Package 'NVIdb'

September 19, 2025

Title Tools to facilitate the use of NVI's databases

Version 0.14.0

Date 2025-09-19

Description Provides tools to facilitate downloading and processing of data from the Norwegian Veterinary Institute's databases. The package comprises several categories of functions.
1) Manage credentials (i.e. password and username),
2) Login functions for database services,
3) Read, copy and update various in-house data registers,
4) Translate codes into descriptions.
NVIdb is dependant of NVIconfig which has to be installed manually from GitHub.
All PJS and EOS related functions are moved to NVIpjsr.

URL https://github.com/NorwegianVeterinaryInstitute/NVIdb

BugReports https://github.com/NorwegianVeterinaryInstitute/NVIdb/issues

Depends R (>= 4.1.0)

License BSD_3_clause + file LICENSE

Imports askpass,
checkmate,
data.table,
DBI,
dplyr,
keyring,
odbc,
RODBC,
RPostgreSQL,
snakecase,
stats,
utils,
NVIcheckmate (>= 0.8.0)

Suggests covr,
desc,
devtools,
knitr,
R.rsp,
remotes,
rmarkdown,
spelling,

testthat,
tibble,
NVIpackager,
NVIrpackages

Remotes NorwegianVeterinaryInstitute/NVIcheckmate,
NorwegianVeterinaryInstitute/NVIrpackages,
NorwegianVeterinaryInstitute/NVIpackager

LazyData true

Encoding UTF-8

Language en-GB

Roxygen list(markdown = FALSE)

RoxygenNote 7.3.2

VignetteBuilder knitr,
R.rsp

Contents

`add_kommune_fylke` *Manage translation from komnr to kommune, fylke and current komnr*

Description

Function to add columns with kommune (name), fylkenr, fylke (name), gjeldende_komnr, gjeldende_kommune, gjeldende_fylkenr, and gjeldende_fylke. In addition there are functions to read and copy the translation tables.

Usage

```
add_kommune_fylke(
  data,
  translation_table = kommune_fylke,
  code_column = "komnr",
  new_column = c("gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr",
    "gjeldende_fylke"),
  year = format(Sys.Date(), "%Y"),
  position = "right",
  overwrite = FALSE
)

copy_kommune_fylke(
  filename = list("komnr_2_gjeldende_komnr2_UTF8.csv", "Fylke_UTF8.csv"),
 from_path = file.path(set_dir_NVI("GrunndataLand", slash = FALSE), "FormaterteData"),
  to_path = NULL
)

read_kommune_fylke(
  filename = list("komnr_2_gjeldende_komnr2_UTF8.csv", "Fylke_UTF8.csv"),
 from_path = file.path(set_dir_NVI("GrunndataLand", slash = FALSE), "FormaterteData"),
  ...
)
```

Arguments

`data`	[data.frame] Data with a column with kommunenummer (komnr) or fylkesnummer (fylkenr).
`translation_table`	[data.frame] The translation table for translating komnr to kommune, fylke etc. Defaults to `kommune_fylke`.
`code_column`	[character(1)] The name of the column with the code value. Valid values are one of c("komnr", "fylkenr"). If the column in data has another name, it can be input as a named vector, see examples. Defaults to "komnr".
`new_column`	[character] The name(s) of the new column(s) that should be added to the data. Defaults to c("gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr", "gjeldende_fylke").
`year`	[integer(1) | character(1)] The year for which the komnr should be translated to valid komnr. Defaults to the current year i.e. format(Sys.Date(), "%Y").
`position`	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
`overwrite`	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if `overwrite = TRUE`. If the new column(s) already exist and `overwrite = FALSE`, an error is issued. Defaults to `FALSE`.

filename	[list] File names of the source files for the translation table. Defaults to list("komnr_2_gjeldende_komnr2_U "Fylke_UTF8.csv").
from_path	[character(1)] Path to the source file(s) used to generate the translation table. Defaults to their standard directory at NVI's internal network.
to_path	[character(1)] Path to where the source file(s) used to generate the translation table should be copied.
...	Other arguments to be passed to utils::read.csv2.

Details

Data sources, like PJS, may provide data with komnr. These functions manage translation of komnr to current komnr, kommune and fylke.

add_kommune_fylke can be used to translate komnr into kommune (name), fylkenr, fylke (name), gjeldende_komnr, gjeldende_kommune, gjeldende_fylkenr, and gjeldende_fylke. The function can also be used to translate fylkenr into fylke (name), gjeldende_fylkenr, and gjeldende_fylke.

add_kommune_fylke will translate komnr from 1977 to the komnr for the year given in the argument year. If a year previous to the current year is given, only years before the given year will be translated. Newer years cannot be translated to older year by this function.

The translation of komnr and fylkenr is simplified. If there have been changes in the borders between kommuner, this is not taken into account. If kommuner have been split the komnr is translated to the new kommune that has taken most of the area in the old kommune. I.e. in 2020, the kommune Snillfjord was split into Orkland, Hitra and Heim, but are translated into Orkland. In 2024, the kommune Ålesund was split into Ålesund and Haram, but are translated into Ålesund. For produsenter you need to first find the the correct new produsent-number by using add_produsent_properties and thereafter assign them to the correct kommune.

When translating the fylkenr, the fylker "Viken", "Vestfold og Telemark" and "Troms og Finnmark" that was established in 2020 and split in 2024, will not be possible to be translate to new fylker without additional information. If the komnr is translated, the new fylke will also be translated.

One has to ensure that the code in the dataset represents a komnr or fylkenr. The function will translate any 4 and 2 digits that has the same ID as a kommune or fylke, respectively.

Standard name for the kommunenummer is komnr. If the column with the komnr that should be translated has another name, the parameter code_column can be input as a named vector. Standard names for the new columns are c("kommune", "fylkenr", "fylke", "gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr", "gjeldende_fylke"). Likewise, if the new columns should be given other names than, the parameter new_column can be input as a named vector, see examples.

The function uses a premade translation table that is made based on information in PJS adresseregister. The translation table is updated when informed that know there is a need, typically when there have been changes in kommune structure.

position is used to give the place if the new columns in the data frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_kommune_fylke read the files "komnr_2_gjeldende_komnr2_UTF8.csv" and "Fylke_UTF8.csv", and generates the translation table as a single data frame that can be used by add_kommune_fylke. Standard setting will read in the file from NVI's internal network. If changing the from_path, the function can be used to read the translation files from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_kommune_fylke copy the files "komnr_2_gjeldende_komnr2_UTF8.csv" and "Fylke_UTF8.csv" to a given directory.

Value

add_kommune_fylke A data frame where one or more of the columns c("kommune", "fylkenr", "fylke", "gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr", "gjeldende_fylke") have been added in the column(s) to the right of the column with the komnr.

read_kommune_fylke A data frame with the original komnr and the corresponding kommune, fylkenr, fylke, and the current komnr, kommune, fylkenr, fylke. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_kommune_fylke copies the source translation table for komnr to kommune, for old komnr to current komnr, and for fylkenr to fylke to given directory. If the target file already exists, the source file is copied only when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
kommune_fylke <- read_kommune_fylke()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_kommune_fylke(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
kommune_fylke <- read_kommune_fylke(from_path = "./Data/")

# Add new column with current komnr and kommune
# The variable gammelt_komnr should be translated and the new variables with gjeldende_komnr" and
# "gjeldende_kommune" is named komnr and kommune, respectively.
newdata <- add_kommune_fylke(olddata,
                             translation_table = kommune_fylke,
                             code_column = c("gammelt_komnr" = "komnr"),
                             new_column = c("komnr" = "gjeldende_komnr",
                                            "kommune" = "gjeldende_kommune"
)
)

## End(Not run)
```

add_lokalitet *Manage adding extra information to aquaculture sites*

Description

Function to add a column with current aquaculture zone and/or geo-coordinates. In addition there are function to read the translation table.

Usage

```
add_lokalitet(
  data,
  translation_table,
  code_column,
  new_column,
  position = "right",
  overwrite = FALSE
)

read_sonetilhorighet(
  filename = "sonetilhorighet.txt",
 from_path = paste0(set_dir_NVI("EksterneDatakilder"), "Lokreg/FormaterteData/Soner/"),
  ...
)
```

Arguments

data	[data.frame] Data with a column with an aquaculture site number ("LokNr")
translation_table	[data.frame] Table for translating from loknr to the property in question.
code_column	[character(1)] The column with the coded value. Valid values are one of c("LokNr"). If the column in data has another name, it can be input as a named vector, see examples.
new_column	[character] The new columns that should be included into the data frame.
position	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
overwrite	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.
filename	[list] The filenames of the source files with the tables for generating the translation table.

from_path	[character(1)] Path to the source file(s) used to generate the translation table. Defaults to their standard directory at NVI's internal network.
...	Other arguments to be passed to data.table::fread.

Details

add_lokalitet can be used to add aquaculture zone and/or geo-coordinates to aquaculture sites. The new columns can be one or more of c("sone", "EastUTM_33N_WGS84", "NorthUTM_33N_WGS84", "Longitude_WGS84", "Latitude_WGS84"). If the new columns in the result data frame should have other names, new_column can be input as a named vector, see examples.

position is used to give the position if the new columns in the data.frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_sonetilhorighet reads the file "sonetilhorighet.txt" into a data frame that can be used by other routines. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

add_lokalitet: data.frame where the aquaculture zone and / or geo-coordinates have been added in the column to the right of the column with the LokNr.

read_sonetilhorighet: data.frame with "LokNr", aquaculture zone and geo-coordinates. If not changing standard input to the function, the standard file at NVI's internal network is read.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:

# READ TRANSLATION TABLE
# Reading from standard directory at NVI's network
sonetilhorighet <- read_sonetilhorighet()

# ADD AQUACULTURE ZONE
eier_lokalitetnr <- c("10298", "10318", "10735", "10814")
olddata <- as.data.frame(eier_lokalitetnr)

# Add new column with aquculture zone
newdata <- add_lokalitet(olddata,
                         translation_table = sonetilhorighet,
                         code_column = c("eier_lokalitetnr" = "LokNr"),
                         new_column = c("produksjonsomraade" = "sone"),
                         position = "left")
```

```
# ADD COORDINATES
eier_lokalitetnr <- c("10298", "10318", "10735", "10814")
olddata <- as.data.frame(eier_lokalitetnr)

# Add new columns with longitude and lattitude
newdata <- add_lokalitet(olddata,
                         translation_table = sonetilhorighet,
                         code_column = c("eier_lokalitetnr" = "LokNr"),
                         new_column = c("longitude" = "Longitude_WGS84",
                                        "latitude" = "Latitude_WGS84"))

## End(Not run)
```

---

add_MT_omrader *Manage translation from komnr to MT-avdeling and MT-region*

Description

Function to add columns with MT_avdelingnr, MT_avdelng (name), MT_regionnr and MT_region (name). In addition there are functions to read and copy the translation tables.

Usage

```
add_MT_omrader(
  data,
  translation_table = komnr_2_MT_omrader,
  code_column = c("komnr"),
  new_column = c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"),
  position = "right",
  overwrite = FALSE
)

copy_MT_omrader(
  filename = list("komnr_2_MT_avdeling.csv", "MT_omrader.csv"),
  from_path = base::paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  to_path = NULL
)

read_MT_omrader(
  filename = list("komnr_2_MT_avdeling.csv", "MT_omrader.csv"),
  from_path = base::paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  ...
)
```

Arguments

data	[data.frame] Data with a column with kommunenummer (komnr).
translation_table	[data.frame] The translation table for translating komnr to MT-areas. Defaults to komnr_2_MT_omrader.

`code_column` [character(1)]
The column with the code value. Valid values are one of c("komnr", "MT_avdelingnr", "MT_regionnr"). If the column in data has another name, it can be input as a named vector, see examples. Defaults to "komnr".

`new_column` [character]
The name(s) of the new column(s) that should be added to the data, see examples. Defaults to c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region").

`position` [character(1)]
The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".

`overwrite` [logical(1)]
When the new column(s) already exist, the content in the existing column(s) is replaced by new data if `overwrite = TRUE`. If the new column(s) already exist and `overwrite = FALSE`, an error is issued. Defaults to `FALSE`.

`filename` [list]
File names of the source files for the translation table. Defaults to list("komnr_2_MT_avdeling.csv", "MT_omrader.csv").

`from_path` [character(1)]
Path to the source file(s) used to generate the translation table. Defaults to their standard directory at NVI's internal network.

`to_path` [character(1)]
Path to where the source file(s) used to generate the translation table should be copied.

`...` Other arguments to be passed to `utils::read.csv2`.

Details

`add_MT_omrader` can be used to translate the komnr into MT_avdelingnr, MT_avdeling, MT_regionnr and MT_region. The function can also be used to translate MT_avdelingnr into MT_avdeling, MT_regionnr and MT_region or to translate MT_regionnr into MT_region. When the `code_column` in the dataframe is not equal to one of c("komnr", "MT_avdelingnr", "MT_regionnr") the `code_column` can be input as a named vector. Likewise, if the new columns should be given other names than c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"), the `new_column` can be input as a named vector, see examples.

`add_MT_omrader` uses a premade translation table ("komnr_2_MT_avdeling.csv"). These data need to be loaded by `read_MT_omrader` before running `add_MT_omrader`, see example. "komnr_2_MT_avdeling.csv" is made based on information in PJS adresseregister. The translation table is updated when we know there is a need.

`position` is used to give the place if the new columns in the data frame. For `position = "right"` the new variables are placed to the right of the code_variable. Likewise, for `position = "left"` the new variables are placed to the left of the code_variable. If `position = "first"` or `position = "last"` the new columns are placed first or last, respectively, in the data.frame. A special case occurs for `position = "keep"` which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

`read_MT_omrader` reads the files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" into a data frame, usually named komnr_2_MT_omrader. This file is used by `add_MT_omrader`. If no options to the function is given, the function will read the latest updated files from NVI's internal network. If

changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a script that don't have access to NVI's internal network.

copy_MT_omrader Copies the csv-files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" to another directory. Thereby, these files are available for read_MT_omrader if they should be read from another directory.

Value

add_MT_omrader A data frame where the MT_avdelingnr has been added in the column to the right of the column with the komnr.

read_MT_omrader A data frame with the table for translating from komnr to c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region") as read from the source csv file. If not changing standard input to the function, the standard files at NVI's internal network is read.

copy_MT_omrader Copies the csv-files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" to another directory. If the target files already exists the source files are only copied if they are newer than the target files.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
komnr_2_MT_omrader <- read_MT_omrader()

# Copy the csv-files used to generate the translation table from the standard location to
# the subdirectory Data below the working directory
copy_MT_omrader(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
komnr_2_MT_omrader <- read_MT_omrader(from_path = "./Data/")

# Add new columns with MT_avdelingnr, MT_avdeling, MT_regionnr, and MT_region based on komnr
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
                          translation_table = list(komnr_2_MT_omrader),
                          code_column = "komnr",
                 new_column = c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"))

# Add new columns with MT_avdelingnr and MT_avdeling based on komnr. The colname of the column
# with komnr is komnr and the new columns are renamed to MT_avdnr and MT_avd.
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
                          translation_table = list(komnr_2_MT_omrader),
                          code_column = c("komnr" = "komnr"),
                    new_column = c("MT_avdnr" = "MT_avdelingnr", "MT_avd" = "MT_avdeling"))

# Add new columns with MT_region based on MT_regionnr. MT_region is renamed to MT_regionnavn
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
```

```
                        translation_table = list(komnr_2_MT_omrader),
                        code_column = "MT_region",
                        new_column = c("MT_regionnavn" = "MT_region"))

## End(Not run)
```

---

add_poststed *Manage translation from postnr to poststed and komnr*

Description

Function to add columns with poststed and komnr. In addition there are functions to read and copy the translation tables.

Usage

```
add_poststed(
  data,
  translation_table = poststed,
  code_column = c("postnr"),
  new_column = c("poststed", "komnr"),
  position = "right",
  overwrite = FALSE
)

copy_poststed(
  filename = "Poststed_UTF8.csv",
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  to_path = NULL
)

read_poststed(
  filename = "Poststed_UTF8.csv",
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  ...
)
```

Arguments

data	[data.frame] Data with a column with postnummer (postnr).
translation_table	[data.frame] The translation table for translating postnr to poststed and komnr. Defaults to poststed.
code_column	[character(1)] The name of the column with the code value. Valid value "postnr". If the column in data has another name, it can be input as a named vector, see examples. Defaults to "postnr".

new_column	[character] The name(s) of the new column(s) that should be added to the data. Defaults to c("poststed", "komnr").
position	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
overwrite	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.
filename	[character(1)] File names of the source files for the translation table. Defaults to "Poststed_UTF8.csv".
from_path	[character(1)] Path to the source file(s) used to generate the translation table. Defaults to their standard directory at NVI's internal network.
to_path	[character(1)] Path to where the source file(s) used to generate the translation table should be copied.
...	Other arguments to be passed to utils::read.csv2.

Details

Data sources may provide data with postnr. These functions manage translating postnr to poststed and komnr.

add_poststed can be used to translate postnr to poststed and komnr.

One has to ensure that the code in the data column represents a postnr. The function will translate any 4 digits that has the same ID as a postnr.

Standard name for the postnr is postnr. If the column with the postnr that should be translated has another name, the parameter code_column can be input as a named vector. Standard names for the new columns are c("poststed", "komnr"). Likewise, if the new columns should be given other names than these, the parameter new_column can be input as a named vector, see examples.

add_poststed uses a premade translation table ("Poststed_UTF8.csv"). These data need to be loaded by read_poststed before running add_poststed, see example. "Poststed_UTF8.csv" is made based on information in PJS adresseregister. The translation table is updated when we know there is a need.

position is used to give the place if the new columns in the data frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_poststed read the file "Poststed_UTF8.csv" a data frame that can be used by add_poststed. Standard setting will read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation files from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_poststed copy the file "Poststed_UTF8.csv" to a given directory.

Value

add_poststed A data frame where one or more of the columns c("poststed", "komnr") have been added in the column(s) to the right of the column with the postnr.

read_poststed A data frame with the original postnr and the corresponding poststed and komnr. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_poststed copies the source translation table for postnr to poststed and komnr to given directory. If the target file already exists, the source file is copied only when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
poststed <- read_poststed()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_poststed(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
poststed <- read_poststed(from_path = "./Data/")

# Add new column with poststed and komnr
# The variable postnummer should be translated into poststed and komnr. For poststed
# the standard name is kept. For komnr the new variable is named postkomnr.
# Remember to load "poststed" by "read_poststed()" before running "add_poststed",
# see above.
newdata <- add_poststed(olddata,
                        translation_table = poststed,
                        code_column = c("postnummer" = "postnr"),
                        new_column = c("poststed", "postkomnr" = "komnr"))

## End(Not run)
```

---

add_produsent_properties

Manage translation from prodnr8 into different produsent properties

---

Description

Function to add a column with gjeldende_prodnr8 and/or geo-coordinates. In addition there are functions to read and copy the translation tables.

Usage

```
add_produsent_properties(
  data,
  translation_table,
```

```
  code_column,
  new_column,
  position = "right",
  overwrite = FALSE,
  impute_old_when_missing = FALSE
)

copy_prodnr_2_current_prodnr(
  filename = "Prodnr2GjeldendeProdnr.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/"),
  to_path = NULL
)

read_prodnr_2_coordinates(
  filename = "Prodnr2Koordinater.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/"),
  ...
)

read_prodnr_2_current_prodnr(
  filename = "Prodnr2GjeldendeProdnr.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/"),
  ...
)
```

Arguments

`data`	[data.frame] Data with a column with produsentnummer (prodnr8).
`translation_table`	[data.frame] The translation table for translating "prodnr8" to "gjeldende_prodnr8" or the translation table for translating "prodnr8" to geo-coordinates.
`code_column`	[character(1)] The name of the column with the code value. Standard value is "prodnr8". If the column in data has another name, it can be input as a named vector, see examples.
`new_column`	[character] The name(s) of the new column(s) that should be added to the data, see examples.
`position`	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
`overwrite`	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if `overwrite = TRUE`. If the new column(s) already exist and `overwrite = FALSE`, an error is issued. Defaults to `FALSE`.
`impute_old_when_missing`	[logical(1)] Should the ID-variable be used as value for the `new_column` if the `new_column`

value is missing? Default is FALSE. To be used when translating prodnr8 to gjeldende_prodnr8, see details.

filename [character(1)]
File name of the source file for the translation table.

from_path [character(1)]
Path to the source file(s) used to generate the translation table. Defaults to their standard directory at NVI's internal network.

to_path [character(1)]
Path to where the source file(s) used to generate the translation table should be copied.

... Other arguments to be passed to data.table::fread.

Details

add_produsent_properties can be used to translate the prodnr8 into gjeldende_prodnr8 and/or geo-coordinates.

position is used to give the place if the new columns in the data frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

impute_old_when_missing = TRUE is used to replace missing values in the new_column with the value in code_column. This is useful when translating prodnr8 to gjeldende_prodnr8 as in the case where gjeldende_prodnr8 is not found, the value from prodnr8 is imputed. It should not be used when translating from prodnr8 to something where imputing the old prodnr8 in the new variables don't have any meaning, for example geo-coordinates.

read_prodnr_2_current_prodnr reads the file "Prodnr2GjeldendeProdnr.csv" into a data frame that can be used by other routines. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_prodnr_2_current_prodnr copies the file "Prodnr2GjeldendeProdnr.csv" to a chosen directory.

read_prodnr_2_coodinates reads the file "Prodnr2Koordinater.csv" into a data frame that can be used to merge with data frames with prodnr8. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

add_produsent_properties returns a data frame where the column with produsent properties has been added to the data.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# CURRENT PRODNR8
# Reading from standard directory at NVI's network
prodnr_2_gjeldende_prodnr <- read_prodnr_2_current_prodnr()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_prodnr_2_current_prodnr(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
prodnr_2_gjeldende_prodnr <- read_prodnr_2_current_prodnr(from_path = "./Data/")

prodnr8 <- c("09140087", "14260856", "17020818", "50060129")
olddata <- as.data.frame(prodnr8)

# Add new column with current prodnr8
newdata <- add_produsent_properties(olddata,
                                    translation_table = prodnr_2_gjeldende_prodnr,
                                    code_column = "prodnr8",
                                    new_column = "gjeldende_prodnr8",
                                    position = "left",
                                    impute_old_when_missing = TRUE)

# COORDINATES
# Reading from standard directory at NVI's network
prodnr_2_koordinater <- read_prodnr_2_coordinates()

newdata <- add_produsent_properties(newdata,
                                    translation_table = prodnr_2_koordinater,
                                    code_column = "prodnr8",
                                    new_column = c("longitude" = "geo_eu89_o",
                                                   "latitude" = "geo_eu89_n"))

## End(Not run)
```

---

copy_Pkode_2_text *Manage translation table for produksjonstilskuddskoder (Pkoder)*

Description

Read and copy the translation table for produksjonstilskuddskoder (Pkoder).

Usage

```
copy_Pkode_2_text(
  filename = "Produksjonstilskuddskoder2_UTF8.csv",
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "StotteData/"),
  to_path = NULL
)

read_Pkode_2_text(
  filename = "Produksjonstilskuddskoder2_UTF8.csv",
```

```
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "StotteData/"),
  keep_old_names = FALSE,
  ...
)
```

Arguments

filename	[character(1)] Name of the translation table. Defaults to "Produksjonstilskuddskoder2_UTF8.csv".
from_path	[character(1)] Path for the translation table for produksjonstilskuddskoder. Defaults to standard directory at the NVI network.
to_path	[character(1)] Path for the target translation table when copying produksjonstilskuddskoder.
keep_old_names	[logical(1)] Keep old column names as were used as standard in NVIdb <= v0.7.1. Defaults to FALSE.
...	Other arguments to be passed to utils::read.csv2.

Details

The translation table for Pkoder contains the Pkode, descriptive text, unit of interest (Dyr), whether the code counts unique animals or not (used when summarising number of animals), and sortering to order the Pkoder. The register covers 2017 and later.

read_Pkode_2_text reads the file "Produksjonstilskuddskoder2_UTF8.csv" into a data frame. The default is to read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_Pkode_2_text copies the file "Produksjonstilskuddskoder2_UTF8.csv" to a given location.

Value

read_Pkode_2_text A data frame with the translation table for Pkoder to description as read from the csv file. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_Pkode_2_text Copies the source translation table "Produksjonstilskuddskoder2_UTF8.csv" to another location. If the target file already exists, the source file is only copied when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
Pkode_2_text <- read_Pkode_2_text()

# Copy standard file from standard location to the sub directory Data below the working directory
copy_Pkode_2_text(to_path = "./Data/")
```

```
# Reading from the sub directory Data below the working directory
Pkode_2_text <- read_Pkode_2_text(from_path = "./Data")

## End(Not run)
```

`copy_Prodtilskudd` *Read Register for søknad om produksjonstilskudd*

Description

Functions to to read and copy versions of the produksjonstilskuddsregister.

Usage

```
copy_Prodtilskudd(
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "FormaterteData/"),
  to_path = NULL,
  Pkode_year = "last",
  Pkode_month = "both",
  extracted_date = NULL
)

read_Prodtilskudd(
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "FormaterteData/"),
  Pkode_year = "last",
  Pkode_month = "both",
  extracted_date = NULL,
  ...
)
```

Arguments

`from_path`	[character(1)] Path for the produksjonstilskuddsregister. Defaults to the standard directory at the NVI network.
`to_path`	[character(1)] Target path for the files with the produksjonstilskuddsregister.
`Pkode_year`	[character] | [numeric] The year(s) from which the register should be read. Options is "last", or a vector with one or more years. Defaults to "last".
`Pkode_month`	[character] The month for which the register should be read. The options are c("05", "10", "both", "last") for `Pkode_year = 2017` and c("03", "10", "both", "last") for `Pkode_year >= 2018`. Defaults to "both".
`extracted_date`	[character] The date the data was extracted from the database of the Norwegian Agricultural Agency. The format should be "yyyy-mm-dd". Defaults to `NULL`.
`...`	Other arguments to be passed to `data.table::fread`.

Details

The produksjonstilskuddsregister includes information on number of animals that the produsent has applied subsidies for at the counting dates. Since 2017, the counting dates are in March and October. Landbruksdirektoratet provides three to four versions of the register for each counting date. The functions automatically selects the last updated version of the register.

`read_Prodtilskudd` reads the produksjonstilskuddsregister into a data frame. The function gives options to select year and season The standard settings will read in the files from NVI's internal network and select the latest updated file for both spring and autumn and combine them into one file. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

`extracted_date` is used if specific versions of the register is required, for example to reproduce the generation of data previously performed using an older version of the register.You should also write in the `extracted_date` in the script to document which version of the register that was used. If so, first extract the last available version of the register. Find the uttrekkdato in the data, and write in the uttrekkdato in `extracted_date`. `extracted_date` cannot be used in combination with `pkode_year = "last"` or `pkode_month = c("last", "both")`.

`copy_Prodtilskudd` copies the source produksjonstilskuddsregister for each of the year and seasons selected to a given directory.

Value

`read_Prodtilskudd` reads one or more data frame(s) with the produksjonstilskuddsregister for each of the year and seasons selected. If the options `Pkode_year = "last"` and `Pkode_month = "last"` is given, one file with the last produksjonstilskuddsregister is given.

`copy_Prodtilskudd` copies the source produksjonstilskuddsregister for each of the year and seasons selected. If the target file already exists, the source files are copied only when newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
Pkode_last <- read_Prodtilskudd()

# Reading from standard directory at NVI's network and
#     selecting a specific version of the register
Pkode201903 <- read_Prodtilskudd(Pkode_year = "2019", Pkode_month = "03")

## End(Not run)
```

cut_slash *Cut away ending slash from string*

Description

Removes ending slash or backslash from string. This is used to clean pathnames so that elements in a path can be combined using `file.path` in stead of `paste0`.

Usage

```
cut_slash(x)
```

Arguments

`x`	[`character`] Strings, typically file paths, where ending slash should be removed.

Value

Object without ending slash in character strings.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# Remove from string
cut_slash("C:/temp/")
cut_slash("C:\\temp\\")
cut_slash(c("C:/temp/", "C:\\temp\\"))
cut_slash(list("C:/temp/", "C:\\temp\\"))
```

find_file_near_date *Finds a file based on date in file name*

Description

Identifies the file name of a file based on date information in the file name. This is usually used to identify the file name of a file with data that has a register date nearest the wanted date. The register date must be included in the file name and written as _per_DATE, see details.

Usage

```
find_file_near_date(
  from_path,
  partial_filename = NULL,
  extension = NULL,
  date_position = "last",
  wanted_date = Sys.Date(),
  wanted_year = NULL,
  wanted_month = NULL,
  nearest_wanted = "before"
)
```

Arguments

from_path	[character(1)] Full path for the directory with the files. Defaults to NULL.
partial_filename	[character] One or more patterns within the file name that will be used to identify a subset of files in the directory that can be selected. Defaults to NULL.
extension	[character(1)] The file extension of the files that can be selected. Defaults to NULL.
date_position	[character(1)] The position of the date in the file name. Must be one of c("last", "per"). Defaults to "last".
wanted_date	[character(1)] The date for which the register date should be nearest. Defaults to Sys.Date().
wanted_year	[character(1)] | [numeric(1)] The year from which the register should be read, see details. Defaults to NULL.
wanted_month	[character] The month for which the register should be read, see details. Defaults to NULL.
nearest_wanted	[character(1)] Whether the nearest register date before the wanted date or the nearest date after the wanted date should be selected, see details. Must be one of c("before", "after"). Defaults to "before".

Details

Reads the file names of files in a given directory and selects the file nearest a wanted date based on date information in the file name. The register date is used to identify the file name of the data that either has the last register date before the wanted date (the default) or the first register date after the wanted date. The function assumes that the register date for the data is written in the file name either as a "DATE" placed last in the file name before the extension or as "_per_DATE" that can be written anywhere in the file name. DATE should be given either as a date (formatted as %Y%m%d), a year_month (formatted as %Y%m) or a year (formatted as %Y). If one want the last register date before the wanted date and the "per_DATE" is given as year, the file will be identified as before the wanted date for any wanted date within the register year. Likewise, If one want the last register date after the wanted date and the "per_DATE" is given as year, the file will be identified as after the wanted date for any wanted date within the register year. I.e. for the file name "data_per_2025.csv" this file name will be found for the dates c("2025-01-01", "2025-12-31")

and all between for both nearest_to_per_date = "before" and nearest_to_per_date = "after". This assumes that there are no other files for that year with a more precise register date. If one want the last register date before the wanted date and the "per_DATE" is given as year_month, the file will be identified as before the wanted date for any wanted date within the register month. Likewise, If one want the last register date after the wanted date and the "per_DATE" is given as year_month, the file will be identified as after the wanted date for any wanted date within the register month. I.e. for the file name "data_per_202503.csv" this file name will be found for the dates c("2025-03-01", "2025-03-31") and all between for both nearest_to_per_date = "before" and nearest_to_per_date = "after". This assumes that there are no other files for that year_month with a more precise register date.

Value

A data frame with the file name of the file that fulfills the selection criteria.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Identify the filename of the last Purkering register before November 2023
filelist <- find_file_near_date(
  from_path = file.path(NVIdb::set_dir_NVI("Ekst", slash = FALSE),
  "Purkeringer", "FormaterteData"),
  partial_filename = c("Purkering"),
  extension = "csv",
  date_position = "last",
  wanted_date = NULL,
  wanted_year = 2023,
  wanted_month = "11",
  nearest_wanted = "before")

## End(Not run)
```

---

login *Log in to data base services*

Description

Log in to NVI's data base services, in particular journal_rapp/PJS.

Usage

```
login(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL
```

```
)

login_by_credentials(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL
)

login_by_input(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL,
  dbtext = NULL
)
```

Arguments

dbservice	[character(1)] Name of the database service, for example "PJS" or "EOS". For database services where one don't use the premade wrappers, the name can be chosen freely, but must be the same as used in set_credentials.
dbdriver	[character(1)] Name of database engine. Defaults to NULL.
db	[character(1)] Name of database. Defaults to NULL.
dbserver	[character(1)] Name of database server. Defaults to NULL.
dbport	[character(1)] Port. Defaults to NULL.
dbprotocol	[character(1)] Protocol to be used. Defaults to NULL.
dbinterface	[character(1)] The R-package that is used for interface towards the data base. Defaults to NULL.
dbtext	[character(1)] Gives the possibility of showing another name than the dbservice in the windows asking for username and password when using login_by_input. Defaults to NULL.

Details

The NVI has access to several database services. These functions log in to such services. The functions provides methods to either log in using credentials set in the user profile by set_credentials

or use input windows for username and password. Thereby the hard coding of username and password can be avoided.

`login` is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter `dbservice`, for example `dbservice = "EOS"`.

Depending on whether username and password have been saved in the users profile at the current computer or not, the user is asked to input credentials.

`login_by_input` is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter `dbservice`. The user is always asked to input username and password.

`login_by_credentials` is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter `dbservice`. The user is never asked for username and password, and the function can only be used when the credentials previously have been set in the user's profile at the current computer.

`login_PJS`, `login_by_input_PJS`, and `login_by_credentials_PJS` are wrappers for the functions above where the specifications for the database journal_rapp/PJS have been pre set. The user only need to input username and password. In the case that the username and password for journal_rapp/PJS have been stored in the user profile at the current computer, the user is automatically logged in to journal_rapp. If the password is no longer valid, an error occur. If so, the user must update the username and password by `set_credentials_PJS`.

The wrapper functions `login_EOS`, `login_by_input_EOS`, and `login_by_credentials_EOS` have been deprecated.

The login functions returns an open ODBC-channel to the database service. The database can then be queried by using functions in the package used for data base interface. The data base interface must be one of `odbc`, `RODBC` or, `RPostgreSQL`. The default is given in NVIconfig and is `RODBC` for "SQL server" and `RPostgreSQL` for "PostgreSQL".

When the session is finished, the script shall close the ODBC-channel by `odbcClose("myodbcchannel")` or `odbcCloseAll` when using `RODBC`.

Value

An open ODBC-channel to the database service.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

See Also

`set_credentials`

Examples

```
## Not run:
library(RODBC)
journal_rapp <- login_PJS()
# Reads hensiktregistret from PJS
hensikter <- sqlQuery(journal_rapp,
                      "select * from v_hensikt",
                      as.is = TRUE,
                      stringsAsFactors = FALSE)
#
odbcClose(journal_rapp)

## End(Not run)
```

NVIdb *NVIdb: A package to facilitate the use of the Norwegian Veterinary Institute's databases.*

Description

The NVIdb package provides functions to facilitate downloading and processing of data from NVI's databases, in particular PJS and EOS. The package comprises four categories of functions: manage credentials, login, read and copy data from in-house data registers and translation of codes into descriptive text.

Manage credentials

Set and remove credentials (i.e. password and user name) in the user profile at the current machine. These functions makes it possible to connect to database services automatically in scripts while avoiding hard coding of the password and the user name.

Login

These functions use the credentials set by functions for managing credentials to automatically login to database services. If the credentials have not been set, there are also login function for interactive input of credentials in windows when running scripts.

Read, copy and update R-data from in-house data registers

The NVI has copies of several data registers like kommuneregister, fylkesregister, register for søknad om Produksjonstilskudd. The aim of these functions are to make these registers easily accessible for R-scripts to ensure that one always uses the latest version of data. These functions reads the registers from where they are saved at NVI internal file system. If necessary, there are function to copy or update the registers to local directories when local versions are needed for example for shiny applications.

Translate codes into descriptive text

Data often only includes codes that should be translated into the description text. These functions perform the translation for PJS-codes and codes like komnr.

Author(s)

Maintainer: Petter Hopp <Petter.Hopp@vetinst.no>

Authors:

- Johan Åkerstedt <Johan.Akerstedt@vetinst.no>

Other contributors:

- Norwegian Veterinary Institute [copyright holder]

See Also

Useful links:

- https://github.com/NorwegianVeterinaryInstitute/NVIdb
- Report bugs at https://github.com/NorwegianVeterinaryInstitute/NVIdb/issues

---

`PJS_code_description_colname`

Data: PJS_code_description_colname, standard column names for description texts for selected code variables in PJS.

Description

A data frame with the standard variable names (column names) for the code variables in PJS, their corresponding standard name of the column with the descriptive text and a column with the PJS type that will can be used to translate from the code variable to the descriptive text. The column names of the code variable are the standardised column names, i.e. after running `NVIdb::standardize_columns`.

The raw data can be edited in the "./data-raw/generate_PJS_code_description_colname.R". The `PJS_code_description_colname` is used by `NVIdb::add_PJS_code_description` when using the options `PJS_variable_type = "auto"` and/or `new_column = "auto"`.

Usage

```
PJS_code_description_colname
```

Format

A data frame with 3 variables:

code_colname column name for selected code variables in PJS and that have been standardized using `NVIdb::standardize_columns`

type the type of PJS variable as used by `NVIdb::add_PJS_code_description` to translate PJS-codes to description text

new_column The new standard column names for the corresponding code column name in PJS

Source

"./data-raw/generate_PJS_code_description_colname.R" in package `NVIdb`

PJS_levels *Data: Variables per PJS-level.*

Description

A data frame with the variable names (column names) in PJS and their corresponding PJS-level. The column names are the standardized column names, i.e. after running `NVIdb::standardize_columns`. The raw data can be edited in the "./data-raw/PJS_levels.xlsx" and the the code for preparing of the data frame is written in "./data-raw/generate_PJS_levels.R". The `PJS_levels` is used as input for `NVIdb::select_PJS_levels`.

Usage

```
PJS_levels
```

Format

A data frame with 9 variables:

variable column name for variables read from PJS and standardized using `NVIdb::standardize_columns`

sak columns at sak-level are given value 1

prove columns at prove-level are given value 1

delprove columns at delprove-level are given value 1

undersokelse columns at undersokelse-level are given value 1

resultat columns at resultat-level are given value 1

konklusjon columns at konklusjon-level are given value 1

subundersokelse columns at subundersokelse-level are given value 1

subresultat columns at subresultat-level are given value 1

Details

The variables included into a specific level is given the value 1, if not included they are given the value 0. To ensure that information on a specific level an be traced to the correct sak, all index variables are given value 1.

Source

"./data-raw/PJS_levels.xlsx" in package `NVIdb`

`read_avlsgris` *Read the register of avlsgris herds*

Description

Functions to read the register of avlsgris herds, i.e. pig nucleus and multiplier herds.

Usage

```
read_avlsgris(
 from_path = file.path(set_dir_NVI("EksterneDatakilder", slash = FALSE), "Avlsgris",
    "FormaterteData"),
  wanted_date = Sys.Date(),
  wanted_year = NULL,
  wanted_month = NULL,
  nearest_wanted = "before",
  ...
)
```

Arguments

`from_path`	[`character(1)`] Full path for the directory with the avlgris-register files. Defaults to standard directory at NVI's home network.
`wanted_date`	[`character(1)`] The date for which the register date should be nearest. Defaults to `Sys.Date()`.
`wanted_year`	[`character(1)`]	[`numeric(1)`] The year from which the register should be read, see details. Defaults to `NULL`.
`wanted_month`	[`character`] The month for which the register should be read, see details. Defaults to `NULL`.
`nearest_wanted`	[`character(1)`] Whether the nearest register date before the wanted date or the nearest date after the wanted date should be selected, see details. Must be one of `c("before", "after")`. Defaults to `"before"`.
`...`	Other arguments to be passed to `utils::read.csv2`.

Details

The avlsgris-register includes information on the avlsgris herds. The register is updated from the industry at least once a year.

`read_avlsgris` reads the avlsgris-register into a `data.frame`. The default is to read the last updated version of the register. You may change this by the input to wanted date or wanted year and month to read the register nearest (either before or after) the wanted date. See help for `find_file_near_date` for a full description of the use of wanted date or wanted year and month.

Value

A `data.frame` with the avlsgris-register.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading the last version from the standard directory at NVI's network
avlsgris <- read_avlsgris()

# Reading from standard directory at NVI's network and
#     selecting a specific version of the register
avlsgris2021 <- read_avlsgris(wanted_year = 2021, wanted_month = "12")

## End(Not run)
```

---

read_leveransereg *Read Leveranseregisteret for slakt*

Description

Functions to read Leveranseregisteret for slakt.

Usage

```
read_leveransereg(
  filename,
  from_path = paste0(set_dir_NVI("LevReg"), "FormaterteData/"),
  ...
)
```

Arguments

filename	[character(1)] Name of the file with Leveranseregisteret.
from_path	[character(1)] Path for the Leveranseregisteret. Defaults to the standard directory at the NVI network.
...	Other arguments to be passed to data.table::fread.

Details

The Leveranseregisteret for slakt includes information on carcasses delivered to slaughter. The register include identity of the farmer, slaughterhouse, date of slaughter, animal species, category (age group and sex), and weight. For poultry the individual animal is not reported, but number of slaughtered poultry per categories, slaughterhouse and date.

read_leveransereg reads the Leveranseregisteret for slakt into a data frame. The standard settings will read in the files from NVI's internal network. If changing the from_path, the function can be used to read Leveranseregisteret from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

read_LevReg A data frame with Leveranseregisteret as in selected csv-file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
LevReg2019 <- read_leveransereg(filename = "LevReg_201901_201912.csv")

## End(Not run)
```

---

read_purkering *Read the register of purkeringer*

Description

Function to read the register of purkeringer, i.e. the central herd and the satellite herds in the sow pools.

Usage

```
read_purkering(
 from_path = file.path(set_dir_NVI("EksterneDatakilder", slash = FALSE), "Purkeringer",
    "FormaterteData"),
  wanted_date = Sys.Date(),
  wanted_year = NULL,
  wanted_month = NULL,
  nearest_wanted = "before",
  ...
)
```

Arguments

from_path	[character(1)] Full path for the directory with the purkering-register files. Defaults to standard directory at NVI's home network.
wanted_date	[character(1)] The date for which the register date should be nearest. Defaults to Sys.Date().
wanted_year	[character(1)] | [numeric(1)] The year from which the register should be read, see details. Defaults to NULL.
wanted_month	[character] The month for which the register should be read, see details. Defaults to NULL.
nearest_wanted	[character(1)] Whether the nearest register date before the wanted date or the nearest date after the wanted date should be selected, see details. Must be one of c("before", "after"). Defaults to "before".
...	Other arguments to be passed to utils::read.csv2.

Details

The purkering-register includes information on the herds in the sow pools. The register is updated from the industry at least once a year.

`read_purkering` reads the purkering-register into a `data.frame`. The default is to read the last updated version of the register. You may change this by the input to wanted date or wanted year and month to read the register nearest (either before or after) the wanted date. See help for `find_file_near_date` for a full description of the use of wanted date or wanted year and month.

Value

A `data.frame` with the purkering-register.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading the last version from the standard directory at NVI's network
purkering <- read_purkering()

# Reading from standard directory at NVI's network and
#     selecting a specific version of the register
purkering2021 <- read_purkering(wanted_year = 2021, wanted_month = "12")

## End(Not run)
```

`read_varekode` *Manage translation table for varekoder til leveransregisteret*

Description

Read the translation table for varekoder til leveransregisteret.

Usage

```
read_varekode(
  filename = "varekoder.csv",
  from_path = paste0(set_dir_NVI("LevReg")),
  year = NULL,
  data_source = "formatted",
  ...
)
```

Arguments

filename	[character(1)] Name of the translation table, defaults to "varekoder.csv". The input is only used when data_source = "formatted".
from_path	[character(1)] Path for the translation table for varekoder. Defaults to standard directory at the NVI network.
year	[integer]	[character] Year(s) for fetching the varekoderegister. Defaults to NULL.
data_source	[character(1)] Reads "formatted" data or "raw" data. Defaults to "formatted".
...	Other arguments to be passed to utils::read.csv2.

Details

The translation table for varekoder comprises the variables: the leveranseaar, varekode, vare (descriptive text), dyreslag, vareart, dyrekategori, and varekategorikode. The register covers 2016 and later.

read_varekoder with the argument type = "formatted" reads the formatted "varekoder.csv" into a data frame. The standard settings will read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

read_varekoder with the argument type = "raw" reads the raw data as supplied from Landbruksdirektoratet into a data frame. Thereafter, these can be used to generate the formatted version. The standard settings will read the file from NVI's internal network and changing the path should be avoided.

Value

read_varekoder A data frame with the translation table for varekoder to descriptive text and metadata.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
varekoder <- read_varekode()

## End(Not run)
```

`remove_credentials` *Manage username and password (credentials) for database services at NVI*

Description

Save or remove the current user's username and password for the data base services at the Norwegian Veterinary Institute in the the user profile.

Usage

```
remove_credentials(dbservice)

set_credentials(dbservice)

set_credentials_PJS()
```

Arguments

`dbservice`	[`character(1)`] Name of the database service, for example "PJS" or "EOS". For database services where one don't use the premade wrappers, the name can be chosen freely, but must be the same as used in `login` and `login_by_credentials`.

Details

The Norwegian Veterinary Institute has access to various database services. To simplify the access to the database services when using R, the function `set_credentials` makes it possible to save the username and password (credentials) in the user profile at the current machine. When the username and password have been saved in the user profile, the functions `login` or `login_by_credentials` will automatically log in to the database services without any need of new input of username and password.

The user profile is not copied between computers. Consequently, if a user runs scripts with `login` on different computers, the credentials have to be saved at each computer separately.

`set_credentials(dbservice)` is used to set the username and password for a database service. The username and password are input using windows and saved in the users profile at the current computer. When the password for the database service have been changed, `set_credentials(dbservice)` can be used to update the password.

`set_credentials_PJS` is a wrapper for `set_credentials(dbservice)` used to set the username and password for journal_rapp/PJS. Journal_rapp has views to information in PJS and some other internal databases at NVI. The username and password are the same as for PJS. When the password for PJS have been changed, `set_credentials_PJS` can be used to update the password.

The wrapper function `set_credentials_EOS` is deprecated. Use `set_credentials(dbservice = "EOS")` instead.

`remove_credentials(dbservice)` is used to delete the credentials for a database service from the user's profile.

Value

`set_credentials` The username and password for a database service are saved in the user profile at the current computer.

`remove_credentials` The username and password for a database service are deleted from the user profile at the current computer.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

See Also

`login` and `login_by_credentials`

Examples

```
## Not run:
set_credentials(dbservice = "dbservice")

set_credentials_PJS()

set_credentials("EOS")

remove_credentials("PJS")

## End(Not run)
```

---

set_dir_NVI *Set directories for data sources at NVI*

Description

Set the directories for various data sources at NVI's network.

Usage

```
set_dir_NVI(datasource, slash = TRUE)
```

Arguments

`datasource`	[character(1)] The data source that one want to access. The input can be abbreviated and case is ignored. To identify short names for the available directories, use `set_dir_NVI(datasource = "?")`.
`slash`	[logical(1)] If `TRUE` the path ends with a slash. Defaults to `TRUE`.

Details

The Norwegian Veterinary Institute has standard data sources at fixed directories. The function returns the standard directory for the given data source. Thereby hard coding of the paths may be avoided.

The path ends with a slash as default. To facilitate the use of `file.path` you can use the argument `slash = FALSE` to avoid ending slash.

Value

The full path for the directory at NVI's network. The path ends with "/" as default, see details.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Set path_ProdTilskudd to path for Prodtilskudd at the NVI network
prodtilskudd_path <- set_dir_NVI(datasource = "ProdTilskudd")

# Set pathname to a file using 'file.path'
pathname <- file.path(set_dir_NVI(datasource = "ProdTilskudd", slash = FALSE),
                      "subdir",
                      "filename")

## End(Not run)
```

`standardize_columns` *Standardize columns for scripts and reports*

Description

Standardizes column names, labels, column width for variables in external databases.

Usage

```
standardize_columns(
  data,
  dbsource = deparse(substitute(data)),
  standards = file.path(NVIdb::set_dir_NVI("ProgrammeringR", slash = FALSE),
    "standardization", "colnames", "column_standards.csv"),
  property,
  language = "no",
  exclude = FALSE,
  ...
)
```

Arguments

data	[data.frame | character(1)] The data source. If property = "colclasses" the path and file name of the csv-file used as data source should be given.
dbsource	[character(1)] The database that is the source of data. Should be the name of the data source as registered in column_standards table. Defaults to deparse(substitute(data)).
standards	[data.frame | list | character(1)] For giving alternatives to the standard table for column_standards using different formats, see details. Defaults to file.path(NVIdb::set_dir_NVI("ProgrammeringR", slash = FALSE), "standardization", "colnames", "column_standards.csv").
property	[character(1)] Property of the column that should be standardized. Must be one of c("colnames", "colclasses", "collabels", "colwidths_Excel", "colorder"). Defaults to NULL.
language	[character(1)] Language for labels. Must be one of c("no", "en"). Defaults to "no".
exclude	[logical(1)] Used in combination with property = "colorder". If TRUE, all columns with no predefined column order are excluded. Defaults to FALSE.
...	Other arguments to be passed to read.csv2 when property = "colclasses".

Details

The standardisation table is under development. This function only works when being connected to the NVI network.

Variables in internal and external data sources uses different variable names for the same content. standardize_columns standardizes column names for use in scripts. In addition, it standardises column labels and column widths for Excel. Furthermore, input values for the parameter colClasses for read.csv2 and data.table::fread can be generated.

standards gives the source file or the data.frame with the standards for formatting the columns. Default is to give the general source csv file. It can also be a data.frame as for example the OKplan::OK_column_standards giving the standards when generating selection files for the Norwegian surveillance programmes. As this file is embedded into OKplan-package, it may be convenient to update the source file and load it as a csv file. On some occasions, it may be most easy to input the column standards directly using a list.

#' The list input to column_standards must follow a specific format. It is a list with at least three named vectors:

- colname: a vector of all columns in in the source file that should be included in the Excel report with the selection list.
- collabel: A vector with the labels that should be used in the Excel report.
- colwidth: A vector with the column width that should be used in the Excel report.

In addition one may input:

- colorder: the order of the columns to be used in the Excel report. The default is to use the same order as they are entered in the vectors.
- column_db: input added as a possibility to keep on the same format as column_standards. Not necessary to input.

- table_db: input added as a possibility to keep on the same format as column_standards. Must be the same as dbsource. Not necessary to input.

All vectors must have the same order and the same length.

property = "colnames" will replace the column names in a data frame with standardized column names. All standard column names is snake_case. If no standard name is defined for a variable name, the variable name is translated to snake_case and the national characters c("æ", "ø", "å") are translated to c("ae", "oe", "aa").

property = "colclasses" will generate a named vector with the column classes for variables that may not be read correct when importing data from a csv-file. This applies for example to numbers with leading zero that must be imported as character. This vector can be used as a parameter for colClasses.

The default fileEncoding is assumed to be "UTF-8". If another encoding is needed, one must give an additional argument like fileEncoding = "latin1".

property = "collabels" will generate a vector with column labels that can be used to replace the column names in the header of the data table. The column names are not standardised automatically but can be standardised by first using standardize_columns with property = "colname". If no standard column label for the column name is defined, the column name as Sentence case is used as column label. If English names are used and no English column label exists, the Norwegian column label is used instead.

property = "colwidths_Excel" will generate a vector with column widths for Excel. To be used as input parameter to openxlsx::setColWidths. If no standard column width is defined, the Excel standard width of 10.78 is used. Be aware that the generation of column widths are based on the column names. Do not change the column names to labels before the column widths are generated.

property = "colorder" will generate a data frame with the column names in a predefined order. The column names should first have been standardised. No standard order will be given unless the dbsource is defined in the column_standards table. If exclude = FALSE (the standard) the columns with no predefined order will be moved to the last columns in the same order as they appeared in the original data frame. If exclude = TRUE all columns with no predefined order is excluded from the data frame. This option is mainly intended for well defined and worked through routines like making selections lists for the Food Safety Authority. Do not use exclude = TRUE unless you are certain that all columns that should be included are defined in the column_standards table for this dbsource. If uncertain, you may first try with exclude = FALSE and thereafter compare with exclude = TRUE to check if you loose important information.

Value

property = "colnames": A data frame with standard column names.

property = "colclasses": A named vector of column classes to be used as input to functions for reading csv-files, see details.

property = "collabels": A vector with labels for the columns in the data frame.

property = "colwidths_Excel": A vector with column widths for Excel. To be used as input parameter to openxlsx::setColWidths.

property = "colorder": A data frame with column names in predefined order. If exclude = TRUE only columns with a defined order is included.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Generate data frame to be standardized
df <- cbind("\u00C5r" = 2020, "Hensiktkode" = "01001", komnr = "5001")
colnames(df)

# Standardize column names
df <- standardize_columns(data = df, property = "colnames")
colnames(df)

# Generate vector with standard labels
labels <- standardize_columns(data = df, property = "collabels")
# use the labels as column names
colnames(df) <- labels

# Generate vector with standard column widths for Excel
colwidths <- standardize_columns(data = df, property = "colwidths_Excel")
colwidths

## End(Not run)
```

Index