Package: gtfstools (via r-universe)

October 28, 2026

Type Package

Title General Transit Feed Specification (GTFS) Editing and Analysing Tools

Version 2.0.0

Description Utility functions to read, manipulate, analyse and write transit feeds in the General Transit Feed Specification (GTFS) data format.

License MIT + file LICENSE

URL https://ipea.github.io/gtfstools/, https://github.com/ipea/gtfstools

BugReports https://github.com/ipea/gtfstools/issues

Depends R (>= 4.1.0)

Imports checkmate, cli, curl, data.table, gtfsio (>= 1.2.2), httr2 (>= 1.0.4), parallelly, processx, rlang, sf, sfheaders, stats, utils, zip

Suggests covr, dplyr, ggplot2, ggraph, igraph, jsonlite, knitr, rmarkdown, sfnetworks, testthat (>= 3.0.0), tidygraph

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

LinkingTo cpp11

Config/roxygen2/version 8.1.0

Author Daniel Herszenhut [aut] (ORCID: <https://orcid.org/0000-0001-8066-1105>), Rafael H. M. Pereira [aut, cre] (ORCID: <https://orcid.org/0000-0003-2125-7465>), Pedro R. Andrade [aut] (ORCID: <https://orcid.org/0000-0001-8675-4046>), Joao Bazzo [aut] (ORCID: <https://orcid.org/0000-0003-4536-5006>), Mark Padgham [ctb], Christopher Higgins [ctb], Gonçalo F. Matos [ctb], Yigit

Han Eroglu [ctb], Arthur Bazolli [ctb], Marcus Saraiva [ctb]
(ORCID: <https://orcid.org/0000-0001-6218-2338>), Ipea -
Institute for Applied Economic Research [cph, fnd]

Maintainer Rafael H. M. Pereira <rafa.pereira.br@gmail.com>

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

Date/Publication 2026-10-28 10:30:16 UTC

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

RemoteRef HEAD

RemoteSha b564b97daccab0196c246e4080839e4db90baa0b

Contents

as_dt_gtfs . . . 3
build_graph . . . 5
combine_nearby_stops . . . 8
convert_seconds_to_time . . . 9
convert_sf_to_shapes . . . 10
convert_shapes_to_sf . . . 11
convert_stops_to_sf . . . 12
convert_stops_to_shapes . . . 13
convert_time_to_seconds . . . 14
crop_gtfs . . . 15
download_validator . . . 17
filter_by_agency_id . . . 18
filter_by_date . . . 19
filter_by_period . . . 20
filter_by_route_id . . . 22
filter_by_route_type . . . 23
filter_by_service_id . . . 26
filter_by_sf . . . 27
filter_by_shape_id . . . 28
filter_by_spatial_extent . . . 29
filter_by_stop_id . . . 30
filter_by_time_of_day . . . 32
filter_by_trip_id . . . 34
filter_by_weekday . . . 35
fix_gtfs . . . 37
frequencies_to_stop_times . . . 39
get_active_services . . . 41
get_calendar_overlap . . . 42
get_children_stops . . . 44
get_dates . . . 45
get_dwell_time . . . 46
get_parent_station . . . 47
get_route_frequency . . . 48
get_route_geometry . . . 50

get_route_speed . . . 51
get_route_timetable . . . 53
get_shape_length . . . 54
get_shapes_geometry . . . 55
get_start_and_end_times . . . 56
get_stop_distances . . . 57
get_stop_frequency . . . 58
get_stop_times_patterns . . . 60
get_stop_timetable . . . 61
get_stops . . . 63
get_stops_geometry . . . 64
get_summary . . . 65
get_trip_duration . . . 68
get_trip_geometry . . . 70
get_trip_length . . . 71
get_trip_segment_duration . . . 74
get_trip_speed . . . 75
get_vkt . . . 78
get_vmt . . . 79
interpolate_shape_dist_traveled . . . 80
interpolate_stop_times . . . 81
list_validator_versions . . . 83
merge_gtfs . . . 84
read_gtfs . . . 85
remove_duplicates . . . 87
remove_unused_ids . . . 87
set_dwell_time . . . 88
set_max_speed . . . 91
set_min_dwell_time . . . 93
set_route_frequency . . . 95
set_route_speed . . . 97
set_trip_speed . . . 100
stop_times_to_frequencies . . . 103
validate_gtfs . . . 104
write_gtfs . . . 106

Index **108**

`as_dt_gtfs` *Coerce lists and GTFS objects from other packages into gtfstools-compatible GTFS objects*

Description

Coerces an existing object, such as a `list` or a GTFS object created from other packages (`{tidytransit}` and `{gtfsio}`, for example) into a gtfstools-compatible GTFS object - i.e. one whose internal tables are represented with `data.table`s and whose fields are formatted like the fields of a feed read with `read_gtfs()`.

`as_dt_gtfs()` is an S3 generic, with methods for:

- `tidygtfs`: the class of GTFS objects read with `tidytransit::read_gtfs()`. This method converts all `tibble`s to `data.table`s and convert time columns, represented as `hms` objects in a `tidygtfs`, to strings in the `"HH:MM:SS"` format.
- `gtfs`: the class of GTFS objects read with `gtfsio::import_gtfs()`. This method convert all date fields, represented as `integer`s in `{gtfsio}`'s representation, to `Date` objects.
- `list`: this method tries to convert the elements of a list into `data.table`s. Please note that all list elements must inherit from `data.frame` and must be named. Like the `gtfs` method, it converts date fields represented as `integer`s (`YYYYMMDD`) to `Date` objects; other fields are kept as they are.

Usage

```
as_dt_gtfs(gtfs, ...)

## S3 method for class 'tidygtfs'
as_dt_gtfs(gtfs, calculate_distance = TRUE, ...)

## S3 method for class 'gtfs'
as_dt_gtfs(gtfs, ...)

## S3 method for class 'list'
as_dt_gtfs(gtfs, ...)
```

Arguments

`gtfs`	The object that should be coerced to a `dt_gtfs`.
`...`	Ignored.
`calculate_distance`	A logical. Passed to `convert_sf_to_shapes()`, which only affects the output when the object to be converted includes a `shapes` element. Controls whether this function, used to convert a `LINESTRING sf` into a GTFS `shapes` table, should calculate and populate the `shape_dist_traveled` column. This column is used to describe the distance along the shape from each one of its points to its first point. Defaults to `TRUE`.

Value

A `dt_gtfs` GTFS object.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfsio_gtfs <- gtfsio::import_gtfs(data_path)
class(gtfsio_gtfs)

gtfstools_gtfs <- as_dt_gtfs(gtfsio_gtfs)
class(gtfstools_gtfs)

gtfs_like_list <- unclass(gtfsio_gtfs)
class(gtfs_like_list)

gtfstools_gtfs <- as_dt_gtfs(gtfs_like_list)
class(gtfstools_gtfs)
```

---

build_graph *Build a transit network graph*

Description

Builds a frequency-based transit network from the trips that depart within a time of day, and returns it as a set of tables in the General Modeling Network Specification (GMNS) format or as an {igraph} or {tidygraph} graph. The network can be used in transit assignment models and in graph analyses, such as finding the shortest travel times between stops.

Usage

```
build_graph(
  gtfs,
  from,
  to,
  format = "gmns",
  max_walk_dist = 300,
  walk_speed = 5
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
from	A string. The starting point of the time of day, in the "HH:MM:SS" format.
to	A string. The ending point of the time of day, in the "HH:MM:SS" format. Must be later than from.
format	A string. The format of the graph: "gmns" (the default), "igraph" or "tidygraph". The last two require the packages of the same name, which the function offers to install in interactive sessions.

max_walk_dist A number. The maximum great-circle distance, in meters, between two stops connected by walking transfer links. Defaults to 300.

walk_speed A number. The walking speed, in km/h, used to calculate the travel time of transfer links. Defaults to 5.

Value

With format = "gmns", a list with the data.tables node, link and config, named after the GMNS files. With format = "igraph", a directed igraph graph, and with format = "tidygraph", a directed tbl_graph. Their vertices and edges have the columns of the node and link tables as attributes, and the name of each vertex is its node_id, as a string.

Details

Nodes (node_type) are "stop" nodes, one per stop visited by the trips, at the stop's coordinates, and "service" nodes, one per route and direction at each of its stops, at the coordinates of the stop (their parent_node_id). Links (link_type) are all directed:

- "in_vehicle" links connect the service nodes of consecutive stops of the trips. travel_time is the mean time from the departure at a stop to the arrival at the next, over the trips with both times (decreasing times are not used), weighted by their departures. departures is the number of departures of the trips of the network that run through the link. length is the great-circle distance between the stops, as in get_stop_distances(), and free_speed the length divided by the travel time (NA when the travel time is 0 or either is missing).
- "boarding" links connect each stop to the service nodes from which trips depart. departures is the number of departures from the stop of the trips of the network, so it may include departures after to from stops far along their routes. travel_time is half the mean headway (the length of the time of day divided by departures).
- "alighting" links connect each service node back to its stop, with a travel_time of 0.
- "transfer" links connect, in both directions, the stops at most max_walk_dist apart. travel_time is the length divided by walk_speed, which is also their free_speed.

Lengths are in meters, travel times in minutes and speeds in km/h. Boarding and alighting links have a length of 0 and an NA free_speed; alighting and transfer links have NA departures. route_id, direction_id and inferred_direction are those of the link's service node, and NA in transfer links.

The network includes the trips that depart from their first stop at or after from and before to, as in get_route_frequency(), so trips already running at from are not included. Trips listed in frequencies are included once per departure generated within the time of day, with their stop_times entries as templates. Trips without stop_times entries are not included. The trips of every service_id are pooled, so the feed should usually be filtered to a single day first, with filter_by_date() or filter_by_weekday(). Departures after midnight have times past "24:00:00", so use a time of day such as "24:00:00"-"26:00:00" for them. Trips whose times wrongly go back to "00:00:00" after midnight are placed by their earliest time.

Within a trip, consecutive stop_times entries at the same stop are one visit, arriving at their first known time and departing at their last. A trip that visits a stop more than once uses a different service node at each visit (visit 1, 2, ...).

Directions come from the direction_id field of trips. Trips without it (or with a blank direction_id) get an inferred_direction instead: trips of a route that share a pair of consecutive stops, in the same order, are in the same direction, and so are trips that share pairs with them. The opposite directions of routes that share a pair of stops in the same order (e.g. through a one-way loop or street) are thus inferred as a single direction. Trips with a blank direction_id are never assigned the direction of other trips of their route.

In-vehicle links whose trips all have blank or decreasing times have an NA travel_time, with a warning; igraph::distances() doesn't accept it as a weight, and blank times can be filled beforehand with interpolate_stop_times(). Alighting links, in-vehicle links between stops with the same times and transfer links between stops at the same coordinates have a travel time of 0; igraph::closeness() and igraph::betweenness() require positive weights.

Departures are counted at every visit but the last of each trip, even with a blank departure time, unlike in get_stop_frequency(). The transfers and pathways tables, parent_stations and the pickup_type and drop_off_type fields are not used. Large values of max_walk_dist in dense feeds raise a warning when comparing the pairs of stops may use a lot of memory, or an error when there are too many pairs. Transfer links are not created across the antimeridian.

In the GMNS format, node_id, x_coord, y_coord, node_type, parent_node_id, link_id, from_node_id, to_node_id, directed, length and free_speed are GMNS fields; the other columns are additional fields, as allowed by GMNS. config lists the units, the coordinate reference system (EPSG:4326), the GMNS version (0.96) and the time of day. Stops not listed in stops or without coordinates have NA coordinates, with a warning (GMNS requires them). Free speeds may exceed the 200 km/h maximum checked by GMNS validators.

See Also

The "Building transit network graphs" vignette (vignette("graphs", package = "gtfstools")), which shows how to analyse and plot the network with {igraph}, {tidygraph}, {sfnetworks} and {ggraph}.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# the services of every day are pooled together, so only the services of a
# single weekday are kept
gtfs <- filter_by_weekday(gtfs, "wednesday")

graph <- build_graph(gtfs, from = "07:00:00", to = "08:00:00")
names(graph)
head(graph$node)
head(graph$link)

# the tables can be written to the csv files of a GMNS network
gmns_dir <- tempfile("gmns")
dir.create(gmns_dir)
for (table in names(graph)) {
  gmns_file <- file.path(gmns_dir, paste0(table, ".csv"))
  data.table::fwrite(graph[[table]], gmns_file)
}
```

```
list.files(gmns_dir)


# the shortest travel times, in minutes, from a stop to the other stops
graph <- build_graph(
  gtfs,
  from = "07:00:00",
  to = "08:00:00",
  format = "igraph"
)
stop_nodes <- igraph::V(graph)[igraph::V(graph)$node_type == "stop"]
travel_times <- igraph::distances(
  graph,
  v = stop_nodes[1],
  to = stop_nodes,
  mode = "out",
  weights = igraph::E(graph)$travel_time
)

# stops that can't be reached have an infinite travel time
is_reachable <- is.finite(travel_times)
sum(is_reachable)
summary(travel_times[is_reachable])
```

`combine_nearby_stops` *Combine nearby stops*

Description

Merges the stops that are within a given distance of each other into a single stop, and updates the tables that refer to them. Useful to build routing networks or to analyse the feed by station rather than by platform.

Usage

```
combine_nearby_stops(gtfs, max_dist)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`max_dist`	A number. The maximum distance, in meters, between two stops merged together.

Value

The GTFS object with the merged stops.

Details

Stops with a parent_station are first merged into one stop with the stop_id of the station, regardless of the distance between them, named after the station if it's listed in stops or else after its platforms. Stops are then merged by distance, closest pairs first, so that every pair of stops merged together is at most max_dist apart (great-circle distance, as in get_stop_distances()). Stops already merged by their parent station are compared by the mean of their coordinates, so their own platforms may be farther apart. Stops without coordinates are only merged by their parent station.

A merged stop has a stop_id and a stop_name made of the ids and names of the merged stops joined by "-" (repeated names listed once) and the mean of their coordinates; its other fields (e.g. zone_id, level_id) are those of the merged stop whose stop_id comes first. Only the first entry of a stop_id listed more than once in stops is used. Merged stop_ids are ambiguous when the original stop_ids include "-", and an error is raised if two resulting stop_ids are identical.

The result has only plain stops (location_type 0) without parent_station: stations are replaced by the stop merged from their platforms, and stations without platforms, entrances, generic nodes, boarding areas and pathways are removed. The stop_ids of merged stops and stations are updated in stop_times, transfers, stop_areas, location_group_stops and fare_leg_join_rules; references to removed stops are left as they are (they and their translations can be removed with remove_unused_ids()). Transfers between stops merged together become transfers within the merged stop, and when several transfers end up with the same stops, routes and trips, only the first is kept, even if their transfer_type or min_transfer_time differ; the other fields of transfers are not changed. Duplicated rows of stop_areas and location_group_stops are removed. Translations of merged stops are not updated.

An error is raised if max_dist would merge all stops into a single one. Large values of max_dist in dense feeds raise a warning when comparing the pairs of stops may use a lot of memory, or an error when there are too many pairs. Feeds that cross the antimeridian are not supported.

See Also

get_stop_distances(), merge_gtfs()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
nrow(gtfs$stops)

combined_gtfs <- combine_nearby_stops(gtfs, max_dist = 100)
nrow(combined_gtfs$stops)
```

---

convert_seconds_to_time

Convert seconds after midnight to time strings

Description

Converts times in seconds after midnight to strings in the "HH:MM:SS" format, as used in stop_times and frequencies. It is the counterpart of convert_time_to_seconds(), useful to write back times calculated from the _secs columns.

Usage

```
convert_seconds_to_time(seconds)
```

Arguments

seconds	A numeric vector with whole numbers of seconds after midnight, between 0 and 35999999 ("9999:59:59"). Values within floating-point tolerance of a whole number are rounded. difftime objects are converted to seconds first.

Value

A character vector with the same length as seconds. NAs are converted to "", as blank times in GTFS. Times past midnight keep counting the hours (e.g. "25:30:00"). Names are dropped.

See Also

convert_time_to_seconds()

Examples

```
convert_seconds_to_time(c(0, 3600, 91800, NA))

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
gtfs <- convert_time_to_seconds(gtfs, file = "stop_times")

# delay every departure by 5 minutes
delayed_departure <- convert_seconds_to_time(
  gtfs$stop_times$departure_time_secs + 300
)
head(delayed_departure)
```

---

convert_sf_to_shapes *Convert a simple feature object into a* shapes *table*

Description

Converts a LINESTRING sf object into a GTFS shapes table.

Usage

```
convert_sf_to_shapes(sf_shapes, shape_id = NULL, calculate_distance = TRUE)
```

Arguments

sf_shapes	A LINESTRING sf associating each shape_ids to a geometry. This object must use CRS WGS 84 (EPSG code 4326).
shape_id	A character vector specifying the shape_ids to be converted. If NULL (the default), all shapes are converted.
calculate_distance	A logical. Whether to calculate and populate the shape_dist_traveled column. This column is used to describe the distance along the shape from each one of its points to its first point, in meters. Distances are great-circle distances on a sphere with the same radius used by {s2} (6,371,010 meters), regardless of whether sf::sf_use_s2() is enabled. Defaults to TRUE.

Value

A data.table representing a GTFS shapes table.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# first converting existing shapes table into a sf object
shapes_sf <- get_shapes_geometry(gtfs)
head(shapes_sf)

# by default converts all shapes
result <- convert_sf_to_shapes(shapes_sf)
result

# shape_id argument controls which shapes are converted
result <- convert_sf_to_shapes(shapes_sf, shape_id = c("17846", "17847"))
result

# calculate_distance argument controls whether to calculate
# shape_dist_traveled or not
result <- convert_sf_to_shapes(shapes_sf, calculate_distance = TRUE)
result
```

---

convert_shapes_to_sf *Convert* shapes *table to simple feature object (deprecated)*

Description

This function is deprecated: calling it raises a warning. Please use get_shapes_geometry() instead, which takes the same arguments (except sort_sequence) and returns the same result.

Usage

```
convert_shapes_to_sf(gtfs, shape_id = NULL, crs = 4326, sort_sequence)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
shape_id	A character vector including the shape_ids to be converted. If NULL (the default), all shapes are converted.
crs	The CRS of the resulting object, either as an EPSG code or as an crs object. Defaults to 4326 (WGS 84).
sort_sequence	Deprecated; do not supply. Shapes are now always sorted by shape_pt_sequence. Passing TRUE raises a warning, and passing FALSE raises an error.

Value

A LINESTRING sf object.

See Also

get_shapes_geometry(), which replaces this function.

---

convert_stops_to_sf *Convert* stops *table to simple feature object (deprecated)*

Description

This function is deprecated: calling it raises a warning. Please use get_stops_geometry() instead, which takes the same arguments and returns the same result.

Usage

```
convert_stops_to_sf(gtfs, stop_id = NULL, crs = 4326)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A character vector including the stop_ids to be converted. If NULL (the default), all stops are converted.
crs	The CRS of the resulting object, either as an EPSG code or as an crs object. Defaults to 4326 (WGS 84).

Value

A POINT sf object.

See Also

get_stops_geometry(), which replaces this function.

convert_stops_to_shapes *Convert stops into shapes*

Description

Creates shapes for the given trips, linking their consecutive stops along straight lines, and assigns them to the trips. Useful for feeds without shapes, or in which some trips are not linked to a shape.

Usage

```
convert_stops_to_shapes(gtfs, trip_id = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to get new shapes, which replace the shapes they are currently linked to. If NULL (the default), only the trips that are not linked to a usable shape get new shapes (see details).

Value

The GTFS object passed to the gtfs parameter, with the new shapes added to the shapes table (created if needed) and assigned to the trips in the trips table.

Details

The shape of a trip links its stops, in stop_sequence order, along straight lines. Stops without coordinates, or not listed in stops, are skipped. Trips that visit the same sequence of stops share the same shape. The new shapes get new shape_ids ("stops_shape_1", "stops_shape_2", etc.), different from any shape_id already listed in shapes or trips. Their shape_dist_traveled is not calculated, because the distance unit used by the feed is unknown, and the shape_dist_traveled of the stop_times entries of the trips that get new shapes, if present, is set to NA. Shapes no longer used by any trip are kept (please use remove_unused_ids() to remove them).

A shape is usable if it has at least two distinct points with coordinates. With trip_id = NULL, the trips that get new shapes are those not linked to a shape_id, or linked to a shape_id that is not listed in shapes or whose shape is not usable (all trips, if the feed doesn't have a shapes table). These are the trips that get_trip_length() and get_trip_geometry() can't measure along a shape.

Trips with fewer than two distinct stops with coordinates, or without stop_times entries, can't get a usable shape, so they keep their current shape_id, with a warning.

See Also

get_trip_geometry(), convert_sf_to_shapes(), remove_unused_ids()

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# a feed without shapes gets one shape for each sequence of stops
gtfs$shapes <- NULL
new_gtfs <- convert_stops_to_shapes(gtfs)
head(new_gtfs$shapes)
new_gtfs$trips |>
  select(trip_id, shape_id) |>
  head()
```

---

convert_time_to_seconds

Convert time fields to seconds after midnight

Description

Converts `stop_times`' and `frequencies`' fields in the "HH:MM:SS" format to seconds after midnight. Instead of overwriting the existing fields, creates new fields with the `_secs` suffix.

Usage

```
convert_time_to_seconds(gtfs, file = NULL, by_reference = FALSE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`file`	A character vector, specifying the file whose fields should be converted. If `NULL` (the default), the function attempts to convert the times from both files, but only raises an error if none of them exist.
`by_reference`	Whether to update the tables by reference. Defaults to `FALSE`.

Details

Other gtfstools functions use existing `_secs` columns as-is. The functions that edit times keep them in sync, but other edits (e.g. made directly to the tables) don't, so re-run `convert_time_to_seconds()` after them.

Value

If `by_reference` is `FALSE`, returns a GTFS object with additional time in seconds columns (identified by a `_secs` suffix). Else, returns a GTFS object invisibly (please note that in such case the original GTFS object is altered).

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

# by default converts both 'stop_times' and 'frequencies' times
converted_gtfs <- convert_time_to_seconds(gtfs)
head(converted_gtfs$stop_times)
head(converted_gtfs$frequencies)

# choose which table to convert with 'file'
converted_gtfs <- convert_time_to_seconds(gtfs, file = "frequencies")
head(converted_gtfs$stop_times)
head(converted_gtfs$frequencies)

# original gtfs remained unchanged, as seen with the frequencies table above
# change original object without creating a copy with 'by_reference = TRUE'
convert_time_to_seconds(gtfs, by_reference = TRUE)
head(gtfs$stop_times)
head(gtfs$frequencies)
```

---

crop_gtfs *Crop a GTFS object to a spatial extent*

Description

Crops a GTFS object to a spatial extent (an `sf`, `sfc` or `bbox` object), keeping only the stops inside it (and their parent stations) and the parts of the trips that serve them. Unlike `filter_by_spatial_extent()`, which keeps or drops entire trips, `crop_gtfs()` cuts the trips that leave the spatial extent, keeping only their stops inside it, and, by default, trims their shapes accordingly.

Usage

```
crop_gtfs(gtfs, geom, trim_shapes = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`geom`	An `sf`, `sfc` or `bbox` object, in WGS 84 (EPSG 4326). Describes the spatial extent used to crop the data. Stops on its boundary are considered inside it. A `bbox` is compared directly with the stops' coordinates, while `sf` and `sfc` objects are tested with `sf::st_intersects()`, whose edges are geodesics when spherical geometry is used (the default in `{sf}`), so stops very close to their edges may be considered outside. Bounding boxes crossing the antimeridian are not supported.
`trim_shapes`	A logical. Whether the shapes of the trips that leave the spatial extent should be trimmed to the part between their first and last stops inside it (defaults to `TRUE`). If `FALSE`, these trips keep their entire shapes.

Value

The GTFS object passed to the `gtfs` parameter, after the cropping process.

Details

The `stop_times` entries at stops outside the spatial extent are dropped. Trips left with fewer than two `stop_times` entries are dropped, as are the entries related to them in other tables, as in `filter_by_trip_id()`. A trip that leaves the spatial extent and enters it again keeps the stops inside it, with gaps in their `stop_sequence`. The resulting `stops` table includes the stops inside the spatial extent served by the remaining trips, the entrances, generic nodes and boarding areas inside it related to these stops, and their parent stations, even if these lie outside the spatial extent. Stops without coordinates are considered outside of it, so generic nodes and boarding areas without coordinates, and the pathways that link to them, are dropped.

GTFS requires times at the first and last stops of each trip. If the new first or last stop of a cropped trip has blank times, the blank times of the trip are interpolated with `interpolate_stop_times()` (along its shape, or along straight lines between stops if the trip is not linked to a shape), before the trip is cropped. Intermediate blank times of these trips are thus filled too. Times that can't be interpolated (see `interpolate_stop_times()`) are left blank, with a warning. If a cropped trip is listed in `frequencies`, the `start_time` and `end_time` of its entries are shifted by the time between the departure (or the arrival, if the departure is blank) at its original first stop and at its new one, since they refer to the first stop of the trip. Entries whose shift can't be calculated (e.g. because of blank times) or is negative are kept unchanged.

With `trim_shapes = TRUE`, the stops of each cropped trip are located along its shape, as in `get_trip_length()`, and its shape is cut from its first to its last remaining stop, keeping the parts of the shape outside the spatial extent between them. A shape cut in only one way keeps its `shape_id`. If a shape is cut in different ways, or is also used by trips that are not cut, each cut gets a new `shape_id` (`"<shape_id>_1"`, `"<shape_id>_2"`, etc., different from any `shape_id` already in use). The `shape_dist_traveled` of the new shape points and of the `stop_times` entries of the trips linked to them is set to `NA`, because the distance unit used by the feed is unknown. Trips not linked to a usable shape keep their shape unchanged.

See Also

`filter_by_spatial_extent()`, `interpolate_stop_times()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

bbox <- sf::st_bbox(
  c(xmin = -46.66, ymin = -23.57, xmax = -46.62, ymax = -23.53),
  crs = 4326
)
object.size(gtfs)

# keeps only the stops inside the bounding box and trims the shapes
cropped_gtfs <- crop_gtfs(gtfs, bbox)
object.size(cropped_gtfs)
```

```
# keeps the entire shapes of the cropped trips
cropped_gtfs <- crop_gtfs(gtfs, bbox, trim_shapes = FALSE)
object.size(cropped_gtfs)
```

`download_validator` *Download MobilityData's GTFS validator*

Description

Downloads MobilityData's command line tool to validate GTFS feeds.

Usage

```
download_validator(
  path = tempdir(),
  version = "latest",
  force = FALSE,
  quiet = TRUE
)
```

Arguments

`path`	A string. The directory where the validator should be saved to. By default, the function downloads the validator to a temporary directory, which is removed when the R session ends. Use a persistent directory to reuse the validator across sessions.
`version`	A string. The version of the validator that should be downloaded. Defaults to `"latest"`, but accepts version numbers as strings (i.e. to download version v6.0.0 please enter `"6.0.0"`). Please check MobilityData/gtfs-validator releases for the full set of available versions.
`force`	A logical. Whether to overwrite a previously downloaded validator in `path`. Defaults to `FALSE`.
`quiet`	A logical. Whether to hide log messages and progress bars. Defaults to `TRUE`.

Value

Invisibly returns the normalized path to the downloaded validator. If the validator can't be downloaded due to network problems (e.g. no internet connection or GitHub being unavailable), it informs that with a message and invisibly returns `NULL`.

See Also

Other validation: `list_validator_versions()`, `validate_gtfs()`

Examples

```
path <- tempdir()

download_validator(path)

# specifying a specific version
download_validator(path, version = "6.0.0")
```

`filter_by_agency_id` *Filter GTFS object by* `agency_id`

Description

Filters a GTFS object by `agency_ids`, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_agency_id(gtfs, agency_id, keep = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`agency_id`	A character vector. The `agency_ids` used to filter the data.
`keep`	A logical. Whether the entries related to the specified `agency_ids` should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

See Also

Other filtering functions: `filter_by_date()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
agency_id <- "92"

object.size(gtfs)

# keeps entries related to passed agency_id
smaller_gtfs <- filter_by_agency_id(gtfs, agency_id)
object.size(smaller_gtfs)
```

```
# drops entries related to passed agency_id
smaller_gtfs <- filter_by_agency_id(gtfs, agency_id, keep = FALSE)
object.size(smaller_gtfs)
```

`filter_by_date` *Filter GTFS object by date*

Description

Filters a GTFS object by the services that run on the given dates, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_date(gtfs, date, keep = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`date`	A `Date` vector (including `IDate`) or a character vector of dates in the `"YYYYMMDD"` or `"YYYY-MM-DD"` formats. The dates used to filter the data.
`keep`	A logical. Whether the entries related to the services that run on the specified dates should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

Details

The services that run on at least one of the given dates are kept (or dropped) as a whole: their `calendar` and `calendar_dates` entries are not trimmed to the given dates, so they may still run on other dates.

A service runs on a date if its `calendar` entry spans the date and flags its weekday, unless the date is removed from it in `calendar_dates` (`exception_type` 2), or if the date is added to it in `calendar_dates` (`exception_type` 1). An addition wins over a removal on the same date. Either table may be missing. If both are missing, the GTFS object is returned unchanged, with a warning.

See Also

`filter_by_weekday()`, `get_calendar_overlap()`

Other filtering functions: `filter_by_agency_id()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# on 2006-07-03 (a monday) the weekday service "WD" is replaced by the
# weekend service "WE", according to the calendar_dates table
gtfs$calendar_dates

smaller_gtfs <- filter_by_date(gtfs, "2006-07-03")
smaller_gtfs$calendar

# dates may also be given as Date objects. drops the services that run on
# 2006-07-05 (a wednesday)
smaller_gtfs <- filter_by_date(gtfs, as.Date("2006-07-05"), keep = FALSE)
smaller_gtfs$calendar

# a vector of dates keeps the services that run on at least one of them
week <- seq(as.Date("2006-07-01"), as.Date("2006-07-07"), by = "day")
smaller_gtfs <- filter_by_date(gtfs, week)
smaller_gtfs$calendar
```

---

filter_by_period *Filter GTFS object by period*

Description

Filters a GTFS object by the services that run on at least one day of a period, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_period(gtfs, start_date, end_date, keep = TRUE, trim = TRUE)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
start_date, end_date	A Date (including IDate) or a string in the "YYYYMMDD" or "YYYY-MM-DD" formats. The first and last days of the period, both included. end_date must not be earlier than start_date.
keep	A logical. Whether the entries related to the services that run in the specified period should be kept or dropped (defaults to TRUE, which keeps the entries).
trim	A logical. Whether the calendars of the kept services should be cut to the period (defaults to TRUE), so that the GTFS object has no service outside it. Ignored when keep is FALSE.

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

Details

The services are selected as when filtering by each day of the period with `filter_by_date()`, whose details also apply here: the services that run on at least one day of the period are kept (or dropped).

With `trim = TRUE`, the kept services are then cut to the period: the `start_date` and `end_date` of their `calendar` entries are clipped to it (entries left without any day are removed), and the `calendar_dates` entries outside it are removed. Each service then runs on exactly the same days within the period as before, and on no day outside it. The `feed_info` table is not changed. With `trim = FALSE`, the services are kept as a whole, so they may still run on days outside the period.

The period is first clipped to the dates covered by the `calendar` and `calendar_dates` tables, on which the services can run, so the days of the period outside them add no work. The memory used still grows with the number of days the period shares with each service's calendar.

See Also

`filter_by_date()`, `get_active_services()`, `get_dates()`

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

gtfs$calendar
gtfs$calendar_dates

# keeps the services that run in the first week of july 2006, with their
# calendars cut to that week
smaller_gtfs <- filter_by_period(gtfs, "2006-07-01", "2006-07-07")
smaller_gtfs$calendar

# keeps the same services, with their whole calendars
smaller_gtfs <- filter_by_period(
  gtfs,
  "2006-07-01",
  "2006-07-07",
  trim = FALSE
)
smaller_gtfs$calendar

# dates may also be given as Date objects. drops the services that run in
# the period
smaller_gtfs <- filter_by_period(
  gtfs,
```

```
  as.Date("2006-07-03"),
  as.Date("2006-07-04"),
  keep = FALSE
)
smaller_gtfs$calendar
```

`filter_by_route_id` *Filter GTFS object by* `route_id`

Description

Filters a GTFS object by `route_id`s, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_route_id(gtfs, route_id, keep = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`route_id`	A character vector. The `route_id`s used to filter the data.
`keep`	A logical. Whether the entries related to the specified `route_id`s should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

See Also

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_period()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
route_ids <- c("6450-51", "CPTM L11")

object.size(gtfs)

# keeps entries related to passed route_ids
smaller_gtfs <- filter_by_route_id(gtfs, route_ids)
object.size(smaller_gtfs)

# drops entries related to passed route_ids
smaller_gtfs <- filter_by_route_id(gtfs, route_ids, keep = FALSE)
object.size(smaller_gtfs)
```

`filter_by_route_type` *Filter GTFS object by* `route_type` *(transport mode)*

Description

Filters a GTFS object by `route_types`, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_route_type(gtfs, route_type, keep = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`route_type`	An integer vector. The `route_types` used to filter the data.
`keep`	A logical. Whether the entries related to the specified `route_types` should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

Route types

Valid options include the route types listed in the GTFS Schedule specification and in the Google Transit implementation. The types specified in the GTFS Schedule specification are:

- 0 - Tram, Streetcar, Light rail. Any light rail or street level system within a metropolitan area.
- 1 - Subway, Metro. Any underground rail system within a metropolitan area.
- 2 - Rail. Used for intercity or long-distance travel.
- 3 - Bus. Used for short- and long-distance bus routes.
- 4 - Ferry. Used for short- and long-distance boat service.
- 5 - Cable tram. Used for street-level rail cars where the cable runs beneath the vehicle, e.g., cable car in San Francisco.
- 6 - Aerial lift, suspended cable car (e.g., gondola lift, aerial tramway). Cable transport where cabins, cars, gondolas or open chairs are suspended by means of one or more cables.
- 7 - Funicular. Any rail system designed for steep inclines.
- 11 - Trolleybus. Electric buses that draw power from overhead wires using poles.
- 12 - Monorail. Railway in which the track consists of a single rail or a beam.

The types defined in Google Transit's extension are listed below, including some examples (not available for all types):

- 100 - Railway Service - Not applicable (N/A)
- 101 - High Speed Rail Service - TGV (FR), ICE (DE), Eurostar (GB)

- 102 - Long Distance Trains - InterCity/EuroCity
- 103 - Inter Regional Rail Service - InterRegio (DE), Cross County Rail (GB)
- 104 - Car Transport Rail Service
- 105 - Sleeper Rail Service - GNER Sleeper (GB)
- 106 - Regional Rail Service - TER (FR), Regionalzug (DE)
- 107 - Tourist Railway Service - Romney, Hythe & Dymchurch (GB)
- 108 - Rail Shuttle (Within Complex) - Gatwick Shuttle (GB), Sky Line (DE)
- 109 - Suburban Railway - S-Bahn (DE), RER (FR), S-tog (Kopenhagen)
- 110 - Replacement Rail Service
- 111 - Special Rail Service
- 112 - Lorry Transport Rail Service
- 113 - All Rail Services
- 114 - Cross-Country Rail Service
- 115 - Vehicle Transport Rail Service
- 116 - Rack and Pinion Railway - Rochers de Naye (CH), Dolderbahn (CH)
- 117 - Additional Rail Service
- 200 - Coach Service
- 201 - International Coach Service - EuroLine, Touring
- 202 - National Coach Service - National Express (GB)
- 203 - Shuttle Coach Service - Roissy Bus (FR), Reading-Heathrow (GB)
- 204 - Regional Coach Service
- 205 - Special Coach Service
- 206 - Sightseeing Coach Service
- 207 - Tourist Coach Service
- 208 - Commuter Coach Service
- 209 - All Coach Services
- 400 - Urban Railway Service
- 401 - Metro Service - Métro de Paris
- 402 - Underground Service - London Underground, U-Bahn
- 403 - Urban Railway Service
- 404 - All Urban Railway Services
- 405 - Monorail
- 700 - Bus Service
- 701 - Regional Bus Service - Eastbourne-Maidstone (GB)
- 702 - Express Bus Service - X19 Wokingham-Heathrow (GB)
- 703 - Stopping Bus Service - 38 London: Clapton Pond-Victoria (GB)
- 704 - Local Bus Service

- 705 - Night Bus Service - N prefixed buses in London (GB)
- 706 - Post Bus Service - Maidstone P4 (GB)
- 707 - Special Needs Bus
- 708 - Mobility Bus Service
- 709 - Mobility Bus for Registered Disabled
- 710 - Sightseeing Bus
- 711 - Shuttle Bus - 747 Heathrow-Gatwick Airport Service (GB)
- 712 - School Bus
- 713 - School and Public Service Bus
- 714 - Rail Replacement Bus Service
- 715 - Demand and Response Bus Service
- 716 - All Bus Services
- 800 - Trolleybus Service
- 900 - Tram Service
- 901 - City Tram Service
- 902 - Local Tram Service - Munich (DE), Brussels (BE), Croydon (GB)
- 903 - Regional Tram Service
- 904 - Sightseeing Tram Service - Blackpool Seafront (GB)
- 905 - Shuttle Tram Service
- 906 - All Tram Services
- 1000 - Water Transport Service
- 1100 - Air Service
- 1200 - Ferry Service
- 1300 - Aerial Lift Service - Telefèric de Montjuïc (ES), Saleve (CH), Roosevelt Island Tramway (US)
- 1301 - Telecabin Service
- 1302 - Cable Car Service
- 1303 - Elevator Service
- 1304 - Chair Lift Service
- 1305 - Drag Lift Service
- 1306 - Small Telecabin Service
- 1307 - All Telecabin Services
- 1400 - Funicular Service - Rigiblick (Zürich, CH)
- 1500 - Taxi Service
- 1501 - Communal Taxi Service - Marshrutka (RU), dolmuş (TR)
- 1502 - Water Taxi Service
- 1503 - Rail Taxi Service

- 1504 - Bike Taxi Service
- 1505 - Licensed Taxi Service
- 1506 - Private Hire Service Vehicle
- 1507 - All Taxi Services
- 1700 - Miscellaneous Service
- 1702 - Horse-drawn Carriage

See Also

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
# read gtfs
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

object.size(gtfs)

# keeps entries related to passed route_types
smaller_gtfs <- filter_by_route_type(gtfs, route_type = 1)
object.size(smaller_gtfs)

# drops entries related to passed route_types
smaller_gtfs <- filter_by_route_type(gtfs, route_type = 1, keep = FALSE)
object.size(smaller_gtfs)
```

---

filter_by_service_id *Filter GTFS object by* service_id

Description

Filters a GTFS object by service_ids, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_service_id(gtfs, service_id, keep = TRUE)
```

Arguments

gtfs	A GTFS object, as created by `read_gtfs()`.
service_id	A character vector. The service_ids used to filter the data.
keep	A logical. Whether the entries related to the specified service_ids should be kept or dropped (defaults to TRUE, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

See Also

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
service_ids <- c("USD", "U__")

object.size(gtfs)

# keeps entries related to the specified service_ids
smaller_gtfs <- filter_by_service_id(gtfs, service_ids)
object.size(smaller_gtfs)

# drops entries related to the specified service_ids
smaller_gtfs <- filter_by_service_id(gtfs, service_ids, keep = FALSE)
object.size(smaller_gtfs)
```

`filter_by_sf` *Filter a GTFS object using a* `simple features` *object (defunct)*

Description

This function was deprecated in gtfstools 1.3.0 and is now defunct: calling it raises an error. Please use `filter_by_spatial_extent()` instead, which takes the same arguments.

Usage

```
filter_by_sf(...)
```

Arguments

`...`	Ignored.

Value

Doesn't return: always raises an error.

See Also

`filter_by_spatial_extent()`, which replaces this function.

filter_by_shape_id *Filter GTFS object by* shape_id

Description

Filters a GTFS object by shape_ids, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_shape_id(gtfs, shape_id, keep = TRUE)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
shape_id	A character vector. The shape_ids used to filter the data.
keep	A logical. Whether the entries related to the specified shape_ids should be kept or dropped (defaults to TRUE, which keeps the entries).

Value

The GTFS object passed to the gtfs parameter, after the filtering process.

See Also

Other filtering functions: filter_by_agency_id(), filter_by_date(), filter_by_period(), filter_by_route_id(), filter_by_route_type(), filter_by_service_id(), filter_by_spatial_extent(), filter_by_stop_id(), filter_by_time_of_day(), filter_by_trip_id(), filter_by_weekday()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
shape_ids <- c("17846", "68962")

object.size(gtfs)

# keeps entries related to passed shape_ids
smaller_gtfs <- filter_by_shape_id(gtfs, shape_ids)
object.size(smaller_gtfs)

# drops entries related to passed shape_ids
smaller_gtfs <- filter_by_shape_id(gtfs, shape_ids, keep = FALSE)
object.size(smaller_gtfs)
```

filter_by_spatial_extent *Filter a GTFS object using a spatial extent*

Description

Filters a GTFS object using a spatial extent (passed as an `sf` object), keeping (or dropping) entries related to shapes and trips whose geometries are selected through a specified spatial operation.

Usage

```
filter_by_spatial_extent(
  gtfs,
  geom,
  spatial_operation = sf::st_intersects,
  keep = TRUE
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`geom`	An `sf`, `sfc` or `bbox` object, in WGS 84 (EPSG 4326). Describes the spatial extent used to filter the data. A `bbox` is treated as a rectangle in longitude and latitude (unlike `sf::st_as_sfc(bbox)`, whose edges are geodesics when `{s2}` is used).
`spatial_operation`	A spatial operation function from the set of options listed in geos_binary_pred. Defaults to `sf::st_intersects`, which tests if the shapes and trips have ANY intersection with the object specified in `geom`. Please note that `geom` is passed as the `x` argument of these functions. `sf::st_equals_exact()` and `sf::st_is_within_distance()` are not supported, since they require additional arguments.
`keep`	A logical. Whether the entries related to the shapes and trips selected by the given spatial operation should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

See Also

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`, `filter_by_weekday()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

shape_id <- "68962"
shape_sf <- get_shapes_geometry(gtfs, shape_id)
bbox <- sf::st_bbox(shape_sf)
object.size(gtfs)

# keeps entries that intersect with the specified polygon
smaller_gtfs <- filter_by_spatial_extent(gtfs, bbox)
object.size(smaller_gtfs)

# drops entries that intersect with the specified polygon
smaller_gtfs <- filter_by_spatial_extent(gtfs, bbox, keep = FALSE)
object.size(smaller_gtfs)

# uses a different function to filter the gtfs
smaller_gtfs <- filter_by_spatial_extent(
  gtfs,
  bbox,
  spatial_operation = sf::st_contains
)
object.size(smaller_gtfs)
```

---

filter_by_stop_id *Filter GTFS object by* stop_id

Description

Filters a GTFS object by stop_ids, keeping (or dropping) relevant entries in each file.

Usage

```
filter_by_stop_id(
  gtfs,
  stop_id,
  keep = TRUE,
  include_children = TRUE,
  include_parents = keep,
  full_trips
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A character vector. The stop_ids used to filter the data.

keep
: A logical. Whether the stop_times entries at the specified stop_ids (and the entries related to them in other tables) should be kept or dropped (defaults to TRUE, which keeps the entries). Trips that still have entries in stop_times are kept. To keep or drop entire trips, pass the trip_ids that serve these stops to filter_by_trip_id().

include_children
: A logical. Whether the filtered output should keep/drop children stops of those specified in stop_id. Defaults to TRUE - i.e. by default children stops are kept if their parents are kept and dropped if their parents are dropped.

include_parents
: A logical. Whether the filtered output should keep/drop parent stations of those specified in stop_id. Defaults to the same value of keep - i.e. by default parent stations are kept both when their children are kept and dropped, because they can be parents of multiple stops that are not necessarily dropped, even if their sibling are.

full_trips
: Defunct. Deprecated in version 1.3.0 and removed in the following major version: using it, with any value, raises an error. The function now always filters by the specified stops. To keep all stops of the trips that pass through them, subset stop_times by stop_id and pass the resulting trip_ids to filter_by_trip_id().

Value

The GTFS object passed to the gtfs parameter, after the filtering process.

See Also

Other filtering functions: filter_by_agency_id(), filter_by_date(), filter_by_period(), filter_by_route_id(), filter_by_route_type(), filter_by_service_id(), filter_by_shape_id(), filter_by_spatial_extent(), filter_by_time_of_day(), filter_by_trip_id(), filter_by_weekday()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
stop_ids <- c("18848", "940004157")

object.size(gtfs)

# keeps the stop_times entries at the specified stop_ids
smaller_gtfs <- filter_by_stop_id(gtfs, stop_ids)
object.size(smaller_gtfs)

# drops the stop_times entries at the specified stop_ids
smaller_gtfs <- filter_by_stop_id(gtfs, stop_ids, keep = FALSE)
object.size(smaller_gtfs)
```

`filter_by_time_of_day` *Filter GTFS object by time of day*

Description

Filters a GTFS object by time of day, keeping (or dropping) the `stop_times` and `frequencies` entries within it, and keeping the entries of the other tables related to the trips that remain.

Usage

```
filter_by_time_of_day(
  gtfs,
  from,
  to,
  keep = TRUE,
  full_trips = FALSE,
  update_frequencies = TRUE
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`from`	A string. The starting point of the time of day, in the "HH:MM:SS" format.
`to`	A string. The ending point of the time of day, in the "HH:MM:SS" format.
`keep`	A logical. Whether the entries related to the specified time of day should be kept or dropped (defaults to `TRUE`, which keeps the entries).
`full_trips`	A logical. Whether trips should be treated as immutable blocks or each of its stops should be considered separately when filtering the `stop_times` table (defaults to `FALSE`, which considers each stop individually). Please check the details section for more information on how this parameter changes the function behaviour.
`update_frequencies`	A logical. Whether the `frequencies` table should have its `start_time` and `end_time` fields updated to fit inside/outside the specified time of day (defaults to `TRUE`, which updates the fields).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

Details

The time of day includes both `from` and `to`.

Trips listed in `frequencies` are filtered through their `frequencies` entries; their `stop_times` entries are templates and are kept whatever their times. With `keep = TRUE`, the entries that overlap the time of day are kept; with `keep = FALSE`, those entirely within it are dropped. With `update_frequencies`

= TRUE and keep = TRUE, an entry that starts before from starts at from (or, if exact_times is 1, at its first departure at or after from), and an entry that ends after to ends one second after to, so that a departure exactly at to is kept. With update_frequencies = TRUE and keep = FALSE, an entry that crosses from ends at from, an entry that crosses to starts one second after to (or, if exact_times is 1, at its first departure after to), and an entry that spans the whole time of day is split in two. Entries left without any departure are removed, and trips left without any entry are dropped from the feed.

The other trips are filtered through their stop_times entries, which are within the time of day when the visit, from arrival_time to departure_time, overlaps it (a blank time is ignored when the other is known). With full_trips = FALSE, each entry is kept or dropped on its own, and entries with both times blank (or, with keep = FALSE, either) are dropped. With full_trips = TRUE, a trip with any entry within the time of day is kept (keep = TRUE) or dropped (keep = FALSE) as a whole, including its blank entries. The other tables are then filtered to the trips left in stop_times, as in filter_by_trip_id(), with a warning if stop_times lists trips missing from trips.

Existing _secs columns in stop_times and frequencies (e.g. created with convert_time_to_seconds()) are used as-is, not recalculated from the time strings.

See Also

Other filtering functions: filter_by_agency_id(), filter_by_date(), filter_by_period(), filter_by_route_id(), filter_by_route_type(), filter_by_service_id(), filter_by_shape_id(), filter_by_spatial_extent(), filter_by_stop_id(), filter_by_trip_id(), filter_by_weekday()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# taking a look at the original frequencies and stop_times
head(gtfs$frequencies)
head(gtfs$stop_times)

smaller_gtfs <- filter_by_time_of_day(gtfs, "05:00:00", "06:00:00")

# filter_by_time_of_day filters the frequencies table but doesn't filter the
# stop_times table because they're just templates
head(smaller_gtfs$frequencies)
head(smaller_gtfs$stop_times)

# frequencies entries are adjusted by default. use update_frequencies =
# FALSE to keep their original times
smaller_gtfs <- filter_by_time_of_day(
  gtfs,
  "05:30:00",
  "06:00:00",
  update_frequencies = FALSE
)
head(smaller_gtfs$frequencies)

# when keep = FALSE, the behaviour of the function in general, and of
# update_frequencies in particular, is a bit different
smaller_gtfs <- filter_by_time_of_day(
```

```
  gtfs,
  "05:30:00",
  "06:00:00",
  keep = FALSE,
  update_frequencies = TRUE
)
head(smaller_gtfs$frequencies)

# let's remove the frequencies table to check the behaviour of full_trips
gtfs$frequencies <- NULL
smaller_gtfs <- filter_by_time_of_day(
  gtfs,
  "05:30:00",
  "06:00:00"
)
head(smaller_gtfs$stop_times)

smaller_gtfs <- filter_by_time_of_day(
  gtfs,
  "05:30:00",
  "06:00:00",
  full_trips = TRUE
)
head(smaller_gtfs$stop_times)
```

`filter_by_trip_id` *Filter GTFS object by* `trip_id`

Description

Filters a GTFS object by `trip_id`s, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_trip_id(gtfs, trip_id, keep = TRUE)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector. The `trip_id`s used to filter the data.
`keep`	A logical. Whether the entries related to the specified `trip_id`s should be kept or dropped (defaults to `TRUE`, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

See Also

Other filtering functions: filter_by_agency_id(), filter_by_date(), filter_by_period(), filter_by_route_id(), filter_by_route_type(), filter_by_service_id(), filter_by_shape_id(), filter_by_spatial_extent(), filter_by_stop_id(), filter_by_time_of_day(), filter_by_weekday()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
trip_ids <- c("CPTM L07-0", "2002-10-0")

object.size(gtfs)

# keeps entries related to passed trip_ids
smaller_gtfs <- filter_by_trip_id(gtfs, trip_ids)
object.size(smaller_gtfs)

# drops entries related to passed trip_ids
smaller_gtfs <- filter_by_trip_id(gtfs, trip_ids, keep = FALSE)
object.size(smaller_gtfs)
```

---

filter_by_weekday *Filter GTFS object by weekday*

Description

Filters a GTFS object by weekday, keeping (or dropping) the relevant entries in each file.

Usage

```
filter_by_weekday(gtfs, weekday, combine = "or", keep = TRUE)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
weekday	A character vector. The weekdays used to filter the data. Possible values are c("monday", "tuesday", "wednesday", "thursday", "friday", "saturday", "sunday").
combine	A string. Specifies which logic operation (OR or AND) should be used to filter the calendar table when multiple weekdays are specified. Defaults to "or". Please check the details and examples sections for more information on this argument usage.
keep	A logical. Whether the entries related to the specified weekdays should be kept or dropped (defaults to TRUE, which keeps the entries).

Value

The GTFS object passed to the `gtfs` parameter, after the filtering process.

`combine` usage

When filtering the calendar table using weekdays, one could reason about the process in different ways. For example, you may want to keep only services who run on mondays AND tuesdays. Or you may want to keep services that run EITHER on mondays OR on tuesdays. The first case is the equivalent of filtering using the expression `monday == 1 & tuesday == 1`, while the second uses `monday == 1 | tuesday == 1`. You can use the `combine` argument to control this behaviour.

Please note that `combine` also works together with `keep`. Using the same examples listed above, you could either keep the entries related to services that run on mondays and tuesdays or drop them, depending on the value you pass to `keep`.

See Also

Other filtering functions: `filter_by_agency_id()`, `filter_by_date()`, `filter_by_period()`, `filter_by_route_id()`, `filter_by_route_type()`, `filter_by_service_id()`, `filter_by_shape_id()`, `filter_by_spatial_extent()`, `filter_by_stop_id()`, `filter_by_time_of_day()`, `filter_by_trip_id()`

Examples

```
library(dplyr)

# read gtfs
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

object.size(gtfs)

# keeps entries related to services than run EITHER on monday OR on sunday
smaller_gtfs <- filter_by_weekday(gtfs, weekday = c("monday", "sunday"))
smaller_gtfs$calendar |>
  select(service_id, monday, sunday)
object.size(smaller_gtfs)

# keeps entries related to services than run on monday AND on sunday
smaller_gtfs <- filter_by_weekday(
  gtfs,
  weekday = c("monday", "sunday"),
  combine = "and"
)
smaller_gtfs$calendar |>
  select(service_id, monday, sunday)
object.size(smaller_gtfs)

# drops entries related to services than run EITHER on monday OR on sunday
# the resulting gtfs shouldn't include any trips running on these days
smaller_gtfs <- filter_by_weekday(
  gtfs,
```

```
  weekday = c("monday", "sunday"),
  keep = FALSE
)
smaller_gtfs$calendar |>
  select(service_id, monday, sunday)
object.size(smaller_gtfs)

# drops entries related to services than run on monday AND on sunday
# the resulting gtfs may include trips that run on these days, but no trips
# that run on both these days
smaller_gtfs <- filter_by_weekday(
  gtfs,
  weekday = c("monday", "sunday"),
  combine = "and",
  keep = FALSE
)
smaller_gtfs$calendar |>
  select(service_id, monday, sunday)
object.size(smaller_gtfs)
```

`fix_gtfs` *Fix common problems in GTFS feeds*

Description

Cleans a GTFS feed in one call, by removing duplicated entries and unused ids, optionally capping the speed of trips, filling blank stop times, optionally setting a minimum dwell time at intermediate stops and recalculating the distances travelled along shapes (replacing any existing ones).

Usage

```
fix_gtfs(
  gtfs,
  max_speed = NULL,
  speed_unit = "km/h",
  min_dwell_time = NULL,
  dwell_time_unit = "s"
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`max_speed`	A number. The maximum speed of trips, in the unit given by `speed_unit`. If `NULL` (the default), speeds are not capped.
`speed_unit`	A string representing the unit in which `max_speed` is given. One of `"km/h"` (the default), `"m/s"` or `"mph"` (miles per hour).

min_dwell_time A non-negative number. The minimum time vehicles remain stopped at each intermediate stop of their trips, with their doors open to serve passengers, in the unit given by dwell_time_unit. If NULL (the default), dwell times are not changed.

dwell_time_unit
A string representing the time unit in which min_dwell_time is given. One of "s" (seconds, the default), "min" (minutes), "h" (hours) or "d" (days).

Value

The fixed GTFS object.

Details

The feed goes through the following steps, in this order:

1. remove_duplicates() removes duplicated entries from every table;
2. remove_unused_ids() removes the trips without stop_times, and the entries of every table that no remaining trip uses (see its documentation for the exceptions). Note that this removes data from the feed;
3. set_max_speed(), only if max_speed is given, slows down the parts of trips that are faster than max_speed. Only spans between stops with known times are capped, and spans that end before they start (e.g. times after midnight written as "00:30:00") are left unchanged, with a warning;
4. interpolate_stop_times() fills blank stop times from the times of the stops around them. Speeds are capped before interpolating, so the filled times follow the capped times. Interpolated times are rounded to the second. Stops whose times can't be interpolated are left blank, with a warning;
5. set_min_dwell_time(), only if min_dwell_time is given, delays the departure from each intermediate stop whose dwell time is shorter than min_dwell_time, so that vehicles stay there for at least that long. The minimum is rounded to the second. The later times of each changed trip are shifted accordingly, so the trip ends later (for trips listed in frequencies, every departure gets longer). This is done after filling blank times, so the filled stops also get the minimum. Only the stops between the first and the last stop with times of each trip get the minimum, so trips keep their start times. Stops with negative dwell times (departures earlier than arrivals) also get it, see set_min_dwell_time() for times after midnight written as "00:30:00";
6. interpolate_shape_dist_traveled(), only if the feed has a shapes table and trips has a shape_id column, replaces the shape_dist_traveled columns of stop_times and shapes with the distances, in meters, from the start of each shape. Existing values, whatever their unit, are replaced, and stops that can't be located along their trip's shape get NA.

The trips and stop_times tables are always required, and stops is required to cap speeds, to fill blank times and to calculate the distances travelled along shapes. The warnings raised are those of the functions above, so some warnings (e.g. about trips not linked to a shape) may be raised more than once.

See Also

remove_duplicates(), remove_unused_ids(), set_max_speed(), interpolate_stop_times(), set_min_dwell_time(), interpolate_shape_dist_traveled()

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

fixed_gtfs <- fix_gtfs(gtfs, max_speed = 40, min_dwell_time = 30)

# duplicated agencies were removed, and the trip is now slower and stays at
# least 30 seconds at each intermediate stop
gtfs$agency
fixed_gtfs$agency

gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0") |>
  slice_tail(n = 1)

fixed_gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0") |>
  slice_tail(n = 1)
```

---

frequencies_to_stop_times

Convert frequencies to stop times

Description

Creates `stop_times` entries based on the frequencies specified in the `frequencies` table.

Usage

```
frequencies_to_stop_times(
  gtfs,
  trip_id = NULL,
  force = FALSE,
  strategy = "exact"
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to have their frequencies converted to stop_times entries. If NULL (the default), the function converts all trips listed in the frequencies table.
force	Whether to convert trips specified in the frequencies table even if they are not described in stop_times (defaults to FALSE). When set to TRUE, these mismatched trip are removed from the frequencies table and their correspondent entries in trips are substituted by what would be their converted counterpart.
strategy	A string. The strategy used to set the departure times of the trips created from frequency-based entries of the frequencies table (i.e. those whose exact_times is not 1). One of: • "exact" (the default): trips depart every headway_secs from start_time; • "half_headway": departures are shifted by half the headway, rounded down to whole seconds; • "random": departures are shifted by a random whole number of seconds, from 0 up to (but not including) the headway, drawn independently for each entry. Uses R's random number generator, so results can be made reproducible with set.seed(). In entries shorter than their headway, the duration of the entry (from start_time to end_time) is used in place of the headway, so that no departure is shifted to or past end_time and each entry still yields at least one trip. As departures are shifted towards end_time, an entry may yield one trip fewer than with "exact". Entries whose exact_times is 1 follow an exact schedule and are always converted as with "exact".

Value

A GTFS object with updated frequencies, stop_times and trips tables.

Details

A single trip described in a frequencies table may yield multiple trips after converting the GTFS. Let's say, for example, that the frequencies table describes a trip called "example_trip", that starts at 08:00 and stops at 09:00, with a 30 minutes headway.

In practice, with the default strategy = "exact", that means that one trip will depart at 08:00 and another at 08:30. As required by the GTFS specification, no trip departs at the end_time (09:00), which is when the headway changes or the service ceases. An entry whose start_time and end_time are equal yields a single trip, departing at that time. frequencies_to_stop_times() appends a "_<n>" suffix to the newly created trips to differentiate each one of them (e.g. in this case, the new trips, described in the trips and stop_times tables, would be called "example_trip_1" and "example_trip_2").

The times of each new trip are those of its template, shifted so that it departs from the first stop at the departure time. When the template has a dwell time at the first stop, a departure close to midnight may yield an arrival before 00:00:00, which is set to 00:00:00 instead.

Existing _secs columns in stop_times and frequencies (e.g. created with convert_time_to_seconds()) are used as-is, not recalculated from the time strings.

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
trip <- "CPTM L07-0"

# converts all trips listed in the frequencies table
converted_gtfs <- frequencies_to_stop_times(gtfs)

# converts only the specified trip_id
converted_gtfs <- frequencies_to_stop_times(gtfs, trip)

# how the specified trip_id was described in the frequencies table
gtfs$frequencies |>
  filter(trip_id == trip) |>
  head()

# the first row of each equivalent stop_times entry in the converted gtfs
converted_gtfs$stop_times |>
  filter(grepl(trip, trip_id)) |>
  distinct(trip_id, .keep_all = TRUE)
```

---

get_active_services *Get active services*

Description

Returns the services that are active on the given dates.

Usage

```
get_active_services(gtfs, date)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
date	A Date vector (including IDate) or a character vector of dates in the "YYYYMMDD" or "YYYY-MM-DD" formats.

Value

A data.table with the columns date (of class Date) and service_id, with one row for each service active on each of the given dates, sorted by date and then by service_id (in the C locale, so uppercase letters come before lowercase ones). Dates on which no service is active have no rows. service_id is never NA.

Details

A service is active on a date if its `calendar` entry spans the date and flags its weekday, unless the date is removed from it in `calendar_dates` (`exception_type` 2), or if the date is added to it in `calendar_dates` (`exception_type` 1). An addition wins over a removal on the same date, and either table may be missing. If the `trips` table has a `service_id` field, services not used by any trip are not returned, as in `get_dates()`. If no service is active, an empty table is returned without a warning.

See Also

`get_dates()`, `filter_by_date()`

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

get_active_services(gtfs, "20060703")

# several dates, also as Date objects. only the weekend service "WE" is
# returned, because the weekday service "WD" is not used by any trip
get_active_services(gtfs, as.Date(c("2006-07-01", "2006-07-05")))
```

---

get_calendar_overlap *Get calendar overlap*

Description

Returns the periods in which all the given GTFS feeds have service, or the number of trips each feed runs on each of its service days. Either result can also be plotted. Useful to pick a date on which several feeds can be analysed together (e.g. when routing on the networks of different agencies).

Usage

```
get_calendar_overlap(gtfs, resolution = "daily", plot = FALSE)
```

Arguments

`gtfs`	Either a character vector with the paths to GTFS `.zip` files or a list of GTFS objects, as created by `read_gtfs()` (a single GTFS object must be wrapped in a list).
`resolution`	A string. `"daily"` (the default) returns the number of trips of each feed on each of its service days, and `"periods"` the periods in which all feeds have service.
`plot`	A logical. Whether to return a plot of the result instead of the result itself (requires the {ggplot2} package). Defaults to `FALSE`.

Value

If `plot = FALSE` and `resolution = "periods"`, a `data.table` with one row per period of consecutive days on which all feeds have service, and the columns `start_date`, `end_date` (both included in the period) and `n_days`. It has no rows if there is no such day.

If `plot = FALSE` and `resolution = "daily"`, a `data.table` with one row per feed and service day, and the columns `feed`, `date`, `n_trips` (the number of trips that run on the day) and `overlap` (whether all feeds have service on the day).

If `plot = TRUE`, a `ggplot` object with dates on the x axis and one row per feed (top to bottom in the given order). With `resolution = "periods"`, it shows a bar spanning each feed's service days. With `resolution = "daily"`, it shows a box plot of when each feed's trips run: the box spans the days by which a quarter and three quarters of its trips have run, with a line on the day by which half have, and the whiskers reach its first and last service days. In both, the overlap periods are shaded.

Details

A feed has service on a day if at least one of its services runs on it, according to `calendar` and `calendar_dates` (either may be missing). A service both added and removed on a day in `calendar_dates` runs on it. If `trips` has a `service_id` field, services not used by any trip are ignored. `feed_info` dates are not used. A warning is raised for feeds without any service day, and a message when two or more feeds have no day with service in all of them.

In `n_trips`, a trip listed in `frequencies` counts once per departure, as in `frequencies_to_stop_times()` with `strategy = "exact"`, except that the departures of overlapping entries of the same trip are added up (identical entries count once). A trip with an invalid `frequencies` entry counts once, with a warning. Departures after midnight (times past `"24:00:00"`) count on the previous day. Without a `trips` table with a `service_id` field, `n_trips` is `NA` and the daily plot shows no box for the feed.

Feeds are named after their file names (without the extension) or list names; unnamed feeds are named `"feed_1"`, `"feed_2"`, etc., and duplicated names are made unique. Overlap periods are shaded only when two or more feeds are given.

See Also

`merge_gtfs()`, `filter_by_service_id()`

Examples

```
spo_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
poa_path <- system.file("extdata/poa_gtfs.zip", package = "gtfstools")

# feeds may be given as paths to their .zip files
daily_trips <- get_calendar_overlap(c(spo_path, poa_path))
head(daily_trips)

# or as a list of GTFS objects, whose names are used as feed names
feeds <- list(spo = read_gtfs(spo_path), poa = read_gtfs(poa_path))
get_calendar_overlap(feeds, resolution = "periods")

if (requireNamespace("ggplot2", quietly = TRUE)) {
```

```
  get_calendar_overlap(feeds, plot = TRUE)
}
```

`get_children_stops` *Get children stops recursively*

Description

Returns the (recursive) children stops of each specified `stop_id`. Recursive in this context means it returns all children's children (i.e. first children, then children's children, and then their children, and so on).

Usage

```
get_children_stops(gtfs, stop_id = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`stop_id`	A string vector including the `stop_id`s to have their children returned. If `NULL` (the default), the function returns the children of every `stop_id` in the GTFS.

Value

A `data.table` containing the `stop_id`s and their children' `stop_id`s. If a stop doesn't have a child, its correspondent `child_id` entry is marked as `""`.

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

children <- get_children_stops(gtfs)
head(children)

# use the stop_id argument to control which stops are analyzed
children <- get_children_stops(gtfs, stop_id = c("F12S", "F12N"))
children
```

get_dates *Get service dates*

Description

Returns the dates on which the GTFS object has service.

Usage

```
get_dates(gtfs, as_date = TRUE)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
as_date	A logical. Whether to return the dates as a Date vector (TRUE, the default) or as a character vector in the "YYYYMMDD" format used by GTFS (FALSE).

Value

A sorted vector of unique dates, either of class Date or a character vector, depending on as_date. It is empty, without a warning, if the GTFS object has no service date.

Details

The GTFS object has service on a date if at least one of its services runs on it. A service runs on the dates between the start_date and end_date of its calendar entry whose weekday is flagged, minus the dates removed from it in calendar_dates (exception_type 2), plus the dates added to it (exception_type 1). Either table may be missing. If the trips table has a service_id field, services not used by any trip are ignored, so each returned date keeps at least one trip when passed to filter_by_date(). feed_info dates are not used.

See Also

get_calendar_overlap(), filter_by_date()

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# the calendar covers all of July 2006, but only weekends (plus 2006-07-03
# and 2006-07-04, added by calendar_dates) are returned, because the
# weekday service "WD" is not used by any trip
get_dates(gtfs)

# dates may also be returned in the "YYYYMMDD" format used by GTFS
get_dates(gtfs, as_date = FALSE)
```

`get_dwell_time` *Get dwell time*

Description

Returns the dwell time of each specified `trip_id` at each specified `stop_id`, i.e. the time the vehicle stays at the stop, as specified in the `stop_times` table.

Usage

```
get_dwell_time(
  gtfs,
  trip_id = NULL,
  stop_id = NULL,
  unit = "s",
  from = NULL,
  to = NULL
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s to have their dwell times calculated. If `NULL` (the default), the function calculates the dwell times of every `trip_id` in the GTFS.
`stop_id`	A character vector including the `stop_id`s to have their dwell times calculated. If `NULL` (the default), the function calculates the dwell times at every `stop_id` in the GTFS.
`unit`	A string representing the time unit in which the dwell times are desired. One of `"s"` (seconds, the default), `"min"` (minutes), `"h"` (hours) or `"d"` (days).
`from`	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or after `from` are included. If `NULL` (the default), there is no lower limit.
`to`	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or before `to` are included. If `NULL` (the default), there is no upper limit.

Value

A `data.table` with the columns `trip_id`, `stop_id`, `stop_sequence` and `dwell_time`, with one row per visit of a specified trip to a specified stop (within the time of day, if `from` or `to` is given), ordered by `trip_id` and `stop_sequence`. `dwell_time` is numeric, in the given `unit`, and `NA` where a time is blank.

Details

The dwell time of a visit is its `departure_time` minus its `arrival_time` in `stop_times`. When both `trip_id` and `stop_id` are given, the visits of every specified trip to every specified stop are returned (not of pairs formed by their elements). The first and last stops of each trip are included. For trips listed in `frequencies`, one dwell time is returned per visit of the template, not per departure.

Blank times (`NA` dwell times) can be filled beforehand with `interpolate_stop_times()`. Negative dwell times (departure earlier than arrival) are returned as they are.

`from` and `to` are compared with the `arrival_time` of each visit as written in `stop_times` (template times, for trips listed in `frequencies`), so use times past `"24:00:00"` for arrivals after midnight. Visits with a blank arrival time are never included when `from` or `to` is given.

`stop_ids` are matched against the `stop_ids` in `stop_times`, which are never stations: to get the dwell times at a station, pass its children that appear in `stop_times` (see `get_children_stops()`). Existing `_secs` columns in `stop_times` are not used; times are read from the time strings.

See Also

`set_dwell_time()`, `set_min_dwell_time()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

dwell_times <- get_dwell_time(gtfs)
head(dwell_times)

# use the trip_id and stop_id arguments to control which trips and stops
# are analyzed
get_dwell_time(gtfs, trip_id = "CPTM L07-0")
get_dwell_time(gtfs, stop_id = "18960")
get_dwell_time(
  gtfs,
  trip_id = c("CPTM L08-0", "CPTM L09-0"),
  stop_id = "18960"
)

# blank times result in NA dwell times. use the unit argument to control in
# which unit the dwell times are calculated
ggl_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
ggl_gtfs <- read_gtfs(ggl_path)
get_dwell_time(ggl_gtfs, trip_id = "AWE1", unit = "min")
```

---

get_parent_station *Get parent stations recursively*

Description

Returns the (recursive) parent stations of each specified stop_id. Recursive in this context means it returns all parents' parents (i.e. first parents, then parents' parents, and then their parents, and so on).

Usage

```
get_parent_station(gtfs, stop_id = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A string vector including the stop_ids to have their parents returned. If NULL (the default), the function returns the parents of every stop_id in the GTFS.

Value

A data.table containing the stop_ids and their parent_stations. If a stop doesn't have a parent, its correspondent parent_station entry is marked as "".

See Also

get_children_stops()

Examples

```
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

parents <- get_parent_station(gtfs)
head(parents)

# use the stop_id argument to control which stops are analyzed
parents <- get_parent_station(gtfs, c("B1", "B2"))
parents
```

---

get_route_frequency *Get route frequency*

Description

Returns the number of departures and the mean headway of each specified route_id within a given time of day, by service_id (and by direction_id, if the field is present in the trips table).

Usage

```
get_route_frequency(gtfs, route_id = NULL, from, to)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
route_id	A character vector including the route_ids to have their frequencies calculated. If NULL (the default), the function calculates the frequency of every route_id in the GTFS. route_ids not listed in trips are ignored, with a warning.
from	A string. The starting point of the time of day, in the "HH:MM:SS" format.
to	A string. The ending point of the time of day, in the "HH:MM:SS" format. Must be later than from.

Value

A data.table with the columns route_id, direction_id (only if present in the trips table), service_id, departures (the number of departures within the time of day, including each departure generated from the frequencies table) and mean_headway (the mean headway within the time of day, in minutes). Routes with no departures within the time of day are not included.

Details

Each departure of a trip is counted once, however many stops it visits within the time of day (unlike get_stop_frequency() and tidytransit::get_route_frequency(), which count stop departures). Scheduled trips depart at their earliest departure_time in stop_times (blank times are ignored); trips without any departure time are not counted. Trips listed in frequencies depart every headway_secs from start_time until (but not including) end_time, as in frequencies_to_stop_times(), even if they are not listed in stop_times; their stop_times entries are templates and are not used. An error is raised if frequencies has invalid entries for the specified routes.

A departure is counted when it happens at or after from and before to (unlike in filter_by_time_of_day(), which also keeps entries at to). Departures after midnight are listed with times past "24:00:00", so use a time of day such as "24:00:00"-"26:00:00" to count them.

The mean headway is the length of the time of day divided by the number of departures, so it overstates the actual headway when service starts or ends within the time of day. Without a direction_id field in trips, the departures of both directions are pooled; trips with an NA direction_id are grouped together.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

route_frequency <- get_route_frequency(
  gtfs,
  from = "07:00:00",
  to = "09:00:00"
)
head(route_frequency)

route_ids <- c("CPTM L07", "2002-10")
route_frequency <- get_route_frequency(
```

```
  gtfs,
  route_id = route_ids,
  from = "07:00:00",
  to = "09:00:00"
)
route_frequency
```

`get_route_geometry` *Get route geometry*

Description

Returns the distinct geometries of the trips of each specified `route_id` (its "variants", e.g. each direction and each short-turn service), with the `trip_id` that identifies each of them and the number of trips that follow them.

Usage

```
get_route_geometry(gtfs, route_id = NULL, method = "shapes", crs = 4326)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`route_id`	A character vector including the `route_id`s to have their geometries generated. If `NULL` (the default), the function generates geometries for every `route_id` in the GTFS. `route_id`s not listed in `trips` are ignored, with a warning.
`method`	A string, either `"shapes"` (the default) or `"euclidean"`, specifying how the geometry of each trip is generated. Please see `get_trip_geometry()` for details. If the GTFS object doesn't have a `shapes` table, or if its `trips` table doesn't have a `shape_id` column, `"euclidean"` is used instead, with a warning.
`crs`	The CRS of the resulting object, either as an EPSG code or as an `crs` object. Defaults to 4326 (WGS 84).

Value

A `LINESTRING sf` with one row per route and distinct trip geometry, with the columns `route_id`, `trip_pattern` (the first `trip_id`, in alphabetical order, of the route's trips with that geometry, which identifies it), `n_trips` (the number of trips of the route with that geometry) and `geometry`. Rows are ordered by `route_id` (with `NA` last) and, within each route, by decreasing `n_trips`.

Details

The geometry of each trip is generated as in `get_trip_geometry()`, so it goes from the trip's first to its last stop, either along its shape or as straight lines between its stops, and the same limitations and warnings apply. The trips of a route whose geometries are identical are then merged into a

single row. Trips that visit stops at different points of the same shape thus result in different rows, even if their geometries overlap.

Only trips listed both in `trips` and in `stop_times` are considered, so routes without any trip listed in `stop_times` are not included in the output. When a `trip_id` is listed more than once in `trips`, only its first entry is considered. With `route_id = NULL`, trips with a missing `route_id` are grouped together, under an `NA route_id`. Trips without any stop with coordinates (and, with `method = "shapes"`, trips that can't be located along a shape) get an empty geometry, as in `get_trip_geometry()`, and these trips are counted in an empty-geometry row of their route.

See Also

`get_trip_geometry()`, `get_shapes_geometry()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

route_geometry <- get_route_geometry(gtfs)
head(route_geometry)

route_geometry <- get_route_geometry(gtfs, route_id = "CPTM L07")
route_geometry
plot(route_geometry["n_trips"])
```

---

get_route_speed *Get route speed*

Description

Returns summary statistics of the average speeds of the trips of each specified `route_id`.

Usage

```
get_route_speed(
  gtfs,
  route_id = NULL,
  method = "shapes",
  unit = "km/h",
  from = NULL,
  to = NULL
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
route_id	A character vector including the route_ids to have their speeds summarised. If NULL (the default), the function summarises the speeds of every route_id in the GTFS. route_ids not listed in trips are ignored, with a warning.
method	A string, either "shapes" (the default) or "euclidean". "shapes" measures lengths along the trip's shape, described in the shapes table, while "euclidean" measures the straight-line (great-circle) distances between consecutive stops. If the GTFS object doesn't have a shapes table, or if its trips table doesn't have a shape_id column, "euclidean" is used instead, with a warning.
unit	A string representing the unit in which the speeds are desired. One of "km/h" (the default), "m/s" or "mph" (miles per hour).
from	A string, in the "HH:MM:SS" format. Only trips that depart from their first stop at or after from are included. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips that depart from their first stop at or before to are included. If NULL (the default), there is no upper limit.

Value

A data.table with the route_id and the minimum (min_speed), mean (mean_speed), median (median_speed), maximum (max_speed) and standard deviation (sd_speed) of the speeds of its trips, ordered by route_id.

Details

The speed of each trip is calculated with get_trip_speed() (with by = "trip"), whose details also apply here. Each trip is linked to its route by the trips table. mean_speed is the unweighted mean of the trip speeds, not the total length of the trips divided by their total duration. Trips listed in frequencies count once, regardless of their headways.

Trips with NA speeds are ignored. Routes whose trips all have NA speeds get NA statistics, and sd_speed is NA for routes with a single trip with a known speed. Routes without any trip in stop_times (or, if from or to is given, without any trip departing within the time of day) are not returned.

See Also

get_trip_speed(), set_route_speed(), set_trip_speed()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

route_speed <- get_route_speed(gtfs)
head(route_speed)

route_ids <- c("CPTM L07", "2002-10")
```

```
route_speed <- get_route_speed(gtfs, route_ids)
route_speed

route_speed <- get_route_speed(gtfs, route_ids, method = "euclidean")
route_speed

route_speed <- get_route_speed(gtfs, route_ids, unit = "m/s")
route_speed
```

`get_route_timetable` *Get route timetable*

Description

Returns the timetable of the given routes on the given dates: one row for each stop time of their trips on each date on which the trip runs.

Usage

```
get_route_timetable(gtfs, route_id = NULL, date = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`route_id`	A character vector including the `route_id`s whose timetable should be built. If `NULL` (the default), the timetable of all routes is built.
`date`	A `Date` vector (including `IDate`) or a character vector of dates in the `"YYYYMMDD"` or `"YYYY-MM-DD"` formats. If `NULL` (the default), all the dates on which the GTFS object has service (as returned by `get_dates()`) are used, which may result in a very large table.

Value

A `data.table` with the column `date` (of class `Date`), followed by all the columns of `trips` and all the columns of `stop_times` (`trip_id` appears only once). It is sorted by `date` and `route_id`, and then the trips of each route are sorted by their first departure time (and `trip_id`), each with its stop times sorted by `stop_sequence`. If the given routes have no trips on the given dates, an empty table with the same columns is returned, without a warning.

Details

A trip runs on a date if its service is active on it, as in `get_active_services()`, so dates outside the period covered by the GTFS object are skipped. Trips listed in `stop_times` but not in `trips` are not included. Departure times are sorted as times, not as strings, and trips without any departure time come last in their route. Duplicated rows in `trips` and `stop_times` are kept.

Without calendar and calendar_dates tables, no trip runs, and an empty table is returned. Columns other than trip_id present in both trips and stop_times (not defined by the GTFS reference) appear twice, the one from stop_times prefixed with i., and the ones from trips are used to sort the timetable.

date is the service date: stop times past 24:00:00 happen on the following calendar day.

The frequencies table is not used: frequency-based trips appear with the times of their template trips, as listed in stop_times. To include all their departures, convert them with frequencies_to_stop_times() first.

See Also

get_stop_timetable(), get_active_services(), get_dates()

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

timetable <- get_route_timetable(gtfs, "1922_3", "20210104")
head(timetable)

# several routes and dates, also as Date objects
timetable <- get_route_timetable(
  gtfs,
  route_id = c("1922_3", "1921_3"),
  date = as.Date(c("2021-01-04", "2021-01-09"))
)
table(timetable$date, timetable$route_id)
```

---

get_shape_length *Get shape length*

Description

Returns the length of each specified shape_id, from its first to its last point.

Usage

```
get_shape_length(gtfs, shape_id = NULL, unit = "km")
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
shape_id	A character vector including the shape_ids to have their length calculated. If NULL (the default), the function calculates the length of every shape_id in the GTFS.
unit	A string representing the unit in which lengths are desired. Either "km" (the default) or "m".

Value

A data.table with the shape_id and the length of each shape.

Details

The points of each shape are sorted by shape_pt_sequence before the length is calculated. Lengths are great-circle distances calculated with the haversine formula, on a sphere with the same radius used by {s2} (6,371,010 meters), regardless of whether sf::sf_use_s2() is enabled. Shapes with any point with missing coordinates have NA lengths.

A trip usually travels only part of its shape, from its first to its last stop. Use get_trip_length() to calculate the length of the trips.

See Also

get_trip_length()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

shape_length <- get_shape_length(gtfs)
head(shape_length)

shape_length <- get_shape_length(gtfs, c("17846", "68962"), unit = "m")
shape_length
```

---

get_shapes_geometry *Get the* shapes *table with spatial geometry*

Description

Converts the shapes table to a LINESTRING sf object.

Usage

```
get_shapes_geometry(gtfs, shape_id = NULL, crs = 4326)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
shape_id	A character vector including the shape_ids to be converted. If NULL (the default), all shapes are converted.
crs	The CRS of the resulting object, either as an EPSG code or as an crs object. Defaults to 4326 (WGS 84).

Value

A LINESTRING sf object.

Examples

```
# read gtfs
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

shapes_sf <- get_shapes_geometry(gtfs)
head(shapes_sf)

shapes_sf <- get_shapes_geometry(gtfs, shape_id = "17846")
shapes_sf
```

---

get_start_and_end_times

Get start and end times

Description

Returns the earliest departure time and the latest arrival time listed in the stop_times table, optionally restricted to the given trips and/or to the trips active on the given dates.

Usage

```
get_start_and_end_times(gtfs, trip_id = NULL, date = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to consider. If NULL (the default), all trips are considered.
date	A Date vector (including IDate) or a character vector of dates in the "YYYYMMDD" or "YYYY-MM-DD" formats. If given, only trips whose services are active on any of these dates are considered. If NULL (the default), trips are not restricted by date.

Value

A named character vector of length 2, with the elements start_time (the earliest departure_time) and end_time (the latest arrival_time), in the "HH:MM:SS" format. Times past midnight keep hours of 24 and above. If no stop time is considered, or if all their times are empty, both elements are NA, without a warning.

Details

Times are compared as seconds, not as strings, so `"5:00:00"` comes before `"10:00:00"` (and is returned as `"05:00:00"`). Empty times are ignored, and malformed ones are ignored with a warning.

When both `trip_id` and `date` are given, only the given trips that are active on the given dates are considered. The services active on each date are found with `get_active_services()`. Trips listed in `stop_times` but not in `trips` are not considered when `date` is given, and if the GTFS object has no `calendar` nor `calendar_dates` tables, no trip is active.

The `frequencies` table is not used: the times of frequency-based trips are taken from their template trips as they are listed in `stop_times`. To consider all their departures, convert them with `frequencies_to_stop_times()` first.

See Also

`get_active_services()`, `get_trip_duration()`

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

get_start_and_end_times(gtfs)

get_start_and_end_times(gtfs, trip_id = c("146389748", "146389727"))

# restricted to the trips active on a given date
data_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

get_start_and_end_times(gtfs, date = "20060701")
```

`get_stop_distances` *Get distances between stops*

Description

Returns the great-circle distance between each pair of stops visited by the given trips and/or routes.

Usage

```
get_stop_distances(gtfs, trip_id = NULL, route_id = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s whose stops should be considered. If `NULL` (the default), all trips are considered. An empty vector selects no trip.
`route_id`	A character vector including the `route_id`s whose stops should be considered. If `NULL` (the default), all routes are considered. An empty vector selects no route.

Value

A `data.table` with the columns `from_stop_id`, `to_stop_id` and `distance` (in meters), with one row for each pair of distinct stops, sorted by `from_stop_id` and `to_stop_id`. It is empty if fewer than two stops are selected.

Details

The stops are selected as in `get_stops()`: only the stops listed in the `stop_times` entries of the selected trips are considered.

Distances are great-circle distances ("as the crow flies") between the stops' coordinates, not distances along the trips' shapes or the street network.

Each pair of stops appears only once, with the `stop_id` that comes first in the C locale (the order used by `data.table`) as `from_stop_id`. If a `stop_id` is listed more than once in `stops`, only its first entry is used. Stops without coordinates have `NA` distances.

The size of the result grows with the square of the number of stops: about 1 GB of memory is used for 5,500 stops, and about 3 GB for 10,000 stops. A warning is raised when more than 1 GB is needed, so on large feeds please select only the trips or routes of interest. More than 65,536 stops can't be used at once, raising an error.

See Also

`get_stops()`, `get_trip_length()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# distances between the stops visited by a trip
distances <- get_stop_distances(gtfs, trip_id = "CPTM L07-0")
head(distances)

# distances between the stops visited by the trips of a route
distances <- get_stop_distances(gtfs, route_id = "CPTM L07")
head(distances)
```

---

get_stop_frequency *Get stop frequency*

Description

Returns the number of departures and the mean headway at each specified `stop_id` within a given time of day, by `service_id`. Optionally, the departures can also be counted by `route_id` (and by `direction_id`, if the field is present in the `trips` table).

Usage

```
get_stop_frequency(gtfs, stop_id = NULL, from, to, by_route = FALSE)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A character vector including the stop_ids to have their frequencies calculated. If NULL (the default), the function calculates the frequency of every stop_id in the GTFS.
from	A string. The starting point of the time of day, in the "HH:MM:SS" format.
to	A string. The ending point of the time of day, in the "HH:MM:SS" format. Must be later than from.
by_route	A logical. Whether the departures should be counted by route_id (and by direction_id, if present in the trips table) at each stop. Defaults to FALSE, in which case the departures of every route that serves a stop are pooled together.

Value

A data.table with the columns stop_id, route_id and direction_id (only if by_route is TRUE, the latter only if present in the trips table), service_id, departures (the number of departures from the stop within the time of day, including each departure generated from the frequencies table) and mean_headway (the mean headway within the time of day, in minutes). Stops with no departures within the time of day are not included.

Details

The function counts stop departures: each stop_times entry with a departure time is a departure from its stop, except for the entry of the last stop of each trip (the one with the highest stop_sequence), from which vehicles don't depart. Entries with blank departure times (such as those of stops that are not timepoints) are not counted, so their times should be interpolated beforehand to count them. A stop visited twice by the same trip is counted twice. Its results are therefore not comparable to those of get_route_frequency(), which counts trips.

The stop_times entries of trips listed in the frequencies table are just templates: these trips depart every headway_secs from start_time until (but not including) end_time, and each of these departures visits the template's stops with the times shifted accordingly, as in frequencies_to_stop_times(). Trips without any departure time in stop_times don't generate departures. An error is raised if the frequencies table has invalid entries for the trips that serve the specified stops, even if they don't have any departure time.

The stops are those listed in stop_times, which are never stations. Use get_children_stops() to get the stops of a station and sum their departures.

A departure is counted when it happens at or after from and before to, so that consecutive time windows (e.g. "06:00:00"-"07:00:00" and "07:00:00"-"08:00:00") don't count the same departure twice. Please note that this differs from filter_by_time_of_day(), which keeps entries whose times are equal to to. Departures after midnight are listed with times greater than "24:00:00" and belong to the previous service day, so a time of day such as "24:00:00"-"26:00:00" should be used to account for them.

The mean headway is the length of the time of day divided by the number of departures. It is an average over the whole time of day, so it is larger than the actual headway when service starts or ends within it. Trips with an NA direction_id are grouped together.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

stop_frequency <- get_stop_frequency(
  gtfs,
  from = "07:00:00",
  to = "09:00:00"
)
head(stop_frequency)

stop_ids <- c("18848", "18960")
stop_frequency <- get_stop_frequency(
  gtfs,
  stop_id = stop_ids,
  from = "07:00:00",
  to = "09:00:00",
  by_route = TRUE
)
stop_frequency
```

---

get_stop_times_patterns

Get stop times patterns

Description

Identifies spatial and spatiotemporal patterns within the `stop_times` table. Please see the details to understand what a "pattern" means in each of these cases.

Usage

```
get_stop_times_patterns(gtfs, trip_id = NULL, type = "spatial", sort_sequence)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s to have their `stop_times` entries analyzed. If `NULL` (the default), the function analyses the pattern of every `trip_id` in the GTFS.
`type`	A string specifying the type of patterns to be analyzed. Either `"spatial"` (the default) or "spatiotemporal".
`sort_sequence`	Deprecated; do not supply. Timetables are now always sorted by `stop_sequence`. Passing `TRUE` raises a warning, and passing `FALSE` raises an error.

Value

A `data.table` associating each `trip_id` to a `pattern_id`.

Details

Two trips are assigned to the same spatial `pattern_id` if they travel along the same sequence of stops. They are assigned to the same spatiotemporal `pattern_id`, on the other hand, if they travel along the same sequence of stops and they take the same time between stops. Please note that, in such case, only the time between stops is taken into account, and the time that the trip started is ignored (e.g. if two trips depart from stop A and follow the same sequence of stops to arrive at stop B, taking both 1 hour to do so, their spatiotemporal pattern will be considered the same, even if one departed at 6 am and another at 7 am). Please also note that only the order given by `stop_sequence` matters, not its values - e.g. if trip X lists stops A followed by stop B with `stop_sequences` 1 and 2, and trip Y lists stops A followed by stop B with `stop_sequences` 1 and 3, they are assigned to the same `pattern_id`.

Existing `_secs` columns in `stop_times` (e.g. created with `convert_time_to_seconds()`) are used as-is, not recalculated from the time strings.

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

patterns <- get_stop_times_patterns(gtfs)
head(patterns)

# use the trip_id argument to control which trips are analyzed
patterns <- get_stop_times_patterns(
  gtfs,
  trip_id = c("143765658", "143765659", "143765660")
)
patterns

# use the type argument to control the type of pattern analyzed
patterns <- get_stop_times_patterns(
  gtfs,
  trip_id = c("143765658", "143765659", "143765660"),
  type = "spatiotemporal"
)
patterns
```

`get_stop_timetable` *Get stop timetable*

Description

Returns the timetable of the given stops on the given dates: one row for each stop time at these stops on each date on which its trip runs.

Usage

```
get_stop_timetable(gtfs, stop_id = NULL, date = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A character vector including the stop_ids whose timetable should be built. If NULL (the default), the timetable of all stops is built.
date	A Date vector (including IDate) or a character vector of dates in the "YYYYMMDD" or "YYYY-MM-DD" formats. If NULL (the default), all the dates on which the GTFS object has service (as returned by get_dates()) are used, which may result in a very large table.

Value

A data.table with the column date (of class Date), followed by all the columns of trips and all the columns of stop_times (trip_id appears only once). It is sorted by date, stop_id, departure time and trip_id. If no trip stops at the given stops on the given dates, an empty table with the same columns is returned, without a warning.

Details

A trip runs on a date if its service is active on it, as in get_active_services(), so dates outside the period covered by the GTFS object are skipped. Trips listed in stop_times but not in trips are not included. Departure times are sorted as times, not as strings, and stop times with an empty departure_time come last at each stop. Duplicated rows in trips and stop_times are kept.

Without calendar and calendar_dates tables, no trip runs, and an empty table is returned. Columns other than trip_id present in both trips and stop_times (not defined by the GTFS reference) appear twice, the one from stop_times prefixed with i., and the ones from trips are used to sort the timetable.

date is the service date: stop times past 24:00:00 happen on the following calendar day.

Only the exact given stops are included: to include the stops of a station, pass them with get_children_stops(). The frequencies table is not used: frequency-based trips appear with the times of their template trips, as listed in stop_times. To include all their departures, convert them with frequencies_to_stop_times() first.

See Also

get_route_timetable(), get_active_services(), get_dates()

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

timetable <- get_stop_timetable(gtfs, "100000720101", "20210104")
head(timetable)

# several dates, also as Date objects
timetable <- get_stop_timetable(
  gtfs,
  stop_id = "100000720101",
  date = as.Date(c("2021-01-04", "2021-01-09"))
)
table(timetable$date)
```

`get_stops` *Get stops*

Description

Returns the stops visited by the given trips and/or routes.

Usage

```
get_stops(gtfs, trip_id = NULL, route_id = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s whose stops should be returned. If `NULL` (the default), all trips are considered. An empty vector selects no trip.
`route_id`	A character vector including the `route_id`s whose stops should be returned. If `NULL` (the default), all routes are considered. An empty vector selects no route.

Value

A `data.table` with the entries of the `stops` table that are visited by the selected trips, with all its columns and in its original order.

Details

A trip is selected if it is listed in `trip_id` (when given) and belongs to one of the routes listed in `route_id` (when given), so when both are given only the trips that satisfy both are selected. When `route_id` is given, trips not listed in the `trips` table can't be matched to a route and are not selected. Ids not found in the feed raise a warning.

Only the stops listed in the `stop_times` entries of the selected trips are returned, so their parent stations, entrances, etc. are not. Please use `get_parent_station()` and `get_children_stops()` to get these. Stops listed in `stop_times` but not in `stops` are not returned.

See Also

get_parent_station(), get_children_stops(), filter_by_trip_id(), filter_by_route_id()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# stops visited by a trip
stops <- get_stops(gtfs, trip_id = "CPTM L07-0")
head(stops)

# stops visited by the trips of a route
stops <- get_stops(gtfs, route_id = "CPTM L07")
head(stops)

# stops visited by any trip
stops <- get_stops(gtfs)
nrow(stops)
```

---

get_stops_geometry *Get the* stops *table with spatial geometry*

Description

Converts the stops table to a POINT sf object.

Usage

```
get_stops_geometry(gtfs, stop_id = NULL, crs = 4326)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
stop_id	A character vector including the stop_ids to be converted. If NULL (the default), all stops are converted.
crs	The CRS of the resulting object, either as an EPSG code or as an crs object. Defaults to 4326 (WGS 84).

Value

A POINT sf object.

Examples

```
# read gtfs
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

stops_sf <- get_stops_geometry(gtfs)
head(stops_sf)

stops_sf <- get_stops_geometry(gtfs, stop_id = "18848")
stops_sf
```

`get_summary` *Get a summary of a GTFS feed*

Description

Runs a set of diagnostics on a GTFS feed (optionally restricted to some dates, routes, modes, agencies and a time of day), writes them to a report with tables and figures, renders the report to HTML and opens it in the browser (in interactive sessions). The diagnostics are also returned as a list of `data.tables`. This function does not validate or look for quality issues in the GTFS feed. To do that, use `validate_gtfs()`.

Usage

```
get_summary(
  gtfs,
  date = NULL,
  route_id = NULL,
  route_type = NULL,
  agency_id = NULL,
  from = NULL,
  to = NULL,
  output_path = NULL,
  html_preview = TRUE,
  by_route_type = FALSE,
  unit = "km/h"
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`, or the path to a GTFS file.
`date`	A `Date` vector (including `IDate`) or a character vector of dates in the `"YYYYMMDD"` or `"YYYY-MM-DD"` formats. If given, only the services that run on these dates are described. If `NULL` (the default), the whole period covered by the feed is described.

route_id	A character vector. If given, only these routes are described. If NULL (the default), all routes are described.
route_type	An integer vector of route_types (see filter_by_route_type()) or, unlike in other functions, a character vector of mode names ("tram", "subway", "rail", "bus", "ferry", "cable_tram", "aerial_lift", "funicular", "trolleybus" and "monorail", which correspond to the basic route_types). If given, only routes of these modes are described. If NULL (the default), all modes are described.
agency_id	A character vector of agency_ids. If given, only the services of these agencies are described. If NULL (the default), all agencies are described.
from, to	Strings in the "HH:MM:SS" format. If given (both must be), only the trips that run within this time of day are described. If NULL (the default), the whole day is described.
output_path	A string. The directory in which the report is saved (created if needed). If NULL (the default), a gtfstools directory inside tempdir() is used. Existing reports in this directory are overwritten.
html_preview	A logical. Whether to open the HTML report in the browser, in interactive sessions (defaults to TRUE).
by_route_type	A logical. Whether to break down the totals, speeds, distances and frequencies by transport mode (route_type), in addition to the whole selection (defaults to FALSE).
unit	A string. Either "km/h" (the default), to report distances in km and speeds in km/h, or "mph", to report them in miles and miles per hour.

Value

A named list of data.tables (invisibly, if the report was opened in the browser):

- files: the tables in the feed (file) and their number of rows (n_rows), before any filtering;
- agency: the agency_id, agency_name, agency_timezone, agency_url and agency_lang columns (those present in the feed) of the agencies that run the selected trips (all agencies, if the selected routes have no agency_id);
- services: a single row summarising the calendar of the selected services, with the columns start_date and end_date (the first and last days with service), n_days (the number of days between them, both included), n_service_days (the number of days with service), n_services (the number of service_ids), mean_trips_per_day (over the days with service), busiest_date and max_trips_per_day (the day with the most trips and its number of trips) and feed_start_date and feed_end_date (from feed_info, NA if absent);
- totals: the columns indicator, value and unit, with the number of agencies, routes, trips and stops, the total and mean length of the shapes and of the trips (in km), the vehicle-kilometres travelled (vkt_total, in km) and their mean per day (vkt_daily_mean, in km/day), the total duration of the trips (in hours), the mean duration of the trips (in minutes), the mean trip speed (in km/h) and the mean route headway (in minutes). The means of lengths, durations and speeds are taken over the shapes and trips whose value is known, and vkt_daily_mean over the days with counted departures (see Details). If by_route_type = TRUE, value is replaced by one column per mode followed by the column all (codes of extended route_types, such as "700", are non-syntactic names, so they need backticks);

- speed, distance and frequency: summary statistics (n, min, q25, median, mean, q75, max and unit) of the trip speeds (in km/h), of the trip lengths (in km) and of the route headways (in minutes). frequency also has the columns date, from and to, with the day and the time of day in which the headways were measured. If by_route_type = TRUE, these tables start with a mode column and have one row per mode followed by the row all.

With unit = "mph", distances are in miles (mi and mi/day) and speeds in miles per hour (mph), and the rows vkt_total and vkt_daily_mean of totals are named vmt_total and vmt_daily_mean (vehicle-miles travelled).

The report is saved as gtfs_summary.md and gtfs_summary.html (together with its figures) in output_path.

Details

The report requires the {ggplot2}, {knitr} and {rmarkdown} packages and Pandoc.

The time-of-day filter keeps whole trips (as filter_by_time_of_day() with full_trips = TRUE), so trip speeds and lengths refer to whole trips.

Trip lengths, durations and speeds are calculated with get_trip_length(), get_trip_duration() and get_trip_speed(), counting each trip once, however many days it runs on or departures frequencies lists for it. Trips with a zero or negative duration count in the durations but have no speed. Shape lengths count each shape used by the selected trips once, and the number of stops counts the stops visited by the selected trips.

The VKT are calculated as in get_vkt() (or get_vmt(), with unit = "mph"): the length of each selected trip counts once per departure on every counted day on which its service runs. The counted days are the given dates or, if date is NULL, every day on which the selected services run; vkt_daily_mean is the mean over the counted days on which at least one counted departure runs. With from and to, only the departures at or after from and at or before to are counted, as listed before the feed is filtered by time of day, so selected trips that depart before from add nothing. vkt_total and vkt_daily_mean are NA if no selected trip departs on the counted days (within the time of day, if given). An error is raised if frequencies has invalid entries for selected trips that run on the counted days.

services and the calendar chart describe the whole calendar of the selected services: with date, the services running on these dates are kept whole (as in filter_by_date()), so they may cover other days too. The trips per day are counted as in get_calendar_overlap().

Route headways are calculated with get_route_frequency(), by route (and direction_id, if present in trips), on a single day: the one with the most trips among the given dates or in the whole feed (the earliest, on ties). They are measured within the given time of day (excluding to, as in get_route_frequency(), unlike the other indicators) or, without from and to, from the first to the last departure of that day, rounded out to whole hours. Trips of the previous service day running after midnight are not counted. With from and to, headways and trips per day are counted on the time-filtered feed, whose frequencies entries crossing from start at from (unless exact_times is 1), so they may differ from get_route_frequency() and get_calendar_overlap() on the full feed.

With by_route_type = TRUE, modes are named as in the route_type argument, extended route_types by their code and trips of unknown route_type as "unknown", ordered by route_type with "unknown" last. Each mode is described as if selected alone (except that all headways share the same day and time of day), so stops, shapes and agencies shared by modes count in each. Only

the numbers of routes and trips, the total trip lengths and durations and `vkt_total` (or `vmt_total`) add up to `all`. Modes whose routes have blank `agency_id`s count all the agencies of the selection. Trips listed in `stop_times` but not in `trips` count only in `all` (and not in the VKT, as they have no service).

Examples

```
if (
  requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("knitr", quietly = TRUE) &&
    requireNamespace("rmarkdown", quietly = TRUE) &&
    rmarkdown::pandoc_available()
) {
  data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

  feed_summary <- get_summary(
    data_path,
    output_path = tempdir(),
    html_preview = FALSE
  )
  feed_summary$totals


  feed_summary <- get_summary(
    data_path,
    route_type = "bus",
    from = "07:00:00",
    to = "09:00:00",
    output_path = tempdir(),
    html_preview = FALSE
  )
  feed_summary$frequency

  feed_summary <- get_summary(
    data_path,
    output_path = tempdir(),
    html_preview = FALSE,
    by_route_type = TRUE
  )
  feed_summary$totals

}
```

`get_trip_duration` *Get trip duration*

Description

Returns the duration of each specified `trip_id`.

Usage

```
get_trip_duration(gtfs, trip_id = NULL, unit = "min", from = NULL, to = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A string vector including the trip_ids to have their duration calculated. If NULL (the default) the function calculates the duration of every trip_id in the GTFS.
unit	A string representing the time unit in which the duration is desired. One of "s" (seconds), "min" (minutes, the default), "h" (hours) or "d" (days).
from	A string, in the "HH:MM:SS" format. Only trips that depart from their first stop at or after from are included. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips that depart from their first stop at or before to are included. If NULL (the default), there is no upper limit.

Value

A data.table containing the duration of each specified trip.

Details

The duration of a trip is defined as the time difference between its last arrival time and its first departure time, as specified in the stop_times table. Trips without any arrival time or without any departure time get a NA duration, with a warning.

Existing _secs columns in stop_times (e.g. created with convert_time_to_seconds()) are used as-is, not recalculated from the time strings.

from and to are compared with the departure time of the first stop of each trip, in stop_sequence order (which must then be an integer column). The time is read as the durations are: from departure_time_secs, if present, or else from departure_time, so use times past "24:00:00" for departures after midnight. For trips listed in frequencies, these are template times, not the times of each departure. Trips whose first departure is blank are never included when from or to is given.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

trip_duration <- get_trip_duration(gtfs)
head(trip_duration)

trip_ids <- c("CPTM L07-0", "2002-10-0")
trip_duration <- get_trip_duration(gtfs, trip_id = trip_ids)
trip_duration

trip_duration <- get_trip_duration(gtfs, trip_id = trip_ids, unit = "h")
trip_duration
```

`get_trip_geometry` *Get trip geometry*

Description

Returns the geometry of each specified `trip_id`, from its first to its last stop, either along the trip's shape or as straight lines between stops.

Usage

```
get_trip_geometry(
  gtfs,
  trip_id = NULL,
  method = "shapes",
  crs = 4326,
  sort_sequence,
  file = NULL
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s to have their geometries generated. If `NULL` (the default), the function generates geometries for every `trip_id` in the GTFS.
`method`	A string, either `"shapes"` (the default) or `"euclidean"`. `"shapes"` returns the part of the trip's shape, described in the `shapes` table, between its first and last stops, while `"euclidean"` links the consecutive stops of the trip along straight lines. If the GTFS object doesn't have a `shapes` table, or if its `trips` table doesn't have a `shape_id` column, `"euclidean"` is used instead, with a warning.
`crs`	The CRS of the resulting object, either as an EPSG code or as an `crs` object. Defaults to 4326 (WGS 84).
`sort_sequence`	Deprecated; do not supply. Timetables and shapes are now always sorted by `stop_sequence` and `shape_pt_sequence`. Passing `TRUE` raises a warning, and passing `FALSE` raises an error.
`file`	Deprecated. Use `method` instead (`file = "stop_times"` corresponds to `method = "euclidean"`).

Value

A `LINESTRING sf` with the `trip_id` and the `geometry` of each trip listed in `stop_times`.

Details

With `method = "shapes"`, the first and last stops of each trip are projected onto its shape, as in `get_trip_length()`, and the geometry is the part of the shape between them. Thus, its length matches the length returned by `get_trip_length()` (when this length is not `NA`, see below), and the limitations described there apply. Trips whose stops are all located at the same point of the shape get a zero-length geometry. Use `get_shapes_geometry()` for the geometry of entire shapes.

With `method = "euclidean"`, the geometry links the consecutive stops of each trip along straight lines (stops' coordinates are retrieved from the `stops` table), so its resolution tends to be much lower than that of the geometry generated from the shapes. Trips with a single stop get a single-point geometry.

Both methods ignore stops with missing coordinates, or not listed in `stops` (for which `get_trip_length()` returns `NA` lengths). Trips without any stop with coordinates get an empty geometry, as do, with `method = "shapes"`, trips with fewer than two stops with coordinates or not linked to a shape with at least two distinct points (the latter with a warning). Geometries along shapes that cross the antimeridian are not handled correctly.

See Also

`get_trip_length()`, `get_shapes_geometry()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

# geometry along the shape from the first to the last stop of each trip
trip_geometry <- get_trip_geometry(gtfs)
head(trip_geometry)

# straight lines between consecutive stops
trip_ids <- c("CPTM L07-0", "2002-10-0")
straight_geometry <- get_trip_geometry(
  gtfs,
  trip_id = trip_ids,
  method = "euclidean"
)
straight_geometry
plot(straight_geometry["trip_id"])
```

---

get_trip_length *Get trip length*

Description

Returns the length of each specified `trip_id`, measured either from the first to the last stop of the trip or between each pair of consecutive stops. Lengths can be measured along the trip's shape or as straight lines between stops.

Usage

```
get_trip_length(
  gtfs,
  trip_id = NULL,
  method = "shapes",
  by = "trip",
  unit = "km",
  sort_sequence,
  file = NULL,
  from = NULL,
  to = NULL
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to have their lengths calculated. If NULL (the default), the function calculates the lengths of every trip_id in the GTFS.
method	A string, either "shapes" (the default) or "euclidean". "shapes" measures lengths along the trip's shape, described in the shapes table, while "euclidean" measures the straight-line (great-circle) distances between consecutive stops. If the GTFS object doesn't have a shapes table, or if its trips table doesn't have a shape_id column, "euclidean" is used instead, with a warning.
by	A string, either "trip" (the default) or "segment". "trip" returns the length from the first to the last stop of each trip, while "segment" returns the length between each pair of consecutive stops.
unit	A string representing the unit in which lengths are desired. Either "km" (the default) or "m".
sort_sequence	Deprecated; do not supply. Timetables and shapes are now always sorted by stop_sequence and shape_pt_sequence. Passing TRUE raises a warning, and passing FALSE raises an error.
file	Deprecated. Use method instead (file = "stop_times" corresponds to method = "euclidean").
from	A string, in the "HH:MM:SS" format. With by = "trip", only trips that depart from their first stop at or after from are included. With by = "segment", only segments that depart from their first stop at or after from are included. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips (or segments) that depart from their first stop at or before to are included. If NULL (the default), there is no upper limit.

Value

With by = "trip", a data.table with the trip_id and the length of each trip. With by = "segment", a data.table with the trip_id, the segment number, the from_stop_id and to_stop_id that delimit the segment and its length.

Details

Segments are numbered as in `get_trip_segment_duration()`, so both outputs can be joined by `trip_id` and `segment`. Trips with a single stop have a length of 0.

With `method = "shapes"`, each stop is projected onto the closest point of its trip's shape, with stops advancing along the shape in `stop_sequence` order, so loops and shapes that pass by the same place more than once are handled. Some limitations remain:

- a stop slightly behind the previous one along the shape (e.g. due to imprecise coordinates) is placed at the previous stop's position, so their segment has a length of 0. On shapes that pass the same place twice in the same direction, such a stop may be matched to the later pass instead;
- on shapes whose outbound and return legs overlap exactly, the stops just before and after the turnaround may be placed at the same position. The trip length is unaffected, but the segment between them gets a length of 0 and the next one is longer.

`shape_dist_traveled` is ignored. Trips not linked to a shape with at least two distinct points have `NA` lengths, with a warning. Use `get_shape_length()` for the length of entire shapes.

Lengths are great-circle (haversine) distances on the sphere used by `{s2}`. Stops with missing coordinates, or not listed in `stops`, result in `NA` lengths for their segments and trips.

`from` and `to` are compared with the `departure_time` of the first stop of each trip (in `stop_sequence` order) or segment, as written in `stop_times`, so use times past `"24:00:00"` for departures after midnight. For trips listed in `frequencies`, these are template times, not the times of each departure. Trips and segments whose departure is blank are never included when `from` or `to` is given.

See Also

`get_shape_length()`, `get_trip_speed()`, `get_trip_segment_duration()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# length along the shape from the first to the last stop of each trip
trip_length <- get_trip_length(gtfs)
head(trip_length)

# length along the shape between consecutive stops
segment_length <- get_trip_length(gtfs, "CPTM L07-0", by = "segment")
head(segment_length)

# straight-line distances between consecutive stops, in meters
straight_length <- get_trip_length(
  gtfs,
  "CPTM L07-0",
  method = "euclidean",
  by = "segment",
  unit = "m"
)
```

```
head(straight_length)
```

`get_trip_segment_duration` *Get trip segments' duration*

Description

Returns the duration of segments between stops of each specified `trip_id`.

Usage

```
get_trip_segment_duration(
  gtfs,
  trip_id = NULL,
  unit = "min",
  sort_sequence,
  from = NULL,
  to = NULL
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A string vector including the `trip_id`s to have their segments' duration calculated. If `NULL` (the default) the function calculates the segments' duration of every `trip_id` in the GTFS.
`unit`	A string representing the time unit in which the duration is desired. One of `"s"` (seconds), `"min"` (minutes, the default), `"h"` (hours) or `"d"` (days).
`sort_sequence`	Deprecated; do not supply. Timetables are now always sorted by `stop_sequence`. Passing `TRUE` raises a warning, and passing `FALSE` raises an error.
`from`	A string, in the "HH:MM:SS" format. Only segments that depart from their first stop at or after `from` are included. If `NULL` (the default), there is no lower limit.
`to`	A string, in the "HH:MM:SS" format. Only segments that depart from their first stop at or before `to` are included. If `NULL` (the default), there is no upper limit.

Value

A `data.table` containing the segments' duration of each specified trip.

Details

A trip segment is defined as the path between two subsequent stops in the same trip. The duration of a segment is defined as the time difference between its arrival time and its departure time, as specified in the `stop_times` file.

Existing `_secs` columns in `stop_times` (e.g. created with `convert_time_to_seconds()`) are used as-is, not recalculated from the time strings.

`from` and `to` are compared with the departure time of the first stop of each segment, read as the durations are (from `departure_time_secs`, if present), so use times past `"24:00:00"` for departures after midnight. For trips listed in `frequencies`, these are template times, not the times of each departure. Segments whose departure is blank are never included when `from` or `to` is given.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

trip_segment_dur <- get_trip_segment_duration(gtfs)
head(trip_segment_dur)

# use the trip_id argument to control which trips are analyzed
trip_segment_dur <- get_trip_segment_duration(gtfs, trip_id = "CPTM L07-0")
trip_segment_dur

# use the unit argument to control in which unit the durations are calculated
trip_segment_dur <- get_trip_segment_duration(gtfs, "CPTM L07-0", unit = "s")
trip_segment_dur

# use the from and to arguments to keep only the segments that depart
# within a time of day
trip_segment_dur <- get_trip_segment_duration(
  gtfs,
  from = "07:00:00",
  to = "09:00:00"
)
head(trip_segment_dur)
```

`get_trip_speed` *Get trip speed*

Description

Returns the average speed of each specified `trip_id`, either from the first to the last stop of the trip or between each pair of consecutive stops.

Usage

```
get_trip_speed(
  gtfs,
  trip_id = NULL,
  method = "shapes",
  by = "trip",
  unit = "km/h",
  sort_sequence,
  file = NULL,
  from = NULL,
  to = NULL
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to have their speeds calculated. If NULL (the default), the function calculates the speed of every trip_id in the GTFS.
method	A string, either "shapes" (the default) or "euclidean". "shapes" measures lengths along the trip's shape, described in the shapes table, while "euclidean" measures the straight-line (great-circle) distances between consecutive stops. If the GTFS object doesn't have a shapes table, or if its trips table doesn't have a shape_id column, "euclidean" is used instead, with a warning.
by	A string, either "trip" (the default) or "segment". "trip" returns the average speed from the first to the last stop of each trip, while "segment" returns the average speed between each pair of consecutive stops.
unit	A string representing the unit in which the speeds are desired. One of "km/h" (the default), "m/s" or "mph" (miles per hour).
sort_sequence	Deprecated; do not supply. Timetables and shapes are now always sorted by stop_sequence and shape_pt_sequence. Passing TRUE raises a warning, and passing FALSE raises an error.
file	Deprecated. Use method instead (file = "stop_times" corresponds to method = "euclidean").
from	A string, in the "HH:MM:SS" format. With by = "trip", only trips that depart from their first stop at or after from are included. With by = "segment", only segments that depart from their first stop at or after from are included. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips (or segments) that depart from their first stop at or before to are included. If NULL (the default), there is no upper limit.

Value

With by = "trip", a data.table with the route_id, the trip_id and the average speed of each trip. With by = "segment", a data.table with the route_id, the trip_id, the segment number, the from_stop_id and to_stop_id that delimit the segment and its average speed.

Details

The speed is the length calculated by get_trip_length() divided by the duration calculated by get_trip_duration() (with by = "trip") or by get_trip_segment_duration() (with by = "segment"). Speeds are NA when the length is NA (e.g. trips not linked to a shape) or when the duration is missing (e.g. blank times at intermediate stops) or not positive. With by = "trip", trips without any arrival or departure time raise the warning of get_trip_duration().

The route_id of each trip comes from the trips table, which is required (also with method = "euclidean"). Trips not listed in trips have an NA route_id.

Existing _secs columns in stop_times (e.g. created with convert_time_to_seconds()) are used as-is, not recalculated from the time strings.

from and to select the trips (or segments) as in get_trip_length(), from the departure_time strings, even if _secs columns exist.

See Also

get_route_speed(), set_trip_speed(), set_route_speed(), get_trip_length(), get_trip_duration(), get_trip_segment_duration()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

trip_speed <- get_trip_speed(gtfs)
head(trip_speed)

trip_ids <- c("CPTM L07-0", "2002-10-0")
trip_speed <- get_trip_speed(gtfs, trip_ids)
trip_speed

trip_speed <- get_trip_speed(gtfs, trip_ids, method = "euclidean")
trip_speed

trip_speed <- get_trip_speed(gtfs, trip_ids, unit = "m/s")
trip_speed

segment_speed <- get_trip_speed(gtfs, "CPTM L07-0", by = "segment")
head(segment_speed)

# speed of the trips that depart within a time of day
peak_speed <- get_trip_speed(gtfs, from = "07:00:00", to = "09:00:00")
head(peak_speed)
```

get_vkt *Get vehicle-kilometres travelled*

Description

Returns the vehicle-kilometres travelled (VKT) by each specified `route_id`: the length of each of its trips, counted once for every departure of the trip on every day on which its service runs. VKT can be restricted to some dates and to a time of day.

Usage

```
get_vkt(gtfs, route_id = NULL, date = NULL, from = NULL, to = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`route_id`	A character vector including the `route_id`s to have their VKT calculated. If `NULL` (the default), the function calculates the VKT of every `route_id` in the GTFS. `route_id`s not listed in `trips` are ignored, with a warning.
`date`	A `Date` vector (including `IDate`) or a character vector of dates in the `"YYYYMMDD"` or `"YYYY-MM-DD"` formats. If given, only these dates are counted. If `NULL` (the default), every day on which the services of the routes run is counted.
`from, to`	Strings in the `"HH:MM:SS"` format. If given (both must be), only the departures at or after `from` and at or before `to` are counted. If `NULL` (the default), every departure is counted.

Value

A `data.table` with the columns `route_id`, `n_days` (the number of counted days on which at least one counted departure of the route runs), `vkt` (the vehicle-kilometres travelled on these days, in km) and `vkt_daily_mean` (`vkt` divided by `n_days`, in km per day), sorted by `route_id`. Routes without any departure on the counted days (and within the time of day) are not included.

Details

The VKT of a route is the sum, over its trips, of the length of each trip times its number of departures times the number of counted days on which its service runs, according to the `calendar` and `calendar_dates` tables. Departures are counted as in `get_route_frequency()`: trips listed in the `frequencies` table depart once per headway, the other trips depart once, and departures after midnight belong to the previous service day. Unlike in `get_route_frequency()`, the time of day includes `to`, as in `get_trip_length()`. An error is raised if the `frequencies` table has invalid entries for trips that run on the counted days.

Trip lengths are calculated with `get_trip_length()`, along the shape of each trip from its first to its last stop, so trips to and from the depot are not included. If the GTFS has no `shapes` table, the straight-line distances between stops are used instead, with a warning. Trips whose length is unknown (e.g. trips without a usable shape, which also raise a warning, or trips listed in `frequencies`

but not in stop_times) add nothing to the VKT, but the days on which they run still count in n_days. If no counted trip of a route has a known length, its vkt is NA.

See Also

get_vmt() for vehicle-miles, get_route_frequency(), get_trip_length(), get_summary()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# VKT of every route over every day of the feed's calendar
vkt <- get_vkt(gtfs)
head(vkt)

# VKT of two routes in the morning peak of a single day
get_vkt(
  gtfs,
  route_id = c("CPTM L07", "2002-10"),
  date = "2019-03-04",
  from = "07:00:00",
  to = "09:00:00"
)
```

---

get_vmt *Get vehicle-miles travelled*

Description

Returns the vehicle-miles travelled (VMT) by each specified route_id: the length of each of its trips, counted once for every departure of the trip on every day on which its service runs. VMT can be restricted to some dates and to a time of day.

Usage

```
get_vmt(gtfs, route_id = NULL, date = NULL, from = NULL, to = NULL)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
route_id	A character vector including the route_ids to have their VKT calculated. If NULL (the default), the function calculates the VKT of every route_id in the GTFS. route_ids not listed in trips are ignored, with a warning.
date	A Date vector (including IDate) or a character vector of dates in the "YYYYMMDD" or "YYYY-MM-DD" formats. If given, only these dates are counted. If NULL (the default), every day on which the services of the routes run is counted.

`from`, `to` Strings in the `"HH:MM:SS"` format. If given (both must be), only the departures at or after `from` and at or before `to` are counted. If `NULL` (the default), every departure is counted.

Value

A `data.table` with the columns `route_id`, `n_days` (the number of counted days on which at least one counted departure of the route runs), `vmt` (the vehicle-miles travelled on these days) and `vmt_daily_mean` (`vmt` divided by `n_days`, in miles per day), sorted by `route_id`. Routes without any departure on the counted days (and within the time of day) are not included.

Details

The VMT is the vehicle-kilometres travelled calculated by `get_vkt()`, converted to international miles (1 mile = 1.609344 km). See `get_vkt()` for how trips, departures, days and trip lengths are counted.

See Also

`get_vkt()`, `get_route_frequency()`, `get_trip_length()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# VMT of every route over every day of the feed's calendar
vmt <- get_vmt(gtfs)
head(vmt)

# VMT of two routes in the morning peak of a single day
get_vmt(
  gtfs,
  route_id = c("CPTM L07", "2002-10"),
  date = "2019-03-04",
  from = "07:00:00",
  to = "09:00:00"
)
```

---

interpolate_shape_dist_traveled

Interpolate the distances travelled along shapes

Description

Calculates the distance, in meters, from the start of each shape to each stop and to each shape point, and writes it to the `shape_dist_traveled` columns of `stop_times` and `shapes`, replacing any existing values.

Usage

```
interpolate_shape_dist_traveled(gtfs)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().

Value

A GTFS object whose stop_times and shapes have a shape_dist_traveled column, in meters. The order of the rows and the other columns are kept. The given GTFS object is not modified.

Details

Each stop is projected onto the closest point of its trip's shape, with stops advancing along the shape in stop_sequence order, as in get_trip_length() (see there for the limitations of this method). Distances are great-circle distances on the sphere used by {s2}.

Existing shape_dist_traveled columns are entirely replaced, since their values may be in any unit, so existing values are lost even where new ones can't be calculated. Stops of trips not linked to a shape with at least two distinct points (with a warning), stops without coordinates and stops of trips with fewer than two stops with coordinates get NA, as do shape points with missing coordinates. Unlike in get_shape_length(), these points are skipped, and the other points of the shape are still measured.

See Also

get_trip_length(), get_shape_length()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

gtfs <- interpolate_shape_dist_traveled(gtfs)
head(gtfs$stop_times[, c("trip_id", "stop_id", "shape_dist_traveled")])
head(gtfs$shapes)
```

---

interpolate_stop_times

Interpolate missing stop times

Description

Fills blank arrival_times and departure_times in stop_times, assuming that vehicles travel at a constant speed between consecutive stops with known times (timepoints).

Usage

```
interpolate_stop_times(gtfs, trip_id = NULL, method = "shapes")
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
trip_id	A character vector including the trip_ids to have their stop times interpolated. If NULL (the default), the function interpolates the stop times of every trip_id in the GTFS.
method	A string, either "shapes" (the default) or "euclidean". "shapes" measures lengths along the trip's shape, described in the shapes table, while "euclidean" measures the straight-line (great-circle) distances between consecutive stops. If the GTFS object doesn't have a shapes table, or if its trips table doesn't have a shape_id column, "euclidean" is used instead, with a warning.

Value

A GTFS object whose stop_times has its blank times filled in. The order of the rows and the other columns are kept. The given GTFS object is not modified.

Details

Within each trip, in stop_sequence order, a stop whose arrival and departure times are both blank gets a time interpolated linearly on the distance travelled from the preceding timepoint, as measured by get_trip_length() (if the distance between two timepoints is 0, their times are spread evenly between the stops in between). Interpolated stops get equal arrival and departure times, rounded to the nearest second. Times past "24:00:00" are supported. A timepoint with only one of its two times blank gets the other time in both columns. If stop_times has a timepoint column, interpolated stops get timepoint = 0; the column is not created otherwise.

Some stops are left blank, with a warning:

- stops before a trip's first or after its last timepoint, including every stop of trips with fewer than two timepoints;
- stops between timepoints whose distance can't be calculated, e.g. trips not linked to a shape or stops without coordinates;
- stops between timepoints in which the arrival is earlier than the previous departure, usually times after midnight written as "00:30:00" instead of "24:30:00".

Times are blank when they are NA, empty or whitespace-only strings, or "NA". Malformed time strings are left unchanged, with a warning, and are not used as timepoints. Rows without a trip_id are left unchanged. For trips listed in frequencies, the template times are interpolated. Times are read from the time strings, and existing _secs columns (e.g. created with convert_time_to_seconds()) are updated to match.

Interpolating between timepoints requires the stops table.

See Also

get_trip_length(), get_trip_speed(), set_trip_speed()

Examples

```
library(dplyr)

data_path <- system.file("extdata/poa_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
trip <- "T2-1@1#520"

# only the first and last stops of each trip have times
gtfs$stop_times |>
  filter(trip_id == trip) |>
  head()

# distances measured along the trips' shapes. some trips of this feed have
# times past midnight written as "00:xx:xx", so they are left blank
interpolated_gtfs <- suppressWarnings(interpolate_stop_times(gtfs))
interpolated_gtfs$stop_times |>
  filter(trip_id == trip) |>
  head()

# the speed is now known between every pair of consecutive stops
head(get_trip_speed(interpolated_gtfs, trip, by = "segment"))

# straight-line distances between stops, for a single trip
interpolated_gtfs <- interpolate_stop_times(
  gtfs,
  trip_id = trip,
  method = "euclidean"
)
interpolated_gtfs$stop_times |>
  filter(trip_id == trip) |>
  head()
```

---

list_validator_versions

List MobilityData's GTFS validator versions

Description

Lists available versions of MobilityData's command line tool to validate GTFS feeds.

Usage

```
list_validator_versions()
```

Value

A data.table with each CLI version, published date, and download URL. If the versions can't be listed due to network problems, it informs that with a message and invisibly returns `NULL`.

See Also

Other validation: download_validator(), validate_gtfs()

`merge_gtfs` *Merge GTFS files*

Description

Combines many GTFS objects into a single one.

Usage

```
merge_gtfs(..., files = NULL, prefix = FALSE)
```

Arguments

`...`	GTFS objects to be merged. Each argument can either be a GTFS or a list of GTFS objects.
`files`	A character vector listing the GTFS tables to be merged. If `NULL` (the default), all tables are merged. Elements that are not tables, such as the `locations` of GTFS-Flex feeds (read from a geojson file), are not merged and are left out of the result, with a warning.
`prefix`	Either a logical or a character vector (defaults to `FALSE`). Whether to add a prefix to the value of id fields that identify from which GTFS object the value comes from. If `TRUE`, the prefixes will range from `"1"` to `n`, where `n` is the number of objects passed to the function. If a character vector, its elements, which must be unique, will be used to identify the GTFS objects, and the length of the vector must equal the total amount of objects passed in `...` (the first element will identify the first GTFS, the second element the second GTFS, and so on). If `FALSE` and two or more GTFS objects are merged, rows that are identical in every column are kept only once (as in remove_duplicates(), also when they come from the same GTFS object; a column that a GTFS object lacks counts as empty), and the GTFS objects may share ids (e.g. the same stop in two feeds) only if their data are identical in all of them: rows are compared on all columns, including columns that only some GTFS objects have, and a trip, for example, must also have the same `stop_times` and `frequencies`. Otherwise an error is raised; use `prefix = TRUE` in that case. This happens with versions of the same feed, but also with unrelated feeds that use simple ids (e.g. both have an `agency_id` `"1"`).

Value

A GTFS object in which each table is a combination (by row) of the tables from the specified GTFS objects.

Examples

```
spo_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
ggl_path <- system.file("extdata/ggl_gtfs.zip", package = "gtfstools")

spo_gtfs <- read_gtfs(spo_path)
names(spo_gtfs)

ggl_gtfs <- read_gtfs(ggl_path)
names(ggl_gtfs)

merged_gtfs <- merge_gtfs(spo_gtfs, ggl_gtfs)
names(merged_gtfs)

# use a list() to programatically merge many GTFS objects
gtfs_list <- list(spo_gtfs, ggl_gtfs)
merged_gtfs <- merge_gtfs(gtfs_list)

# 'prefix' helps disambiguating from which GTFS each id comes from.
# if TRUE, the ids range from 1:n, where n is the number of gtfs
merged_gtfs <- merge_gtfs(gtfs_list, prefix = TRUE)
merged_gtfs$agency

# if a character vector, its elements will be used to identify the each gtfs
merged_gtfs <- merge_gtfs(gtfs_list, prefix = c("spo", "ggl"))
merged_gtfs$agency
```

---

read_gtfs *Read GTFS files*

Description

Reads GTFS text files from either a local `.zip` file or an URL.

Usage

```
read_gtfs(
  path,
  files = NULL,
  fields = NULL,
  skip = NULL,
  quiet = TRUE,
  encoding = "unknown"
)
```

Arguments

`path`	The path to a GTFS `.zip` file.
`files`	A character vector containing the text files to be read from the GTFS (without the `.txt` extension). If `NULL` (the default) all existing files are read.

fields	A named list containing the fields to be read from each text file, in the format list(file = c("field1", "field2")). If NULL (the default), all fields from the files specified in files are read. If a file is specified in files but not in fields, all fields from that file will be read (i.e. you may specify in fields only files whose fields you want to subset).
skip	A character vector containing the text files that should not be read from the GTFS, without the .txt extension. If NULL (the default), no files are skipped. Cannot be used if files is already set.
quiet	Whether to hide log messages and progress bars (defaults to TRUE).
encoding	A string, ultimately passed to data.table::fread(). Defaults to "unknown". Other possible options are "UTF-8" and "Latin-1". Please note that this is not used to re-encode the input, but to enable handling encoded strings in their native encoding.

Value

A data.table-based GTFS object: a list of data.tables in which each table represents a GTFS text file.

Details

The column types of each data.table in the final GTFS object conform as closely as possible to the Google's Static GTFS Reference. Exceptions are date-related columns (such as calendar.txt's start_date and end_date, for example), which are converted to Date objects, instead of being kept as integers, allowing for easier data manipulation. These columns are converted back to integers when writing the GTFS object to a .zip file using write_gtfs().

See Also

Other io functions: write_gtfs()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)
names(gtfs)

gtfs <- read_gtfs(data_path, files = c("trips", "stop_times"))
names(gtfs)

gtfs <- read_gtfs(data_path, skip = "trips")
names(gtfs)

gtfs <- read_gtfs(data_path, fields = list(agency = "agency_id"))
names(gtfs)
names(gtfs$agency)
```

remove_duplicates *Remove duplicated entries*

Description

Removes duplicated entries from GTFS objects tables.

Usage

```
remove_duplicates(gtfs)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().

Value

A GTFS object containing only unique entries.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# this gtfs includes some duplicated entries
gtfs$agency

gtfs <- remove_duplicates(gtfs)
gtfs$agency
```

remove_unused_ids *Remove unused ids*

Description

Removes the ids that the feed doesn't use, with their entries in every file. A trip is used if it has stop_times, and anything that a used trip refers to, directly or indirectly, is used too. Entries that refer to removed ids are removed as well, including those in attributions and translations. When there are no fare_rules, fare_attributes and their agencies are kept, since these fares apply to the whole feed. Fares v2 and GTFS-Flex files are left unchanged, so they may still refer to removed ids.

Usage

```
remove_unused_ids(gtfs)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().

Value

The GTFS object without unused ids.

See Also

filter_by_trip_id(), remove_duplicates()

Examples

```
data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)
nrow(gtfs$agency)

gtfs <- remove_unused_ids(gtfs)
nrow(gtfs$agency)
```

---

set_dwell_time *Set dwell time*

Description

Sets the dwell time of each specified trip_id at each specified stop_id, i.e. the time the vehicle stays at the stop, by changing the departure_time at these stops in stop_times and shifting the later times of the trips accordingly.

Usage

```
set_dwell_time(
  gtfs,
  dwell_time,
  unit = "s",
  trip_id = NULL,
  stop_id = NULL,
  from = NULL,
  to = NULL,
  route_type = NULL,
  by_reference = FALSE
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
dwell_time	A non-negative number, the dwell time to be set, in unit. It's rounded to the nearest whole second and applies to every specified visit (call the function once per value to set different dwell times).
unit	A string representing the time unit in which dwell_time is given. One of "s" (seconds, the default), "min" (minutes), "h" (hours) or "d" (days).
trip_id	A character vector including the trip_ids to have their dwell times set. If NULL (the default), the dwell times of every trip_id in the GTFS (or of every trip of the given route_types) are set.
stop_id	A character vector including the stop_ids at which the dwell times are set. If NULL (the default), the dwell times are set at every stop_id in the GTFS.
from	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or after from are changed. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or before to are changed. If NULL (the default), there is no upper limit.
route_type	An integer vector of route_types (see filter_by_route_type()). If given, only trips of routes of these types are changed; together with the other selection arguments, it restricts their selection. route_types not listed in routes are ignored. If NULL (the default), trips of any type are changed.
by_reference	Whether to update stop_times' data.table by reference. Defaults to FALSE.

Value

If by_reference is set to FALSE, returns a GTFS object with the time columns of its stop_times adjusted. Else, returns a GTFS object invisibly (note that in this case the original GTFS object is altered).

Details

The dwell time of a visit is its departure_time minus its arrival_time in stop_times, as returned by get_dwell_time(). It is set at the visits of every specified trip to every specified stop (not of pairs formed by their elements): the arrival time is kept and the departure time is set to the arrival time plus dwell_time. All the later times of the trip, in stop_sequence order, are shifted by the change, and changes at several visits of the same trip add up. At a trip's last stop, only the departure time changes. A stop visited more than once by the same trip has its dwell time set at each visit.

Blank times are never filled. Selected visits with a blank arrival or departure time are not changed and don't shift the later times, with a warning when stop_id is given; fill them beforehand with interpolate_stop_times(). An error is raised if a shift would result in a negative time (times not in chronological order) or in a time later than "9999:59:59". stop_sequence is assumed to be unique within each trip and not NA. When both trip_id and stop_id are given but none of the trips visits any of the stops, a warning is raised and the GTFS is returned unchanged.

For trips listed in frequencies, the times in stop_times are a template (see frequencies_to_stop_times()), so setting the dwell time at the first stop only changes how long before each departure the vehicle arrives there, while changes at later stops lengthen or shorten every departure.

from and to are compared with the arrival_time of each visit as written in stop_times before any change (template times, for trips listed in frequencies), so use times past "24:00:00" for arrivals after midnight. A shift caused by an earlier visit may thus move a changed visit out of the time of day, or another visit into it. Visits with a blank arrival time are never selected when from or to is given. When the time of day contains no visit of the specified trips to the specified stops, the GTFS is returned unchanged.

stop_ids are matched against the stop_ids in stop_times, which are never stations: to set the dwell times at a station, pass its children that appear in stop_times (see get_children_stops()). Existing _secs columns in stop_times (e.g. created with convert_time_to_seconds()) are updated to match.

See Also

get_dwell_time(), set_min_dwell_time(), set_trip_speed()

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# set a 30 seconds dwell time at a stop of a trip. the later times of the
# trip are shifted by 30 seconds
new_gtfs <- set_dwell_time(
  gtfs,
  trip_id = "CPTM L07-0",
  stop_id = "18920",
  dwell_time = 30
)
new_gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0") |>
  head()

# set the dwell time at a stop in every trip that visits it. use the unit
# argument to give the dwell time in another unit
new_gtfs <- set_dwell_time(
  gtfs,
  stop_id = "18960",
  dwell_time = 1,
  unit = "min"
)
get_dwell_time(new_gtfs, stop_id = "18960")

# set the dwell time only at the visits that arrive within a time of day
new_gtfs <- set_dwell_time(
  gtfs,
  trip_id = "CPTM L07-0",
  dwell_time = 30,
  from = "05:00:00",
```

```
  to = "06:00:00"
)
get_dwell_time(new_gtfs, trip_id = "CPTM L07-0")

# original gtfs remains unchanged
get_dwell_time(gtfs, stop_id = "18960")

# when doing by reference, original gtfs is changed
set_dwell_time(
  gtfs,
  trip_id = "CPTM L07-0",
  stop_id = "18920",
  dwell_time = 30,
  by_reference = TRUE
)
gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0") |>
  head()
```

`set_max_speed` *Set trip maximum speed*

Description

Caps the speed of the specified trips: each part of a trip that is faster than `max_speed` is slowed down to `max_speed`, by changing the `arrival_time` and `departure_time` columns in `stop_times`. Parts of the trips that are already slower are left unchanged.

Usage

```
set_max_speed(
  gtfs,
  max_speed,
  unit = "km/h",
  route_id = NULL,
  trip_id = NULL,
  first_stop = NULL,
  last_stop = NULL,
  from = NULL,
  to = NULL,
  route_type = NULL
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`max_speed`	A number. The maximum speed allowed, in the unit given by `unit`.

unit	A string representing the unit in which the speed is given. One of "km/h" (the default), "m/s" or "mph" (miles per hour).
route_id	A character vector including the route_ids whose trips have their speeds capped. If route_id, trip_id and route_type are all NULL (the default), the speeds of every trip in the GTFS are capped. route_ids not listed in trips are ignored, with a warning.
trip_id	A character vector including the trip_ids to have their speeds capped. When given together with route_id, the trips of both are capped.
first_stop	A string. The stop_id where the segment whose speeds are capped starts. If NULL (the default), the segment starts at each trip's first stop.
last_stop	A string. The stop_id where the segment ends. If NULL (the default), the segment ends at each trip's last stop.
from	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or after from are changed. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or before to are changed. If NULL (the default), there is no upper limit.
route_type	An integer vector of route_types (see filter_by_route_type()). If given, only trips of routes of these types are changed; together with the other selection arguments, it restricts their selection. route_types not listed in routes are ignored. If NULL (the default), trips of any type are changed.

Value

A GTFS object with the time columns of its stop_times adjusted.

Details

A span is the part of a trip between two consecutive stops with times; stops with blank times inside it are kept blank. Its speed is its length, calculated by get_trip_length() along the trip's shape (or, with a warning, as straight lines between stops if there are no shapes; the stops table is required), divided by the time from the departure at its first stop to the arrival at its second. Spans whose length is NA (e.g. in trips not linked to a shape) are left unchanged.

Each span faster than max_speed, or with a zero duration, has its duration raised to its length divided by max_speed, rounded up to the next second. All the later times of the trip are shifted by the accumulated extra time, so dwell times are kept. Spans whose arrival is earlier than the previous departure (usually times after midnight written as "00:30:00" instead of "24:30:00") are left unchanged, with a warning. An error is raised if a shift would result in a time later than "9999:59:59". Only stop_times is modified; its existing _secs columns are updated to match.

With first_stop and last_stop, matched as in set_trip_speed(), only the spans between them are capped: times before first_stop don't change, and the times at and after last_stop are shifted. The given stops must have a time, so that no span crosses them (if only last_stop is given, the trip's first stop needs none). Trips that don't visit first_stop and then last_stop, or have blank times there, are left unchanged, with a warning if they were given in trip_id or route_id.

from and to are compared with the departure at the segment's first stop (or its arrival, if the departure is blank) as written in stop_times (template times, for trips listed in frequencies), so use times past "24:00:00" for departures after midnight. Trips without a time there are left unchanged.

See Also

set_trip_speed(), get_trip_speed()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

# the trip is faster than 30 km/h between some of its stops
get_trip_speed(gtfs, "CPTM L07-0", by = "segment")

gtfs_capped <- set_max_speed(gtfs, max_speed = 30, trip_id = "CPTM L07-0")
get_trip_speed(gtfs_capped, "CPTM L07-0", by = "segment")

# the later stops are shifted, so the trip ends later
 get_start_and_end_times(gtfs, trip_id = "CPTM L07-0")
 get_start_and_end_times(gtfs_capped, trip_id = "CPTM L07-0")

# cap the speed only between two stops. the spans before and after them
# are not capped
gtfs_capped <- set_max_speed(
  gtfs,
  max_speed = 30,
  trip_id = "CPTM L07-0",
  first_stop = "18921",
  last_stop = "18974"
)
get_trip_speed(gtfs_capped, "CPTM L07-0", by = "segment")

# cap the speed of all the trips of all routes of the GTFS feed
gtfs_capped <- set_max_speed(gtfs, max_speed = 8)
```

---

set_min_dwell_time *Set a minimum dwell time*

Description

Raises the dwell time of each specified trip_id at each specified stop_id to at least min_dwell_time, i.e. makes vehicles stay at least that long at the stops, by delaying the departure_time at these stops in stop_times and shifting the later times of the trips accordingly.

Usage

```
set_min_dwell_time(
  gtfs,
  min_dwell_time,
  unit = "s",
  trip_id = NULL,
  stop_id = NULL,
  from = NULL,
  to = NULL,
  route_type = NULL,
  by_reference = FALSE
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
min_dwell_time	A non-negative number, the minimum dwell time, in unit. It's rounded to the nearest whole second.
unit	A string representing the time unit in which min_dwell_time is given. One of "s" (seconds, the default), "min" (minutes), "h" (hours) or "d" (days).
trip_id	A character vector including the trip_ids to have their dwell times raised. If NULL (the default), the dwell times of every trip_id in the GTFS (or of every trip of the given route_types) are raised.
stop_id	A character vector including the stop_ids at which the dwell times are raised. If NULL (the default), the dwell times are raised at every stop_id in the GTFS.
from	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or after from are changed. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only visits whose arrival time is at or before to are changed. If NULL (the default), there is no upper limit.
route_type	An integer vector of route_types (see filter_by_route_type()). If given, only trips of routes of these types are changed; together with the other selection arguments, it restricts their selection. route_types not listed in routes are ignored. If NULL (the default), trips of any type are changed.
by_reference	Whether to update stop_times' data.table by reference. Defaults to FALSE.

Value

If by_reference is set to FALSE, returns a GTFS object with the time columns of its stop_times adjusted. Else, returns a GTFS object invisibly (note that in this case the original GTFS object is altered).

Details

Visits are selected as in set_dwell_time(). At each selected visit whose dwell time is shorter than min_dwell_time, the arrival time is kept and the departure time is set to the arrival time plus min_dwell_time.

The first and last visits with both an arrival and a departure time of each trip are never raised (use `set_dwell_time()` for them). They are found among all the visits of the trip, not only the selected ones, and the times of the last visit may still be shifted by an earlier visit.

Negative dwell times (departure earlier than arrival) are raised like any other shorter dwell time, so a `min_dwell_time` of 0 raises only these. When they come from times after midnight written as `"00:30:00"` instead of `"24:30:00"`, the later times of the trip are shifted by about a day.

Otherwise it works as `set_dwell_time()`.

See Also

`set_dwell_time()`, `get_dwell_time()`, `fix_gtfs()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# vehicles stay 0 seconds at the stops of this trip. make them stay at least
# 30 seconds at each intermediate stop
head(get_dwell_time(gtfs, trip_id = "CPTM L07-0"))

new_gtfs <- set_min_dwell_time(gtfs, 30, trip_id = "CPTM L07-0")
head(get_dwell_time(new_gtfs, trip_id = "CPTM L07-0"))
```

---

set_route_frequency *Set route frequency*

Description

Sets the headway of each specified `route_id` within a given time of day, by replacing its trips in that time of day with a single frequency-based trip, described in the `frequencies` table.

Usage

```
set_route_frequency(
  gtfs,
  route_id = NULL,
  headway,
  from,
  to,
  route_type = NULL
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
route_id	A character vector including the route_ids to have their frequency set. route_ids not listed in trips are ignored, with a warning. If NULL (the default), route_type must be given, and the frequency of every route of these types is set.
headway	A number. The headway to be set, in minutes. It is rounded to whole seconds and applies to every route_id (call the function once per route to set different headways).
from	A string. The starting point of the time of day, in the "HH:MM:SS" format.
to	A string. The ending point of the time of day, in the "HH:MM:SS" format. Must be later than from.
route_type	An integer vector of route_types (see filter_by_route_type()). If given, only routes of these types are changed; together with route_id, it restricts the given routes. route_types not listed in routes are ignored. If NULL (the default), routes of any type are changed.

Value

A GTFS object with updated frequencies, trips and stop_times tables (and transfers, when present). If the given GTFS object has no frequencies table, one is created.

Details

Departures are grouped by route_id, service_id and direction_id (if present in trips) and counted within the time of day (at or after from and before to) as in get_route_frequency(), which thus reports the new headway (exactly, when the length of the time of day is a multiple of headway). Departures after midnight are listed with times past "24:00:00", so use a time of day such as "24:00:00"-"26:00:00" for them.

The function changes existing service, it does not create it: groups with no departure within the time of day are not modified, and scheduled trips that depart before from are not changed, even if they run within the time of day. In each other group:

- the trips that depart within the time of day are replaced by a single template: among the trips of the stop pattern (see get_stop_times_patterns()) with the most departures within the time of day, the one that departs first. Its travel times apply to all the new departures. The other trips that depart within the time of day are removed (frequency-based ones instead lose their frequencies entries within the time of day, see below);
- a frequencies entry is added for the template, from from to to, with headway_secs set to headway and exact_times = 0; the transfers entries that refer to the template then apply to all of its departures;
- the existing frequencies entries of the group's trips are cut at from and to, keeping their departures outside the time of day exactly as before (as converted by frequencies_to_stop_times() with strategy = "exact"). Trips left without any departure are removed from trips, stop_times and transfers.

Groups whose trips within the time of day have no departure time in stop_times can't have a template and are left unchanged, with a warning. Removed trips may leave unused entries in other tables, such as shapes and calendar, which remove_unused_ids() removes.

stop_times requires the stop_id and stop_sequence fields. Existing _secs columns in frequencies (e.g. created with convert_time_to_seconds()) are updated.

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

get_route_frequency(
  gtfs,
  route_id = "CPTM L07",
  from = "07:00:00",
  to = "09:00:00"
)

# sets a 10 minutes headway from 7 to 9 am
new_gtfs <- set_route_frequency(
  gtfs,
  route_id = "CPTM L07",
  headway = 10,
  from = "07:00:00",
  to = "09:00:00"
)
get_route_frequency(
  new_gtfs,
  route_id = "CPTM L07",
  from = "07:00:00",
  to = "09:00:00"
)

# use frequencies_to_stop_times() to get the scheduled trips
new_gtfs <- frequencies_to_stop_times(new_gtfs)
```

---

set_route_speed *Set route average speed*

Description

Sets the average speed of the trips of each specified route_id, or of a segment of them, by changing the arrival_time and departure_time columns in stop_times.

Usage

```
set_route_speed(
  gtfs,
```

```
  route_id = NULL,
  speed,
  unit = "km/h",
  first_stop = NULL,
  last_stop = NULL,
  from = NULL,
  to = NULL,
  route_type = NULL,
  by_reference = FALSE
)
```

Arguments

gtfs	A GTFS object, as created by read_gtfs().
route_id	A character vector including the route_ids to have the average speed of their trips set. route_ids not listed in trips are ignored, with a warning. If NULL (the default), route_type must be given, and the speed of every trip of these types is set.
speed	A number representing the speed to be set for the trips of all route_ids. Must be positive and finite.
unit	A string representing the unit in which the speed is given. One of "km/h" (the default), "m/s" or "mph" (miles per hour).
first_stop	A string. The stop_id where the segment whose speed is set starts. If NULL (the default), the segment starts at each trip's first stop.
last_stop	A string. The stop_id where the segment ends. If NULL (the default), the segment ends at each trip's last stop.
from	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or after from are changed. If NULL (the default), there is no lower limit.
to	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or before to are changed. If NULL (the default), there is no upper limit.
route_type	An integer vector of route_types (see filter_by_route_type()). If given, only trips of routes of these types are changed; together with the other selection arguments, it restricts their selection. route_types not listed in routes are ignored. If NULL (the default), trips of any type are changed.
by_reference	Whether to update stop_times' data.table by reference. Defaults to FALSE.

Value

If by_reference is set to FALSE, returns a GTFS object with the time columns of its stop_times adjusted. Else, returns a GTFS object invisibly (note that in this case the original GTFS object is altered).

Details

Every trip of each route, as listed in `trips`, gets the given speed, as in `set_trip_speed()`, whose details also apply here: the duration of each trip's segment (by default, the whole trip) is set to its length divided by `speed`, the times at stops inside the segment are set to `""`, and `from` and `to` restrict the changes to the trips that depart from the segment's first stop within the time of day. Warnings about trips that can't be changed refer to their `trip_ids`. Trips without `stop_times` entries are ignored.

See Also

`set_trip_speed()`, `get_trip_speed()`, `get_route_speed()`

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

get_route_speed(gtfs, route_id = "CPTM L07")

new_speed_gtfs <- set_route_speed(gtfs, route_id = "CPTM L07", speed = 50)
get_route_speed(new_speed_gtfs, route_id = "CPTM L07")

# set the speed only of the trips that depart within a time of day
poa_path <- system.file("extdata/poa_gtfs.zip", package = "gtfstools")
poa_gtfs <- read_gtfs(poa_path)

poa_new_speed <- set_route_speed(
  poa_gtfs,
  route_id = "T2",
  speed = 25,
  from = "07:00:00",
  to = "09:00:00"
)

# the arrival at the last stop of a trip that departs at 07:02 changes
poa_gtfs$stop_times |>
  filter(trip_id == "T2-1@1#702") |>
  slice_tail(n = 1)

poa_new_speed$stop_times |>
  filter(trip_id == "T2-1@1#702") |>
  slice_tail(n = 1)
```

set_trip_speed *Set trip average speed*

Description

Sets the average speed of each specified `trip_id`, or of a segment of it, by changing the `arrival_time` and `departure_time` columns in `stop_times`.

Usage

```
set_trip_speed(
  gtfs,
  trip_id = NULL,
  speed,
  unit = "km/h",
  first_stop = NULL,
  last_stop = NULL,
  from = NULL,
  to = NULL,
  route_type = NULL,
  by_reference = FALSE
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A string vector including the `trip_id`s to have their average speed set. If `NULL` (the default), `route_type` must be given, and the speed of every trip of these types is set.
`speed`	A numeric representing the speed to be set. Its length must either equal 1, in which case the value is recycled for all `trip_id`s, or equal `trip_id`'s length. Must be positive and finite. A `trip_id` listed more than once must be given the same speed. Must be a single number when `trip_id` is `NULL`.
`unit`	A string representing the unit in which the speed is given. One of `"km/h"` (the default), `"m/s"` or `"mph"` (miles per hour).
`first_stop`	A string. The `stop_id` where the segment whose speed is set starts. If `NULL` (the default), the segment starts at each trip's first stop.
`last_stop`	A string. The `stop_id` where the segment ends. If `NULL` (the default), the segment ends at each trip's last stop.
`from`	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or after `from` are changed. If `NULL` (the default), there is no lower limit.
`to`	A string, in the "HH:MM:SS" format. Only trips that depart from the segment's first stop at or before `to` are changed. If `NULL` (the default), there is no upper limit.

route_type — An integer vector of route_types (see filter_by_route_type()). If given, only trips of routes of these types are changed; together with the other selection arguments, it restricts their selection. route_types not listed in routes are ignored. If NULL (the default), trips of any type are changed.

by_reference — Whether to update stop_times' data.table by reference. Defaults to FALSE.

Value

If by_reference is set to FALSE, returns a GTFS object with the time columns of its stop_times adjusted. Else, returns a GTFS object invisibly (note that in this case the original GTFS object is altered).

Details

The duration of each trip's segment, from the departure at first_stop to the arrival at last_stop, is set to its length divided by speed. Lengths are calculated by get_trip_length() along the trip's shape, or as straight lines between stops, with a warning, if there are no shapes; the stops table is required. Trips whose length is NA (e.g. not linked to a shape) are left unchanged.

The times at stops strictly inside the segment are set to "" (written as NA by write_gtfs()). Times after last_stop are shifted by the change in the segment's duration. Dwell times at first_stop and last_stop are kept, except at the trip's first and last stops, whose dwell time becomes 0. first_stop is matched at its first visit in stop_sequence order, and last_stop at its next visit after that (which may be the same stop). Blank times at last_stop are filled if it's the trip's last stop.

With first_stop or last_stop, trips that don't visit first_stop and then last_stop, or whose departure at first_stop is blank, or whose times at last_stop are both blank (unless it's the trip's last stop), are left unchanged, with a warning if they were given in trip_id. Without first_stop, last_stop, from and to, a blank departure at the trip's first stop leaves all the times of the trip blank, without a warning.

from and to are compared with the departure_time at first_stop as written in stop_times (template times, for trips listed in frequencies), so use times past "24:00:00" for departures after midnight; trips whose departure there is blank are then left unchanged. Existing _secs columns in stop_times are updated to match.

See Also

set_route_speed(), get_trip_speed(), get_route_speed()

Examples

```
library(dplyr)

data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")

gtfs <- read_gtfs(data_path)

gtfs_new_speed <- set_trip_speed(gtfs, trip_id = "CPTM L07-0", 50)
gtfs_new_speed$stop_times |>
```

```
  filter(trip_id == "CPTM L07-0")

# use the unit argument to change the speed unit
gtfs_new_speed <- set_trip_speed(
  gtfs,
  trip_id = "CPTM L07-0",
  speed = 15,
  unit = "m/s"
)
gtfs_new_speed$stop_times |>
  filter(trip_id == "CPTM L07-0")

# set the speed only between two stops. later stops are shifted
gtfs_new_speed <- set_trip_speed(
  gtfs,
  trip_id = "CPTM L07-0",
  speed = 30,
  first_stop = "18917",
  last_stop = "18922"
)
gtfs_new_speed$stop_times |>
  filter(trip_id == "CPTM L07-0")

# set the speed only of the trips that depart within a time of day
poa_path <- system.file("extdata/poa_gtfs.zip", package = "gtfstools")
poa_gtfs <- read_gtfs(poa_path)

poa_new_speed <- set_trip_speed(
  poa_gtfs,
  trip_id = poa_gtfs$trips$trip_id,
  speed = 25,
  from = "07:00:00",
  to = "09:00:00"
)

# the arrival at the last stop of a trip that departs at 07:02 changes
poa_gtfs$stop_times |>
  filter(trip_id == "T2-1@1#702") |>
  slice_tail(n = 1)

poa_new_speed$stop_times |>
  filter(trip_id == "T2-1@1#702") |>
  slice_tail(n = 1)

# original gtfs remains unchanged
gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0")

# when doing by reference, original gtfs is changed
set_trip_speed(gtfs, trip_id = "CPTM L07-0", 50, by_reference = TRUE)

gtfs$stop_times |>
  filter(trip_id == "CPTM L07-0")
```

stop_times_to_frequencies *Convert stop times to frequencies*

Description

Converts scheduled trips, described only in `stop_times`, into frequency-based trips, described in `frequencies`. Trips that follow the same route and sequence of stops are summarised, in each hour of the day, by a single template trip and the headway between their departures. This is a lossy approximation of the original schedule.

Usage

```
stop_times_to_frequencies(gtfs, trip_id = NULL)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`trip_id`	A character vector including the `trip_id`s to be converted. If `NULL` (the default), the function converts all trips listed in `stop_times` that are not already listed in `frequencies`.

Value

A GTFS object with updated `frequencies`, `stop_times` and `trips` tables (and `transfers`, when present). If the given GTFS object has no `frequencies` table, one is created.

Details

Trips are grouped by `route_id`, `service_id`, `direction_id` and `shape_id` (the last two only if present in `trips`) and by their sequence of stops, as identified by `get_stop_times_patterns()`. Each group is split into one-hour slots of the clock, from `HH:00:00` (included) to the next `HH:00:00` (excluded), by the earliest `departure_time` of each trip in `stop_times`, regardless of `stop_sequence` (so times after midnight written as `"00:30:00"` instead of `"24:30:00"` misplace the trip). Slots after midnight are kept as such (e.g. `"25:00:00"` to `"26:00:00"`).

In each slot with `n` trips, the trip with the earliest departure (ties broken by `trip_id`) becomes the template and keeps its `trip_id` and its `trips` and `stop_times` entries. A `frequencies` entry is added for it, from the start to the end of the slot, with `headway_secs = ceiling(3600 / n)` and `exact_times = 0`. The other trips are removed from `trips`, `stop_times` and `transfers`; `transfers` entries of the template now apply to all of its departures. Other tables that may refer to trips, such as `attributions` and `translations`, are not changed.

Slots with a single trip are also converted. Only the trips in `trip_id` are grouped and converted; other trips are never removed, even if they share a group and a slot with converted ones. Trips

already listed in `frequencies` (with a warning, if listed in `trip_id`), trips not listed in `trips` and trips without any `departure_time` are left unchanged.

Converting the result back with `frequencies_to_stop_times()` yields the same number of trips in each slot with at most 60 trips (approximately the same otherwise), evenly spaced from the start of the slot.

Existing `_secs` columns in `stop_times` and `frequencies` (e.g. created with `convert_time_to_seconds()`) are used as-is and filled in for the new `frequencies` entries.

Examples

```
library(dplyr)

data_path <- system.file("extdata/ber_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

# converts all trips
converted_gtfs <- stop_times_to_frequencies(gtfs)
nrow(gtfs$trips)
nrow(converted_gtfs$trips)
head(converted_gtfs$frequencies)

# converts only the trips of one route
route_trips <- gtfs$trips |>
  filter(route_id == gtfs$trips$route_id[1]) |>
  pull(trip_id)
converted_gtfs <- stop_times_to_frequencies(gtfs, route_trips)

# converting back yields evenly spaced scheduled trips
back_gtfs <- frequencies_to_stop_times(converted_gtfs)
```

---

validate_gtfs *Validate GTFS feed*

Description

Uses MobilityData's GTFS validator to perform a GTFS business rule validation. The results are available as an HTML report (if validator v3.1.0 or higher is used) and in JSON format. Please check the complete set of rules used in the validation here. Please note that this function requires a working installation of Java 11 or higher to work. Recent versions of the validator require Java 17 or higher.

Usage

```
validate_gtfs(
  gtfs,
```

```
  output_path,
  validator_path,
  overwrite = TRUE,
  html_preview = TRUE,
  pretty_json = FALSE,
  quiet = TRUE,
  n_threads = 1
)
```

Arguments

gtfs	The GTFS to be validated. Can be in the format of a GTFS object, of a path to a GTFS file, of a path to a directory or an URL to a feed.
output_path	A string. The path to the directory that the validator will create and in which the results will be saved to.
validator_path	A string. The path to the GTFS validator, previously downloaded with download_validator().
overwrite	A logical. Whether to overwrite existing validation results in output_path. Defaults to TRUE.
html_preview	A logical. Whether to show HTML report in a viewer, such as RStudio or a browser. Defaults to TRUE (only works on interactive sessions).
pretty_json	A logical. Whether JSON results should be printed in a readable way, that allows it to be inspected without manually formatting. Defaults to FALSE.
quiet	A logical. Whether to hide informative messages. Defaults to TRUE.
n_threads	An integer between 1 and the number of cores in the running machine. Control how many threads are used during the validation. Defaults to 1.

Value

Invisibly returns the normalized path to the directory where the validation results were saved to.

See Also

Other validation: download_validator(), list_validator_versions()

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
output_path <- tempfile("validation_result")
validator_path <- download_validator(tempdir())

# download_validator() returns NULL if the validator can't be downloaded
if (!is.null(validator_path)) {
  gtfs <- read_gtfs(data_path)

  validate_gtfs(gtfs, output_path, validator_path)
  print(list.files(output_path))

  # works with feeds saved to disk
```

```
  new_output_path <- tempfile("new_validation_result")
  validate_gtfs(data_path, new_output_path, validator_path)
  print(list.files(new_output_path))

  # and with feeds pointed by an url
  newer_output_path <- tempfile("newer_validation_result")
  gtfs_url <- "https://github.com/ipea/gtfstools/raw/main/inst/extdata/spo_gtfs.zip"
  validate_gtfs(gtfs_url, newer_output_path, validator_path)
  print(list.files(newer_output_path))
}
```

`write_gtfs` *Write GTFS files*

Description

Writes GTFS objects as GTFS `.zip` files.

Usage

```
write_gtfs(
  gtfs,
  path,
  files = NULL,
  standard_only = FALSE,
  as_dir = FALSE,
  overwrite = TRUE,
  quiet = TRUE,
  compression_level = 6
)
```

Arguments

`gtfs`	A GTFS object, as created by `read_gtfs()`.
`path`	The path to the `.zip` file in which the feed should be written to.
`files`	A character vector containing the name of the elements to be written to the feed. If `NULL` (the default), all elements inside the GTFS object are written.
`standard_only`	Whether to write only standard files and fields (defaults to `FALSE`, which doesn't drop extra files and fields).
`as_dir`	Whether to write the feed as a directory, instead of a `.zip` file (defaults to `FALSE`, which means that the field is written as a zip file).
`overwrite`	Whether to overwrite existing `.zip` file (defaults to `TRUE`).
`quiet`	Whether to hide log messages and progress bars (defaults to `TRUE`).

compression_level
: A number between 1 and 9 indicating how much the `.zip` file should be compressed. Lower levels write faster but create larger files. Defaults to 6, which is usually several times faster than 9 and creates files of very similar size. Ignored when `as_dir = TRUE`.

Value

Invisibly returns the same GTFS object passed to the `gtfs` parameter.

See Also

Other io functions: `read_gtfs()`

Examples

```
data_path <- system.file("extdata/spo_gtfs.zip", package = "gtfstools")
gtfs <- read_gtfs(data_path)

tmp_dir <- file.path(tempdir(), "tmpdir")
dir.create(tmp_dir)
list.files(tmp_dir) #'
tmp_file <- tempfile(pattern = "gtfs", tmpdir = tmp_dir, fileext = ".zip")
write_gtfs(gtfs, tmp_file)
list.files(tmp_dir)

gtfs_all_files <- read_gtfs(tmp_file)
names(gtfs_all_files)

write_gtfs(gtfs, tmp_file, files = "stop_times")
gtfs_stop_times <- read_gtfs(tmp_file)
names(gtfs_stop_times)
```

Index

∗ **filtering functions**
  filter_by_agency_id, 18
  filter_by_date, 19
  filter_by_period, 20
  filter_by_route_id, 22
  filter_by_route_type, 23
  filter_by_service_id, 26
  filter_by_shape_id, 28
  filter_by_spatial_extent, 29
  filter_by_stop_id, 30
  filter_by_time_of_day, 32
  filter_by_trip_id, 34
  filter_by_weekday, 35
∗ **io functions**
  read_gtfs, 85
  write_gtfs, 106
∗ **validation**
  download_validator, 17
  list_validator_versions, 83
  validate_gtfs, 104

as_dt_gtfs, 3

build_graph, 5

combine_nearby_stops, 8
convert_seconds_to_time, 9
convert_sf_to_shapes, 10
convert_sf_to_shapes(), *4*, *13*
convert_shapes_to_sf, 11
convert_stops_to_sf, 12
convert_stops_to_shapes, 13
convert_time_to_seconds, 14
convert_time_to_seconds(), *9, 10*, *33*, *40*, *61*, *69*, *75*, *77*, *82*, *90*, *97*, *104*
crop_gtfs, 15

data.table::fread(), *86*
download_validator, 17
download_validator(), *84*, *105*

filter_by_agency_id, 18
filter_by_agency_id(), *19*, *21, 22*, *26–29*, *31*, *33*, *35, 36*
filter_by_date, 19
filter_by_date(), *6*, *18*, *21, 22*, *26–29*, *31*, *33*, *35, 36*, *42*, *45*, *67*
filter_by_period, 20
filter_by_period(), *18, 19*, *22*, *26–29*, *31*, *33*, *35, 36*
filter_by_route_id, 22
filter_by_route_id(), *18, 19*, *21*, *26–29*, *31*, *33*, *35, 36*, *64*
filter_by_route_type, 23
filter_by_route_type(), *18, 19*, *21, 22*, *27–29*, *31*, *33*, *35, 36*, *66*, *89*, *92*, *94*, *96*, *98*, *101*
filter_by_service_id, 26
filter_by_service_id(), *18, 19*, *21, 22*, *26*, *28, 29*, *31*, *33*, *35, 36*, *43*
filter_by_sf, 27
filter_by_shape_id, 28
filter_by_shape_id(), *18, 19*, *21, 22*, *26, 27*, *29*, *31*, *33*, *35, 36*
filter_by_spatial_extent, 29
filter_by_spatial_extent(), *15, 16*, *18, 19*, *21, 22*, *26–28*, *31*, *33*, *35, 36*
filter_by_stop_id, 30
filter_by_stop_id(), *18, 19*, *21, 22*, *26–29*, *33*, *35, 36*
filter_by_time_of_day, 32
filter_by_time_of_day(), *18, 19*, *21, 22*, *26–29*, *31*, *35, 36*, *49*, *59*, *67*
filter_by_trip_id, 34
filter_by_trip_id(), *16*, *18, 19*, *21, 22*, *26–29*, *31*, *33*, *36*, *64*, *88*
filter_by_weekday, 35
filter_by_weekday(), *6*, *18, 19*, *21, 22*, *26–29*, *31*, *33*, *35*
fix_gtfs, 37

fix_gtfs(), *95*
frequencies_to_stop_times, 39
frequencies_to_stop_times(), *43*, *49*, *54*, *57*, *59*, *62*, *89*, *96*, *104*

geos_binary_pred, *29*
get_active_services, 41
get_active_services(), *21*, *53, 54*, *57*, *62*
get_calendar_overlap, 42
get_calendar_overlap(), *19*, *45*, *67*
get_children_stops, 44
get_children_stops(), *47, 48*, *59*, *62–64*, *90*
get_dates, 45
get_dates(), *21*, *42*, *53, 54*, *62*
get_dwell_time, 46
get_dwell_time(), *89, 90*, *95*
get_parent_station, 47
get_parent_station(), *63, 64*
get_route_frequency, 48
get_route_frequency(), *6*, *59*, *67*, *78–80*, *96*
get_route_geometry, 50
get_route_speed, 51
get_route_speed(), *77*, *99*, *101*
get_route_timetable, 53
get_route_timetable(), *62*
get_shape_length, 54
get_shape_length(), *73*, *81*
get_shapes_geometry, 55
get_shapes_geometry(), *11, 12*, *51*, *71*
get_start_and_end_times, 56
get_stop_distances, 57
get_stop_distances(), *6*, *9*
get_stop_frequency, 58
get_stop_frequency(), *7*, *49*
get_stop_times_patterns, 60
get_stop_times_patterns(), *96*, *103*
get_stop_timetable, 61
get_stop_timetable(), *54*
get_stops, 63
get_stops(), *58*
get_stops_geometry, 64
get_stops_geometry(), *12*
get_summary, 65
get_summary(), *79*
get_trip_duration, 68
get_trip_duration(), *57*, *67*, *77*
get_trip_geometry, 70
get_trip_geometry(), *13*, *50, 51*
get_trip_length, 71
get_trip_length(), *13*, *16*, *55*, *58*, *67*, *71*, *77–82*, *92*, *101*
get_trip_segment_duration, 74
get_trip_segment_duration(), *73*, *77*
get_trip_speed, 75
get_trip_speed(), *52*, *67*, *73*, *82*, *93*, *99*, *101*
get_vkt, 78
get_vkt(), *67*, *80*
get_vmt, 79
get_vmt(), *67*, *79*
gtfsio::import_gtfs(), *4*

interpolate_shape_dist_traveled, 80
interpolate_shape_dist_traveled(), *38, 39*
interpolate_stop_times, 81
interpolate_stop_times(), *7*, *16*, *38, 39*, *47*, *89*

list_validator_versions, 83
list_validator_versions(), *17*, *105*

merge_gtfs, 84
merge_gtfs(), *9*, *43*

read_gtfs, 85
read_gtfs(), *4, 5*, *8*, *12–15*, *18–20*, *22, 23*, *26*, *28–30*, *32*, *34, 35*, *37*, *40–42*, *44–46*, *48–50*, *52–57*, *59, 60*, *62–65*, *69, 70*, *72*, *74*, *76*, *78, 79*, *81, 82*, *87–89*, *91*, *94*, *96*, *98*, *100*, *103*, *106, 107*
remove_duplicates, 87
remove_duplicates(), *38, 39*, *84*, *88*
remove_unused_ids, 87
remove_unused_ids(), *9*, *13*, *38, 39*, *97*

set.seed(), *40*
set_dwell_time, 88
set_dwell_time(), *47*, *94, 95*
set_max_speed, 91
set_max_speed(), *38, 39*
set_min_dwell_time, 93
set_min_dwell_time(), *38, 39*, *47*, *90*
set_route_frequency, 95
set_route_speed, 97
set_route_speed(), *52*, *77*, *101*

set_trip_speed, 100
set_trip_speed(), *52*, *77*, *82*, *90*, *92, 93*, *99*
sf::st_equals_exact(), *29*
sf::st_intersects, *29*
sf::st_intersects(), *15*
sf::st_is_within_distance(), *29*
stop_times_to_frequencies, 103

tempdir(), *66*

validate_gtfs, 104
validate_gtfs(), *17*, *65*, *84*

write_gtfs, 106
write_gtfs(), *86*, *101*