

# Package ‘rapbase’

October 14, 2022

**Type** Package

**Title** Base Functions and Resources for Rapporteket

**Version** 1.23.0

**Maintainer** Are Edvardsen <biorakel@gmail.com>

**Description** Provide common functions and resources for registry specific R-packages at Rapporteket  
<[https://rapporteket.github.io/rapporteket/articles/short\\_introduction.html](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 (>= 3.5.0)

**Imports** base64enc, bookdown, DBI, digest, dplyr, DT, jsonlite, kableExtra, knitr, magrittr, readr, rlang, RMariaDB, rmarkdown, rpivotTable, sendmailR, shiny, shinyalert, sship (>= 0.8.0), utils, yaml

**RoxygenNote** 7.1.2

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

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

**Suggests** httpptest, lifecycle, rvest, testthat

**NeedsCompilation** no

**Author** Are Edvardsen [aut, cre] (<<https://orcid.org/0000-0002-5210-3656>>),  
Kevin Otto Thon [aut]

**Repository** CRAN

**Date/Publication** 2022-08-17 14:20:02 UTC

**R topics documented:**

.getFun	3
.testAutoReport	3
appLog	4
appNavbarUserWidget	4
autoReport	6
createAutoReport	9
deleteAutoReport	11
export	11
exportGuide	12
filterAutoRep	13
findNextRunDate	14
fireInTheHole	15
getConfig	15
getGithub	16
getRapPackages	17
getRegs	17
getUserEmail	18
getUserFullName	18
getUserGroups	19
getUserName	20
getUserPhone	21
getUserReshId	22
getUserRole	22
halloRapporteket	23
howWeDealWithPersonalData	24
isPkgRapReg	24
isRapContext	25
loadRegData	25
logger	26
makeAutoReportTab	29
makeRunDayOfYearSequence	30
makeStandardTable	31
navbarWidget	32
noOptOutOk	33
rapbase	34
rapCloseDbConnection	34
rapOpenDbConnection	34
readAutoReportData	35
renderRmd	35
runAutoReport	36
runBulletin	37
runNoweb	37
sendEmail	38
stagingData	39
stats	40
statsGuide	42

<code>.getFun</code>	3
<code>upgradeAutoReportData</code> . . . . .	43
<code>userInfo</code> . . . . .	43
<code>writeAutoReportData</code> . . . . .	44
<b>Index</b>	<b>46</b>

---

<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 7 variables:

**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,  
  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)
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()
```

---

`autoReport`*Shiny modules and helper functions for registry auto reports*

---

## Description

These shiny modules may be used to set up auto reporting from registries at Rapporteket.

## Usage

```
autoReportUI(id)

autoReportOrgInput(id)

autoReportOrgServer(id, orgs)

autoReportFormatInput(id)

autoReportFormatServer(id)

autoReportInput(id)

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

autoReportApp(
  registryName = "rapbase",
  type = "subscription",
  reports = NULL,
  paramNames = shiny::reactive(c("")),
  orgs = NULL
)

orgList2df(orgs)
```

## Arguments

`id` Character string providing the shiny module id.

orgs	Named list of organizations (names) and ids (values). When set to NULL (default) the ids found in auto report data will be used in the table listing existing auto reports.
registryName	Character string with the registry name key. Must correspond to the registry R package name.
type	Character string defining the type of auto reports. Must be one of <code>c("subscription", "dispatchment", "bulletin")</code>
org	Shiny reactive or NULL (default) defining the organization (id) of the data source used for dispatchments and bulletins (in which case it cannot be set to NULL) and its value will be used to populate the <i>organization</i> field in auto report data (autoReport.yml) for these auto report types. On the other hand, since subscriptions are personal (per user) the only relevant organization id will implicit be that of the user and in this case any value of <i>org</i> will be disregarded.
paramNames	Shiny reactive value as a vector of parameter names of which values are to be set interactively at application run time. Each element of this vector must match exactly those of <i>paramValues</i> . Default value is <code>shiny::reactiveVal("")</code> .
paramValues	Shiny reactive value as a vector of those parameter values to be set interactively, <i>i.e.</i> as per user input in the application. Default value is set to <code>shiny::reactiveVal("")</code> in which case parameter values defined in reports will be used as is. In other words, explicit use of <i>paramValues</i> will only be needed if parameter values must be changed during application run time. If so, each element of this vector must correspond exactly to those of <i>paramNames</i> .
reports	List of a given structure that provides meta data for the reports that are made available as automated reports. See Details for further description.
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.
freq	Character string defining default frequency set in the auto report GUI. Must be one of <code>c("day", "week", "month", "quarter", "year")</code> . Default value is "month".

## Details

The *reports* argument must be a list where each entry represents one report and its name will be used in the auto report user interface for selecting reports, *e.g.* `reports = list(MagicReport = ...)` will produce the entry "MagicReport" in the GUI list of reports to select from. The value of each entry must be another list with the following names and values:

**synopsis** character string describing the report

**fun** report function base name (without "()")

**paramNames** character vector naming all arguments of *fun*

**paramValues** vector with values corresponding to *paramNames*

These named values will be used to run reports none-interactively on a given schedule and must therefore represent existing and exported functions from the registry R package. For subscriptions the *reports* list can be used as is, more specifically that the values provided in *paramValues* can go

unchanged. For dispatchments and bulletins it is likely that parameter values must be set dynamically in which case *paramValues* must be a reactive part of the application. See Examples on how function arguments may be used as reactivities in an application.

## Value

In general, shiny objects. In particular, `autoReportOrgServer` returns a list with names "name" and "value" with corresponding reactive values for the selected organization name and id. This may be used when parameter values of auto report functions needs to be altered at application run time. `orgList2df` returns a data frame with columns "name" and "id".

## 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 = TRUE
)
}

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

```

---

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 Sys.Date() + 1 <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 <a href="#">seq.POSIXt</a> . 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</i>
------------------	---

---

**Description**

Delete existing report from config

**Usage**

```
deleteAutoReport(autoReportId)
```

**Arguments**

autoReportId    String providing the auto report unique id

**See Also**

[createAutoReport](#)

---

export	<i>Shiny modules providing GUI and server logic for Export</i>
--------	--

---

**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, registryName, repoName = registryName, eligible = TRUE)
```

```
exportUCApp(registryName = "rapbase")
```

```
selectListPubkey(pubkey)
```

```
exportDb(registryName, compress = FALSE, session)
```

**Arguments**

id	Character string module ID
registryName	Character string registry name key
repoName	Character string defining the github repository name of the registry. Default value is registryName.
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
compress	Logical if export data is to be compressed (using gzip). FALSE by default.
session	Shiny session object

**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", registryName = "rapbase")
}

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

**Description**

Shiny modules providing the Export Guide

**Usage**

```
exportGuideUI(id)

exportGuideServer(id, registryName)

exportGuideApp()
```

**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(
  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 `rapbase::getConfig(fileName = "autoReport.yml")`

`by`                      Character string defining the filtering entity and must be one of `c("package", "type", "owner", "organization")`. 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 <- list(ar1 = list(type = "A"), ar2 = list(type = "B"))
filterAutoRep(ar, "type", "B") # ar2
```

---

`findNextRunDate` *Find next run date for automated reports*

---

### 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 = 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.

`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(c(10, 20, 30), 20)
```

---

fireInTheHole	<i>Kick off functions at Rapporteket</i>
---------------	--

---

### Description

This function will normally be executed by a cron daemon. Once started this function will nest through schedule functions defined in a configuration file, e.g. "rapbaseConfig.yml".

### Usage

```
fireInTheHole(flipPeriod = FALSE)
```

### Arguments

flipPeriod      Logical only used for testing. FALSE by default

### Details

This is a crontab example running fireInTheHole() every night at 01 hours, Monday through Friday and with emails suppressed:

```
0 1 * * 1-5 Rscript -e 'rapbase::fireInTheHole()' >/dev/null
2>&1
```

### Examples

```
# Depends on the env var R_RAP_CONFIG_PATH being properly set
fireInTheHole()
```

---

getConfig	<i>Get configuration for package, if any</i>
-----------	--

---

### 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 = "dbConfig.yml", packageName = "rapbase")
```

**Arguments**

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

**Value**

A list of (yaml) configuration

**Examples**

```
getConfig()
```

---

getGithub

*Collect various data from the GitHub API*

---

**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



---

getRapPackages	<i>Get all installed Rapporteket packages</i>
----------------	---

---

**Description**

Get all installed packages that depends on 'rapbase' which itself will not be reported

**Usage**

```
getRapPackages()
```

**Value**

Character vector of packages names

**Examples**

```
## Relevant only in a Rapporteket-like context
if (isRapContext()) {
  getRapPackages()
}
```

---

getRegs	<i>Provide vector of registries (i.e. their R packages) in config</i>
---------	---

---

**Description**

Provide vector of registries (*i.e.* their R packages) in config

**Usage**

```
getRegs(config)
```

**Arguments**

config            list of configuration for automated reports

**Value**

character vector of registry (package) names

getUserEmail                      *Get user email from config or session object*

---

### Description

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

### Usage

```
getUserEmail(shinySession = NULL)
```

### Arguments

**shinySession**      Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

### Value

String email address

### See Also

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

### Examples

```
# Requires a valid shiny session object
try(getUserEmail())
try(getUserEmail(shinySessionObject))
```

---

getUserFullName                      *Get user full name from config or session object*

---

### Description

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

**Usage**

```
getUserFullName(shinySession = NULL)
```

**Arguments**

`shinySession` Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

**Value**

String full name

**See Also**

[getUserName](#), [getUserGroups](#), [getUserReshId](#), [getUserEmail](#), [getUserPhone](#)

**Examples**

```
# Requires a valid shiny session object
try(getUserFullName())
try(getUserFullName(shinySessionObject))
```

---

getUserGroups

*Get user groups from config or session object*

---

**Description**

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

**Usage**

```
getUserGroups(shinySession = NULL)
```

**Arguments**

`shinySession` Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

**Value**

String user name

**See Also**

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

**Examples**

```
# Requires a valid shiny session object
try(getUserGroups())
try(getUserGroups(shinySessionObject))
```

---

getUserName	<i>Get user name from config or session object</i>
-------------	--

---

**Description**

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

**Usage**

```
getUserName(shinySession = NULL)
```

**Arguments**

**shinySession** Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

**Value**

String user name

**See Also**

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

## Examples

```
# Requires a valid shiny session object
try(getUserName())
try(getUserName(shinySessionObject))
```

---

getUserPhone	<i>Get user phone (number) from config or session object</i>
--------------	--

---

## Description

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

## Usage

```
getUserPhone(shinySession = NULL)
```

## Arguments

**shinySession** Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

## Value

String phone number

## See Also

[getUserName](#), [getUserGroups](#), [getUserReshId](#), [getUserEmail](#), [getUserFullName](#)

## Examples

```
# Requires a valid shiny session object
try(getUserPhone())
try(getUserPhone(shinySessionObject))
```

---

getUserReshId	<i>Get user resh ID from config or session object</i>
---------------	---

---

### Description

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

### Usage

```
getUserReshId(shinySession = NULL)
```

### Arguments

**shinySession** Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

### Value

String user name

### See Also

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

### Examples

```
# Requires a valid shiny session object
try(getUserReshId())
try(getUserReshId(shinySessionObject))
```

---

getUserRole	<i>Get user role from config or session object</i>
-------------	--

---

### Description

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

**Usage**

```
getUserRole(shinySession = NULL)
```

**Arguments**

`shinySession` Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.

**Value**

String user name

**See Also**

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

**Examples**

```
# Requires a valid shiny session object
try(getUserRole())
try(getUserRole(shinySessionObject))
```

---

halloRapporteket      *Plain testing tool*

---

**Description**

To be used for testing purposes

**Usage**

```
halloRapporteket()
```

**Value**

message A test message

---

howWeDealWithPersonalData

*Render text in pop-up*

---

### Description

Render text on how Rapporteket deals with personal data

### Usage

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

### Arguments

session	A shiny session object used to personalize the text
callerPkg	Character string naming the package that makes a call to this function in case version number of the caller package should be added to the returned (html) info text. Default to NULL in which case no version number for the caller will be added to the info text

### Value

fragment html info text

---

isPkgRapReg

*Test if a package is part of Rapporteket*

---

### Description

Test if an installed package is linked to Rapporteket based on dependency to the package 'rapbase'

### Usage

```
isPkgRapReg(pkg)
```

### Arguments

pkg	String providing the package name
-----	-----------------------------------

### Value

Logical TRUE if pkg depends on 'rapbase', FALSE if not

### See Also

[getRapPackages](#) on how to list all packages that depend on 'rapbase'



**Examples**

```
# returns FALSE, rapbase has no explicit dependency to itself
isPkgRapReg("rapbase")
```

---

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, query, dbType = "mysql")

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

**Arguments**

registryName	String Name of the registry as defined in dbConfig.yml
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

**Value**

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

---

logger

*Log user events in shiny applications at Rapporteket*

---

**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")
```

```
repLogger(
  session,
  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



```

    }
    ...
}

```

## 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(autLogger(user = "ttester", registryName = "rapbase", reshId = "999999"))

```

---

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 shiny session object provided.

## Usage

```

makeAutoReportTab(
  session,
  namespace = character(),
  type = "subscription",
  mapOrgId = NULL,
  includeReportId = FALSE
)

```

## Arguments

session	A shiny session object
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.
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 a 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. <code>"2019-03-17"</code> or as class <code>"Date"</code> . Defaults to today
interval	String representing a valid seq.POSIXt interval such as <code>"DSTday"</code> , <code>"week"</code> , <code>"month"</code> , <code>"quarter"</code> or <code>"year"</code> )

**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 <code>colnames(tab)</code> .
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 <code>\@ref(tab:my_table)</code> . Please note that for this to work for both LaTeX and HTML the bookdown document processing functions must be used, <i>i.e.</i> <code>bookdown::pdf_document2()</code> and <code>bookdown::html_document2()</code> , respectively. Default value is <code>knitr::opts_current\$get("label")</code> 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> <code>c("l", "c", "r")</code> will align the first column to the left, center the second and right-align the last one. Default is <code>NULL</code> 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, ])
```

---

 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)
```

```
navbarWidgetServer(id, orgName, caller = environmentName(rlang::caller_env()))
```

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

**Arguments**

id	Character string providing module namespace
addUserInfo	Logical defining if an "about" hyperlink is to be added
orgName	Character string naming the organization
caller	Character string naming the environment this function was called from. Default value is environmentName(rlang::caller_env()). The value is used to display the current version of the R package representing the registry at Rapporteket. If this module is called from exported functions in the registry R package use the default value. If the module is called from outside the registry environment caller must be set to the actual name of the R package.

**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.



**Value**

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

**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) {
  navbarWidgetServer("testWidget", orgName = "Test org", caller = "Rpkg")
}

## 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                      *rapbase: Base Functions and Resources for Rapporteket*

---

### 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    *Close down data connection handle*

---

### Description

Close down data connection handle

### Usage

```
rapCloseDbConnection(con)
```

### Arguments

con                      Open connection object that is to be closed

---

rapOpenDbConnection    *Provide connection handle for data source at Rapporteket*

---

### Description

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

### Usage

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

### Arguments

registryName    String id used for the registry in global configuration file from which information on the database connection is provided

dbType            String providing type of data source, one of "mysql" and "mssql". Defaults to "mysql"

### Value

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

---

readAutoReportData      *Read automated report metadata*

---

**Description**

Read automated report metadata

**Usage**

```
readAutoReportData(fileName = "autoReport.yml", packageName = "rapbase")
```

**Arguments**

fileName            String defining name of the yaml configuration file. Default 'autoReport.yml'  
packageName        String defining the package in which the above configuration file resides. A configuration file within an R-package is only used in case the environmental variable 'R\_RAP\_CONFIG\_PATH' is not defined (empty)

**Value**

a list of yaml data

**Examples**

```
readAutoReportData()
```

---

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"  
)
```

**Arguments**

sourceFile	Character string providing the path to the rmarkdown source file.
outputType	Character string specifying the output format. Must be one of <code>c("pdf", "html", "html_fragment")</code> . Default value is "html".
logoFile	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.
params	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.
template	Character string defining which template to use for making pdf documents. Must be one of "default" or "document" where the first 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").

---

runAutoReport	<i>Run reports as defined in yaml config and ship content by email</i>
---------------	--

---

**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,
  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)
-----------	---

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
runAutoReport()
```

---

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

---

**Description**

This is a wrapper for `runAutoReport()` to issue bulletins. Purpose is to ease simplify fire-in-the-hole at Rapporteket

**Usage**

```
runBulletin()
```

**Value**

Whatever `runAutoReport()` might provide

---

runNoweb	<i>runNoweb</i>
----------	-----------------

---

**Description**

Function to run noweb file contained in a package. Assume all noweb files of the package are placed flat under the *inst* directory

**Usage**

```
runNoweb(nowebFileName, packageName, weaveMethod = "knitr")
```

**Arguments**

nowebFileName	Basename of the noweb file, <i>e.g.</i> 'myFile.Rnw'.
packageName	Name of the package containing noweb file(s)
weaveMethod	Method to apply for weaving. Currently available are 'Sweave' and 'knitr', default to the latter.

---

sendEmail

*Send email from Rapporteket*


---

**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. The string is converted within this function to conform to RFC 1342
text	Character string providing the plain email text
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, dir = Sys.getenv("R_RAP_CONFIG_PATH"))

mtimeStagingData(registryName, dir = Sys.getenv("R_RAP_CONFIG_PATH"))

saveStagingData(
  registryName,
  dataName,
  data,
  dir = Sys.getenv("R_RAP_CONFIG_PATH")
)

loadStagingData(registryName, dataName, dir = Sys.getenv("R_RAP_CONFIG_PATH"))

deleteStagingData(
  registryName,
  dataName,
  dir = Sys.getenv("R_RAP_CONFIG_PATH")
)

cleanStagingData(eolAge, dryRun = TRUE)

pathStagingData(registryName, dir)
```

### Arguments

registryName	Character string providing the registry name.
dir	Character string providing the path to where the staging data directory resides. Default value is <code>Sys.getenv("R_RAP_CONFIG_PATH")</code> .
dataName	Character string providing the data set name.
data	A data object such as a <code>data.frame</code> to be stored as <code>dataName</code> .
eolAge	Numeric providing the staging file end-of-life age in seconds. Based on the current time and the file modification time stamp staging files older than <code>eolAge</code> will be identified as subject for removal.
dryRun	Logical defining if function is to be run in dry (none destructive) mode.

## Details

`cleanStagingData()` globally removes all staging data files older than the end-of-life age provided. This is potentially a vastly destructive function that should be used with great care.

## Value

- `listStagingData()` returns a character vector of staging data files for the given registry (`registryName`).
- `mtimeStagingData()` returns a staging file-named POSIXct vector of modification times for the given registry (`registryName`).
- `saveStagingData()` returns the data object (`data`), invisibly.
- `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 file was deleted and `FALSE` if not.
- `cleanStagingData()` returns a list of files (to be) removed.
- `rapbase:::pathStagingData()` is an internal helper function and returns a character string with the path to the staging directory of `registryName`. If its parent directory (`dir`) does not exist an error is returned.

## Examples

```
## Prep test data
registryName <- "myReg"
dataName <- "testData"
data <- mtcars
dir <- tempdir()

## Save data for staging
saveStagingData(registryName, dataName, data, dir)

## List files currently in staging
listStagingData(registryName, dir)

## Retrieve data set from staging
loadStagingData(registryName, dataName, dir)

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



**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, eligible = TRUE)

statsApp()

logFormat(log)

logTimeFrame(log, startDate, endDate)
```

**Arguments**

<code>id</code>	Character string shiny module id
<code>registryName</code>	Character string registry name key
<code>eligible</code>	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.
<code>log</code>	Data frame containing log data (in Rapporteket format)
<code>startDate</code>	Date object defining start of interval (character representation "YYYY-MM-DD")
<code>endDate</code>	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)
}
```

---

statsGuide

*Shiny modules providing the Stats Guide*

---

## 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)
}
```

---

upgradeAutoReportData *Upgrade auto reports*

---

### Description

Upgrade auto report config as new features emerge. Currently, the type definition is added and set to 'subscription' that historically has been the only type used

### Usage

```
upgradeAutoReportData(config)
```

### Arguments

config            List of auto report configuration

### Value

List of (upgraded) auto report configuration

---

userInfo            *Provide user attributes based on environment context*

---

### Description

Extracts elements from either config, url (shiny) or session (shiny) relevant for user data such as name, group, role and reshId. 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,  
  shinySession = NULL,  
  devContexts = c("DEV"),  
  testContexts = c("TEST"),  
  prodContexts = c("QA", "PRODUCTION")  
)
```

**Arguments**

entity	String defining the element to return. Currently, one of 'user', 'groups', 'resh_id', 'role', 'email', 'full_name' or 'phone'
shinySession	Shiny session object (list, NULL by default). Must be provided when the source of user attributes is either the shiny app url or an external authentication provider. By default this will apply to the 'TEST', 'QA' and 'PRODUCTION' contexts in which case the shiny session object must be provided.
devContexts	A character vector providing unique instances to be regarded as a development context. In this context user attributes will be read from configuration as provided by 'rapbaseConfig.yml'. The instances provided cannot overlap instances provided in any other contexts. By default set to c("DEV").
testContexts	A character vector providing unique instances to be regarded as a test context. In this context user attributes will be read from the url call to a shiny application. Hence, for this context the corresponding shiny session object must also be provided. The instances provided cannot overlap instances provided in any other contexts. By default set to c("TEST").
prodContexts	A character vector providing unique instances to be regarded as a production context. In this context user attributes will be read from the shiny session object (as shiny server interacts with an external log-in service). Hence, for this context the corresponding shiny session object must also be provided. The instances provided cannot overlap instances provided in any other contexts. By default set to c("QA", "PRODUCTION").

**Value**

String of single user data element

**See Also**

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

---

writeAutoReportData     *Write automated report metadata*

---

**Description**

Write automated report metadata

**Usage**

```
writeAutoReportData(
  fileName = "autoReport.yml",
  config,
  packageName = "rapbase"
)
```

**Arguments**

fileName	String defining name of the yaml configuration file. Default 'autoReport.yml'
config	a list of yaml configuration
packageName	String defining the package in which the above configuration file resides. A configuration file within an R-package is only used in case the environmental variable 'R_RAP_CONFIG_PATH' is not defined (empty)

**Examples**

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

# Index

## \* datasets

- appLog, 4
- .getFun, 3
- .testAutoReport, 3
  
- appLog, 4
- appLogger (logger), 26
- appNavbarUserWidget, 4
- autLogger (logger), 26
- autoReport, 6
- autoReportApp (autoReport), 6
- autoReportFormatInput (autoReport), 6
- autoReportFormatServer (autoReport), 6
- autoReportFormatServer (autoReport), 6
- autoReportInput (autoReport), 6
- autoReportOrgInput (autoReport), 6
- autoReportOrgServer (autoReport), 6
- autoReportServer (autoReport), 6
- autoReportUI (autoReport), 6
  
- cleanStagingData (stagingData), 39
- createAutoReport, 9, 11
  
- deleteAutoReport, 10, 11
- deleteStagingData (stagingData), 39
- describeRegistryDb (loadRegData), 25
  
- export, 11
- exportDb (export), 11
- exportGuide, 12
- exportGuideApp (exportGuide), 12
- exportGuideServer (exportGuide), 12
- exportGuideUI (exportGuide), 12
- exportUcApp (export), 11
- exportUcInput (export), 11
- exportUcServer (export), 11
  
- filterAutoRep, 13
- findNextRunDate, 14
- fireInTheHole, 15
  
- getConfig, 15
- getGithub, 16
- getRapPackages, 17, 24
- getRegs, 17
- getUserEmail, 18, 18, 19, 21
- getUserFullName, 18, 21
- getUserGroups, 18, 19, 19, 20–23, 44
- getUserNames, 18–20, 20, 21–23, 44
- getUserPhone, 19, 21
- getUserReshId, 18–21, 22, 23, 44
- getUserRole, 20, 22, 22, 44
  
- halloRapporteket, 23
- howWeDealWithPersonalData, 24
  
- isPkgRapReg, 24
- isRapContext, 25
  
- listStagingData (stagingData), 39
- loadRegData, 25
- loadStagingData (stagingData), 39
- logFormat (stats), 40
- logger, 26
- logTimeFrame (stats), 40
  
- makeAutoReportTab, 29
- makeRunDayOfYearSequence, 30
- makeStandardTable, 31
- mst (makeStandardTable), 31
- mtimeStagingData (stagingData), 39
  
- navbarWidget, 32
- navbarWidgetApp (navbarWidget), 32
- navbarWidgetInput (navbarWidget), 32
- navbarWidgetServer (navbarWidget), 32
- noOptOutOk, 33
  
- orgList2df (autoReport), 6
  
- pathStagingData (stagingData), 39

- rapbase, [34](#)
- rapCloseDbConnection, [34](#)
- rapOpenDbConnection, [34](#)
- readAutoReportData, [35](#)
- renderRmd, [35](#)
- repLogger (logger), [26](#)
- runAutoReport, [36](#)
- runBulletin, [37](#)
- runNoweb, [37](#)
  
- saveStagingData (stagingData), [39](#)
- selectListPubkey (export), [11](#)
- sendEmail, [38](#)
- seq.POSIXt, [10](#)
- stagingData, [39](#)
- stats, [40](#)
- statsApp (stats), [40](#)
- statsGuide, [42](#)
- statsGuideApp (statsGuide), [42](#)
- statsGuideServer (statsGuide), [42](#)
- statsGuideUI (statsGuide), [42](#)
- statsInput (stats), [40](#)
- statsServer (stats), [40](#)
- statsUI (stats), [40](#)
- strptime, [14](#)
  
- upgradeAutoReportData, [43](#)
- userInfo, [18–22](#), [43](#)
  
- writeAutoReportData, [44](#)