Package: rlas (via r-universe)

June 2, 2026

Type Package

Title Read and Write 'las' and 'laz' Binary File Formats Used for Remote Sensing Data

Version 1.9.3

Description Read and write 'las' and 'laz' binary file formats. The LAS file format is a public file format for the interchange of 3-dimensional point cloud data between data users. The LAS specifications are approved by the American Society for Photogrammetry and Remote Sensing <https://community.asprs.org/leadership-restricted/leadership-content/public-documents/standards>. The LAZ file format is an open and lossless compression scheme for binary LAS format versions 1.0 to 1.4 <https://laszip.org/>.

URL https://github.com/r-lidar/rlas

BugReports https://github.com/r-lidar/rlas/issues

License GPL-3

Depends R (>= 3.6.0)

Imports Rcpp, data.table, utils

RoxygenNote 7.3.3

LinkingTo Rcpp

SystemRequirements C++17, GDAL (>= 2.2.3), GEOS (>= 3.4.0), PROJ (>= 4.9.3), sqlite3, GNU make

Suggests tinytest

Encoding UTF-8

NeedsCompilation yes

Author Jean-Romain Roussel [aut, cre, cph], Florian De Boissieu [aut, ctb] (Enable the support of .lax file and extra byte attributes), Martin Isenburg [cph] (Is the author of the LASlib and LASzip libraries), David Auty [ctb] (Reviewed the documentation), Pierrick Marie [ctb] (Helped to compile LASlib

code in R), Tiago de Conto [ctb] (Implemented the -thin_with_voxel filter method)

Maintainer Jean-Romain Roussel <info@r-lidar.com>

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev make libproj-dev libsqlite3-dev

Repository https://cran.r-universe.dev

Date/Publication 2026-06-02 07:30:08 UTC

RemoteUrl https://github.com/cran/rlas

RemoteRef HEAD

RemoteSha 6bd768f292e3899524fc350d112252cfb523f68a

Contents

`check_las_validity` *Check if data and headers respect the LAS specification*

Description

las files are normalized files. These functions perform tests of compliance with LAS specification.

- `check_las_validity` tests if the data and the header contain information that cannot be written into a las file. For example it tests is the intensities do not exeed 65535. It throws an error for each deviance to the specification. It is useful for testing if modified R objects are still valid.
- `check_las_compliance` test if the data and the header contain information that can be written into a las file but are invalid with repect of the specification. For example it test if the RGB colors are recoded on 16 bits. It is possible to store colors recorded on 8 bits (0 to 255) but it is not correct to do that. It throws a warning for each deviance to the specification. It is useful for testing if the data read from a file are correct.

Usage

```
check_las_validity(header, data)

check_las_compliance(header, data)
```

Arguments

header	a list containing the header of a las file
data	a data.frame or a data.table containing a point cloud

---

compression *ALTREP utilities*

Description

Test if an a vector is compressed using the ALTREP framework

Usage

```
is_compressed(x)

true_size(x)
```

Arguments

x	an R object

Examples

```
lazfile <- system.file("extdata", "example.las", package = "rlas")
las <- read.las(lazfile)
is_compressed(las)

# The difference is more substantial on bigger point clouds (~30%)
object.size(las)
true_size(las)
```

crs_tools *Coordinate Reference System Tools*

Description

Functions that update a header to describe coordinates reference system according to the LAS specifications

Usage

```
header_get_epsg(header)

header_set_epsg(header, epsg)

header_get_wktcs(header)

header_set_wktcs(header, WKT)

find_epsg_position(header)
```

Arguments

header	list
epsg	integer. An EPSG code
WKT	string. A string of an WKT OGC CS

See Also

Other header_tools: extra_bytes_attribute_tools, fwf_interpreter(), public_header_block_tools

extra_bytes_attribute_tools *Extra Bytes Attributes Tools*

Description

Functions that update a header to describe Extra Bytes Attributes according to the LAS specifications

Usage

```
header_add_extrabytes(header, data, name, desc)

header_add_extrabytes_manual(
  header,
  name,
  desc,
  type,
  offset = NULL,
  scale = NULL,
  max = NULL,
  min = NULL,
  NA_value = NULL
)
```

Arguments

header	list
data	vector. Data that must be added in the extrabytes attributes.
name	character. The name of the extrabytes attributes to add in the file.
desc	character. The description of the extrabytes attributes to add in the file.
type	integer. The data type of the extrabytes attributes (page 25 of the spec).
scale, offset	numeric. The scale and offset of the data. NULL if not relevant.
min, max	numeric or integer. The minimum and maximum value of the data. NULL if not relevant.
NA_value	numeric or integer. NA is not a valid value. At writing time it will be replaced by this value that will be considered as NA. NULL if not relevant.

See Also

Other header_tools: crs_tools, fwf_interpreter(), public_header_block_tools

Examples

```
data = data.frame(X = c(339002.889, 339002.983, 339002.918),
                  Y = c(5248000.515, 5248000.478, 5248000.318),
                  Z = c(975.589, 974.778, 974.471),
                  gpstime = c(269347.281418006, 269347.281428006, 269347.281438006),
                  Intensity = c(82L, 54L, 27L),
                  ReturnNumber = c(1L, 1L, 2L),
                  NumberOfReturns = c(1L, 1L, 2L),
                  ScanDirectionFlag = c(1L, 1L, 1L),
                  EdgeOfFlightline = c(1L, 0L, 0L),
                  Classification = c(1L, 1L, 1L),
                  ScanAngleRank = c(-21L, -21L, -21L),
                  UserData = c(32L, 32L, 32L),
                  PointSourceID = c(17L, 17L, 17L),
                  treeID = c(1L, 1L, 1L))
```

```
lasheader = header_create(data)
lasheader[["Variable Length Records"]]

lasheader = header_add_extrabytes(lasheader, data$treeID, "treeID", "An id for each tree")
lasheader[["Variable Length Records"]]
```

`fwf_interpreter` *Full Waveform Interpreter*

Description

This is an experimental function that may change
Raw full waveform data read from LAS files might be cryptic even with a good understanding of the LAS specifications. This function interpret the full waveform data as a set of XYZ coordinates and an amplitude which is the digitized voltage.

Usage

```
fwf_interpreter(header, data)
```

Arguments

`header`	list. A header
`data`	data.frame or data.table

Value

A list containing a `data.frame` per pulse with the XYZ coordinates of the waveform and the voltage of the record (`Amplitude`)

See Also

Other header_tools: `crs_tools`, `extra_bytes_attribute_tools`, `public_header_block_tools`

Examples

```
## Not run:
f <- system.file("extdata", "fwf.laz", package="rlas")
head <- read.lasheader(f)
data <- read.las(f)
fwf <- fwf_interpreter(head, data)

## End(Not run)
```

help_filter *LASlib filter and transform operation*

Description

Print the existing LASlib filter and transform commands

Usage

```
help_filter()

help_transform()
```

las_specification_tools *A set of function to test conformance with LAS specifications*

Description

Tools reserved to developpers and not intended to be used by regular users. The functions return TRUE or FALSE by default without more information. If behavior is 'warning' functions throw a warning for each fail and return FALSE if any warning TRUE otherwise. If behavior is 'stop' functions throw an error for the first fail and return TRUE if 0 error. If behavior is 'vector' returns a character vector with the decription of each error an never fail. Is it useful to make a detailed inspection.

Usage

```
is_defined_offsets(header, behavior = "bool")

is_valid_offsets(header, behavior = "bool")

is_defined_scalefactors(header, behavior = "bool")

is_valid_scalefactors(header, behavior = "bool")

is_defined_filesourceid(header, behavior = "bool")

is_valid_filesourceid(header, behavior)

is_defined_globalencoding(header, behavior = "bool")

is_valid_globalencoding(header, behavior = "bool")

is_defined_version(header, behavior = "bool")
```

```
is_valid_version(header, behavior = "bool")

is_defined_date(header, behavior = "bool")

is_valid_date(header, behavior = "bool")

is_defined_pointformat(header, behavior = "bool")

is_valid_pointformat(header, behavior = "bool")

is_defined_extrabytes(header, behavior = "bool")

is_valid_extrabytes(header, behavior = "bool")

is_empty_point_cloud(header, behavior = "bool")

is_defined_coordinates(data, behavior = "bool")

is_valid_XYZ(data, behavior = "bool")

is_valid_Intensity(data, behavior = "bool")

is_valid_ReturnNumber(data, header, behavior = "bool")

is_valid_NumberOfReturns(data, header, behavior = "bool")

is_valid_ScanDirectionFlag(data, behavior = "bool")

is_valid_EdgeOfFlightline(data, behavior = "bool")

is_valid_Classification(data, header, behavior = "bool")

is_valid_ScannerChannel(data, behavior = "bool")

is_valid_SyntheticFlag(data, behavior = "bool")

is_valid_KeypointFlag(data, behavior = "bool")

is_valid_WithheldFlag(data, behavior = "bool")

is_valid_OverlapFlag(data, behavior = "bool")

is_valid_ScanAngle(data, behavior = "bool")

is_valid_ScanAngleRank(data, behavior = "bool")

is_valid_UserData(data, behavior = "bool")
```

```
is_valid_gpstime(data, behavior = "bool")

is_valid_PointSourceID(data, behavior = "bool")

is_valid_RGB(data, behavior = "bool")

is_valid_NIR(data, behavior = "bool")

is_compliant_ReturnNumber(data, behavior = "bool")

is_compliant_NumberOfReturns(data, behavior = "bool")

is_compliant_ReturnNumber_vs_NumberOfReturns(data, behavior = "bool")

is_compliant_RGB(data, behavior = "bool")

is_compliant_ScanAngle(data, behavior = "bool")

is_compliant_ScanAngleRank(data, behavior = "bool")

is_NIR_in_valid_format(header, data, behavior = "bool")

is_gpstime_in_valid_format(header, data, behavior = "bool")

is_RGB_in_valid_format(header, data, behavior = "bool")

is_ScanAngle_in_valid_format(header, data, behavior = "bool")

is_ScannerChannel_in_valid_format(header, data, behavior = "bool")

is_XY_larger_than_bbox(header, data, behavior = "bool")

is_XY_smaller_than_bbox(header, data, behavior = "bool")

is_Z_in_bbox(header, data, behavior = "bool")

is_number_of_points_in_accordance_with_header(header, data, behavior = "bool")

is_number_of_points_by_return_in_accordance_with_header(
  header,
  data,
  behavior = "bool"
)

is_extrabytes_in_accordance_with_data(header, data, behavior = "bool")
```

Arguments

header	a list containing the header of a las file
behavior	character. Defines the behavior of the function. 'bool' returns TRUE or FALSE. 'warning' throw a warning for each fails and return FALSE if any warning TRUE otherwise. 'vector' returns a character vector of each warning but does not thrown any warning.
data	a data.frame or a data.table containing a point cloud

---

public_header_block_tools

Public Header Block Tools

Description

Create or update a header for a las file from a dataset. A las file consists of two parts. A header that describes the data and the data itself. These functions make valid headers (public header block only) that can be used in write.las.

Usage

```
header_create(data)

header_update(header, data)
```

Arguments

data	data.frame or data.table
header	list. A header

Details

`header_create` makes a full header from data. `header_update` modifies the information that needs to be updated. But most of the original information is not modified, for example point data format is kept 'as is'.

Value

A list containing the metadata required to write a las file.

See Also

Other header_tools: `crs_tools`, `extra_bytes_attribute_tools`, `fwf_interpreter()`

Examples

```
lasdata = data.frame(X = c(339002.889, 339002.983, 339002.918),
                     Y = c(5248000.515, 5248000.478, 5248000.318),
                     Z = c(975.589, 974.778, 974.471),
                     gpstime = c(269347.281418006, 269347.281428006, 269347.281438006),
                     Intensity = c(82L, 54L, 27L),
                     ReturnNumber = c(1L, 1L, 2L),
                     NumberOfReturns = c(1L, 1L, 2L),
                     ScanDirectionFlag = c(1L, 1L, 1L),
                     EdgeOfFlightline = c(1L, 0L, 0L),
                     Classification = c(1L, 1L, 1L),
                     ScanAngleRank = c(-21L, -21L, -21L),
                     UserData = c(32L, 32L, 32L),
                     PointSourceID = c(17L, 17L, 17L),
                     treeID = c(1L, 1L, 1L))

lasheader = header_create(lasdata)
```

---

read.las *Read data from a .las or .laz file*

Description

Reads data from .las or .laz files according to LAS specifications and returns a `data.table` labelled according to LAS specifications. See the ASPRS documentation for the LAS file format. The optional parameters enable the user to save memory by choosing to load only the attributes they need. Moreover, the function provides a streaming filter to load only the points of interest into the memory and hence avoids allocating any superfluous memory.

Usage

```
read.las(files, select = "*", filter = "", transform = "")

read_and_write.las(
  ifiles,
  ofile = "",
  select = "*",
  filter = "",
  polygons = list()
)
```

Arguments

`files`	array of characters
`select`	character. select only columns of interest to save memory (see details)
`filter`	character. streaming filters - filter data while reading the file (see details)

transform	character. streaming transformation - transform data while reading the file (see details)
ifiles, ofile	characters. Streaming operations.
polygons	list. Internal use only.

Details

Select: the 'select' argument specifies the data that will actually be loaded. For example, 'xyzia' means that the x, y, and z coordinates, the intensity and the scan angle will be loaded. The supported entries are t - gpstime, a - scan angle, i - intensity, n - number of returns, r - return number, c - classification, s - synthetic flag, k - keypoint flag, w - withheld flag, o - overlap flag (format 6+), u - user data, p - point source ID, e - edge of flight line flag, d - direction of scan flag, R - red channel of RGB color, G - green channel of RGB color, B - blue channel of RGB color, N - near-infrared channel, C - scanner channel (format 6+), W - Full waveform. Also numbers from 1 to 9 for the extra bytes data numbers 1 to 9. 0 enables all extra bytes to be loaded and '*' is the wildcard that enables everything to be loaded from the LAS file.
Note that x, y, z are implicit and always loaded. 'xyzia' is equivalent to 'ia'.

Filter: the 'filter' argument allows filtering of the point cloud while reading files. `rlas` relies on the well-known `LASlib` library written by Martin Isenburg to read the binary files. Thus the package inherits the filter commands available in LAStools. To use these filters the user can pass the common commands from `LAStools` into the parameter `'filter'`. Type `read.las(filter = "-help")` to display the `LASlib` documentation and the available filters.

Transform: the 'transform' argument allows transformation of the point cloud while reading files. `rlas` relies on the well-known `LASlib` library written by Martin Isenburg to read the binary files. Thus the package inherits the transform commands available in LAStools. To use these transformations the user can pass the common commands from `LAStools` into the parameter `'transform'`. Type `read.las(transform = "-help")` to display the `LASlib` documentation and the available transformations.

Value

A `data.table`

Full Waveform

The support of full waveform is still in development. The version 1.4.1 introduced the support of point formats 4, 5, 9 and 10. The current state consists in reading the raw data. We also introduced the function fwf_interpreter to interpret the raw data into something easier to manipulate (but that uses more memory). The current behaviour is not set in stone and is prone to design modification until version 1.5.0 where we aims to get enough insight to lock our engineering choices to something that suit best the needs.

Examples

```
lasfile <- system.file("extdata", "example.las", package="rlas")

lasdata <- read.las(lasfile)
```

```
lasdata <- read.las(lasfile, filter = "-keep_first")
lasdata <- read.las(lasfile, filter = "-drop_intensity_below 80")
lasdata <- read.las(lasfile, select = "xyzia")
```

`read.lasheader` *Read header from a .las or .laz file*

Description

Reads header from .las or .laz files according to LAS specifications and returns a `list` labeled according to LAS specifications. See the ASPRS documentation for the LAS file format.

Usage

```
read.lasheader(file)
```

Arguments

`file`	filepath character string to the .las or .laz file

Value

A `list`

See Also

Other rlas: `write.las()`

Examples

```
lazfile   <- system.file("extdata", "example.las", package="rlas")
lasheader <- read.lasheader(lazfile)
```

`write.las` *Write a .las or .laz file*

Description

Write a .las or .laz file. The user provides a table with the data in columns. Column names must respect the specified allowed names (see details). A correct and complete header must also be provided. This header can optionally be generated with header_create.

Usage

```
write.las(file, header, data)
```

Arguments

`file`	character. file path to .las or .laz file
`header`	list. Can be partially recycled from another file (see read.lasheader) and updated with header_update or generated with header_create.
`data`	data.frame or data.table that contains the data to write in the file. Column names must respect the imposed nomenclature (see details)

Details

Allowed names are "X", "Y", "Z", "gpstime", "Intensity", "ReturnNumber", "NumberOfReturns", "ScanDirectionFlag", "EdgeOfFlightline", "Classification", "ScanAngle", "UserData", "PointSourceID", "R", "G", "B", "NIR". All other extra columns will be written in extra bytes attributes only if the header specifically states these data should be saved into extra bytes attributes. To use the full potential of the function `write.las` it is recommended users read the complete specifications of the LAS file format. Otherwise users can rely on automated procedures that are expected to be sufficient for most usages.

Value

void

See Also

Other rlas: `read.lasheader()`

Examples

```
lasdata = data.frame(X = c(339002.889, 339002.983, 339002.918),
                     Y = c(5248000.515, 5248000.478, 5248000.318),
                     Z = c(975.589, 974.778, 974.471),
                     gpstime = c(269347.281418006, 269347.281428006, 269347.281438006),
                     Intensity = c(82L, 54L, 27L),
                     ReturnNumber = c(1L, 1L, 2L),
                     NumberOfReturns = c(1L, 1L, 2L),
                     ScanDirectionFlag = c(1L, 1L, 1L),
                     EdgeOfFlightline = c(1L, 0L, 0L),
                     Classification = c(1L, 1L, 1L),
                     ScanAngleRank = c(-21L, -21L, -21L),
                     UserData = c(32L, 32L, 32L),
                     PointSourceID = c(17L, 17L, 17L))

lasheader = header_create(lasdata)
file = file.path(tempdir(), "temp.las")

write.las(file, lasheader, lasdata)
```

writelax *Write a .lax file from a .las or .laz file*

Description

Write a lax file from a las or laz file. A lax file is a tiny file which can come with a las or laz and which spatially index the data to make faster spatial queries. It has been invented by Martin Isenburg in LASlib. rlas support lax file and enable to write a lax file with default settings. For more options, use lasindex from binaries provided by LASlib (for more informations see references)

Usage

```
writelax(file, verbose = FALSE)
```

Arguments

file	character. filename of .las or .laz file
verbose	boolean. Verbose switch.

References

https://rapidlasso.com/
https://rapidlasso.com/2012/12/03/lasindex-spatial-indexing-of-lidar-data/
https://github.com/LAStools/LAStools

Index