

Package: bcaquiferdata (via r-universe)

October 3, 2026

Title BC Aquifer Data Tools

Version 1.0.0

Description Set of tools for processing BC Aquifer lithology and yield data.

License Apache License (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports bcmaps (>= 1.2.0), dplyr (>= 1.0.7), httr (>= 1.4.2), janitor (>= 2.1.0), lubridate (>= 1.9.2), shinycssloaders (>= 1.1.0), purrr (>= 0.3.4), rappdirs (>= 0.3.3), readr (>= 2.1.1), rlang (>= 0.4.12), shiny (>= 1.7.2), shinyFiles (>= 0.9.3), bslib (>= 0.5.0), sf (>= 0.9.8), stars (>= 0.5.3), stringdist (>= 0.9.8), stringr (>= 1.4.0), tidyr (>= 1.1.4), fs (>= 1.6.1), DT (>= 0.23), ggplot2 (>= 3.3.5), ggthemes (>= 4.2.4), shinyjs (>= 2.1.0), htmltools (>= 0.5.7), bsicons (>= 0.1.2), R.utils (>= 2.13.0), bcdata (>= 0.5.2), units (>= 1.0.1)

Depends R (>= 4.1)

Suggests knitr, gt, testthat (>= 3.0.0), glue, markdown, rmarkdown, openxlsx

LazyData true

Config/testthat/edition 3

URL <https://bcgov.github.io/bcaquiferdata/>,
<https://github.com/bcgov/bcaquiferdata>

VignetteBuilder knitr

Config/pak/sysreqs
libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

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

Date/Publication 2026-05-25 21:04:34 UTC

RemoteUrl <https://github.com/bcgov/bcaquiferdata>
RemoteRef HEAD
RemoteSha 26ae4a2abeef5dc3a537036f90ec9d6339fb59ff

Contents

aq_app	2
cache_clean	3
data_read	3
data_update	4
dem_region	5
drawdown	7
fix_depth_mismatch	8
fix_yield_zero	9
flags	9
lith_fix	10
tiles	10
wells_eg_fixed	11
wells_eg_unfixed	11
wells_elev	12
wells_export	13
wells_subset	15
wells_yield	16
Index	17

aq_app	<i>Launch Aquifer Data Shiny App</i>
--------	--------------------------------------

Description

This app allows you to load a shapefile and filter aquifer/well data according to region, explore data, and export cleaned files.

Usage

aq_app()

Examples

aq_app()

cache_clean	<i>Clean cache</i>
-------------	--------------------

Description

Removes data cache

Usage

```
cache_clean(bcmmaps_cded = FALSE)
```

Arguments

bcmmaps_cded	Logical. Whether or not to also remove CDED files cached with the bcmmaps package. These are used by bcaquifertools for acquiring TRIM data, but may also be cached for use by other workflows.
--------------	---

Examples

```
# cache_clean()
# cache_clean(bcmmaps_cded = TRUE)
```

data_read	<i>Download, Update, and/or load data</i>
-----------	---

Description

This function downloads, updates or loads locally stored data. Currently this function returns wells, wells_sf, or lithology data. Note that these data are originally from GWELLS, but are cleaned and summarized for use in the bcaquiferdata package. For example wells_sf is a spatial version of the data, and lithology is a cleaned and standardized version of lithology. wells also contains the new standardized lithology data, along with the original lithology observations and intermediate classification steps to simplify error tracing.

Usage

```
data_read(type, update = FALSE, permission = FALSE)
```

Arguments

type	Character. Type of data to return, one of wells, wells_sf, or lithology
update	Logical. Force update of the data?
permission	Logical. Permission to create the cache folder. If FALSE, user is asked for permission, if TRUE, permission is implied.

Details

Under normal circumstances, users will not need to use this function as it is used internally by the main workflow functions. However, users may wish to overview entire datasets.

Bear in mind that the lithology cleaning and standardizing, while better than the original data, will almost certainly still have errors!

Value

Data frame or spatial features object of the requested data.

Examples

```
wells <- data_read("wells")
```

data_update	<i>Update cached data</i>
-------------	---------------------------

Description

Update the GWELLS data stored locally.

Usage

```
data_update(  
  type = c("wells", "lithology"),  
  download = TRUE,  
  permission = FALSE  
)
```

Arguments

type	Character. Type of data to update. One of "all", "wells", "lithology"
download	Logical. Whether to re-download and process the data (TRUE), or just re-process it (FALSE).
permission	Logical. Permission to create the cache folder. If FALSE, user is asked for permission, if TRUE, permission is implied.

Examples

```
data_update(type = "wells")  
data_update(type = "lithology")
```

dem_region

*Fetch and trim DEM of a region***Description**

This function takes a shape file of a region and creates a DEM of the region. Lidar data is stored locally as tiles. Tiles are only downloaded if they don't already exist unless `only_new = FALSE`. TRIM data is obtained via the `bcmeps` package and stored locally as tiles. **Note:** TRIM elevation is coarser than Lidar. Use Lidar unless it is missing for your region of interest.

Usage

```
dem_region(
  region,
  source = "lidar",
  out_file = NULL,
  overwrite = FALSE,
  buffer = 1,
  lidar_dir = NULL,
  only_new = TRUE,
  progress = httr::progress()
)
```

Arguments

<code>region</code>	sf simple features object. Shape file of the region of interest.
<code>source</code>	Character. Source of DEM, "lidar", "trim" or a file path (or vector of file paths) to a custom DEM file. See Details.
<code>out_file</code>	Character. File path of where to save tif of DEM clipped to region.
<code>overwrite</code>	Logical. If <code>out_file</code> supplied, whether to overwrite this file if it already exists.
<code>buffer</code>	Numeric. Percent buffer to apply to the region spatial file before cropping the DEM data to match. Increase this value if you find that wells on the edge of your area aren't been matched to elevations when using <code>wells_elev()</code> .
<code>lidar_dir</code>	Character. File path of where Lidar tiles should be stored. Defaults to the cache directory. Only applies when <code>source = "lidar"</code> .
<code>only_new</code>	Logical. Whether to download all Lidar tiles, or only new tiles that don't exist locally. Defaults to TRUE. Only applies when <code>source = "lidar"</code> .
<code>progress</code>	Function. Progress bar to use. Generally leave as is.

Details

Lidar tiles are the newest tile available. If you have reason to need a historical file, contact the `bcaquiferdata` team to discuss your use case.

Value

stars spatiotemporal array object

Data Source

Lidar data is obtained from the LidarBC portal. The tiles data frame contains is an internally created data frame listing tiles and their respective download locations. Tiles to download are selected based on overlap between map tiles and the provided shapefile (region). These Lidar tiles can be browsed and downloaded manually via the [LidarBC Open LiDAR Data Portal](#)

The grid of map tiles is obtained from the BC Data Catalogue, [BCGS 1:20,000 Grid](#)

TRIM data is obtained via the bcmeps package from the BC government [Data Catalogue](#) based on overlap between map tiles and the provided shapefile (region).

If a file path or vector of file paths are provided, a local DEM is loaded with stars and combined with `stars::st_mosaic()`. Note that it is assumed the data is elevation in metres.

Saving to disk or memory

The fastest way to process these data is to use an stars proxy object which keeps manipulations fast. However, to save a copy of the DEM, it is *much* faster to use `sf::gdal_utils()` (rather than `stars::write_stars()` after).

Therefore, if `out_file` is NULL `stars::st_mosaic()` and `sf::st_crop()` will be used to manipulate a stars proxy object. Alternatively, if `out_file` is provided, `sf::gdal_utils()` will be used to combine, crop, and save a copy of the Lidar or Trim DEM for the region provided. These means that slightly different methods are used and it results in a very slightly different cropped region. However, since the region file is buffered before cropping in either method, it shouldn't affect well elevation calculations downstream.

Examples

```
library(sf)
library(stars)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Fetch Lidar DEM
creek_lidar <- dem_region(creek_sf)
plot(creek_lidar)

# Fetch TRIM DEM
creek_trim <- dem_region(creek_sf, source = "trim")
plot(creek_trim)

# Get dem and save to file
creek_lidar2 <- dem_region(creek_sf, out_file = "clinton_lidar.tif")
plot(creek_lidar)

creek_trim <- dem_region(creek_sf, source = "trim", out_file = "clinton_trim.tif")
```

```
plot(creek_trim)

# Use local DEM
koksilah_sf <- st_read("misc/data/Koksilah_watershed4/Koksilah_watershed4.shp")
koksilah_dem <- dem_region(
  koksilah_sf, source = "misc/data/Koksilah_Watershed_DEM_2km_Buffer.tif")
```

drawdown

Create Excel Drawdown template file

Description

This function is still in development. Use at your own risk! Creates an Excel file with formulas for calculating drawdown. File is pre-filled with wells within 100km of location (well or coordinates).

Usage

```
drawdown(
  location,
  rate = NA,
  duration = NA,
  overwrite = FALSE,
  file_name = NULL,
  update = FALSE
)
```

Arguments

location	Numeric. Either a vector with longitude/latitude, or a well tag number.
rate	Numeric. The pumping rate in m3/day (defaults to NA, fillable in Excel file).
duration	Numeric. The duration of pumping in days (defaults to NA, fillable in Excel file).
overwrite	Logical. Overwrite existing file?
file_name	Character. Optional file name for created excel file.
update	Logical. Force update of the data?

Value

Creates excel file

Examples

```
drawdown(85199, rate = 343, duration = 180)
drawdown(85199, rate = 343, duration = 180, overwrite = TRUE)
drawdown(22966, rate = 343, duration = 180)
drawdown(22966)

drawdown(c(-123.5593, 48.647), rate = 3.97, duration = 180)
drawdown(c(-123.5593, 48.647))
```

fix_depth_mismatch	<i>Fix the depth of the well if not equal to final lithology</i>
--------------------	--

Description

The `flag_depth_mismatch` flag identifies wells where the depth is not the same as the depth of the final lithology layer. Fixing these well depths means replacing the well depth with the depth of the final lithology layer.

Usage

```
fix_depth_mismatch(wells_sub)
```

Arguments

`wells_sub` `sf` spatial data frame. Subset of wells data output by `wells_subset()`

Details

This is done automatically for leapfrog exports, but otherwise not.

Value

Fixed `wells_sub` data frame

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)

# Fix well depths
creek_wells_fixed <- fix_depth_mismatch(creek_wells)
```

```
# Explore all fixes
dplyr::select(creek_wells_fixed, dplyr::starts_with("fix"))
```

fix_yield_zero	<i>Fix the yield of a well if equal to zero</i>
----------------	---

Description

The `flag_yield_zero` flag identifies wells where the yield was marked as 0 in GWELLS but probably should have been recorded as NA. Fixing these well yields means replacing the yield of 0 with NA.

Usage

```
fix_yield_zero(wells_sub, fix = TRUE)
```

Arguments

<code>wells_sub</code>	Data frame. The subsetted Wells data frame.
<code>fix</code>	Logical. Whether to apply the fix.

flags	<i>Flags</i>
-------	--------------

Description

A glossary of flag terms

Usage

```
flags
```

Format

`flags:`

A data frame with 10 rows and 2 columns:

Flag flag name

Description Flag description

Solution Suggested solution to the problem

<code>lith_fix</code>	<i>Fix lithology descriptions</i>
-----------------------	-----------------------------------

Description

Clean and categorize lithology descriptions into primary, secondary, tertiary and final lithology categories. Generally this function is used internally when loading and cleaning GWELLS lithology.

Usage

```
lith_fix(desc = NULL)
```

Arguments

desc Character. Text string to convert/fix.

Details

However statements can be tested directly with this function to see how it works and for troubleshooting.

Value

Data frame of lithology categorizations

Examples

```
lith_fix("sandy gravel")

# basic spell checks
lith_fix("saandy gravel")
```

<code>tiles</code>	<i>tiles</i>
--------------------	--------------

Description

A spatial data frame of map tiles with corresponding links to Lidar tiles.

Usage

```
tiles
```

Format

tiles:

A data frame with 7,129 rows and 5 columns:

map_tile Tile name

geometry Spatial data

utm Projection

tile_name Lidar tile name

url Link to Lidar tile

Details

The spatial grid of map tiles is obtained from the BC Data Catalogue, [BCGS 1:20,000 Grid](#)

Links to Lidar tile urls are extracted from the list at the [LidarBC Open LiDAR Data Portal](#)

wells_eg_fixed	<i>wells_eg_fixed</i>
----------------	-----------------------

Description

Example data of fixed wells subsetted and with elevation added

Usage

```
wells_eg_fixed
```

Format

wells_eg_fixed:

A data frame with 31 rows and 71 columns.

wells_eg_unfixed	<i>wells_eg_unfixed</i>
------------------	-------------------------

Description

Example data of unfixed wells subsetted and with elevation added

Usage

```
wells_eg_unfixed
```

Format

wells_eg_unfixed:

A data frame with 31 rows and 71 columns.

wells_elev	<i>Subset wells and add elevation</i>
------------	---------------------------------------

Description

This function takes a region shape file and the DEM of a region (output of `dem_region()`), subsets the wells data (from GWELLS) to this region and adds the elevation data.

Usage

```
wells_elev(wells_sub, dem, dem_extra = NULL, update = FALSE)
```

Arguments

<code>wells_sub</code>	sf spatial data frame. Subset of wells data output by <code>wells_subset()</code>
<code>dem</code>	stars simple features object. Output of <code>dem_region()</code> . Primary source of elevation data.
<code>dem_extra</code>	stars simple features object. Output of <code>dem_region()</code> . Optional secondary source of elevation data. Useful in situations where the primary source is incomplete. Use with caution: Combining elevations measured though different techniques may introduce artifacts (See Details).
<code>update</code>	Logical. Force update of the data?

Details

Because combining elevation data measured from different sources can introduce artifacts, if two sources of elevation are provided (i.e. both `dem` and `dem_extra`), the data will contain extra columns for assessment. Specifically, there will be three elevation columns, rather than just one. `elev1` contains elevations from the primary source (`dem`), `elev2` contains elevations from the secondary source (`dem_extra`), `elev` contains the combined elevation data, `elev1` unless missing, then `elev2`.

Value

sf spatial data frame

Examples

```
library(sf)
library(ggplot2)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)

# Fetch Lidar DEM
```

```

creek_lidar <- dem_region(creek_sf)

# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_lidar)

ggplot() +
  geom_sf(data = creek_sf) +
  geom_sf(data = creek_wells, aes(colour = elev), size = 0.5,
    fill = "NA", show.legend = FALSE) +
  coord_sf(datum = st_crs(3005)) # BC Albers

# OR Fetch TRIM DEM
creek_trim <- dem_region(creek_sf, type = "trim")

# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_trim)

ggplot() +
  geom_sf(data = creek_sf) +
  geom_sf(data = creek_wells, aes(colour = elev), size = 0.5,
    fill = "NA", show.legend = FALSE) +
  coord_sf(datum = st_crs(3005)) # BC Albers

# Dealing with missing data
mill_sf <- st_read("misc/data/MillBayWatershed.shp")
mill_wells <- wells_subset(mill_sf)

mill_lidar <- dem_region(mill_sf)
mill_trim <- dem_region(mill_sf, type = "trim")

mill_wells <- wells_elev(mill_wells, dem = mill_lidar, dem_extra = mill_trim)

# See how the elevation data is combined, `dem` (elev1) is the primary source.
select(mill_wells, well_tag_number, elev1, elev2, elev)

# Use local DEM
koksilah_sf <- st_read("misc/data/Koksilah_watershed4/Koksilah_watershed4.shp")
koksilah_wells <- wells_subset(koksilah_sf)
koksilah_dem <- dem_region(
  koksilah_sf, source = "misc/data/Koksilah_Watershed_DEM_2km_Buffer.tif")
koksilah_wells <- wells_elev(koksilah_wells, dem = koksilah_dem)

# Plot
p <- koksilah_wells |>
  st_transform(crs = st_crs(koksilah_dem))
plot(koksilah_dem, reset = FALSE, key.pos = NULL)
plot(p[,"elev"], add = TRUE, pch = 20)

```

Description

Export wells data for use in Strater and Voxler

Usage

```
wells_export(wells_sub, id, type, dir = ".", zip = FALSE, preview = FALSE)
```

Arguments

wells_sub	Data frame. Output of wells_elev()
id	Character. Id to prepend to all output files e.g., "id_lith.csv"
type	Character. Format in which to export. One of "strater", "voxler", "archydro", "leapfrog", or "surfer" (case-insensitive).
dir	Character. Directory where files should be exported to. Defaults to working directory.
zip	Logical. Whether to export a zip archive of the files to dir.
preview	Logical. Whether to preview the exports (TRUE, return a list of data frames) or to actually export the data (FALSE, write the necessary files to the dir folder, default).

Value

If preview = FALSE, a vector of file names (or if zip = TRUE, a single filename of the zipped archive); if preview = TRUE, a list of data frames.

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek)

# Fetch Lidar DEM
creek_lidar <- dem_region(creek)

# Collect wells in this region with added elevation from Lidar
creek_wells <- wells_elev(creek_wells, creek_lidar)

# Preview data for Strater
p <- wells_export(creek_wells, id = "clinton", type = "strater", preview = TRUE)
names(p)
p[["strater_lith"]]
p[["strater_collars"]]
p[["strater_wells"]]
```

```

# Export data for Strater
wells_export(creek_wells, id = "clinton", type = "strater", zip = TRUE)

# Export data for Voxler
wells_export(creek_wells, id = "clinton", type = "voxler", zip = TRUE)

# Export Arc Hydro
wells_export(creek_wells, id = "clinton", type = "archydro", zip = TRUE)

# Export Surver
wells_export(creek_wells, id = "clinton", type = "surfer", zip = TRUE)

wells_export(creek_wells, id = "clinton", type = "leapfrog", zip = TRUE)

```

wells_subset	<i>Subset wells to region</i>
--------------	-------------------------------

Description

Filter the GWELLS data returning only wells within the provided shapefile.

Usage

```

wells_subset(
  region,
  fix_bottom_intervals = TRUE,
  fix_depth_missing = TRUE,
  fix_yield_zero = TRUE,
  update = FALSE
)

```

Arguments

region	sf simple features object. Shape file of the region of interest.
fix_bottom_intervals	Logical. Whether to add 1m to bottom lithology intervals that has no thickness (identified by flat_int_bottom). Default TRUE.
fix_depth_missing	Logical. Whether to fix missing well depths by making them equal to the depth of the final lithology layer. Default TRUE.
fix_yield_zero	Logical. Whether to fix well yields of 0 by making them NA. Default TRUE.
update	Logical. Force update of the data?

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)
```

wells_yield	<i>Add yield lithology data to wells subset</i>
-------------	---

Description

Yield records are extracted from lithology observations and added to the wells data.

Usage

```
wells_yield(wells_sub)
```

Arguments

`wells_sub` sf spatial data frame. Subset of wells data output by `wells_subset()`

Value

Data frame or sf spatial data frame with wells data and added yield from lithology.

Examples

```
library(sf)

# Load a shape file defining the region of interest
creek_sf <- st_read("misc/data/Clinton_Creek.shp")

# Get wells within this region
creek_wells <- wells_subset(creek_sf)

# Get yield data for these wells
creek_yield <- wells_yield(creek_wells)
```

Index

- * **datasets**
 - flags, [9](#)
 - tiles, [10](#)
 - wells_eg_fixed, [11](#)
 - wells_eg_unfixed, [11](#)
- aq_app, [2](#)
- cache_clean, [3](#)
- data_read, [3](#)
- data_update, [4](#)
- dem_region, [5](#)
- drawdown, [7](#)
- fix_depth_mismatch, [8](#)
- fix_yield_zero, [9](#)
- flags, [9](#)
- lith_fix, [10](#)
- sf::gdal_utils(), [6](#)
- sf::st_crop(), [6](#)
- stars::st_mosaic(), [6](#)
- stars::write_stars(), [6](#)
- tiles, [10](#)
- wells_eg_fixed, [11](#)
- wells_eg_unfixed, [11](#)
- wells_elev, [12](#)
- wells_export, [13](#)
- wells_subset, [15](#)
- wells_yield, [16](#)