing full name of owner. Defaults to an empty string to maintain backwards compatibility
email	String with email address to recipient of email containing the report

organization	String identifying the organization the owner belongs to
runDayOfYear	Integer vector with day numbers of the year when the report is to be run
startDate	Date-class date when report will be run first time. Default value is set to <code>Sys.Date() + 1</code> <i>i.e.</i> tomorrow.
terminateDate	Date-class date after which report is no longer run. Default value set to NULL in which case the function will provide an expiry date adding 3 years to the current date if in a PRODUCTION context and 1 month if not
interval	String defining a time interval as defined in seq.POSIXt . Default value is an empty string
intervalName	String providing a human understandable representation of interval. Default value is an empty string
dryRun	Logical defining if global auto report config actually is to be updated. If set to TRUE the actual config (all of it) will be returned by the function. FALSE by default

Value

Nothing unless dryRun is set TRUE in which case a list of all config will be returned

See Also

[deleteAutoReport](#)

deleteAutoReport	<i>Delete existing report from config/db</i>
------------------	--

Description

Delete existing report from config/db

Usage

```
deleteAutoReport(autoReportId)
```

Arguments

autoReportId	String providing the auto report unique id
--------------	--

See Also

[createAutoReport](#)

export

Shiny modules providing GUI and server logic for Export

Description

Functions for registries that wants to implement exporting of registry databases, *e.g.* for local development purposes. Also includes relevant helper functions

Usage

```
exportUCInput(id)

exportUCServer(
  id,
  dbName,
  teamName = NULL,
  eligible = shiny::reactiveVal(TRUE)
)

exportUCApp(dbName = "rapbase")

selectListPubkey(pubkey)

exportDb(dbName, dropTabs = NULL, tableNames = "", compress = FALSE, session)

queryToFile(
  dbName,
  tableName = "",
  query,
  format = "RDS",
  compress = FALSE,
  session
)
```

Arguments

id	Character string module ID
dbName	Character string database name. If this is 'data', then the database name is taken from the environment variable 'MYSQL_DB_DATA'.
teamName	Character string defining the github team name containing members allowed to export the database. Default value is dbName.
eligible	Logical defining if the module should be allowed to work at full capacity. This might be useful when access to module products should be restricted. Default is TRUE, <i>i.e.</i> no restrictions.
pubkey	Character vector with public keys

dropTabs	Character vector with names of tables to be excluded from export.
tableNames	Character vector with names of tables to export. Only used when exporting the whole database, not a single table.
compress	Logical if export data is to be compressed (using gzip). FALSE by default.
session	Shiny session object
tableName	Character string with name of table to export. Only used when exporting a single table, not the whole database.
query	Character string with SQL query to fetch data to export. Only used when exporting a single table, not the whole database.
format	Character string with format to export data in. Only used when exporting a single table, not the whole database. Possible values are

Value

Shiny objects, mostly. Helper functions may return other stuff too.

Examples

```
## client user interface function
ui <- shiny::fluidPage(
  shiny::sidebarLayout(
    shiny::sidebarPanel(
      exportUCInput("test"),
    ),
    shiny::mainPanel(
      NULL
    )
  )
)

## server function
server <- function(input, output, session) {
  exportUCServer("test", dbName = "rapbase")
}

## run the shiny app in an interactive environment
if (interactive()) {
  shiny::shinyApp(ui, server)
}
```

exportApp

Shiny app with database export functionality

Description

Shiny app with database export functionality

Usage

```
exportApp(teamName = "", dbName = "data", logAsJson = TRUE)
```

Arguments

teamName	Character string, corresponding to github team name
dbName	Character string, can be used to specify name of database if needed. Defaults to "data", which will work for most registries.
logAsJson	Logical, if TRUE (default) logging will be done in JSON format.

exportGuide	<i>Shiny modules providing the Export Guide</i>
-------------	---

Description

Shiny modules providing the Export Guide

Usage

```
exportGuideUI(id)

exportGuideServer(id, dbName)

exportGuideServer2(id, dbName)

exportGuideApp()
```

Arguments

id	Character string module ID
dbName	Character string registry name key

Value

Functions ui and server representing the (module) app

Examples

```
ui <- shiny::fluidPage(
  exportGuideUI("exportGuide")
)

server <- function(input, output, session) {
  exportGuideServer("exportGuide", "test")
}

if (interactive()) {
  shiny::shinyApp(ui, server)
}
```

filterAutoRep	<i>Filter auto report data</i>
---------------	--------------------------------

Description

Generic function to filter various entities from auto report data

Usage

```
filterAutoRep(data, by, pass)
```

Arguments

data	List (nested) specifying auto reports to be filtered. May be obtained by <code>rapbase::getConfig(fileName = "autoReport.yml")</code>
by	Character string defining the filtering entity and must be one of <code>c("package", "type", "owner", "organization")</code> . The term 'package' represents the registry name
pass	Character vector defining the values of the filtering entity that will allow reports to pass through the filter

Value

List of auto reports matching the filtering criteria

Examples

```
ar <- data.frame(type = c("A", "B"))
filterAutoRep(ar, "type", "B") # ar2
```

findNextRunDate	<i>Find next run date for automated reports</i>
-----------------	---

Description

Find the next date that an automated report is supposed to be run. Likely, this function will only be relevant for shiny apps when this date is to be printed

Usage

```
findNextRunDate(
  runDayOfYear,
  baseDayNum = as.POSIXlt(Sys.Date())$yday + 1,
  startDate,
  terminateDate,
  interval = NULL,
  returnFormat = "%A %e. %B %Y"
)
```

Arguments

runDayOfYear	Numeric vector providing year-day numbers
baseDayNum	Numeric defining base year-day. Default is today
startDate	Character string of format "YYYY-MM-DD" defining the date of the very first run. If set to NULL (default) or a none future date (compared to the date represented by baseDayNum for the current year) it will be disregarded.
terminateDate	Date-class date after which report is no longer run. Default value set to NULL in which case the function will provide an expiry date adding 3 years to the current date if in a PRODUCTION context and 1 month if not
interval	String defining a time interval as defined in seq.POSIXt . Default value is an empty string
returnFormat	String providing return format as described in strptime in the current locale. Defaults to "%A %d. %B %Y" that will provide something like 'Mandag 20. januar 2019' in a Norwegian locale

Value

String date for printing

Examples

```
# Will return Jan 30 in the current year and locale with default formatting
findNextRunDate(runDayOfYear = c(10, 20, 30),
  baseDayNum = 20, startDate = 1, terminateDate = 50)
```

getConfig

Get configuration for package, if any

Description

Try to obtain yaml-formatted configuration placed either as given by the environment variable `R_RAP_CONFIG_PATH` or as provided by the package itself. If none can be found the function exits with an error

Usage

```
getConfig(fileName, packageName = "rapbase")
```

Arguments

fileName	String providing configuration file base name
packageName	String providing the package name

Value

A list of (yaml) configuration

getGithub	<i>Collect various data from the GitHub API</i>
-----------	---

Description

Collect various data from the GitHub API

Usage

```
getGithub(what, value, .token = NULL)
```

Arguments

what	Character string defining the api endpoint. Currently one of c("contributors", "members", "keys").
value	Character string specifying what to collect from the given endpoint. For "contributors" this should be the name of the repository, for "members" value should be the team slug and for "keys" this should be a github user name.
.token	Character string providing a valid token that will be used if the api call requires authentication. Listing of team members do require a token with the appropriate credentials.

Value

Character vector with results from the GitHub api request

halloRapporteket	<i>Plain testing tool</i>
------------------	---------------------------

Description

To be used for testing purposes

Usage

```
halloRapporteket()
```

Value

message A test message

howWeDealWithPersonalData	<i>Render text in pop-up</i>
---------------------------	------------------------------

Description

Render text on how Rapporteket deals with personal data

Usage

```
howWeDealWithPersonalData(..., callerPkg = NULL)
```

Arguments

...	Further arguments, currently not used
callerPkg	Character string naming the package this function was called from. The value is used to display the current version of the R package representing the registry at Rapporteket.

Value

fragment html info text

isRapContext	<i>Rapporteket context</i>
--------------	----------------------------

Description

Call to this function will return TRUE when run on a system where the environment variable R_RAP_INSTANCE is set to either "DEV", "TEST", "QA" or "PRODUCTION" and FALSE otherwise

Usage

```
isRapContext()
```

Value

Logical if system has a defined Rapporteket context

Examples

```
isRapContext()
```

loadRegData	<i>Provider of data for registries at Rapporteket</i>
-------------	---

Description

Generic to registries, provide reporting data obtained from sql databases Underlying this function is rapbase::RapporteketDbConnection

Usage

```
loadRegData(registryName = "data", query, dbType = "mysql")
```

```
describeRegistryDb(registryName, tabs = c())
```

```
nlinesRegistryDb(registryName, tab = "")
```

Arguments

registryName	String providing the name of the database to connect to. If it is "data" it will use the MYSQL_DB_DATA environment variable, if it is "autoreport" it will use the MYSQL_DB_AUTOREPORT environment variable, and if it is "raplog" it will use the MYSQL_DB_LOG environment variable. If none of these are set, it will use the name provided.
--------------	--

query	String SQL query to obtain the data
-------	-------------------------------------

dbType	String Type of db to query, currently "mysql" (default) and "mssql"
tabs	Character vector for optional definition of tables to describe. Defaults to an empty vector in which case all tables are used
tab	String name of the table for which to get number of lines

Value

data frame containing registry data or a list with table names and corresponding fields with attributes

logger	<i>Log user events in shiny applications at Rapporteket</i>
--------	---

Description

To be used for logging at application level (*i.e.* when a shiny session is started) or at report level (*i.e.* each time a report is run). Logging of single report events should be made from reactive environments within the shiny server function or from within the (report) functions used by the same reactive environments.

Usage

```
appLogger(
  session,
  msg = "No message provided",
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)

userLogger(user, msg = "No message provided")

repLogger(
  session = NULL,
  msg = "No message provided",
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)

autLogger(
  user,
  name,
  registryName,
  reshId,
  type,
  pkg,
  fun,
  param,
```

```

    msg = "No message provided",
    .topenv = parent.frame()
)

```

Arguments

session	Shiny session object to be used for getting user data. For testing and development purposes session can be replaced by <code>list()</code> in which case various config options might be used to provide something sensible.
msg	String providing a user defined message to be added to the log record. Default value is 'No message provided'.
.topcall	Parent call (if any) calling this function. Used to provide the function call with arguments. Default value is <code>sys.call(-1)</code> .
.topenv	Name of the parent environment calling this function. Used to provide package name (<i>i.e.</i> register) this function was called from. Default value is <code>parent.frame()</code> .
user	String providing owner of an automated report. Its value should correspond to the actual user name as provided in a shiny session at Rapporteket. Only used for subscription reports that are run outside a shiny session.
name	String providing full name of the report owner. Only used for automated reports that are run outside a shiny session.
registryName	String providing registry name. Only used for automated reports that are run outside a shiny session.
reshId	String providing the organization id of the (subscription) report author. Only used for automated reports that are run outside a shiny session.
type	Character string defining the type of report. Only used for automated reports that are run outside a shiny session in which case its value will replace that of <code>.topcall</code> .
pkg	Character string naming the package of the function that is to be logged. Only used for automated reports that are run outside a shiny session.
fun	Character string naming the function that should be logged. Only used for automated reports that are run outside a shiny session.
param	List of named function parameter. Only used for automated reports that are run outside a shiny session.

Details

The below fields will be appended to the log, in the following order:

1. time: date-time as event is logged as `format(time, "%Y-%m-%d %H:%M:%S")`
2. user: username as found in the shiny session object or as provided by function argument
3. name: full name of user as found in the shiny session object
4. group: users group membership as provided by the shiny session object. Normally, this will correspond to the registry the user belongs to
5. role: users role as provided by the shiny session object. Its value will depend on whatever is delivered by the authorization provider, but for OpenQReg registries 'LU' (local user) and 'SC' (system coordinator) are typical values

6. `resh_id`: the organization id of the current user as provided by the shiny session object, OR, when source of logging is auto reports, the organization ID of the data source used to make the report
7. `environment`: environment from where the logger function was called (only provided by `repLogger()`)
8. `call`: function (with arguments) from where the logger was called (only provided by `repLogger()`)
9. `message`: an optional message defined as argument to the function

The `autLogger()` function is a special case to be used for automated reports. Since such reports are run outside a reactive (shiny) context shiny session data are not available to the logger. Hence, logging data must be provided as arguments directly. As of rapbase version 1.12.0 logging of automated reports are already taken care of. Hence, this function should not be applied per registry application.

Value

Returns nothing but calls a logging appender

Note

Pseudo code of how `appLogger()` may be implemented:

```
library(shiny)
library(raplog)

server <- function(input, output, session) {
  raplog::appLogger(session, msg = "Smertere registeret: starting shiny app")
  ...
}
```

Pseudo code on how `repLogger()` can be implemented as part of a function in a reactive (shiny) context. First, this is an example of the shiny server function with the (reactive) function `renderPlot()` calling a function that provides a histogram:

```
library(shiny)
library(raplog)

server <- function(input, output, session) {
  ...
  output$hist <- renderPlot({
    makeHist(data, var = input$var, bins = input$bins, session = session)
  })
  ...
}
```

Then, logging is called within the function `makeHist()`:

```
makeHist <- function(data, var, bins, ...) {

  if ("session" %in% names(list(...))) {
    raplog::repLogger(session = list(...)[["session"]],
                      msg = "Providing histogram")
  }
  ...
}
```

Examples

```
# Depend on the environment variable R_RAP_CONFIG_PATH being set
try(appLogger(list()))
```

```
# Depend on the environment variable R_RAP_CONFIG_PATH being set
try(repLogger(list()))
```

```
# Depend on the environment variable R_RAP_CONFIG_PATH being set
try(autoLogger(user = "ttester", registryName = "rapbase", reshId = "999999"))
```

loggerSetup

Settings for logging as json

Description

Every info, warning and error will be logged in json format.

Usage

```
loggerSetup(
  usernameEnv = "SHINYPROXY_USERNAME",
  appidEnv = "SHINYPROXY_APPID",
  hooks = TRUE
)
```

Arguments

usernameEnv	Global variable containing user name
appidEnv	Global variable containing application name
hooks	Logical defining if hooks for automatic logging should be set

logShinyInputChanges	<i>Wrapper around logger::log_shiny_input_changes</i>
----------------------	---

Description

Wrapper around logger::log_shiny_input_changes

Usage

```
logShinyInputChanges(...)
```

Arguments

...	Arguments passed to logger::log_shiny_input_changes function
-----	--

makeAutoReportTab	<i>Make table of automated reports</i>
-------------------	--

Description

Make a table to be rendered in a shiny app providing automated reports from a given user or registry as obtained from the environmental variables.

Usage

```
makeAutoReportTab(
  namespace = character(),
  user = getUserName(),
  group = getUserGroups(),
  orgId = getUserReshId(),
  type = "subscription",
  mapOrgId = NULL,
  includeReportId = FALSE
)
```

Arguments

namespace	String naming namespace. Defaults to character() in which case no namespace will be created. When this function is used by shiny modules namespace must be provided.
user	Character string providing the username. Introduced as a new argument when running apps inside containers. Default value is set to rapbase::getUserName() to allow backward compatibility.

group	Character string defining the registry, normally corresponding to the R package name and the value stemming from the SHINYPROXY_GROUPS environment variable. Introduced as a new argument when running apps inside containers. Default value is set to <code>rapbase::getUserGroups()</code> to allow backward compatibility.
orgId	Character string or integer defining the organization (id) for user. Default value is set to <code>rapbase::getUserReshId()</code> to allow backward compatibility.
type	Character string defining the type of auto reports to tabulate. Must be one of "subscription", "dispatchment" or "bulletin". Default value set to "subscription".
mapOrgId	Data frame containing the two columns 'name' and 'id' corresponding to unique name and id of organizations. Default is NULL in which case the ids provided in auto report data will be used. In case mapOrgId is not NULL but no id match is found the id found in the auto report data will also be used
includeReportId	Logical if the unique report id should be added as the last column in the table. FALSE by default.

Details

Each table record (line) represents a uniquely defined automated report. For each line two shiny action buttons are provided to allow for editing and deleting of each entry. For applications implementing this table observing events on these action buttons may be used to allow users to manage automated reports by GUI. The action buttons for editing and deleting are provided with the static input ids `edit_button` and `del_button` and upon clicking the `button` part of their ids will change to the unique id of the report. Hence, a GUI call for editing a report can be caught by `shiny::observeEvent("edit_button")` and within this event the report id is obtained by collecting the string after the double underscore, e.g. `strsplit(input$edit_button, "__")[[1]][2]`.

Optionally, report id may be provided as the last column in the table to allow further development for registry specific purposes. Regardless, this column should normally be hidden in the GUI.

Take a look at the [example shiny server function in rapRegTemplate](#) on how this function may be implemented.

Value

Matrix providing a table to be rendered in a shiny app

makeRunDayOfYearSequence

Make a sequence of day numbers from av given date and interval

Description

This function provides an even sequence of day numbers spanning 365/366 days from the start date and interval provided. Mainly to be used in setting up automated reports at Rapporteket

Usage

```
makeRunDayOfYearSequence(startDay = Sys.Date(), interval)
```

Arguments

startDay Start date of sequence. May be provided as a string, e.g. `"2019-03-17"` or as class `"Date"`. Defaults to today

interval String representing a valid seq.POSIXt interval such as `"DSTday"`, `"week"`, `"month"`, `"quarter"` or `"year"`)

Value

Integer vector of day numbers

Examples

```
makeRunDayOfYearSequence(interval = "month")
```

makeStandardTable	<i>Make standard table for rmarkdown reports</i>
-------------------	--

Description

Function that will return tables used in reports.

Usage

```
mst(
  tab,
  col_names = colnames(tab),
  type = "pdf",
  cap = "",
  label = knitr::opts_current$get("label"),
  digs = 0,
  align = NULL,
  fs = 8,
  lsd = FALSE
)
```

Arguments

tab Data frame or matrix representing the table.

col_names Character vector with column names. Defaults `colnames(tab)`.

type Character string defining output, either `"html"` or `"pdf"`. Default is `"pdf"`.

cap Character string with table caption. Empty string by default.

label	Character string defining the label in case the table needs to be referenced elsewhere in the overall document. For instance, setting this to 'my_table' the corresponding inline rmarkdown reference to use is \@ref(tab:my_table). Please note that for this to work for both LaTeX and HTML the bookdown document processing functions must be used, <i>i.e.</i> bookdown::pdf_document2() and bookdown::html_document2(), respectively. Default value is knitr::opts_current\$get("label") in which case the name of the current R chunk will be used as label.
digs	Integer number of digits to use. 0 by default.
align	Character vector specifying column alignment in the LaTeX way, <i>e.g.</i> c("l", "c", "r") will align the first column to the left, center the second and right-align the last one. Default is NULL in which case numeric columns are right-aligned and all other columns are left-aligned.
fs	Integer providing the font size. Applies only for pdf output. Default value is 8.
lsd	Logical if table is to be scaled down. Applies only for pdf output. FALSE by default.

Details

mst() creates RMarkdown code for creating standard tables.

Value

Character string containing RMarkdown table code

Examples

```
mst(tab = mtcars[1:10, ])
```

memoryDisplay

Display memory usage in the app

Description

This function creates a reactive text output for displaying memory usage. In your server function, assign it to output\$memory:

Usage

```
memoryDisplay(session = shiny::getDefaultReactiveDomain())
```

Arguments

session Shiny session object.

Details

```
output$memory <- memoryDisplay(session)
```

The output ID memory is required and must match the textOutput ID used in memoryTitle().

Value

A reactive text output displaying memory usage.

Examples

```
server <- function(input, output, session) {  
  output$memory <- memoryDisplay(session)  
}
```

memoryTitle

Title on the top left of the app, including the logo and memory usage

Description

Title on the top left of the app, including the logo and memory usage

Usage

```
memoryTitle(title)
```

Arguments

title	Title of the app
-------	------------------

Value

a tagList containing the logo, the title, and memory usage

Examples

```
app_ui <- function() {  
  shiny::navbarPage(  
    title = memoryTitle("My App")  
  )  
}
```

 navbarWidget

Shiny modules providing GUI and server logic for user info widget

Description

Shiny modules for making a user information widget in registry shiny apps at Rapporteket. One benefit using these modules will be reduced complexity and number of code lines for each registry.

Usage

```
navbarWidgetInput(id, addUserInfo = TRUE, selectOrganization = FALSE)
```

```
navbarWidgetServer2(id, orgName, map_orname = NULL, caller = NULL)
```

```
navbarWidgetApp(orgName = "Org Name")
```

Arguments

id	Character string providing module namespace
addUserInfo	Logical defining if an "about" hyperlink is to be added
selectOrganization	Logical providing option for selecting among available organizations and roles.
orgName	Character string naming the organization
map_orname	A data.frame containing two columns: UnitId unit ids orgname corresponding organization names
caller	Character string naming the environment this function was called from. The value is used to display the current version of the R package representing the registry at Rapporteket.

Details

These modules take use of the shiny session object to obtain data for the widget. Hence, a Rapporteket like context will be needed for these modules to function properly. For deployment of (shiny) application as containers make sure to migrate to `navbarWidgetServer2()`. In addition to serving the user information widget, this function provides a list of reactive user attributes. Hence, when using `navbarWidgetServer2()` the source of (static) user attributes is no longer the shiny session object but rather the list object (of reactive user attributes) returned by this function.

Value

Shiny objects, mostly. `navbarWidgetServer2()` invisibly returns a list of reactive values representing user metadata and privileges. See [userAttribute](#) for further details on these values.

Examples

```
## client user interface function
ui <- shiny::tagList(
  shiny::navbarPage(
    "Testpage",
    shiny::tabPanel(
      "Testpanel",
      shiny::mainPanel(
        navbarWidgetInput("testWidget")
      )
    )
  )
)

## server function
server <- function(input, output, session) {
  navbarWidgetServer2("testWidget", orgName = "Test org")
}

## run the app in an interactive session and a Rapporteket like environment
if (interactive() && isRapContext()) {
  shiny::shinyApp(ui, server)
}
```

noOptOutOk

Provide a no-opt-out ok message

Description

To be applied for user input when there is actually no choice :-)

Usage

```
noOptOutOk()
```

Value

String with possible state of mind (in Norwegian) once left with no options

Examples

```
noOptOutOk()
```

rapbase	<i>rapbase: Base Functions and Resources for Rapporteket</i>
---------	--

Description

Provide common functions and resources for registry specific R-packages at Rapporteket. This packages is relevant for developers of packages/registries at Rapporteket

rapCloseDbConnection	<i>Close down data connection handle</i>
----------------------	--

Description

Close down data connection handle

Usage

```
rapCloseDbConnection(con)
```

Arguments

con	Open connection object that is to be closed
-----	---

rapOpenDbConnection	<i>Provide connection handle for data source at Rapporteket</i>
---------------------	---

Description

Generic to registries, handle the data source connections, including usernames and passwords needed to open these connections

Usage

```
rapOpenDbConnection(dbName, dbType = "mysql")
```

Arguments

dbName	String providing the name of the database to connect to. If it is "data" it will use the MYSQL_DB_DATA environment variable, if it is "autoreport" it will use the MYSQL_DB_AUTOREPORT environment variable, and if it is "raplog" it will use the MYSQL_DB_LOG environment variable. If none of these are set, it will use the name provided.
dbType	String providing type of data source, one of "mysql" and "sqlite". Defaults to "mysql". "mssql" is not supported anymore.

Value

A named list of con and drv representing the db connection handle and driver, respectively.

rapTheme	<i>Theme of the app.</i>
----------	--------------------------

Description

This is a wrapper around bslib::bs_theme

Usage

```
rapTheme(theme = "flatly", version = 3)
```

Arguments

theme	Name of the theme. See bslib::bootswatch_themes() for available themes.
version	Version of bootstrap to use (3, 4 or 5).

Value

a Bootstrap theme object

readAutoReportData	<i>Read automated report metadata</i>
--------------------	---------------------------------------

Description

Read automated report metadata

Usage

```
readAutoReportData()
```

Value

a table of autoreport data

Examples

```
try(readAutoReportData())
```

readLog	<i>Read log entries</i>
---------	-------------------------

Description

Function that provide log entries

Usage

```
readLog(type, name = "", app_id = NULL)
```

Arguments

type	Character string defining which log to request data from. Must be one of c("app", "report").
name	Character string with registry filter. Default value is an empty string that will return all log entries. If not empty its value must correspond to an existing registry (<i>i.e.</i> R package) name.
app_id	An identifier for a particular registry. Default value is NULL, in which case no action is taken. If value is provided, the log is filtered to show only entries matching chosen app_id.

Value

A data frame of log entries

regTitle	<i>Title on the top left of the app, including the logo</i>
----------	---

Description

Title on the top left of the app, including the logo

Usage

```
regTitle(regTitle = "rapbase")
```

Arguments

regTitle	Title of the app
----------	------------------

Value

a div containing the logo and the title

renderRmd

*Render documents from rmarkdown files at Rapporteket***Description**

Function that renders documents at Rapporteket from rmarkdown source files. Output formats may be (vanilla) HTML or PDF based on our own pandoc latex template or fragments of html when the result is to be embedded in existing web pages. Rmarkdown allow parameters to be part of report processing. Thus, parameters that are specific to reports must be provided (as a named list) when calling `renderRmd()`.

Usage

```
renderRmd(
  sourceFile,
  outputType = "html",
  logoFile = NULL,
  params = list(),
  template = "default",
  quiet = TRUE
)
```

Arguments

<code>sourceFile</code>	Character string providing the path to the rmarkdown source file.
<code>outputType</code>	Character string specifying the output format. Must be one of <code>c("pdf", "html", "html_fragment")</code> . Default value is "html".
<code>logoFile</code>	Character string with path to the logo to be used for PDF output. Often, this will be the registry logo. Only PNG and PDF graphics are allowed. Default value is NULL in which case no such logo will be added to the output document.
<code>params</code>	List of report parameters (as named values) to override the corresponding entries under <i>params</i> in the rmarkdown document yaml header. Default is NULL in which case no parameters as defined in the rmarkdown document will be overridden.
<code>template</code>	Character string defining which template to use for making pdf documents. Must be one of "default", "document", or NULL where the first is assumed if this argument is not set. NULL means that the default pandoc template is used.
<code>quiet</code>	Logical. If TRUE, suppresses output from the rendering process. TRUE is assumed if this argument is not set.

Value

Character string with path to the rendered file or, if `outputType` is set to "html_fragment", a character string providing an html fragment. Files are named according to `tempfile()` with an empty pattern and with the extension according to `outputType` ("pdf" or "html").

repLogger2

*Log report events in shiny applications at Rapporteket.***Description**

Updated version of repLogger() to be used in reactive contexts where user data is provided as reactive values. This is the case for the when user <- navbarWidgetServer2 which the way of providing user data in shiny applications at Rapporteket. The use of the original repLogger() should be changed to repLogger2 as repLogger() doesn't handle change of user roles during a session, which can happen in the shiny applications at Rapporteket.

Usage

```
repLogger2(
  user,
  msg = "No message provided",
  .topcall = sys.call(-1),
  .topenv = parent.frame()
)
```

Arguments

user	This needs to be a navbarWidgetServer2 object. Make sure to be in a reactive context when calling this function.
msg	String providing a description of the event to be logged. Default value is 'No message provided'.
.topcall	Parent call (if any) calling this function. Used to provide the function call with arguments. Default value is sys.call(-1).
.topenv	Name of the parent environment calling this function. Used to provide package name (<i>i.e.</i> register) this function was called from. Default value is parent.frame().

Note

If you are in the main shiny server, you can call user without passing it as an argument to the server function, as user is available in the main server environment. However, if you are in a module server function, you need to pass user as an argument to the module server function before you call repLogger2(user).

Examples

```
## Not run:
# Inside a reactive context, for example within a
# downloadHandler() function, you can call:
rapbase::repLogger2(user, msg = "This is a test")
# make sure user exists inside server modules
# by passing it as an argument to the module server function:
module_server <- function(id, user) { ... }
```

```
## End(Not run)
```

runAutoReport

Run reports as defined in yaml config and ship content by email

Description

Usually to be called by a scheduler, e.g. cron. If the provided day of year matches those of the config the report is run as otherwise specified in config. Functions called upon are expected to return a character string providing a path to a file that can be attached to an email or, in case of a bulletin, the email body itself. For bulletins, files cannot be attached. The email itself is prepared and sent to recipients defined in the config

Usage

```
runAutoReport(
  dayNumber = as.POSIXlt(Sys.Date())$yday + 1,
  dato = Sys.Date(),
  group = NULL,
  type = c("subscription", "dispatchment"),
  dryRun = FALSE
)
```

Arguments

dayNumber	Integer day of year where January 1st is 1. Defaults to current day, <i>i.e.</i> <code>as.POSIXlt(Sys.Date())\$yday + 1</code> (POSIXlt yday is base 0)
dato	Date-class date when report will be run first time. Default value is set to <code>Sys.Date()</code>
group	Character string defining the registry, normally corresponding to the R package name and the value stemming from the <code>SHINYPROXY_GROUPS</code> environment variable. Introduced as a new argument when running apps inside containers. Default value is set to <code>rapbase::getUserGroups()</code> to allow backward compatibility.
type	Character vector defining the type of reports to be processed. May contain one or more of <code>c("subscription", "dispatchment", "bulletin")</code> . Defaults value set to <code>c("subscription", "dispatchment")</code> .
dryRun	Logical defining if emails are to be sent. If TRUE a message with reference to the payload file is given but no emails will actually be sent. Default is FALSE

Value

Emails with corresponding file attachment. If `dryRun == TRUE` just a message

Examples

```
# Example depend on environment variable R_RAP_CONFIG_PATH being set
try(runAutoReport())
```

runBulletin	<i>Run bulletin auto reports</i>
-------------	----------------------------------

Description

This is a wrapper for runAutoReport() to issue bulletins.

Usage

```
runBulletin()
```

Value

Whatever runAutoReport() might provide

sanitizeLog	<i>Sanitize log entries that have reached end of life</i>
-------------	---

Description

Function that removes log entries older than a given number of days.

Usage

```
sanitizeLog(eolDays = 730)
```

Arguments

eolDays	Number of days to keep log entries. Entries older than this will be removed. Default value is 730 days (2 years).
---------	--

Value

NULL on success

sendEmail	<i>Send email from Rapporteket</i>
-----------	------------------------------------

Description

This function can be used to send email from within R at Rapporteket. It relies on (and must hence be provided) specific settings through local configuration to work properly.

Usage

```
sendEmail(conf, to, subject, text, attFile = NULL)
```

Arguments

conf	List containing (Rapporteket) config such as sender email address, SMTP server url and port number
to	Character vector containing email addresses. May also contain full names like 'Jane Doe <janed@nowhere.com>'
subject	Character string providing email subject.
text	Character string providing the plain email text in HTML format.
attFile	Character string providing the full file path to an attachment. Default is NULL in which case no attachment is made

Value

Invisible sending of email

stagingData	<i>Staging data functions</i>
-------------	-------------------------------

Description

Low level functions for handling registry staging data at Rapporteket. As such, these functions does not provide staging data management *per se*. Proper management, *e.g.* staging data updates and fallback logic must therefore be established within each registry that take staging data into use.

Usage

```
listStagingData(registryName, dbTable = "data")

mtimeStagingData(registryName, dbTable = "data")

saveStagingData(registryName, dataName, data, dbTable = "data")
```

```
loadStagingData(registryName, dataName, dbTable = "data")

deleteStagingData(registryName, dataName, dbTable = "data")

cleanStagingData(eolAge, dryRun = TRUE, dbTable = "data")
```

Arguments

registryName	Character string providing the registry name.
dbTable	Character string providing the database table name to be used
dataName	Character string providing the data set name.
data	A data object such as a data.frame to be stored as dataName.
eolAge	Numeric providing the staging data end-of-life age in seconds. Based on the current time and the time of storage staging files older than eolAge will be identified as subject for removal.
dryRun	Logical defining if function is to be run in dry (none destructive) mode.

Details

Staging data is stored as binary large objects in a database. A per registry symmetric encryption of storage content is enforced. Keys used for encryption are generated from existing database credentials. Therefore, please note that removing or changing such credentials will render any historic staging data inaccessible.

`cleanStagingData()` globally removes all staging data with store date prior to the end-of-life age provided. This is a vastly destructive function that should be used with great care.

Value

- `listStagingData()` returns a character vector of staging data sets for the given registry (registryName).
- `mtimeStagingData()` returns a staging data set named POSIXct vector of modification times for the given registry (registryName).
- `saveStagingData()` when successful returns the data object (data), invisibly. If saving fails a warning is issued and the function returns FALSE.
- `loadStagingData()` returns the data object corresponding to the name given upon saving (dataName). If the requested data set for loading does not exist the function returns FALSE.
- `deleteStagingData()` returns TRUE if the data set was deleted and FALSE if not.
- `cleanStagingData()` returns a list of data sets (to be) removed.

Examples

```
## Prep test data
registryName <- "rapbase"
dataName <- "testData"
data <- mtcars
```

```
## Save data for staging
try(saveStagingData(registryName, dataName, data))

## List data currently in staging
try(listStagingData(registryName))

## Retrieve data set from staging and compare to outset
try(stagedData <- loadStagingData(registryName, dataName))
try(identical(data, stagedData))

## Get modification time for staging file(s)
try(mtimeStagingData(registryName))
```

stats

Shiny modules and helper functions for registry usage reports

Description

These modules may be used by registries for easy setup of usage reports. The intended purpose is to provide registry staff access to when and by whom the resources at Rapporteket were used, *i.e.* application start-up and single report usage. As such, this will be a tool to provide useful statistics. However, it might also serve as a formal monitor utility but only if logging is carefully implemented throughout the relevant functions that make up the registry application at Rapporteket.

Usage

```
statsInput(id)

statsUI(id)

statsServer(id, registryName, app_id = NULL, eligible = TRUE)

statsServer2(
  id,
  registryName,
  app_id = NULL,
  eligible = shiny::reactiveVal(TRUE)
)

statsApp()

logFormat(log)

logTimeFrame(log, startDate, endDate)
```

Arguments

id	Character string shiny module id
registryName	Character string registry name key
app_id	An identifier for a particular registry. Default value is NULL, in which case no action is taken. If value is provided, the log is filtered to show only entries matching chosen app_id.
eligible	Logical defining if the module should be allowed to work at full capacity. This might be useful when access to module products should be restricted. Default is TRUE, <i>i.e.</i> no restrictions.
log	Data frame containing log data (in Rapporteket format)
startDate	Date object defining start of interval (character representation "YYYY-MM-DD")
endDate	Date object defining end of interval (character representation "YYYY-MM-DD")

Value

Shiny objects, mostly. Helper functions may return other stuff too.

Examples

```
# client user interface function
ui <- shiny::fluidPage(
  shiny::sidebarLayout(
    shiny::sidebarPanel(statsInput("test")),
    shiny::mainPanel(statsUI("test"))
  )
)

# server function
server <- function(input, output, session) {
  statsServer("test", registryName = "rapbase", eligible = TRUE)
}

# run the shiny app in an interactive environment
if (interactive()) {
  shiny::shinyApp(ui, server)
}
```

Description

Shiny modules providing the Stats Guide

Usage

```
statsGuideUI(id)

statsGuideServer(id, registryName)

statsGuideApp()
```

Arguments

```
id           Character string module ID
registryName Character string registry name key
```

Value

Functions ui and server representing the (module) app

Examples

```
ui <- shiny::fluidPage(
  statsGuideUI("statsGuide")
)

server <- function(input, output, session) {
  statsGuideServer("statsGuide", "test")
}

if (interactive()) {
  shiny::shinyApp(ui, server)
}
```

testdata	<i>Test dataset.</i>
----------	----------------------

Description

A dataset containing test entries.

Usage

```
testdata
```

Format

A data frame with 10 rows and 6 variables:

```
id int id
someText char text
someInt int integer
```

someBigInt int big integer
someFloat num float number
someTime time date

userAttribute

User attributes in container apps running behind shinyproxy

Description

For apps running as containers particular environment variables must be defined for an orderly handling of dynamic user privileges. This function makes use of environmental variables defined by shinyproxy to provide available privileges for the shiny application.

These are helper function for [userInfo](#). When used without a shiny session object calls to these functions is made without any arguments. If redefining contexts is needed, please use [userInfo](#) instead.

Usage

```
userAttribute(unit = NULL, map_orname = NULL)

getUserEmail(...)

getUserFullName(...)

getUserGroups(...)

getAppId(...)

getUserName(...)

getUserPhone(...)

getUserReshId(...)

getUserRole(...)
```

Arguments

unit	Integer providing the look-up unit id. Default value is NULL in which case all privileges for group are returned.
map_orname	A data.frame containing two columns: UnitId unit ids orgname corresponding organization names
...	To keep the same interface as before. Not used.

Value

a list of user metadata and privileges:

name The username for whom the privileges apply.

fullName User full name

phone User phone number

email User email

app SHINYPROXY_APPID environmental variable (id as string).

group FALK_APP_ID environmental variable (id as integer).

org Organization id for the user.

role Role of the user.

orgName Name of the organization as defined under the unit id.

String with user attribute

Examples

```
try(getUserEmail())
```

userInfo

Provide user attributes based on environment context

Description

Extracts elements from either config, url (shiny), shiny session or environmental variables relevant for user data such as name, group, role and org id (*e.g.* resh id). Source of info is based on environment context and can be controlled by altering the default settings for which contexts that will apply for the various sources of user data. This function will normally be used via its helper functions (see below).

Usage

```
userInfo(entity)
```

Arguments

entity String defining the element to return. Currently, one of 'user', 'groups', 'resh_id', 'role', 'email', 'full_name' or 'phone'.

Value

String of single user data element

See Also

[getUserName](#), [getUserGroups](#), [getUserReshId](#), [getUserRole](#)

writeAutoReportData	<i>Write automated report metadata</i>
---------------------	--

Description

Write automated report metadata

Usage

```
writeAutoReportData(config)
```

Arguments

config	a list of yaml configuration
--------	------------------------------

Examples

```
# Example depend on environment variable R_RAP_CONFIG_PATH being set
try(config <- readAutoReportData())
try(writeAutoReportData(config = config))
```

Index

* datasets

- appLog, 4
- testdata, 44
- .getFun, 3
- .testAutoReport, 3
- appLog, 4
- appLogger (logger), 22
- appNavbarUserWidget, 4
- autLogger (logger), 22
- autoReportFormatInput, 6
- autoReportFormatServer, 6
- autoReportInput, 7
- autoReportOrgInput, 7
- autoReportOrgServer, 8
- autoReportServer, 8
- autoReportServer2, 10
- autoReportUI, 11
- cleanStagingData (stagingData), 40
- createAutoReport, 12, 13
- deleteAutoReport, 13, 13
- deleteStagingData (stagingData), 40
- describeRegistryDb (loadRegData), 21
- export, 14
- exportApp, 15
- exportDb (export), 14
- exportGuide, 16
- exportGuideApp (exportGuide), 16
- exportGuideServer (exportGuide), 16
- exportGuideServer2 (exportGuide), 16
- exportGuideUI (exportGuide), 16
- exportUCApp (export), 14
- exportUCInput (export), 14
- exportUCServer (export), 14
- filterAutoRep, 17
- findNextRunDate, 17
- getAppId (userAttribute), 45
- getConfig, 18
- getGithub, 19
- getUserEmail (userAttribute), 45
- getUserFullName (userAttribute), 45
- getUserGroups, 46
- getUserGroups (userAttribute), 45
- getUserName, 46
- getUserName (userAttribute), 45
- getUserPhone (userAttribute), 45
- getUserReshId, 46
- getUserReshId (userAttribute), 45
- getUserRole, 46
- getUserRole (userAttribute), 45
- halloRapporteket, 20
- howWeDealWithPersonalData, 20
- isRapContext, 21
- listStagingData (stagingData), 40
- loadRegData, 21
- loadStagingData (stagingData), 40
- logFormat (stats), 42
- logger, 22
- loggerSetup, 25
- logShinyInputChanges, 26
- logTimeFrame (stats), 42
- makeAutoReportTab, 26
- makeRunDayOfYearSequence, 27
- makeStandardTable, 28
- memoryDisplay, 29
- memoryTitle, 30
- mst (makeStandardTable), 28
- mtimeStagingData (stagingData), 40
- navbarWidget, 31
- navbarWidgetApp (navbarWidget), 31
- navbarWidgetInput (navbarWidget), 31
- navbarWidgetServer2 (navbarWidget), 31

nlinesRegistryDb (loadRegData), [21](#)
noOptOutOk, [32](#)

queryToFile (export), [14](#)

rapbase, [33](#)
rapCloseDbConnection, [33](#)
rapOpenDbConnection, [33](#)
rapTheme, [34](#)
readAutoReportData, [34](#)
readLog, [35](#)
regTitle, [35](#)
renderRmd, [36](#)
repLogger (logger), [22](#)
repLogger2, [37](#)
runAutoReport, [38](#)
runBulletin, [39](#)

sanitizeLog, [39](#)
saveStagingData (stagingData), [40](#)
selectListPubkey (export), [14](#)
sendEmail, [40](#)
seq.POSIXt, [13](#), [18](#)
stagingData, [40](#)
stats, [42](#)
statsApp (stats), [42](#)
statsGuide, [43](#)
statsGuideApp (statsGuide), [43](#)
statsGuideServer (statsGuide), [43](#)
statsGuideUI (statsGuide), [43](#)
statsInput (stats), [42](#)
statsServer (stats), [42](#)
statsServer2 (stats), [42](#)
statsUI (stats), [42](#)
strptime, [18](#)

testdata, [44](#)

userAttribute, [31](#), [45](#)
userInfo, [45](#), [46](#)
userLogger (logger), [22](#)

writeAutoReportData, [47](#)