Package: GTFSwizard (via r-universe)

November 18, 2026

Type Package

Title Creating, Exploring, and Manipulating GTFS Files

Version 1.2.2

Author Nelson de Oliveira Quesado Filho [aut, cre], Caio Gustavo Coelho Guimarães [aut], Francisco Moraes de Oliveira Neto [aut]

Maintainer Nelson de Oliveira Quesado Filho <nquesado@gmail.com>

Date 2026-11-16

URL https://github.com/nelsonquesado/GTFSwizard, https://nelsonquesado.github.io/GTFSwizard/, https://opatp.github.io/GTFSwizard/

BugReports https://github.com/nelsonquesado/GTFSwizard/issues

Description Creating, exploring, analyzing, and manipulating General Transit Feed Specification (GTFS) files, which represent public transportation schedules and geographic data. The package allows users to filter data by routes, trips, stops, service dates, and time, generate spatial visualizations, and perform detailed analyses of transit networks, including headway, dwell times, route frequencies, service span, scheduled vehicle-hours, and trip duration. Methods follow common public transport planning and operation concepts described in Ceder (2007, ISBN:978-0-7506-6166-6), Vuchic (2005, ISBN:978-0-471-63265-8), Vuchic (2007, ISBN:978-0-471-75823-5), Cascetta (2009) <doi:10.1007/978-0-387-75857-2>, and Gentile and Noekel (2016) <doi:10.1007/978-3-319-25082-3>.

Imports sf, tidyr, dplyr, ggplot2, gtfsio, rlang, tibble, zip

Suggests data.table, gtfstools, hms, jsonlite, knitr, leaflet, openssl, plotly, rmarkdown, rstudioapi, shiny, testthat, tidytransit

Depends R (>= 4.1.0)

License GPL-3

Encoding UTF-8

LazyData true

LazyDataCompression xz

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

VignetteBuilder knitr

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

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

Date/Publication 2026-11-17 23:20:12 UTC

RemoteUrl https://github.com/nelsonquesado/gtfswizard

RemoteRef HEAD

RemoteSha 444a5830c1ef72ea6c84b11ae48a53c465b7f31e

Contents

as_wizardgtfs . . . 3
bcn_bus_gtfs . . . 4
compare_scenarios . . . 5
create_gtfs . . . 6
delay_trip . . . 7
edit_dwelltime . . . 8
edit_speed . . . 9
explore_gtfs . . . 10
filter_area . . . 11
filter_servicepattern . . . 12
for_bus_gtfs . . . 13
for_rail_gtfs . . . 14
get_1stdeparture . . . 16
get_corridor . . . 16
get_distances . . . 17
get_durations . . . 18
get_dwelltimes . . . 19
get_fleet . . . 21
get_frequency . . . 22
get_headways . . . 23
get_hubs . . . 24
get_servicepattern . . . 24
get_shapes . . . 25
get_shapes_df . . . 26
get_shapes_sf . . . 27
get_speeds . . . 27
get_stops_sf . . . 28
latlon2epsg . . . 29
merge_gtfs . . . 29

plot_calendar . . . 30
plot_corridor . . . 31
plot_frequency . . . 32
plot_headways . . . 33
plot_hubs . . . 33
plot_routeduration . . . 34
plot_routefrequency . . . 35
plot_serviceheatmap . . . 36
plot_servicespan . . . 36
plot_servicesupply . . . 37
read_gtfs . . . 38
repair_gtfs . . . 39
repair_report . . . 40
selection . . . 40
set_dwelltime . . . 41
split_trip . . . 42
tidy_raptor . . . 43
validate_gtfs . . . 45
wizardgtfs-methods . . . 47
write_gtfs . . . 48

Index **49**

`as_wizardgtfs` *Convert a GTFS Feed to 'wizardgtfs'*

Description

Converts a list-like GTFS feed to the class used by GTFSwizard. Character GTFS dates are converted to [Date] values, a date-to-service lookup is created, and missing shapes can optionally be inferred from stop locations.

Usage

```
as_wizardgtfs(gtfs_list, build_shapes = TRUE, repair = c("legacy", "none"))
```

Arguments

`gtfs_list`	A named list of GTFS tables or a 'tidygtfs' object.
`build_shapes`	Logical. If 'TRUE', infer 'shapes' when the table is absent. Inferred shapes connect stops with straight line segments and are intended for analysis and visualization, not map-matched routing.
`repair`	'legacy' preserves historical reference cleanup, recording changes in [repair_report()]. Use 'none' to validate without deleting or creating reference records. Use [repair_gtfs()] for explicit time repairs.

Details

The input is checked for required tables, required fields, primary-key duplication, foreign-key consistency, increasing stop and shape sequences, valid service dates, and valid GTFS time strings. Times remain character values because GTFS permits hours greater than 24.

Value

A ‘wizardgtfs‘ object.

See Also

[GTFSwizard::create_gtfs()], [GTFSwizard::get_shapes()]

Examples

```
gtfs_wizard <- as_wizardgtfs(for_rail_gtfs, build_shapes = TRUE)
```

---

bcn_bus_gtfs *GTFS Data for Barcelona (Bus System), Spain*

Description

A historical AMB Mobilitat bus feed for the Barcelona metropolitan area. It contains six agencies, 128 routes, 25,122 trips, 3,385 stops, 776,187 stop-time records, and 134,170 shape points. It is a historical snapshot.

Format

A `wizardgtfs` object containing `feed_info`, `agency`, `routes`, `stops`, `shapes`, `trips`, `calendar_dates`, `calendar`, `stop_times`, and the internal `dates_services` lookup.

Details

Imported from `es-barcelona-autos-castellbisbal-gtfs-892.zip` using `read_gtfs()`. The publisher is AMBMobilitat and the feed version is `20240621`. Import removes 48 trips with fewer than two stop-time records. Publisher metadata states validity from 2024-06-21 to 2024-06-26; the supplied calendars cover 2017-01-27 through 2026-12-31 and produce 239 service patterns. These source calendar ranges are preserved. For a representative analysis of the published snapshot, filter to its stated validity dates with `filter_date()`.

Source

AMB Mobilitat. Publisher website: https://www.ambmobilitat.cat.

See Also

for_bus_gtfs, for_rail_gtfs

Examples

```
data(bcn_bus_gtfs)
head(bcn_bus_gtfs$routes)
head(bcn_bus_gtfs$stops)
```

`compare_scenarios` *Compare Scheduled Supply in Two Scenarios*

Description

Compare one common service date by route or across the network. No changes are made to either feed.

Usage

```
compare_scenarios(
  baseline,
  scenario,
  dates,
  by = c("route", "network"),
  distance_method = c("shapes", "straight")
)
```

Arguments

`baseline`, `scenario`	GTFS objects.
`dates`	One service date available in both feeds. Required explicitly so calendar patterns are never accidentally compared as calendar days.
`by`	Either ‘"route"‘ (default) or ‘"network"‘.
`distance_method`	Distance method passed to [get_distances()].

Details

Metrics are departure instances, scheduled vehicle-hours, vehicle-kilometres, service span, mean departure headway, and maximum simultaneous active trips. Vehicle-hours include intermediate dwell but exclude layover/deadheading. Fleet is concurrent active trips, not an operational vehicle requirement. Headways compare first departures within route and direction and then average intervals; network headway pools these intervals rather than interleaving different routes. Frequency departures are nominal for ‘exact_times = 0‘. Both feeds must use comparable route IDs.

Value

A long tibble with ‘route_id‘ (route comparisons), ‘metric‘, ‘unit‘, ‘baseline‘, ‘scenario‘, ‘change‘, and ‘percent_change‘. Percentage changes from a zero baseline are ‘NA‘. Missing routes have zero additive supply; their span and mean headway are undefined (‘NA‘).

Examples

```
compare_scenarios(for_rail_gtfs, for_rail_gtfs, dates = "2021-12-13")
```

`create_gtfs` *Create a GTFS Feed from Tables*

Description

Creates and validates a GTFS Schedule feed from data frames. The required files and fields follow the current GTFS Schedule reference, including its conditional rules for route names, stop locations, and service calendars.

Usage

```
create_gtfs(
  agency,
  routes,
  trips,
  stop_times,
  stops,
  calendar = NULL,
  calendar_dates = NULL,
  shapes = NULL,
  frequencies = NULL,
  transfers = NULL,
  fare_attributes = NULL,
  fare_rules = NULL,
  build_shapes = TRUE,
  zipfile = NULL,
  repair = c("legacy", "none"),
  ...
)
```

Arguments

`agency`, `routes`, `trips`, `stop_times`, `stops`	Required GTFS tables supplied as data frames. ‘agency_id‘ may be omitted when the feed contains one agency.
`calendar`, `calendar_dates`	Service calendar tables. At least one must define every ‘service_id‘ used by ‘trips‘.
`shapes`, `frequencies`, `transfers`, `fare_attributes`, `fare_rules`	Optional GTFS tables.
`build_shapes`	Logical. If ‘TRUE‘, infer straight-line shapes when ‘shapes‘ is not supplied.
`zipfile`	Optional output path ending in ‘.zip‘.
`repair`	Reference cleanup policy passed to [as_wizardgtfs()].

... Additional named GTFS tables, such as ‘pathways‘, ‘levels‘, ‘feed_info‘, or GTFS Fares v2 tables.

Value

A validated ‘wizardgtfs‘ object, invisibly written as well when ‘zipfile‘ is supplied.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/)

Examples

```
feed <- create_gtfs(
  agency = data.frame(
    agency_id = "A", agency_name = "Demo Transit",
    agency_url = "https://example.com",
    agency_timezone = "America/Fortaleza"
  ),
  routes = data.frame(
    route_id = "R1", agency_id = "A", route_short_name = "1",
    route_long_name = "Central", route_type = 3
  ),
  trips = data.frame(route_id = "R1", service_id = "WK", trip_id = "T1"),
  stop_times = data.frame(
    trip_id = "T1", arrival_time = c("08:00:00", "08:10:00"),
    departure_time = c("08:00:00", "08:10:00"),
    stop_id = c("S1", "S2"), stop_sequence = c(1, 2)
  ),
  stops = data.frame(
    stop_id = c("S1", "S2"), stop_name = c("First", "Second"),
    stop_lat = c(-3.73, -3.74), stop_lon = c(-38.52, -38.53)
  ),
  calendar = data.frame(
    service_id = "WK", monday = 1, tuesday = 1, wednesday = 1,
    thursday = 1, friday = 1, saturday = 0, sunday = 0,
    start_date = "20260101", end_date = "20261231"
  )
)
```

---

delay_trip *Shift Trips in Time*

Description

Adds a number of seconds to every non-empty arrival and departure time for selected trips. Edited times that would move two or more days after midnight are wrapped back to the equivalent time on the next service day while keeping stop times non-decreasing within each trip.

Usage

```
delay_trip(gtfs, trip, duration)
```

Arguments

gtfs	A GTFS object.
trip	Character vector of ‘trip_id‘ values.
duration	Numeric seconds or an object coercible to seconds, such as a ‘difftime‘.

Value

A modified ‘wizardgtfs‘ object.

See Also

[GTFSwizard::edit_speed()], [GTFSwizard::set_dwelltime()]

Examples

```
delayed <- delay_trip(
  for_rail_gtfs,
  trip = for_rail_gtfs$trips$trip_id[1:2],
  duration = 300
)
```

---

edit_dwelltime *Scale Dwell Times*

Description

Multiplies dwell time at selected trip-stop calls and propagates each change to all later times in the same trip. Arrival at the edited stop is retained; departure and subsequent calls move by the dwell-time difference. Edited times that would move two or more days after midnight are wrapped back to the equivalent time on the next service day while keeping stop times non-decreasing within each trip.

Usage

```
edit_dwelltime(gtfs, trips = "all", stops = "all", factor)
```

Arguments

gtfs	A GTFS object.
trips, stops	Character ID vectors or ‘"all"‘.
factor	One non-negative numeric multiplier.

Value

A modified ‘wizardgtfs‘ object.

See Also

[GTFSwizard::set_dwelltime()], [GTFSwizard::get_dwelltimes()]

Examples

```
edited <- edit_dwelltime(
  for_rail_gtfs,
  trips = for_rail_gtfs$trips$trip_id[1:2],
  stops = for_rail_gtfs$stops$stop_id[1:2],
  factor = 1.5
)
```

---

edit_speed *Scale In-Vehicle Travel Speed*

Description

Changes travel time between consecutive stops by dividing it by a speed multiplier. Dwell times are preserved and all downstream times are shifted. Edited times that would move two or more days after midnight are wrapped back to the equivalent time on the next service day while keeping stop times non-decreasing within each trip.

Usage

```
edit_speed(gtfs, trips = "all", stops = "all", factor)
```

Arguments

gtfs	A GTFS object.
trips	Character trip IDs or ‘"all"‘.
stops	Character stop IDs or ‘"all"‘. A segment is edited when either endpoint is selected.
factor	One positive speed multiplier. For example, ‘2‘ halves segment travel times.

Value

A modified ‘wizardgtfs‘ object.

See Also

[GTFSwizard::get_speeds()], [GTFSwizard::get_durations()]

Examples

```
edited <- edit_speed(
  for_rail_gtfs,
  trips = for_rail_gtfs$trips$trip_id[1:2],
  stops = "all",
  factor = 1.25
)
```

`explore_gtfs` *Explore GTFS Data in an Interactive Shiny Dashboard*

Description

Opens a lightweight Shiny dashboard for exploring a GTFS feed. The dashboard shows summary cards, route and stop maps, planning indicators, service calendars, operational charts, and corridor and hub views. Route, service, service-pattern, stop, date, and time filters update the dashboard without changing the original object. Edits and exports operate on an in-memory copy.

Usage

```
explore_gtfs(gtfs = NULL, plotly = FALSE)
```

Arguments

`gtfs`	A GTFS object, preferably of class ‘wizardgtfs‘. When omitted or ‘NULL‘ in an interactive session, a file-selection window opens so the user can choose a GTFS ‘.zip‘ archive.
`plotly`	Logical. If ‘TRUE‘, eligible dashboard charts are rendered as interactive plotly widgets. The calendar and trip-duration boxplot remain static because conversion reduces their readability. ‘plotly‘ is optional and only required when this argument is ‘TRUE‘.

Value

A Shiny app object.

References

Ceder, A. (2007). *Public Transit Planning and Operation*. Vuchic, V. R. (2005). *Urban Transit: Operations, Planning, and Economics*. White, P. (2008). *Public Transport: Its Planning, Management and Operation*. Cascetta, E. (2009). *Transportation Systems Analysis*.

See Also

[GTFSwizard::as_wizardgtfs()], [GTFSwizard::get_shapes()], [GTFSwizard::plot_calendar()]

Examples

```
if (interactive()) {
  explore_gtfs()
  explore_gtfs(GTFSwizard::for_rail_gtfs)
}
```

`filter_area` *Filter a Feed by Stop Locations in an Area*

Description

Select stops intersecting a polygon or bounding box, retaining either whole trips that serve those stops or only the calls inside the area.

Usage

```
filter_area(gtfs, area, full_trips = TRUE)
```

Arguments

`gtfs`	A GTFS object.
`area`	An ‘sf‘/‘sfc‘ polygon with a known CRS, an ‘st_bbox‘, or a numeric WGS84 bounding box ordered xmin, ymin, xmax, ymax.
`full_trips`	Logical. Retain complete trips (default). If ‘FALSE‘, retain only inside calls; trips with fewer than two remaining calls are removed. Cropped endpoints must have known arrival and departure times.

Details

Selection uses stop points, including polygon boundaries, rather than shape intersection. Cropped trips keep original stop sequences and shapes; distance calculations cover only retained stops. Frequency windows are shifted when cropping changes the first stop. Original times and measures are retained. A crop may connect retained stops across an excluded area; it does not split trips into separate contiguous sections.

Value

A ‘wizardgtfs‘ object with related tables pruned. Parent stations are retained. No matching service returns an empty feed with a warning.

Examples

```
filter_area(for_rail_gtfs, c(-39, -4, -38, -3))
```

`filter_servicepattern` *Filter a GTFS Feed*

Description

Filters a GTFS feed while preserving referential integrity across routes, trips, stops, shapes, calendars, frequencies, fares, and transfers.

Usage

```
filter_servicepattern(gtfs, servicepattern = NULL)

filter_date(gtfs, dates = NULL)

filter_service(gtfs, service)

filter_route(gtfs, route, keep = TRUE)

filter_trip(gtfs, trip, keep = TRUE)

filter_stop(gtfs, stop)

filter_time(gtfs, from = "00:00:00", to = "48:00:00")
```

Arguments

`gtfs`	A GTFS object.
`servicepattern`	Character vector of service-pattern IDs returned by [get_servicepattern()]. A pattern represents a distinct set of services active on a date. When ‘NULL‘, the most frequent active pattern is used. The synthetic ‘"No service"‘ pattern cannot be used to filter trips.
`dates`	Dates to retain. Values accepted by [as.Date()] are supported. When ‘NULL‘, the latest available service date is used.
`service`, `route`, `trip`, `stop`	Character vectors of IDs to retain.
`keep`	Logical. If ‘FALSE‘, the specified route or trip IDs are excluded.
`from`, `to`	Inclusive GTFS time bounds in ‘"HH:MM:SS"‘ form. Hours may exceed 24.

Details

‘filter_stop()‘ retains the requested stop calls and therefore may return partial trips. ‘filter_time()‘ retains individual stop calls whose arrival or departure falls inside the inclusive time interval and may also return partial trips. This behavior is useful for network experiments. Route, trip, service, service-pattern, and date filters retain complete trips. Parent stations referenced by retained stops are preserved. Before applying a time filter, frequency-based trips are expanded at their headway intervals with generated trip IDs, so each retained departure can have its own partial stop sequence.

The expanded trips replace their ‘frequencies‘ rows. For ‘exact_times = 0‘, these are nominal departures for analysis rather than guaranteed timetable departures.

‘filter_date()‘ rewrites service availability as ‘calendar_dates‘ additions for exactly the selected dates. Other filters preserve the selected services’ original date ranges and exceptions.

Value

A ‘wizardgtfs‘ object.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/)

See Also

[GTFSwizard::as_wizardgtfs()], [GTFSwizard::get_servicepattern()]

Examples

```
typical <- filter_servicepattern(for_rail_gtfs)
one_day <- filter_date(for_rail_gtfs, "2021-02-10")
one_route <- filter_route(for_rail_gtfs, for_rail_gtfs$routes$route_id[1])
two_trips <- filter_trip(for_rail_gtfs, for_rail_gtfs$trips$trip_id[1:2])
serving_stop <- filter_stop(for_rail_gtfs, for_rail_gtfs$stops$stop_id[1])
morning <- filter_time(for_rail_gtfs, from = "06:30:00", to = "10:00:00")
```

---

for_bus_gtfs *GTFS Data for Fortaleza (Bus System), Brazil*

Description

A dataset containing GTFS (General Transit Feed Specification) data for Fortaleza’s transit system by bus. The data includes information on routes, trips, stops, stop times, and other elements necessary for transit planning and analysis.

Format

An object of class `wizardgtfs`, containing multiple data frames:

agency Data frame with 1 row and 7 columns, providing information about the transit agency, including agency name, URL, timezone, and contact details.

calendar Data frame with 3 rows and 10 columns, detailing service availability by day of the week, start and end dates for each service.

fare_attributes Data frame with 2 rows and 6 columns, showing fare information, including price, currency, payment method, and transfer rules.

fare_rules Data frame with 345 rows and 5 columns, linking fare IDs to routes, along with optional restrictions on origins, destinations, and zones.

routes Data frame with 345 rows and 9 columns, listing route details such as route ID, agency ID, route short and long names, route type, and colors.

shapes Data frame with 125,776 rows and 5 columns, representing the spatial paths of routes with latitude, longitude, point sequence, and cumulative distance traveled.

stop_times Data frame with 2,659,737 rows and 9 columns, including stop times for each trip, with arrival and departure times, stop sequence, and stop ID information.

stops Data frame with 4,676 rows and 12 columns, containing information about each stop, including stop ID, name, location (latitude and longitude), and accessibility.

trips Data frame with 85,410 rows and 9 columns, detailing trips associated with routes, including trip IDs, route IDs, direction, block, and shape IDs.

Details

The GTFS data format is widely used for representing public transportation schedules and associated geographic information. This dataset follows the GTFS standard and includes elements for advanced analysis in transit planning.

Source

Fortaleza transit agency (ETUFOR).

Examples

```
# Load the dataset
data(for_bus_gtfs)

# Access trips data
head(for_bus_gtfs$trips)

# Access stops data
head(for_bus_gtfs$stops)
```

---

for_rail_gtfs *GTFS Data for Fortaleza (Rail System), Brazil*

Description

This dataset contains GTFS (General Transit Feed Specification) data for Fortaleza's rail transit system, managed by METROFOR. The data includes information on routes, trips, stops, stop times, shapes, and other necessary elements for transit analysis and planning.

Format

An object of class `wizardgtfs`, consisting of multiple data frames:

agency Data frame with 1 row and 7 columns, providing information about the transit agency, including agency name, URL, timezone, language, and contact details.

calendar Data frame with 1 row and 10 columns, detailing the service availability by day of the week, along with start and end dates for each service.

calendar_dates Data frame with 26 rows and 3 columns, listing specific dates and exceptions (e.g., holidays) that modify the usual service pattern.

routes Data frame with 3 rows and 9 columns, listing route details such as route ID, short and long names, route type, and colors associated with each route.

stops Data frame with 39 rows and 10 columns, containing information about each stop, including stop ID, name, location (latitude and longitude), and additional details.

stop_times Data frame with 3,420 rows and 10 columns, detailing arrival and departure times for each trip, along with stop sequences and stop IDs.

trips Data frame with 215 rows and 7 columns, providing trip-specific information such as trip ID, headsign, direction, associated service ID, route ID, and shape ID.

shapes Data frame with 80 rows and 5 columns, representing spatial paths of routes using latitude, longitude, point sequence, and cumulative distance traveled.

Details

The GTFS data format is widely adopted for representing public transportation schedules and spatial information. This dataset follows GTFS standards and is tailored for advanced analysis, particularly in transit planning and operations. Key tables included are ‘agency‘, ‘routes‘, ‘stops‘, ‘stop_times‘, ‘trips‘, and ‘shapes‘, each providing essential attributes for a comprehensive transit analysis.

Source

Cia Cearense de Transportes Metropolitanos (METROFOR).

Examples

```
# Load the dataset
data(for_rail_gtfs)

# Access trips data
head(for_rail_gtfs$trips)

# Access stops data
head(for_rail_gtfs$stops)
```

get_1stdeparture *Get the First Departure of Each Trip*

Description

Returns the departure at the lowest ‘stop_sequence‘ for every trip.

Usage

```
get_1stdeparture(gtfs)
```

Arguments

gtfs	A GTFS object.

Value

A tibble with ‘route_id‘, ‘trip_id‘, ‘departure_time‘, and ‘stop_id‘.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/#stop_timestxt)

Examples

```
head(get_1stdeparture(for_rail_gtfs))
```

get_corridor *Identify High-Frequency Transit Corridors*

Description

Finds frequently served adjacent stop pairs, joins connected pairs into corridors, and measures them in a local metric CRS.

Usage

```
get_corridor(gtfs, i = 0.01, min_length = 1500, ...)
```

Arguments

gtfs	A GTFS object.
i	Proportion of the highest-frequency stop pairs to retain. Must be greater than 0 and no greater than 1.
min_length	Minimum corridor length in meters.
...	Supports the legacy argument ‘min.length‘.

Value

An ‘sf‘ object with ‘corridor‘, list-columns ‘stop_id‘ and ‘trip_id‘, numeric ‘length‘ in meters, and WGS84 geometry. Segments are straight lines between stop coordinates, not shape geometry.

See Also

[GTFSwizard::plot_corridor()]

Examples

```
corridors <- get_corridor(for_rail_gtfs, i = 0.2, min_length = 100)
```

---

get_distances *Calculate Consistent Trip and Segment Distances*

Description

Calculate distances in metres using the same segments for every summary.

Usage

```
get_distances(
  gtfs,
  method = "by_trip",
  trips = "all",
  distance_method = c("shapes", "straight")
)
```

Arguments

gtfs	A GTFS object.
method	One of ‘"by_trip"‘, ‘"by_route"‘, or ‘"detailed"‘.
trips	Character trip IDs or ‘"all"‘.
distance_method	Either ‘"shapes"‘ (default) or ‘"straight"‘.

Details

Shape distances use stop and shape ‘shape_dist_traveled‘ values together when available, converting their common units to metres using the shape geometry. Otherwise stops are projected onto shape segments in sequence order. This is an approximation, especially on loops; supplied distance measures are preferred. Missing shapes fall back to straight geodesic segments with a warning. Trip lengths cover only served segments, rather than the complete shared shape. Straight distances are geodesic.

Value

A tibble. Detailed rows identify trips and their originating ‘stop_sequence‘; trip and route summaries retain service patterns.

Examples

```
get_distances(for_rail_gtfs, "by_trip")
```

`get_durations` *Calculate Trip Durations in GTFS Data*

Description

Calculates scheduled trip and segment durations in seconds.

Usage

```
get_durations(gtfs, method = "by_trip", trips = "all")
```

Arguments

`gtfs`	A GTFS object, ideally of class ‘wizardgtfs‘. If not, it will be converted.
`method`	A character string specifying the calculation method. Options include:
	"by_route" Calculates the average duration for each route.
	"by_trip" Calculates the total duration for each trip.
	"detailed" Calculates detailed durations for each stop-to-stop segment within a trip.
`trips`	A character vector of trip IDs to consider. When set to ‘all‘, includes all trips.

Details

This function calls specific sub-functions based on the selected method:

- "by_route": Calculates average durations for each route.

- "by_trip": Calculates the total duration of each trip.

- "detailed": Calculates detailed durations between consecutive stops within each trip, excluding dwell times.

Legacy dotted values remain accepted. If an invalid ‘method‘ is specified, the function defaults to ‘"by_trip"‘ and provides a warning.

Value

A data frame containing trip durations based on the specified method:

If ‘method = "by_route"‘ Columns ‘route_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, ‘average.duration‘, and ‘trips‘.

If ‘method = "by_trip"‘ Columns ‘route_id‘, ‘trip_id‘, optional ‘direction_id‘, ‘duration‘, ‘service_pattern‘, and ‘pattern_frequency‘.

If ‘method = "detailed"‘ Excludes dwell time and returns ‘route_id‘, ‘trip_id‘, optional ‘direction_id‘, next ‘arrival_time‘, departure ‘hour‘, ‘from_stop_id‘, ‘to_stop_id‘, ‘duration‘, ‘service_pattern‘, and ‘pattern_frequency‘.

See Also

[GTFSwizard::as_wizardgtfs()], [GTFSwizard::get_servicepattern()]

Examples

```
# Calculate average route durations
durations_by_route <- get_durations(gtfs = for_rail_gtfs, method = "by_route", trips = 'all')

# Calculate trip durations
durations_by_trip <- get_durations(gtfs = for_rail_gtfs, method = "by_trip", trips = 'all')

# Calculate detailed durations between stops
detailed_durations <- get_durations(gtfs = for_rail_gtfs, method = "detailed", trips = 'all')
```

---

get_dwelltimes *Calculate Dwell Times in GTFS Data*

Description

The ‘get_dwelltimes‘ function calculates dwell times within a ‘wizardgtfs‘ object using different methods. Depending on the selected ‘method‘, it can provide average dwell times per route, per trip, by hour, or detailed dwell times at each stop.

Usage

```
get_dwelltimes(gtfs, max_dwelltime = 90, method = "by_trip", ...)
```

Arguments

gtfs	A GTFS object, ideally of class ‘wizardgtfs‘. If not, it will be converted.
max_dwelltime	Numeric. Maximum allowable dwell time in seconds.
method	A character string specifying the calculation method. Options include:
	"by_hour" Calculates the average dwell time per hour of the day across all trips.

"by_route" Calculates the average dwell time for each route.

"by_trip" Calculates the average dwell time for each trip.

"detailed" Calculates detailed dwell times at each stop within every trip.

... Supports the legacy argument ‘max.dwelltime‘.

Details

This function calls specific sub-functions based on the selected method:

- "by_hour": Calculates the average dwell time for each hour of the day.

- "by_route": Calculates average dwell times across each route.

- "by_trip": Calculates the mean dwell time for each trip.

- "detailed": Calculates departure minus arrival at each stop call.

Legacy dotted values remain accepted. If an invalid ‘method‘ is specified, the function defaults to ‘"by_trip"‘ and provides a warning.

Value

A data frame containing dwell times based on the specified method:

If ‘method = "by_hour"‘ Returns a data frame with columns: ‘hour‘, ‘trips‘, ‘average.dwelltime‘, ‘service_pattern‘, and ‘pattern_frequency‘.

If ‘method = "by_route"‘ Returns ‘route_id‘, optional ‘direction_id‘, ‘trips‘, ‘average.dwelltime‘, ‘service_pattern‘, and ‘pattern_frequency‘.

If ‘method = "by_trip"‘ Returns ‘route_id‘, ‘trip_id‘, optional ‘direction_id‘, ‘average.dwelltime‘, ‘service_pattern‘, and ‘pattern_frequency‘.

If ‘method = "detailed"‘ Returns ‘route_id‘, ‘trip_id‘, optional ‘direction_id‘, ‘stop_id‘, ‘hour‘, ‘dwell_time‘, ‘service_pattern‘, and ‘pattern_frequency‘.

See Also

[GTFSwizard::as_wizardgtfs()], [GTFSwizard::get_servicepattern()]

Examples

```
# Calculate dwell times by hour
dwelltimes_by_hour <- get_dwelltimes(gtfs = for_rail_gtfs, max_dwelltime = 120, method = "by_hour")

# Calculate dwell times by route
dwelltimes_by_route <- get_dwelltimes(gtfs = for_rail_gtfs, max_dwelltime = 90, method = "by_route")

# Calculate dwell times by trip
dwelltimes_by_trip <- get_dwelltimes(gtfs = for_rail_gtfs, max_dwelltime = 45, method = "by_trip")

# Calculate detailed dwell times between stops
detailed_dwelltimes <- get_dwelltimes(gtfs = for_rail_gtfs, max_dwelltime = 60, method = "detailed")
```

`get_fleet` *Estimate Simultaneous Vehicles*

Description

Estimates the number of active trip instances from their first departure to final arrival. Frequency-based services are expanded from ‘frequencies.txt‘.

Usage

```
get_fleet(gtfs, method = "by_route")
```

Arguments

`gtfs`	A GTFS object.
`method`	One of ‘"by_route"‘, ‘"by_hour"‘, ‘"peak"‘, or ‘"detailed"‘. Legacy dotted values remain accepted.

Value

A tibble whose observational unit depends on ‘method‘:

‘"by_route"‘ ‘route_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, and maximum simultaneous ‘fleet‘.

‘"by_hour"‘ ‘service_pattern‘, ‘pattern_frequency‘, ‘hour‘, and maximum simultaneous ‘fleet‘.

‘"peak"‘ The three highest hourly fleet estimates per service pattern, with the same columns as ‘"by_hour"‘.

‘"detailed"‘ ‘route_id‘, optional ‘direction_id‘, vehicle-change event ‘net.fleet‘, cumulative ‘fleet‘, event ‘time‘ in seconds, ‘service_pattern‘, and ‘pattern_frequency‘.

GTFS times beyond 24 hours remain in the following service day rather than being wrapped.

Examples

```
get_fleet(for_rail_gtfs, "by_route")
get_fleet(for_rail_gtfs, "by_hour")
```

`get_frequency` *Calculate Scheduled Service Frequency*

Description

Counts trip departures by route, shape, stop, or hour. Trips referenced by ‘frequencies.txt‘ are expanded using the period’s inclusive ‘start_time‘, exclusive ‘end_time‘, and ‘headway_secs‘, as required by GTFS.

Usage

```
get_frequency(gtfs, method = "by_trip")
```

Arguments

`gtfs`	A GTFS object.
`method`	One of ‘"by_trip"‘, ‘"by_route"‘, ‘"by_shape"‘, ‘"by_stop"‘, or ‘"detailed"‘. Legacy dotted values remain accepted.

Value

A tibble whose observational unit depends on ‘method‘:

‘"by_trip"‘ ‘route_id‘, ‘trip_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, and ‘daily.frequency‘.

‘"by_route"‘ ‘route_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, and ‘daily.frequency‘.

‘"by_shape"‘ ‘shape_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, and ‘daily.frequency‘.

‘"by_stop"‘ ‘stop_id‘, optional ‘direction_id‘, ‘service_pattern‘, ‘pattern_frequency‘, and ‘daily.frequency‘.

‘"detailed"‘ ‘route_id‘, optional ‘direction_id‘, departure ‘hour‘, ‘service_pattern‘, ‘pattern_frequency‘, and ‘frequency‘.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/#frequenciestxt)

See Also

[GTFSwizard::get_headways()]

Examples

```
get_frequency(for_rail_gtfs, "by_route")
get_frequency(for_rail_gtfs, "detailed")
```

`get_headways` *Calculate Scheduled Headways*

Description

Calculates elapsed minutes between successive service instances. Frequency based trips are expanded according to ‘frequencies.txt‘.

Usage

```
get_headways(gtfs, method = "by_trip")
```

Arguments

`gtfs`	A GTFS object.
`method`	One of ‘"by_trip"‘, ‘"by_route"‘, ‘"by_hour"‘, ‘"by_stop"‘, ‘"by_shape"‘, or ‘"detailed"‘. Legacy dotted values remain accepted.

Details

Route, hour, and trip methods compare first departures. Stop and detailed methods compare arrivals at the same stop, route, direction, and service pattern. The first service instance in each group has no headway and is omitted.

Value

A tibble. Headway values are always returned in ‘headway_minutes‘; ‘valid_trips‘ is the number of intervals summarized.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/#frequenciestxt)

See Also

[GTFSwizard::get_frequency()]

Examples

```
get_headways(for_rail_gtfs, "by_route")
get_headways(for_rail_gtfs, "by_hour")
```

get_hubs *Identify Stops Serving Multiple Routes*

Description

Identify Stops Serving Multiple Routes

Usage

```
get_hubs(gtfs)
```

Arguments

gtfs	A GTFS object.

Value

A WGS84 ‘sf‘ object with one row per served stop, list-columns ‘trip_id‘ and ‘route_id‘, counts ‘n_trip‘ and ‘n_routes‘, and point geometry.

See Also

[GTFSwizard::plot_hubs()], [GTFSwizard::get_stops_sf()]

Examples

```
get_hubs(for_rail_gtfs)
```

get_servicepattern *Identify Recurring Daily Service Patterns*

Description

Identifies each distinct set of ‘service_id‘ values active on a calendar date. The same service may belong to more than one pattern when it operates alongside different services on different dates.

Usage

```
get_servicepattern(gtfs)
```

Arguments

gtfs	A GTFS object.

Value

A tibble with one row per service-to-pattern association and columns ‘service_id‘, ‘service_pattern‘, and ‘pattern_frequency‘. The frequency is the number of dates having that exact set of active services. Pattern numbers are ordered from most to least frequent. Calendar dates without active services are represented by ‘service_pattern = "No service"‘ and ‘service_id = NA‘.

See Also

[GTFSwizard::filter_servicepattern()]

Examples

```
get_servicepattern(for_rail_gtfs)
```

---

get_shapes *Infer Straight-Line Shapes from Stop Sequences*

Description

Builds one shape for each unique ordered stop pattern and assigns it to the corresponding trips.

Usage

```
get_shapes(gtfs)
```

Arguments

gtfs	A GTFS object.

Details

Consecutive stop coordinates are connected directly. The result is useful for visualization and approximate distance analysis, but it is not a map-matched representation of the vehicle path. Existing stop distance measures are recalculated in metres to agree with the inferred shapes.

Value

A ‘wizardgtfs‘ object with a new ‘shapes‘ table and updated ‘trips$shape_id‘.

See Also

[GTFSwizard::get_shapes_sf()], [GTFSwizard::get_shapes_df()]

Examples

```
gtfs_with_shapes <- get_shapes(for_rail_gtfs)
```

get_shapes_df *Convert Shape Geometries to a GTFS Shapes Table*

Description

Convert Shape Geometries to a GTFS Shapes Table

Usage

```
get_shapes_df(shape)
```

Arguments

shape	An ‘sf‘ object with a ‘shape_id‘ field and line geometries.

Details

Distances are geodesic approximations between consecutive coordinates. GTFS permits any consistent distance unit; this function uses meters.

Value

A tibble containing ‘shape_id‘, WGS84 point coordinates, ‘shape_pt_sequence‘, and cumulative ‘shape_dist_traveled‘ in meters.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/#shapestxt)

See Also

[GTFSwizard::get_shapes_sf()], [GTFSwizard::get_shapes()]

Examples

```
shapes_sf <- get_shapes_sf(for_rail_gtfs)$shapes
shapes_df <- get_shapes_df(shapes_sf)
```

get_shapes_sf *Convert GTFS Shapes to Simple Features*

Description

Convert GTFS Shapes to Simple Features

Usage

```
get_shapes_sf(gtfs)
```

Arguments

gtfs	A GTFS object or a ‘shapes‘ data frame.

Value

For a data frame, one WGS84 ‘LINESTRING‘ feature per ‘shape_id‘. For a GTFS object, the same object with its ‘shapes‘ table replaced by that ‘sf‘ object. If present, ‘shape_dist_traveled‘ is summarized as its maximum value for each shape.

See Also

[GTFSwizard::get_shapes_df()], [GTFSwizard::get_shapes()]

Examples

```
gtfs_sf <- get_shapes_sf(for_rail_gtfs)
shapes_sf <- get_shapes_sf(for_rail_gtfs$shapes)
```

get_speeds *Calculate Scheduled Speeds*

Description

Combines distance in meters and duration in seconds to calculate kilometers per hour.

Usage

```
get_speeds(
  gtfs,
  method = "by_trip",
  trips = "all",
  distance_method = c("shapes", "straight")
)
```

Arguments

gtfs	A GTFS object.
method	One of ‘"by_trip"‘, ‘"by_route"‘, or ‘"detailed"‘. Legacy dotted values remain accepted.
trips	Character trip IDs or ‘"all"‘.
distance_method	Distance calculation passed to [get_distances()].

Details

Every summary uses the same segment geometry selected by ‘distance_method‘.

Value

A tibble with ‘average.speed‘ for route and trip methods, or segment-level ‘speed‘ for the detailed method. ‘direction_id‘ is retained when available.

See Also

[GTFSwizard::get_distances()], [GTFSwizard::get_durations()]

Examples

```
get_speeds(for_rail_gtfs, "by_route")
get_speeds(for_rail_gtfs, "by_trip")
```

---

get_stops_sf *Convert GTFS Stops to Simple Features*

Description

Convert GTFS Stops to Simple Features

Usage

```
get_stops_sf(gtfs)
```

Arguments

gtfs	A GTFS object or a ‘stops‘ data frame.

Value

For a data frame, a WGS84 point ‘sf‘ object. For a GTFS object, the same object with its ‘stops‘ table converted to ‘sf‘.

Examples

```
gtfs_sf <- get_stops_sf(for_rail_gtfs)
stops_sf <- get_stops_sf(for_rail_gtfs$stops)
```

---

latlon2epsg *Transform Spatial Data to a Local Metric CRS*

Description

Selects a UTM zone from the geographic centroid, or a polar stereographic CRS outside UTM's latitude range.

Usage

```
latlon2epsg(sf_obj)
```

Arguments

sf_obj	An ‘sf‘ or ‘sfc‘ object with a defined CRS.

Value

The input transformed to EPSG 326xx/327xx, EPSG 3413, or EPSG 3031.

See Also

[sf::st_transform()]

Examples

```
shapes <- get_shapes_sf(for_rail_gtfs)$shapes
metric_shapes <- latlon2epsg(shapes)
```

---

merge_gtfs *Merge Two GTFS Feeds*

Description

Combines two feeds table by table. By default, all identifiers and their foreign-key references receive feed-specific suffixes, preventing accidental collisions.

Usage

```
merge_gtfs(gtfs_x = NULL, gtfs_y = NULL, suffix = TRUE, ...)
```

Arguments

gtfs_x, gtfs_y	GTFS objects.
suffix	Logical. Append '.x' and '.y' to identifiers when 'TRUE'.
...	Supports the legacy arguments 'gtfs.x' and 'gtfs.y'.

Details

Standard references in routes, trips, stop times, shapes, calendars, frequencies, transfers, pathways, fare tables, networks, and booking rules are updated together. Non-standard tables are row-bound without identifier rewriting.

Value

A validated 'wizardgtfs' object.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/)

Examples

```
merged <- merge_gtfs(for_rail_gtfs, for_rail_gtfs, suffix = TRUE)
```

---

plot_calendar *Plot the GTFS Service Calendar*

Description

Creates a calendar heatmap of scheduled trips by service date.

Usage

```
plot_calendar(
  gtfs,
  ncol = 4,
  facet_by_year = FALSE,
  fill = c("trips", "service_pattern"),
  max_patterns = 20L
)
```

Arguments

gtfs	A GTFS object.
ncol	Number of facet columns when ‘facet_by_year = FALSE‘.
facet_by_year	Logical. Arrange years in rows and months in columns.
fill	Calendar fill. ‘"trips"‘ shows trip counts and draws dates with no active service as zero trips; ‘"service_pattern"‘ uses a discrete color for each service pattern and labels dates with no active service as ‘"No service"‘.
max_patterns	Maximum number of service patterns given individual colors when ‘fill = "service_pattern"‘. Remaining patterns share an ‘"Other service patterns"‘ category. Dates and original pattern IDs remain available in the plot data.

Value

A ‘ggplot‘ object.

See Also

[GTFSwizard::get_servicepattern()]

Examples

```
plot_calendar(for_rail_gtfs, ncol = 4)
plot_calendar(
  for_rail_gtfs,
  facet_by_year = TRUE,
  fill = "service_pattern"
)
```

---

plot_corridor *Plot High-Frequency Corridors*

Description

Plot High-Frequency Corridors

Usage

```
plot_corridor(gtfs, i = 0.01, min_length = 1500, ...)
```

Arguments

gtfs	A GTFS object.
i	Proportion of highest-frequency segments to retain.
min_length	Minimum corridor length in meters.
...	Supports the legacy argument ‘min.length‘.

Value

A ‘ggplot‘ map.

See Also

[GTFSwizard::get_corridor()]

Examples

```
plot_corridor(for_rail_gtfs, i = 0.2, min_length = 100)
```

`plot_frequency` *Plot System Frequency by Hour*

Description

Shows the distribution and weighted mean of scheduled trip departures by hour. Service-pattern date counts provide the weights.

Usage

```
plot_frequency(gtfs)
```

Arguments

`gtfs`	A GTFS object.

Value

A ‘ggplot‘ object.

See Also

[GTFSwizard::get_frequency()]

Examples

```
plot_frequency(for_rail_gtfs)
```

plot_headways *Plot System Headway by Hour*

Description

Plot System Headway by Hour

Usage

```
plot_headways(gtfs, max_patterns = 12L)
```

Arguments

gtfs	A GTFS object.
max_patterns	Maximum number of the most frequent service patterns to display. Use 'Inf' to display all patterns.

Value

A 'ggplot' object with headway in minutes.

See Also

[GTFSwizard::get_headways()]

Examples

```
plot_headways(for_rail_gtfs)
```

plot_hubs *Plot Transit Hubs*

Description

Plot Transit Hubs

Usage

```
plot_hubs(gtfs, i = 0.05)
```

Arguments

gtfs	A GTFS object.
i	Proportion of stops with the most routes to retain.

Details

To keep dense networks readable, at most 40 of the highest-ranked stops are drawn. Ranking uses distinct route count and then trip count.

Value

A ‘ggplot‘ map.

See Also

[GTFSwizard::get_hubs()]

Examples

```
plot_hubs(for_rail_gtfs, i = 0.1)
```

---

plot_routeduration *Plot Scheduled Trip Duration by Route*

Description

Shows the distribution of scheduled end-to-end trip duration for the busiest routes.

Usage

```
plot_routeduration(gtfs, top_n = 20L, max_patterns = 12L)
```

Arguments

gtfs	A GTFS object.
top_n	Maximum number of routes to display.
max_patterns	Maximum number of the most frequent service patterns to display. Use ‘Inf‘ to display all patterns.

Value

A ‘ggplot‘ object. Each observation is one scheduled trip; boxes summarize duration in minutes by route.

References

[FTA Evaluation Introduction](https://www.transit.dot.gov/research-innovation/evaluation-introduction)

Examples

```
plot_routeduration(for_rail_gtfs)
```

plot_routefrequency *Plot Route Frequency by Hour*

Description

Creates a heatmap of scheduled departures by route, hour, and service pattern. The tile fill is the number of scheduled trips.

Usage

```
plot_routefrequency(gtfs, route = NULL, top_n = 25L, max_patterns = 12L)
```

Arguments

gtfs	A GTFS object.
route	Optional character vector of route IDs. ‘NULL‘ includes all routes after applying ‘top_n‘.
top_n	Maximum number of routes to display when ‘route‘ is ‘NULL‘.
max_patterns	Maximum number of the most frequent service patterns to display. Use ‘Inf‘ to display all patterns.

Value

A ‘ggplot‘ object.

See Also

[GTFSwizard::get_frequency()]

Examples

```
plot_routefrequency(
  for_rail_gtfs,
  route = for_rail_gtfs$routes$route_id[1:2]
)
```

plot_serviceheatmap *Plot Average Scheduled Departures by Weekday and Hour*

Description

Creates a heatmap of average scheduled departures on active service dates.

Usage

```
plot_serviceheatmap(gtfs)
```

Arguments

gtfs	A GTFS object.

Value

A 'ggplot' object. Each tile is a weekday-hour combination and its fill is the mean number of scheduled trip departures per active date.

References

[GTFS Schedule Reference](https://gtfs.org/documentation/schedule/reference/)

Examples

```
plot_serviceheatmap(for_rail_gtfs)
```

plot_servicespan *Plot Scheduled Service Span by Route*

Description

Shows the first departure and final arrival for route and service-pattern combinations. The longest and most frequently operated route-pattern combinations are retained when the feed is large. Frequency-based trips contribute each departure at its headway interval.

Usage

```
plot_servicespan(gtfs, top_n = 30L)
```

Arguments

gtfs	A GTFS object.
top_n	Maximum number of route-pattern combinations to display.

Value

A ‘ggplot‘ object. Each horizontal line is one route and service pattern; its endpoints are scheduled clock hours.

References

[GTFS Routes, Stops, and Trips](https://gtfs.org/documentation/schedule/examples/routes-stops-trips/)

Examples

```
plot_servicespan(for_rail_gtfs)
```

`plot_servicesupply` *Plot Scheduled Vehicle-Hours by Route*

Description

Sums scheduled end-to-end trip durations for each route and service pattern. Frequency-based trips contribute all departures at their headway intervals.

Usage

```
plot_servicesupply(gtfs, top_n = 20L, max_patterns = 12L)
```

Arguments

`gtfs`	A GTFS object.
`top_n`	Maximum number of routes to display.
`max_patterns`	Maximum number of the most frequent service patterns to display. Use ‘Inf‘ to display all patterns.

Value

A ‘ggplot‘ object. Each bar is scheduled vehicle-hours for one route on one representative service-pattern day.

References

[FTA National Transit Database](https://www.transit.dot.gov/ntd)

Examples

```
plot_servicesupply(for_rail_gtfs)
```

read_gtfs *Read a GTFS Feed*

Description

Imports a GTFS zip archive and converts it to `wizardgtfs`.

Usage

```
read_gtfs(
  file_path = NULL,
  files = NULL,
  quiet = TRUE,
  repair = c("legacy", "none"),
  ...
)
```

Arguments

`file_path`	Optional path to a GTFS `.zip` archive. When omitted in an interactive session, a file-selection window opens.
`files`	Optional character vector of table names without `.txt`.
`quiet`	Logical. Suppress importer messages and UTF-16 conversion warnings.
`repair`	Reference cleanup policy passed to `as_wizardgtfs()`.
`...`	Additional arguments passed to `gtfsio::import_gtfs()`, including the legacy argument `file.path`.

Details

UTF-16LE and UTF-16BE GTFS text files are converted to UTF-8 in a temporary copy before import. The original ZIP is not modified. When `quiet = FALSE`, a warning identifies the converted files. Other import arguments, including `fields`, `extra_spec`, and `skip`, are preserved.

Value

A validated `wizardgtfs` object.

See Also

`write_gtfs()`, `as_wizardgtfs()`

Examples

```
path <- tempfile(fileext = ".zip")
write_gtfs(for_rail_gtfs, path)
gtfs <- read_gtfs(path)
```

repair_gtfs *Explicitly Repair a GTFS Feed*

Description

Apply selected repairs and record every changed cell or removed row.

Usage

```
repair_gtfs(gtfs, repairs = "references", build_shapes = TRUE)
```

Arguments

gtfs	A named list of GTFS tables or a ‘wizardgtfs‘ object.
repairs	Character vector containing ‘"references"‘, ‘"overnight"‘, and/or ‘"stop_times"‘. Reference repair uses the legacy import cleanup. Overnight repair adds days only to backward jumps greater than 12 hours. Stop-time repair interpolates between known times using straight distances; no extrapolation is performed. Defaults to reference repair only.
build_shapes	Logical. Infer missing shapes after repair.

Details

Repairs are estimates. Review their report before exporting. Interpolation preserves known times and dwell, and sets ‘timepoint = 0‘ for estimated calls when that field exists. Missing endpoint times and intervals without usable coordinates remain missing. Overnight adjustments precede interpolation when both are requested.

Value

A validated ‘wizardgtfs‘. Its ‘repair_report‘ attribute is a tibble with table, original row, field, old value, new value, and action. Read it with [repair_report()]. The input remains unchanged.

Examples

```
fixed <- repair_gtfs(for_rail_gtfs, repairs = "references")
repair_report(fixed)
```

repair_report *Inspect Changes Made by Feed Repair*

Description

Inspect Changes Made by Feed Repair

Usage

```
repair_report(gtfs)
```

Arguments

gtfs	A feed returned by [repair_gtfs()], [read_gtfs()], or [as_wizardgtfs()].

Value

A tibble of recorded changes; empty when none were recorded.

selection *Group or Select GTFS Records Without Filtering the Feed*

Description

Creates persistent selection metadata while leaving every GTFS table unchanged. Bare columns and computed values define groups, similarly to [dplyr::group_by()]. Logical expressions restrict the records represented by those groups.

Usage

```
selection(gtfs, ..., add = FALSE)

unselection(gtfs)
```

Arguments

gtfs	A GTFS object.
...	Unquoted grouping columns, computed grouping expressions, or logical selection expressions. Columns from ‘stop_times‘, ‘trips‘, ‘routes‘, and ‘stops‘ are available. A ‘geometry‘ column is available for stop-based spatial predicates.
add	Logical. When ‘TRUE‘, add the new grouping or selection expressions to an existing selection. When ‘FALSE‘, replace it.

Details

A bare expression such as ‘route_id‘ creates one group per route. Multiple grouping expressions create combinations, for example ‘selection(gtfs, route_id, direction_id)‘. Logical expressions such as ‘route_id %in% c("1", "2")‘ restrict the records represented in the selection but do not remove rows from the GTFS feed.

Spatial expressions may use ‘geometry‘ with ‘%intersects%‘, ‘%touches%‘, ‘%crosses%‘, ‘%within%‘, ‘%contains%‘, ‘%overlaps%‘, or ‘%equals%‘.

Value

The unchanged feed with class ‘wizardgtfs_selected‘ and a ‘selection‘ attribute. The attribute contains ‘groups‘, ‘group_vars‘, selected route, trip, and stop IDs, and the original expressions.

See Also

[GTFSwizard::unselection()], [dplyr::group_by()]

Examples

```
grouped <- selection(for_rail_gtfs, route_id, direction_id)
attr(grouped, "selection")$groups

selected <- selection(
  for_rail_gtfs,
  route_id,
  stop_id %in% for_rail_gtfs$stops$stop_id[1:3]
)

bbox <- sf::st_bbox(
  c(xmin = -38.58, ymin = -3.81, xmax = -38.50, ymax = -3.75),
  crs = sf::st_crs(4326)
)
spatial <- selection(
  for_rail_gtfs,
  geometry %intersects% sf::st_as_sfc(bbox)
)
```

---

set_dwelltime *Set Dwell Times*

Description

Sets dwell time at selected trip-stop calls and propagates each change to all later times in the same trip. Arrival at the edited stop is retained. Edited times that would move two or more days after midnight are wrapped back to the equivalent time on the next service day while keeping stop times non-decreasing within each trip.

Usage

```
set_dwelltime(gtfs, duration = 30, trips = "all", stops = "all")
```

Arguments

gtfs	A GTFS object.
duration	One non-negative dwell time in seconds.
trips, stops	Character ID vectors or ‘"all"‘.

Value

A modified ‘wizardgtfs‘ object.

See Also

[GTFSwizard::edit_dwelltime()], [GTFSwizard::get_dwelltimes()]

Examples

```
edited <- set_dwelltime(
  for_rail_gtfs,
  duration = 30,
  trips = for_rail_gtfs$trips$trip_id[1:2],
  stops = for_rail_gtfs$stops$stop_id[1:2]
)
```

---

split_trip *Split Trips into Consecutive Parts*

Description

Splits each selected trip into consecutive parts. Trips can be split into an approximately even number of parts with ‘split‘, or at specific internal stop IDs with ‘stops‘. The boundary stop is included at the end of one part and the start of the next, producing valid complete stop sequences.

Usage

```
split_trip(gtfs, trip, split = 1L, stops = NULL)
```

Arguments

gtfs	A GTFS object.
trip	Character vector of ‘trip_id‘ values.
split	Positive integer number of split points. ‘split = 1‘ creates two parts. For each trip, the maximum is the number of stop-time records minus two, so every resulting part contains at least two stops. Used when ‘stops = NULL‘.

`stops` Optional character vector of stop IDs where the trip should be split. When supplied, 'split' must be omitted. Each internal occurrence of these stops becomes a split boundary. First and last stops cannot be used as split boundaries because they would create one-stop parts.

Details

New IDs use '.part1', '.part2', and so on. New straight-line shapes are inferred from stop coordinates for the split parts. Frequency periods are shifted by each part's offset from the original first departure. Trip-level transfers are reassigned to the part containing their transfer stop. Trips with fewer than three retained stop-time records cannot be split. When using 'stops', each selected trip must contain at least one matching internal stop.

Value

A modified 'wizardgtfs' object.

See Also

[GTFSwizard::get_shapes()], [GTFSwizard::merge_gtfs()]

Examples

```
gtfs_split <- split_trip(
  for_rail_gtfs,
  trip = for_rail_gtfs$trips$trip_id[1],
  split = 2
)

first_trip_stops <- for_rail_gtfs$stop_times[
  for_rail_gtfs$stop_times$trip_id == for_rail_gtfs$trips$trip_id[1],
]
gtfs_split_at_stop <- split_trip(
  for_rail_gtfs,
  trip = for_rail_gtfs$trips$trip_id[1],
  stops = first_trip_stops$stop_id[2]
)
```

---

tidy_raptor *Calculate Travel Times with RAPTOR Algorithm*

Description

The 'tidy_raptor' function calculates travel times from a set of origin stops to all reachable stops within a GTFS dataset. It uses the RAPTOR (Round-Based Public Transit Routing) algorithm from the 'tidytransit' package and integrates it with the GTFSwizard framework.

Usage

```
tidy_raptor(
  gtfs,
  min_departure = "0:0:0",
  max_arrival = "23:59:59",
  dates = NULL,
  stop_ids,
  arrival = FALSE,
  time_range = 3600,
  max_transfers = NULL,
  keep = "all",
  filter = TRUE,
  separate_starts = FALSE
)
```

Arguments

gtfs	A GTFS feed, optionally of class ‘wizardgtfs‘.
min_departure	A string representing the earliest departure time, in "HH:MM:SS" format. Defaults to ‘"0:0:0"‘.
max_arrival	A string representing the latest arrival time, in "HH:MM:SS" format. Defaults to ‘"23:59:59"‘.
dates	One service date. When ‘NULL‘, the latest active service date is used.
stop_ids	A character vector of stop IDs from where journeys should start (or end, if ‘arrival = TRUE‘).
arrival	Logical. If ‘FALSE‘ (default), journeys start from ‘stop_ids‘. If ‘TRUE‘, journeys end at ‘stop_ids‘.
time_range	Either a range in seconds (numeric) or a vector with the minimal and maximal departure time (e.g., ‘c(0, 3600)‘ or ‘"HH:MM:SS"‘) describing the journey window.
max_transfers	Maximum number of transfers allowed. Defaults to ‘NULL‘ (no limit).
keep	One of ‘"all"‘, ‘"shortest"‘, ‘"earliest"‘, or ‘"latest"‘. Determines which journeys to retain: - ‘"all"‘: All journeys are returned (default). - ‘"shortest"‘: Only journeys with the shortest travel time. - ‘"earliest"‘: Journeys arriving at stops the earliest. - ‘"latest"‘: Journeys arriving at stops the latest.
filter	Logical. Apply ‘min_departure‘, ‘max_arrival‘, and ‘dates‘ before routing.
separate_starts	Logical. Keep results for each origin start time separately; passed to [tidytransit::raptor()].

Details

With ‘filter = TRUE‘, the service date is selected before frequency trips are expanded. Stop calls are kept when their arrival or departure falls within the inclusive time bounds. With ‘filter = FALSE‘, all services are combined. Departures are nominal when ‘exact_times = 0‘.

Value

A tibble containing the RAPTOR algorithm results, including:

from_stop_id The ID of the stop where the journey starts.

to_stop_id The ID of the stop where the journey ends.

departure_time Departure time from the origin stop.

arrival_time Arrival time at the destination stop.

travel_time Total travel time in seconds.

transfers Number of transfers in the journey.

Note

Ensure that the ‘stop_times‘ is present and correctly structured in the GTFS dataset. Time values in ‘min_departure‘, ‘max_arrival‘, and ‘time_range‘ should be correctly formatted to avoid errors. GTFS times beyond 24:00:00 are supported.

See Also

[tidytransit::raptor()], [GTFSwizard::as_wizardgtfs()], [GTFSwizard::filter_time()]

Examples

```
tidy_raptor(for_rail_gtfs,
   min_departure = '06:20:00',
   max_arrival = '09:40:00',
   dates = "2021-12-13",
   max_transfers = 2,
   keep = "all",
   stop_ids = '66')
```

---

validate_gtfs *Validate a GTFS Schedule Feed*

Description

Run the MobilityData Canonical GTFS Schedule Validator on a ZIP archive or a GTFS object and return its findings. The validator checks the official GTFS Schedule specification and produces additional best-practice notices.

Usage

```
validate_gtfs(
  gtfs,
  validator_jar = NULL,
  output_dir = NULL,
  quiet = FALSE,
  download_validator = FALSE
)
```

Arguments

gtfs	Path to a local GTFS ZIP archive or a named list of GTFS tables, including a wizardgtfs object.
validator_jar	Optional path to a MobilityData validator CLI JAR. When NULL, use the path in GTFS_VALIDATOR_JAR. The JAR must support --skip_validator_update.
output_dir	Optional directory in which to keep the validator's reports. It must not exist or must be empty. When NULL, temporary files are removed after validation and no reports are saved.
quiet	Logical. If FALSE (the default), print a summary of the findings. Set to TRUE to return the result without printing.
download_validator	Logical. If TRUE and no JAR path is supplied, download the official validator to the user's cache if necessary and reuse it on subsequent calls. Requires internet access on the first download and the suggested package openssl for SHA-256 verification.

Details

Requires Java 17 or newer and the suggested package jsonlite. Supply a MobilityData CLI JAR from https://github.com/MobilityData/gtfs-validator/releases (version 8.0.1 or newer recommended), or explicitly set download_validator = TRUE to download version 8.0.1. The cached download's SHA-256 is checked against the published release checksum before each use. The ZIP is passed directly to the validator without importing or repairing it. An object is first exported with write_gtfs(); its report describes the exported representation, not the original file or any previously repaired data. No data are uploaded by this function. Results depend on the validator version and its implemented rules; no errors found is not a certification.

The output directory must be empty to avoid overwriting existing reports or confusing a failed execution with a previous successful one.

Value

A list with class gtfs_validation: notices, a data frame with rule codes, severities, occurrence counts and sample details; system_notices, a data frame of system errors; report and system_errors, the parsed JSON reports; has_errors, TRUE or FALSE when validation completed with recognized severities and without system errors, otherwise NA; and report_path, system_errors_path, html_path, and output_dir, which are NULL when reports are not saved. The result is returned invisibly.

See Also

read_gtfs(), write_gtfs(), https://gtfs.org/getting-started/validate/

Examples

```
# Requires Java 17 or newer, jsonlite and openssl, and initially internet:
## Not run:
result <- validate_gtfs(for_rail_gtfs, download_validator = TRUE)
result$notices
```

```
# Supply output_dir to keep the HTML and JSON reports:
result <- validate_gtfs(for_rail_gtfs, output_dir = "validation-report",
                        download_validator = TRUE)

## End(Not run)
```

`wizardgtfs-methods` *Print, Summarize, and Plot ‘wizardgtfs‘ Objects*

Description

Print, Summarize, and Plot ‘wizardgtfs‘ Objects

Usage

```
## S3 method for class 'wizardgtfs'
print(x, ..., n = 5L)

## S3 method for class 'wizardgtfs'
summary(object, ...)

## S3 method for class 'summary.wizardgtfs'
print(x, ...)

## S3 method for class 'wizardgtfs'
plot(x, ...)
```

Arguments

`x, object`	A ‘wizardgtfs‘ object or its summary.
`...`	Additional arguments passed to print methods.
`n`	Number of rows shown per GTFS table.

Value

‘print()‘ returns ‘x‘ invisibly; ‘summary()‘ returns a ‘summary.wizardgtfs‘ object; ‘plot()‘ returns a ‘ggplot‘ object.

`write_gtfs` *Write a GTFS Feed*

Description

Exports a GTFS object to a zip archive. Internal ‘dates_services‘ data are omitted, GTFS dates use ‘YYYYMMDD‘, and spatial stops/shapes are converted back to standard text-table columns.

Usage

```
write_gtfs(gtfs, zipfile = NULL, ...)
```

Arguments

`gtfs`	A GTFS object.
`zipfile`	Optional output ‘.zip‘ path. When omitted in an interactive session, a save-file window opens. The ‘.zip‘ extension is added when needed.
`...`	Additional arguments passed to the format-specific writer.

Value

The normalized output path, invisibly.

See Also

[GTFSwizard::read_gtfs()]

Examples

```
path <- tempfile(fileext = ".zip")
write_gtfs(for_rail_gtfs, path)
```

Index

as_wizardgtfs, 3
as_wizardgtfs(), *38*

bcn_bus_gtfs, 4

compare_scenarios, 5
create_gtfs, 6

delay_trip, 7

edit_dwelltime, 8
edit_speed, 9
explore_gtfs, 10

filter_area, 11
filter_date *(*filter_servicepattern*)*, 12
filter_date(), *4*
filter_route *(*filter_servicepattern*)*, 12
filter_service *(*filter_servicepattern*)*, 12
filter_servicepattern, 12
filter_stop *(*filter_servicepattern*)*, 12
filter_time *(*filter_servicepattern*)*, 12
filter_trip *(*filter_servicepattern*)*, 12
for_bus_gtfs, *4*, 13
for_rail_gtfs, *4*, 14

get_1stdeparture, 16
get_corridor, 16
get_distances, 17
get_durations, 18
get_dwelltimes, 19
get_fleet, 21
get_frequency, 22
get_headways, 23
get_hubs, 24
get_servicepattern, 24
get_shapes, 25
get_shapes_df, 26
get_shapes_sf, 27
get_speeds, 27
get_stops_sf, 28
gtfsio::import_gtfs(), *38*

latlon2epsg, 29

merge_gtfs, 29

plot.wizardgtfs *(*wizardgtfs-methods*)*, 47
plot_calendar, 30
plot_corridor, 31
plot_frequency, 32
plot_headways, 33
plot_hubs, 33
plot_routeduration, 34
plot_routefrequency, 35
plot_serviceheatmap, 36
plot_servicespan, 36
plot_servicesupply, 37
print.summary.wizardgtfs *(*wizardgtfs-methods*)*, 47
print.wizardgtfs *(*wizardgtfs-methods*)*, 47

read_gtfs, 38
read_gtfs(), *4*, *46*
repair_gtfs, 39
repair_report, 40

selection, 40
set_dwelltime, 41
split_trip, 42
summary.wizardgtfs *(*wizardgtfs-methods*)*, 47

tidy_raptor, 43

unselection *(*selection*)*, 40

validate_gtfs, 45

wizardgtfs-methods, 47
write_gtfs, 48
write_gtfs(), *38*, *46*