

Package: gps2gtfs (via r-universe)

October 3, 2026

Type Package

Title High-Performance GPS to GTFS Converter

Version 0.6.0

Description Preprocesses raw GPS trajectory data of public transit and transforms it to GTFS format. Provides a high-performance R port of the 'gps2gtfs' Python package by Aaivu (Ratneswaran et al., 2023) <[doi:10.1109/ICCT56969.2023.10075789](https://doi.org/10.1109/ICCT56969.2023.10075789)>, whose software description is published in Ratneswaran et al. (2025) <[doi:10.1016/j.simpa.2025.100780](https://doi.org/10.1016/j.simpa.2025.100780)>. Heavy computational tasks are offloaded to an extremely fast compiled Rust backend or a C++ (Rcpp) backend. Automatic backend selection prefers Rust, then Rcpp, and falls back to a slower pure R implementation utilizing 'data.table' and 'sf'.

License MIT + file LICENSE

Depends R (>= 4.2)

Imports data.table, Rcpp (>= 1.0.0)

Suggests gtfsio, gtfsrealtime, knitr, rmarkdown, sf, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

LinkingTo Rcpp

Config/rextendr/version 0.5.0

Encoding UTF-8

LazyData true

SystemRequirements Cargo (Rust's package manager), rustc >= 1.65.0, xz

URL <https://e-kotov.github.io/gps2gtfs/>,
<https://github.com/e-kotov/gps2gtfs>

BugReports <https://github.com/e-kotov/gps2gtfs/issues>

Config/roxygen2/version 8.1.0

Config/pak/sysreqs xz-utils libclang-dev

Repository <https://e-kotov.r-universe.dev>
Date/Publication 2026-10-03 16:59:20 UTC
RemoteUrl <https://github.com/e-kotov/gps2gtfs>
RemoteRef HEAD
RemoteSha 3228c763deb82926891be22a0bbbb3b7b5093315

Contents

g2g_clean_gps	2
g2g_data_gps	4
g2g_data_stops	5
g2g_data_terminals	6
g2g_diagnostics	6
g2g_extract_trips	7
g2g_extract_trips_and_stop_times	11
g2g_shapes_from_trips	17
g2g_stops_from_geometries	17
g2g_stops_from_gtfs	19
g2g_stops_from_positions	20
g2g_terminals_from_geometries	21
g2g_terminals_from_gtfs	22

Index	24
--------------	-----------

g2g_clean_gps	<i>Clean raw GPS data</i>
---------------	---------------------------

Description

Cleans raw GPS or GTFS-Realtime vehicle position data: maps input columns to the canonical schema, removes records with zero coordinates or missing vehicle identifiers, deduplicates repeated pings, parses timestamps, and sorts by vehicle, date, and time.

Usage

```
g2g_clean_gps(  
  raw_gps_df,  
  projected = NULL,  
  vehicle_col = "vehicle_id",  
  time_col = "timestamp",  
  tz = NULL,  
  dedupe = TRUE,  
  drop_missing_vehicle = TRUE  
)
```

Arguments

<code>raw_gps_df</code>	A <code>data.frame</code> containing raw GPS data. Must include latitude and longitude (WGS-84 degrees unless <code>projected = TRUE</code>), a vehicle identifier column, and a timestamp column. A speed column is recommended (used for dwell-time estimation); GTFS-Realtime reports it in meters per second, but only <code>speed == 0</code> is interpreted. An id row identifier is generated when absent or not unique.
<code>projected</code>	Logical. Is the coordinates data already projected? Default is <code>NULL</code> (auto-detect).
<code>vehicle_col</code>	Character. Name of the column holding the vehicle identifier. Default <code>"vehicle_id"</code> (GTFS-Realtime convention).
<code>time_col</code>	Character. Name of the column holding the observation time. Default <code>"timestamp"</code> (GTFS-Realtime convention). <code>POSIXct</code> input keeps its own timezone (service days split at feed-local midnight) and <code>tz</code> is ignored for it. Character/numeric input is parsed in <code>tz</code> .
<code>tz</code>	Character. Timezone in which non- <code>POSIXct</code> timestamps are interpreted, and therefore the timezone whose midnight bounds service days. Because service-day attribution (and every downstream GTFS clock string) depends on it, there is no safe default for character input: pass the feed's operating timezone (e.g. <code>"America/New_York"</code>), or pass already-localized <code>POSIXct</code> timestamps. <code>NULL</code> (default) parses character input as UTC <i>with a warning</i> ; set <code>tz = "UTC"</code> explicitly to silence it when UTC is truly intended.
<code>dedupe</code>	Logical. Drop repeated (<code>vehicle_id</code> , <code>timestamp</code>) rows, as produced by archived GTFS-Realtime feeds that re-report unchanged positions. Default <code>TRUE</code> .
<code>drop_missing_vehicle</code>	Logical. Drop rows with missing/empty vehicle identifiers with a warning (<code>TRUE</code> , default) instead of failing.

Details

The canonical column names follow the GTFS-Realtime convention, so the output of `gtfsrealtime::read_gtfsrt_position` is accepted directly. Any other source (AVL exports, logger CSVs) can be mapped via `vehicle_col` and `time_col`. Columns beyond the canonical set (e.g. `trip_id`, `route_id`, `stop_id`) are passed through untouched.

Value

A sorted `data.table` with canonical columns `id`, `vehicle_id`, `latitude`, `longitude`, `timestamp`, `speed`, additional `date` and `time_str` columns, and all other input columns passed through.

Examples

```
data(g2g_data_gps)
cleaned_gps <- g2g_clean_gps(g2g_data_gps)
head(cleaned_gps)
```

g2g_data_gps

*Sample bus GPS trajectories***Description**

A lightweight subset of raw bus GPS pings from Kandy, Sri Lanka, intended for examples and testing of the `gps2gtfs` pipeline. It is the `gps_data` input, and works together with [g2g_data_stops](#) and [g2g_data_terminals](#), which describe the same service.

Usage

```
g2g_data_gps
```

Format

A data frame with 2167 rows and 6 variables:

id Unique identifier for the GPS record.

vehicle_id Unique identifier for the tracking vehicle (bus). Column naming follows the GTFS-Realtime convention.

timestamp Timestamp of the GPS ping (UTC).

latitude Latitude in WGS-84 degrees.

longitude Longitude in WGS-84 degrees.

speed Recorded speed of the vehicle.

Source

Original data from the Python `gps2gtfs` package.

References

Ratneswaran, S., & Thayasivam, U. (2023). Extracting potential Travel time information from raw GPS data and Evaluating the Performance of Public transit - a case study in Kandy, Sri Lanka. *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 1-7. doi:10.1109/ICCT56969.2023.10075789

g2g_data_stops	<i>Sample bus stops</i>
----------------	-------------------------

Description

Coordinates and metadata for the bus stops of the same Kandy, Sri Lanka service as [g2g_data_gps](#). It is the `stops_data` input, and works together with that dataset and [g2g_data_terminals](#) to extract stop times.

Usage

```
g2g_data_stops
```

Format

A data frame with 23 rows and 6 variables:

stop_id Unique identifier for the bus stop.

route_id Route identifier the stop belongs to.

direction Direction of the route the stop serves.

latitude Latitude in WGS-84 degrees.

longitude Longitude in WGS-84 degrees.

address Address or name of the stop location.

Source

Original data from the Python `gps2gtfs` package.

References

Ratneswaran, S., & Thayasivam, U. (2023). Extracting potential Travel time information from raw GPS data and Evaluating the Performance of Public transit - a case study in Kandy, Sri Lanka. *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 1-7. doi:10.1109/ICCT56969.2023.10075789

g2g_data_terminals	<i>Sample bus terminals</i>
--------------------	-----------------------------

Description

Coordinates for the two route terminals of the same Kandy, Sri Lanka service as [g2g_data_gps](#). It is the `terminals_data` input, and works together with that dataset and [g2g_data_stops](#): the terminal buffers are what bound the start and end of each trip under `segmentation = "terminals"`.

Usage

`g2g_data_terminals`

Format

- A data frame with 2 rows and 4 variables:
- terminal_id** Unique identifier for the bus terminal.
 - terminal_name** Name of the terminal.
 - latitude** Latitude in WGS-84 degrees.
 - longitude** Longitude in WGS-84 degrees.

Source

Original data from the Python `gps2gtfs` package.

References

Ratneswaran, S., & Thayasivam, U. (2023). Extracting potential Travel time information from raw GPS data and Evaluating the Performance of Public transit - a case study in Kandy, Sri Lanka. *2023 3rd International Conference on Intelligent Communication and Computational Techniques (ICCT)*, 1-7. [doi:10.1109/ICCT56969.2023.10075789](#)

g2g_diagnostics	<i>Extraction coverage diagnostics</i>
-----------------	--

Description

Retrieve the coverage diagnostics attached to the return value of [g2g_extract_trips](#) or [g2g_extract_trips_and_stop_times](#). The diagnostics record how many GPS pings, trip segments, and trips survived each stage of extraction, so a non-random loss of coverage (a feed whose vehicle positions carry no usable trip identity, routes the two-terminal model cannot segment, stops out of buffer range) is visible as data instead of vanishing silently.

Usage

```
g2g_diagnostics(x)
```

Arguments

x The value returned by `g2g_extract_trips_and_stop_times` (a list) or by `g2g_extract_trips` (a data.table). Diagnostics are attached to that top-level object, not to the individual trips/stop_times tables inside the list.

Details

The extractors also emit a one-line `warning()` whenever a run loses coverage worth surfacing, pointing back here — so unattended or agent-driven pipelines notice the loss without having to inspect attributes. Duplicate removal is treated as routine and never triggers that warning (it still appears in the table). Suppress the warning with `diagnostics_warn = FALSE` or `options(gps2gtfs.diagnostics_warn = FALSE)`.

Value

A `g2g_diagnostics` data.table with columns `stage`, `metric`, and `n` (integer count; NA where a metric does not apply to the run). It has a compact print method; treat it as a normal data.table for programmatic use.

Examples

```
data(g2g_data_gps)
data(g2g_data_terminals)
data(g2g_data_stops)
res <- g2g_extract_trips_and_stop_times(
  gps_data = g2g_data_gps,
  terminals_data = g2g_data_terminals,
  stops_data = g2g_data_stops,
  terminals_buffer_radius = 50,
  stops_buffer_radius = 30,
  stops_extended_buffer_radius = 50
)
g2g_diagnostics(res)
```

<code>g2g_extract_trips</code>	<i>Extract trips from GPS trajectories</i>
--------------------------------	--

Description

Cuts a stream of vehicle GPS pings into individual trips and returns one row of features per trip: its vehicle, direction, start and end times, duration, and day of week. Takes a data.frame or a path to a CSV, and optionally writes the result to `output_path`.

Usage

```

g2g_extract_trips(
  gps_data,
  terminals_data = NULL,
  terminals_buffer_radius = NULL,
  output_path = NULL,
  projected_crs = NULL,
  backend = "auto",
  projected = NULL,
  vehicle_col = "vehicle_id",
  time_col = "timestamp",
  tz = NULL,
  trip_col = NULL,
  direction_col = NULL,
  cut_on_direction_change = FALSE,
  direction_debounce_min_pings = 2L,
  direction_debounce_min_seconds = 600,
  session_gap = 4 * 3600,
  segmentation = c("auto", "terminals", "layover"),
  layover_gap = 10 * 60,
  layover_radius = 50,
  diagnostics_warn = getOption("gps2gtfs.diagnostics_warn", TRUE)
)

```

Arguments

<code>gps_data</code>	A data.frame or path to the raw GPS CSV.
<code>terminals_data</code>	A data.frame or path to the terminal coordinates CSV. Optional (NULL) with <code>segmentation = "layover"</code> or with <code>direction_col</code> ; required for terminal-buffer segmentation.
<code>terminals_buffer_radius</code>	Numeric. Buffer radius for terminals (in meters). Only consumed by terminal-buffer segmentation; may be omitted otherwise.
<code>output_path</code>	Character. Optional path to write output trip features as CSV. Default is NULL (no file written).
<code>projected_crs</code>	Numeric. The EPSG code of a projected coordinate system used for metric distance calculations. Required when <code>projected = TRUE</code> ; otherwise a suitable UTM projection is selected when omitted.
<code>backend</code>	Character. The backend to use: "auto", "rust", "rcpp", or "pure_r". Automatic selection prefers Rust, then Rcpp, then pure R. Explicit unavailable backends produce an error.
<code>projected</code>	Logical. Whether plain-table coordinates are already projected. Out-of-bounds coordinates require explicit TRUE.
<code>vehicle_col</code>	Character. Name of the vehicle identifier column in <code>gps_data</code> . Default "vehicle_id" (GTFS-Realtime convention).

time_col	Character. Name of the timestamp column in gps_data. Default "timestamp" (GTFS-Realtime convention).
tz	Character. Timezone in which non-POSIXct timestamps are interpreted; also the timezone whose midnight bounds service days and thus every downstream GTFS clock string. Pass the feed's operating timezone, or supply localized POSIXct timestamps. NULL (default) parses character input as UTC with a warning. Passed to g2g_clean_gps .
trip_col	Character. Optional name(s) of column(s) holding supplied trip identities (e.g. "trip_id" from GTFS-Realtime Vehicle Positions). When given, trips are segmented by those identities (fast path) instead of inferred from terminal-buffer crossings. Without direction_col, the first/last ping of each segment is matched to the nearest terminal for direction assignment. May be several columns that jointly identify a trip - e.g. c("route_id", "direction_id", "start_date", "start_time"), the GTFS-Realtime TripDescriptor, for feeds whose positions carry no trip_id; they are combined into one identity. Default NULL (spatial inference).
direction_col	Character. Optional column giving each ping's travel direction (e.g. GTFS-Realtime "direction_id"). Only used with trip_col. When supplied, trip direction is taken from the data rather than inferred from which of two terminals a trip started at, so terminals_data becomes optional and routes that are not simple two-terminal lines (short-turns, variants, one-way services) are handled. For stop-time extraction, stops are grouped for matching by their direction label, which must use the same values as direction_col. By default this column labels a trip's direction; it does not cut trips. A trip has exactly one direction, so if direction_col takes more than one value inside a segment that trip_col declared to be one trip, the two inputs contradict each other and extraction stops with an error: either add the direction column to trip_col to segment by it, set cut_on_direction_change = TRUE, or collapse it to one value per supplied trip identity. Default NULL.
cut_on_direction_change	Logical. When TRUE, a change in direction_col cuts a trip rather than raising the mid-trip direction conflict error above. Raw direction fields flip spuriously, so the change is debounced first: only a run of consecutive identical values that is both long enough in pings and long enough in seconds counts as a real direction change, and shorter contrary bursts are absorbed into the direction around them. Requires direction_col. Default FALSE, which leaves behaviour unchanged.
direction_debounce_min_pings	Integer. Minimum consecutive pings for a direction run to be treated as a real direction change. Inert unless cut_on_direction_change = TRUE. Note this floor only binds on high-cadence feeds or at small direction_debounce_min_seconds: at a 15 s ping cadence a run needs about min_seconds / 15 + 1 pings to span the seconds threshold at all, so from roughly 300 s upwards the seconds threshold binds first. Default 2L.
direction_debounce_min_seconds	Numeric. Minimum intra-run span, in seconds, for a direction run to be treated as a real direction change. A single-ping run spans 0 seconds and is therefore always absorbed. Inert unless cut_on_direction_change = TRUE. Default 600.

session_gap	Numeric. Successive observations of a vehicle further apart than this many seconds start a new driving session; trips never span sessions. This replaces the former calendar-date boundary, so overnight trips crossing midnight stay intact while overnight parking still separates one day's operations from the next. Default 4 hours (4 * 3600).
segmentation	Character. How the raw-GPS spatial path cuts trips when no trip_col is supplied: "terminals" (classic two-terminal buffer model), "layover" (cut wherever the vehicle dwells longer than layover_gap, anywhere on the route - handles short-turn, loop, and multi-branch services; all trips share a single direction group), or "auto" (default: "terminals" when terminals_data is given, "layover" otherwise). Not used with trip_col.
layover_gap	Numeric. Layover segmentation only: dwells longer than this many seconds bound trips - whether a silent gap between successive pings or a stationary spell (pings present but staying within layover_radius). Mode-dependent: must exceed the feed's ping interval and normal in-service stop dwell, and stay below the shortest real layover. Default 10 minutes (10 * 60).
layover_radius	Numeric. Layover segmentation only: a vehicle counts as stationary while successive pings stay within this many meters of the dwell's first ping (anchoring absorbs GPS jitter while parked). Default 50.
diagnostics_warn	Logical. Emit a one-line warning() when the run drops coverage worth surfacing (pings with no usable trip identity, unmatched segments, stops out of range) so unattended or agent-driven pipelines notice silent loss. The full breakdown is always available via g2g_diagnostics regardless of this flag. Default <code>getOption("gps2gtfs.diagnostics_warn", TRUE)</code> .

Details

Where the cuts fall is the whole problem, and there are three regimes. If the data already carries trip identities (trip_col, e.g. a GTFS-Realtime trip_id), segmentation follows them and nothing is inferred. Otherwise the trips are inferred from the trajectory: segmentation = "terminals" cuts on crossings of two known terminal buffers, and segmentation = "layover" cuts wherever the vehicle dwells longer than layover_gap, anywhere on the route, which is what handles short-turns, loops and multi-branch service. session_gap bounds all three: a vehicle unseen for that long starts a new driving session and no trip spans one, so overnight trips stay intact while overnight parking still separates one day's operations from the next.

Inferred boundaries are estimates, and layover_gap in particular is mode- and feed-dependent: it must exceed the ping interval and the normal in-service dwell, and stay below the shortest real layover. The result carries a coverage table ([g2g_diagnostics](#)) reporting what the run dropped, which is the way to tell a clean extraction from a plausible-looking one.

To get stop times as well, use [g2g_extract_trips_and_stop_times](#), which runs this segmentation and then matches the pings of each trip to stop buffers.

Value

A data.table containing extracted trip features. start_time and end_time are absolute POSIXct times in the timezone of the input timestamps. Rows are returned in a stable, backend-invariant

order: sorted by the internal integer `trip_id`. The result carries an `attr(, "diagnostics")` coverage table (see [g2g_diagnostics](#)). See the "Inference tables, not GTFS files" section of [g2g_extract_trips_and_stop_times](#).

Examples

```
data(g2g_data_gps)
data(g2g_data_terminals)
trips <- g2g_extract_trips(
  gps_data = g2g_data_gps,
  terminals_data = g2g_data_terminals,
  terminals_buffer_radius = 50
)
```

`g2g_extract_trips_and_stop_times`

Extract trips and stop times from GPS trajectories

Description

The package's main entry point. Cuts a stream of vehicle GPS pings into individual trips, then matches each trip's pings to stop buffers, returning both a trip-feature table and a stop-time table with arrival, departure and dwell per stop. Each of `gps_data`, `terminals_data` and `stops_data` may be a `data.frame` or a path to a CSV, and either table can optionally be written out.

Usage

```
g2g_extract_trips_and_stop_times(
  gps_data,
  terminals_data = NULL,
  stops_data,
  terminals_buffer_radius = NULL,
  stops_buffer_radius,
  stops_extended_buffer_radius,
  output_trips_path = NULL,
  output_stops_path = NULL,
  projected_crs = NULL,
  backend = "auto",
  projected = NULL,
  stop_direction_map = NULL,
  vehicle_col = "vehicle_id",
  time_col = "timestamp",
  tz = NULL,
  trip_col = NULL,
  direction_col = NULL,
  cut_on_direction_change = FALSE,
  direction_debounce_min_pings = 2L,
```

```

direction_debounce_min_seconds = 600,
session_gap = 4 * 3600,
segmentation = c("auto", "terminals", "layover"),
layover_gap = 10 * 60,
layover_radius = 50,
return_trajectory = FALSE,
diagnostics_warn = getOption("gps2gtfs.diagnostics_warn", TRUE),
min_dwell = 0
)

```

Arguments

gps_data	A data.frame or path to the raw GPS CSV.
terminals_data	A data.frame or path to the terminal coordinates CSV. Optional (NULL) with segmentation = "layover" or with direction_col; required for terminal-buffer segmentation.
stops_data	A data.frame or path to the bus stops coordinates CSV. Its direction column groups stops for matching; with layover segmentation the column is optional and any labels are ignored (all stops match all trips).
terminals_buffer_radius	Numeric. Buffer radius for terminals (in meters). Only consumed by terminal-buffer segmentation; may be omitted otherwise.
stops_buffer_radius	Numeric. Buffer radius for bus stops (in meters).
stops_extended_buffer_radius	Numeric. Extended buffer radius for bus stops (in meters).
output_trips_path	Character. Optional path to write output trip features as CSV. Default is NULL (no file written).
output_stops_path	Character. Optional path to write output stop times as CSV. Default is NULL (no file written).
projected_crs	Numeric. The EPSG code of a projected coordinate system used for metric distance calculations. Required when projected = TRUE; otherwise a suitable UTM projection is selected when omitted.
backend	Character. The backend to use: "auto", "rust", "rcpp", or "pure_r". Automatic selection prefers Rust, then Rcpp, then pure R. Explicit unavailable backends produce an error.
projected	Logical. Whether plain-table coordinates are already projected. Out-of-bounds coordinates require explicit TRUE.
stop_direction_map	Named character vector mapping each raw stop-direction label to the terminal ID that trips in that direction <i>start from</i> . Required whenever stops_data labels its directions with anything other than the terminal IDs themselves: which label belongs to which terminal is not recoverable from the data (both direction groups span the same corridor), and getting it backwards silently reverses every

	direction in the output while still producing plausible stop times. Omitting it in that case is an error listing both candidate maps. Not needed when the labels are already terminal IDs, as with <code>g2g_stops_from_gtfs</code> , nor in data-driven direction mode (<code>direction_col</code>) or layover segmentation.
<code>vehicle_col</code>	Character. Name of the vehicle identifier column in <code>gps_data</code> . Default <code>"vehicle_id"</code> (GTFS-Realtime convention).
<code>time_col</code>	Character. Name of the timestamp column in <code>gps_data</code> . Default <code>"timestamp"</code> (GTFS-Realtime convention).
<code>tz</code>	Character. Timezone in which non-POSIXct timestamps are interpreted; also the timezone whose midnight bounds service days and thus every downstream GTFS clock string. Pass the feed's operating timezone, or supply localized POSIXct timestamps. NULL (default) parses character input as UTC with a warning. Passed to <code>g2g_clean_gps</code> .
<code>trip_col</code>	Character. Optional name(s) of column(s) holding supplied trip identities (e.g. <code>"trip_id"</code> from GTFS-Realtime Vehicle Positions). When given, trips are segmented by those identities (fast path) instead of inferred from terminal-buffer crossings. Without <code>direction_col</code> , the first/last ping of each segment is matched to the nearest terminal for direction assignment. May be several columns that jointly identify a trip - e.g. <code>c("route_id", "direction_id", "start_date", "start_time")</code> , the GTFS-Realtime TripDescriptor, for feeds whose positions carry no <code>trip_id</code> ; they are combined into one identity. Default NULL (spatial inference).
<code>direction_col</code>	Character. Optional column giving each ping's travel direction (e.g. GTFS-Realtime <code>"direction_id"</code>). Only used with <code>trip_col</code> . When supplied, trip direction is taken from the data rather than inferred from which of two terminals a trip started at, so <code>terminals_data</code> becomes optional and routes that are not simple two-terminal lines (short-turns, variants, one-way services) are handled. For stop-time extraction, stops are grouped for matching by their direction label, which must use the same values as <code>direction_col</code> . By default this column labels a trip's direction; it does not cut trips. A trip has exactly one direction, so if <code>direction_col</code> takes more than one value inside a segment that <code>trip_col</code> declared to be one trip, the two inputs contradict each other and extraction stops with an error: either add the direction column to <code>trip_col</code> to segment by it, set <code>cut_on_direction_change = TRUE</code> , or collapse it to one value per supplied trip identity. Default NULL.
<code>cut_on_direction_change</code>	Logical. When TRUE, a change in <code>direction_col</code> cuts a trip rather than raising the mid-trip direction conflict error above. Raw direction fields flip spuriously, so the change is debounced first: only a run of consecutive identical values that is both long enough in pings and long enough in seconds counts as a real direction change, and shorter contrary bursts are absorbed into the direction around them. Requires <code>direction_col</code> . Default FALSE, which leaves behaviour unchanged.
<code>direction_debounce_min_pings</code>	Integer. Minimum consecutive pings for a direction run to be treated as a real direction change. Inert unless <code>cut_on_direction_change = TRUE</code> . Note this floor only binds on high-cadence feeds or at small <code>direction_debounce_min_seconds</code> : at a 15 s ping cadence a run needs about <code>min_seconds / 15 + 1</code> pings to span the

	seconds threshold at all, so from roughly 300 s upwards the seconds threshold binds first. Default 2L.
<code>direction_debounce_min_seconds</code>	Numeric. Minimum intra-run span, in seconds, for a direction run to be treated as a real direction change. A single-ping run spans 0 seconds and is therefore always absorbed. Inert unless <code>cut_on_direction_change = TRUE</code> . Default 600.
<code>session_gap</code>	Numeric. Successive observations of a vehicle further apart than this many seconds start a new driving session; trips never span sessions. This replaces the former calendar-date boundary, so overnight trips crossing midnight stay intact while overnight parking still separates one day's operations from the next. Default 4 hours ($4 * 3600$).
<code>segmentation</code>	Character. How the raw-GPS spatial path cuts trips when no <code>trip_col</code> is supplied: "terminals" (classic two-terminal buffer model), "layover" (cut wherever the vehicle dwells longer than <code>layover_gap</code> , anywhere on the route - handles short-turn, loop, and multi-branch services; all trips share a single direction group), or "auto" (default: "terminals" when <code>terminals_data</code> is given, "layover" otherwise). Not used with <code>trip_col</code> .
<code>layover_gap</code>	Numeric. Layover segmentation only: dwells longer than this many seconds bound trips - whether a silent gap between successive pings or a stationary spell (pings present but staying within <code>layover_radius</code>). Mode-dependent: must exceed the feed's ping interval and normal in-service stop dwell, and stay below the shortest real layover. Default 10 minutes ($10 * 60$).
<code>layover_radius</code>	Numeric. Layover segmentation only: a vehicle counts as stationary while successive pings stay within this many meters of the dwell's first ping (anchoring absorbs GPS jitter while parked). Default 50.
<code>return_trajectory</code>	Logical. Also return the ping-level trajectory (cleaned GPS records with assigned <code>trip_id</code> and direction) as a trajectory element — the input for g2g_shapes_from_trips . Default FALSE.
<code>diagnostics_warn</code>	Logical. Emit a one-line <code>warning()</code> when the run drops coverage worth surfacing (pings with no usable trip identity, unmatched segments, stops out of range) so unattended or agent-driven pipelines notice silent loss. The full breakdown is always available via g2g_diagnostics regardless of this flag. Default <code>getOption("gps2gtfs.diagnostics_warn", TRUE)</code> .
<code>min_dwell</code>	Numeric. Minimum dwell, in seconds, for a stop visit to be kept; larger values also drop short stops. Default 0 keeps every visit. Dwell is measured from zero-speed pings, so three kinds of visit have zero dwell: a vehicle that crossed the stop's buffer without stopping; one that stopped between two pings; and one seen stationary only on its last ping inside the buffer. <code>min_dwell = 1</code> removes all three, so it is not a pure pass-through filter. With pings a few seconds apart the last two are rare; at 15-30 s intervals they are common, and many visits to stops that were served are removed. Check the number dropped, reported in a message, before relying on it. Without a speed on any ping inside a stop buffer every dwell is 0, so any positive value drops every visit (with a warning).

Details

Segmentation works exactly as in [g2g_extract_trips](#) - supplied `trip_col` identities, terminal buffers, or layover dwells, all bounded by `session_gap` - and this function adds the stop-matching stage on top. A ping is attributed to a stop when it falls inside `stops_buffer_radius`, with `stops_extended_buffer_radius` as the wider fallback; the stops of a trip are restricted to those labeled with its direction, so `stops_data$direction` and the trips' direction labels must use the same values (see `stop_direction_map`).

Both stages estimate. Trip boundaries can be misplaced without the trip count looking wrong, and a stop the vehicle passed outside every buffer simply produces no row. The returned tables therefore carry a coverage table ([g2g_diagnostics](#)) reporting pings with no usable trip identity, unmatched segments and out-of-range stops, and that is what to read before treating the output as complete.

The result is inference tables, not GTFS files; see the section below for what still has to happen before this is a publishable feed.

Value

A list containing two data.tables: `trips` and `stop_times`, plus trajectory when `return_trajectory = TRUE`. All times (`start_time`, `end_time`, `arrival_time`, `departure_time`) are absolute POSIXct values in the timezone of the input timestamps — never clock strings, so trips running past midnight stay unambiguous. In `stop_times`, `date`, `day_of_week`, `is_weekday` and `hour_of_day` describe the visit's `arrival_time` in that timezone.

Row order is a stable, backend-invariant contract: trips are ordered by the internal integer `trip_id`, and `stop_times` by (`trip_id`, `arrival_time`, `stop_id`, `departure_time`) with every remaining column appended as a final tie-breaker. The sort is therefore total, so the three backends (Rust, Rcpp, pure R) return identical row order for identical input and summaries built on the result are reproducible regardless of backend or parallelism.

The result also carries an `attr(, "diagnostics")` coverage table (see [g2g_diagnostics](#)).

Inference tables, not GTFS files

The returned `trips` and `stop_times` use GTFS-style column names but are **inference tables**, not valid `trips.txt`/ `stop_times.txt`: they carry no `route_id`, `service_id`, `stop_sequence`, or GTFS clock strings ("HH:MM:SS", with >24:00:00 for post-midnight stops). This is deliberate: `route_id`, `service_id`, and canonical stop identities cannot be inferred from coordinates alone, so `gps2gtfs` stops at inference rather than synthesizing them.

Turning these tables into a standard-compliant feed — deriving `stop_sequence`, attributing each trip to a service day, encoding >24:00:00 clock strings, and linking or synthesizing IDs — is the job of the companion package `gtfsrt2static` (a separate, optional install), which yields a feed the `MobilityData gtfs-validator` accepts:

```
res    <- g2g_extract_trips_and_stop_times(...)
events <- gtfsrt2static::snapshot_from_stop_times(
  res$stop_times, trip_id_col = "provided_trip_id" # keep official IDs
)
feed   <- gtfsrt2static::snapshot_assemble(events) # baseline: real IDs
# or, with no planned feed to lean on:
# feed <- gtfsrt2static::snapshot_scaffold(events, strict = TRUE)
```

```
gtfsio::export_gtfs(feed, "realized_gtfs.zip")    # gtfstools/tidytransit-ready
```

Both tables carry a `provided_trip_id` column: the caller's official trip identity (the `trip_col` value, e.g. a GTFS-Realtime `trip_id`) when the fast path is used, NA otherwise. It lets a downstream assembler preserve official trip IDs instead of synthesizing them. The internal integer `trip_id` remains the join key between the two tables.

Orientation and pattern labels (C5 contract)

Both tables also carry an additive, versioned set of inference-label columns. They are the contract slots for a future baseline-free orientation/turnaround-detection stage and are currently always in the "no detector applied" state (no such detector is enabled yet), so the legacy `direction` column - retained unchanged - is what downstream consumers read today:

`orientation_id` Integer 0/1, geometric travel direction only; NA when unavailable/abstained (always NA today).

`orientation_status` Factor, never NA. "none" (no orientation method applied) today; reserved values "ok" / "single_group" / "abstain_<reason>" make a future detector's abstentions observable rather than silent.

`orientation_confidence` Double in [0,1], NA when `orientation_id` is NA (always NA today).

`pattern_ref` Character K-way branch/short-turn variant identity; reserved nullable (always NA today). The handoff to `gtfsrt2static`'s cross-trip stop-order stage.

`start_anchor_ref`, `end_anchor_ref` Character discovered turnaround-cluster ids; trips **only** (trip-level metadata, not per-stop), NA today.

Downstream, `gtfsrt2static` maps `orientation_id` into the GTFS `direction_id` (falling back to `direction` while `orientation_id` is NA); `pattern_ref` rides through to C6; anchors stay in C5. `orientation_*/pattern_ref` propagate from trips onto `stop_times` on the internal `trip_id`, the same mechanism as `provided_trip_id`.

Examples

```
data(g2g_data_gps)
data(g2g_data_terminals)
data(g2g_data_stops)
result <- g2g_extract_trips_and_stop_times(
  gps_data = g2g_data_gps,
  terminals_data = g2g_data_terminals,
  stops_data = g2g_data_stops,
  terminals_buffer_radius = 50,
  stops_buffer_radius = 30,
  stops_extended_buffer_radius = 50,
  # Each stop-direction label maps to the terminal its trips start from.
  # BT01 is Kandy, BT02 is Digana.
  stop_direction_map = c(
    "Kandy-Digana" = "BT01",
    "Digana-Kandy" = "BT02"
  )
)
head(result$trips)
```

```
head(result$stop_times)
```

`g2g_shapes_from_trips` *Build shapes.txt-shaped traces from a pipeline trajectory*

Description

Converts the ping-level trajectory of extracted trips into a table shaped like the GTFS `shapes.txt` file: one shape per trip, points in travel order, with cumulative distance. Unlike planned shapes, these traces record the geometry *actually driven*, including detours.

Usage

```
g2g_shapes_from_trips(trajectory, shape_id_prefix = "SHP_")
```

Arguments

<code>trajectory</code>	A data.table of trajectory GPS records with columns <code>trip_id</code> , <code>latitude</code> , <code>longitude</code> , <code>date</code> , <code>time_str</code> (as returned in the pipeline's <code>\$trajectory</code>).
<code>shape_id_prefix</code>	Character prefix for generated shape ids. Default "SHP_" (shape ids become SHP_<trip_id>).

Details

Obtain the trajectory by running `g2g_extract_trips_and_stop_times(..., return_trajectory = TRUE)` and using the `$trajectory` element of the result.

Value

A data.table with GTFS-standard columns `shape_id`, `shape_pt_lat`, `shape_pt_lon`, `shape_pt_sequence`, `shape_dist_traveled` (meters).

`g2g_stops_from_geometries`

Derive a stops table from supplied route geometries

Description

Builds the `stops_data` input for `g2g_extract_trips_and_stop_times` from the route's own linestrings and a plain table of stop coordinates, with no GTFS feed involved: a stop belongs to a direction when it lies within `buffer_m` of that direction's line.

Usage

```
g2g_stops_from_geometries(geometries, stops, route_id, buffer_m = 50)
```

Arguments

geometries	The route geometries, in either of two forms: • a data.frame of vertices in long form, with columns route_id, direction, vertex_seq (the numeric order of the vertices within a direction), latitude and longitude; or • an sf object of LINESTRING or MULTILINESTRING geometries with route_id and direction columns (requires the 'sf' package). MULTILINESTRING parts are concatenated in part order.
stops	A data.frame of stop coordinates with columns stop_id, latitude, longitude. Any other columns are dropped. Rows with a missing coordinate, or a missing or blank stop_id, are dropped with a warning giving the counts.
route_id	A single route identifier present in geometries. The gps2gtfs pipeline models one route (two terminals) at a time.
buffer_m	Numeric. Maximum distance in meters from a direction's line for a stop to be assigned to that direction. Default 50.

Details

Distance is measured to the *polyline*, not to its vertices. A stop sitting halfway along a 300 m segment is on the route; a nearest-vertex rule would measure 150 m and drop it.

Direction labels are the terminal_ids that [g2g_terminals_from_geometries](#) emits for the same input, so the two tables drop into terminals_data and stops_data together and no stop_direction_map is needed.

A stop within buffer_m of both lines - the usual case for the two sides of one street - is returned once per direction, which is what makes the direction labels do any work. Stops farther than buffer_m from every direction are simply absent, and their count is reported in a message. If that is every stop, the result is a zero-row table and a warning, not an error, so a caller looping over many routes can take that answer for one of them. Terminals are not treated specially and are not removed.

Value

A data.table with columns stop_id, latitude, longitude, direction (the terminal_id of the direction the stop was matched to), sorted by direction then stop_id. Stops matched in both directions appear once per direction. Zero rows, with the same four columns and types, when no stop is within buffer_m of either direction.

See Also

[g2g_terminals_from_geometries](#) for the matching terminals_data, and [g2g_stops_from_gtfs](#) for the same table derived from a planned GTFS feed.

Examples

```
geometries <- data.frame(
  route_id = "r1",
  direction = rep(c("east", "west"), each = 3),
  vertex_seq = rep(1:3, times = 2),
  latitude = c(7.29, 7.30, 7.31, 7.31, 7.30, 7.29),
  longitude = c(80.63, 80.64, 80.65, 80.65, 80.64, 80.63)
)
stops <- data.frame(
  stop_id = c("S1", "S2"),
  latitude = c(7.295, 7.305),
  longitude = c(80.635, 80.645)
)
g2g_stops_from_geometries(geometries, stops, route_id = "r1")
```

g2g_stops_from_gtfs	<i>Derive a stops table from a static GTFS feed</i>
---------------------	---

Description

Builds the stops_data input for [g2g_extract_trips_and_stop_times](#) from a planned (baseline) GTFS feed: all non-terminal stops served by a route, labeled with the direction group they belong to. Direction labels are the terminal_id a trip starts from, so they map onto the terminals derived by [g2g_terminals_from_gtfs](#) without a manual stop_direction_map. Note this means the labels are terminal stop_ids, not GTFS direction_id values; they are not interchangeable with a direction_col taken from the positions.

Usage

```
g2g_stops_from_gtfs(gtfs, route_id)
```

Arguments

gtfs	A GTFS feed object (named list of data.frames, as returned by <code>gtfsio::import_gtfs()</code> or <code>gtfstools::read_gtfs()</code>) or a path to a GTFS zip file (requires the 'gtfsio' package).
route_id	A single route identifier present in <code>trips.txt</code> . The <code>gps2gtfs</code> pipeline models one route (two terminals) at a time.

Details

Trips that start at neither derived terminal have no direction group and are dropped with a warning, so stops served only by short turns or branch variants do not appear in the result.

Value

A data.table with columns stop_id, latitude, longitude, direction (the starting terminal_id of trips serving the stop in that direction). Stops served in both directions appear once per direction.

Examples

```
gtfs <- list(
  trips = data.frame(trip_id = c("t1", "t2"), route_id = "r1"),
  stop_times = data.frame(
    trip_id = c("t1", "t1", "t1", "t2", "t2", "t2"),
    stop_id = c("A", "S1", "B", "B", "S1", "A"),
    stop_sequence = c(1, 2, 3, 1, 2, 3)
  ),
  stops = data.frame(
    stop_id = c("A", "S1", "B"),
    stop_lat = c(7.29, 7.30, 7.31),
    stop_lon = c(80.63, 80.64, 80.65)
  )
)
g2g_stops_from_gtfs(gtfs, route_id = "r1")
```

g2g_stops_from_positions

Estimate stop coordinates from vehicle positions

Description

Baseline-free helper: when no static GTFS feed exists, stop locations can be estimated from GTFS-Realtime Vehicle Positions that carry a `stop_id` annotation. Each stop's coordinates are the median position of the pings observed at it — by default only pings labeled `STOPPED_AT`, the most precise signal.

Usage

```
g2g_stops_from_positions(positions, statuses = "STOPPED_AT", min_obs = 3L)
```

Arguments

<code>positions</code>	A <code>data.frame</code> of vehicle positions (e.g. from <code>gtfsrealtime::read_gtfsrt_positions()</code> or <code>g2g_clean_gps()</code>) with columns <code>stop_id</code> , <code>latitude</code> , <code>longitude</code> , and optionally <code>current_status</code> .
<code>statuses</code>	Character vector of <code>current_status</code> values to use. Default <code>"STOPPED_AT"</code> . Ignored (all <code>stop_id</code> -annotated pings used, with a message) when the <code>current_status</code> column is absent.
<code>min_obs</code>	Integer. Minimum number of pings required per stop; stops with fewer observations are dropped with a warning. Default 3.

Details

The result covers the `stop_id/latitude/longitude` columns of the `stops_data` input to [g2g_extract_trips_and_stop_t](#); the direction label (starting terminal of trips serving the stop) must still be supplied, e.g. from the RT `trip_id/direction_id` annotations or a baseline feed. It also fills the spec-required `stop_lat/stop_lon` of a scaffolded `stops.txt`.

Value

A data.table with columns stop_id, latitude, longitude, n_obs, sorted by stop_id.

g2g_terminals_from_geometries

Derive trip terminals from supplied route geometries

Description

Determines the two terminals of a route from the route's own linestrings, with no GTFS feed involved: each direction's terminal is the *first* vertex of that direction's line, which is where a vehicle running that direction starts. The result feeds directly into terminals_data of [g2g_extract_trips](#) and [g2g_extract_trips_and_stop_times](#).

Usage

```
g2g_terminals_from_geometries(geometries, route_id)
```

Arguments

geometries	The route geometries, in either of two forms: • a data.frame of vertices in long form, with columns route_id, direction, vertex_seq (the numeric order of the vertices within a direction), latitude and longitude; or • an sf object of LINESTRING or MULTILINESTRING geometries with route_id and direction columns (requires the 'sf' package). MULTILINESTRING parts are concatenated in part order.
route_id	A single route identifier present in geometries. The gps2gtfs pipeline models one route (two terminals) at a time.

Details

The terminal_id is the direction label itself. That is deliberate: [g2g_stops_from_geometries](#) labels stops with the same values, so the stop labels already are terminal IDs and no stop_direction_map is needed (the convention [g2g_stops_from_gtfs](#) documents).

Exactly two directions are required, because the terminal model is one route with two ends. A route drawn as one loop, or with a third branch variant, errors; use segmentation = "layover" for those. Two first vertices a few meters apart are not two ends of a line either, and warn.

Value

A data.table with columns terminal_id, latitude, longitude, direction, one row per direction. terminal_id equals direction.

See Also

[g2g_stops_from_geometries](#) for the matching stops_data, and [g2g_terminals_from_gtfs](#) for the same table derived from a planned GTFS feed.

Examples

```
geometries <- data.frame(
  route_id = "r1",
  direction = rep(c("east", "west"), each = 3),
  vertex_seq = rep(1:3, times = 2),
  latitude = c(7.29, 7.30, 7.31, 7.31, 7.30, 7.29),
  longitude = c(80.63, 80.64, 80.65, 80.65, 80.64, 80.63)
)
g2g_terminals_from_geometries(geometries, route_id = "r1")
```

```
g2g_terminals_from_gtfs
```

Derive trip terminals from a static GTFS feed

Description

Determines the two terminals of a route from a planned (baseline) GTFS feed: the two most frequent trip endpoints (first/last stop per trip in stop_times.txt). The result feeds directly into terminals_data of [g2g_extract_trips](#) and [g2g_extract_trips_and_stop_times](#).

Usage

```
g2g_terminals_from_gtfs(gtfs, route_id)
```

Arguments

gtfs	A GTFS feed object (named list of data.frames, as returned by <code>gtfsio::import_gtfs()</code> or <code>gtfstools::read_gtfs()</code>) or a path to a GTFS zip file (requires the 'gtfsio' package).
route_id	A single route identifier present in trips.txt. The <code>gps2gtfs</code> pipeline models one route (two terminals) at a time.

Details

Frequency ranking cannot tell two ends of a line from two platforms of one place, so the function warns when fewer than 90% of trip endpoints match the two it derived (variants or branches) and when the two are close enough together to be the same physical terminal under platform-level stop_ids. Loop routes have no two terminals at all and error; use `segmentation = "layover"` for those.

Value

A data.table with columns terminal_id, latitude, longitude.

Examples

```
gtfs <- list(  
  trips = data.frame(trip_id = c("t1", "t2"), route_id = "r1"),  
  stop_times = data.frame(  
    trip_id = c("t1", "t1", "t2", "t2"),  
    stop_id = c("A", "B", "B", "A"),  
    stop_sequence = c(1, 2, 1, 2)  
  ),  
  stops = data.frame(  
    stop_id = c("A", "B"),  
    stop_lat = c(7.29, 7.31),  
    stop_lon = c(80.63, 80.65)  
  )  
)  
g2g_terminals_from_gtfs(gtfs, route_id = "r1")
```

Index

* datasets

- g2g_data_gps, [4](#)
- g2g_data_stops, [5](#)
- g2g_data_terminals, [6](#)

- g2g_clean_gps, [2](#), [9](#), [13](#)
- g2g_data_gps, [4](#), [5](#), [6](#)
- g2g_data_stops, [4](#), [5](#), [6](#)
- g2g_data_terminals, [4](#), [5](#), [6](#)
- g2g_diagnostics, [6](#), [10](#), [11](#), [14](#), [15](#)
- g2g_extract_trips, [6](#), [7](#), [7](#), [15](#), [21](#), [22](#)
- g2g_extract_trips_and_stop_times, [6](#), [7](#),
[10](#), [11](#), [11](#), [17](#), [19–22](#)
- g2g_shapes_from_trips, [14](#), [17](#)
- g2g_stops_from_geometries, [17](#), [21](#), [22](#)
- g2g_stops_from_gtfs, [13](#), [18](#), [19](#), [21](#)
- g2g_stops_from_positions, [20](#)
- g2g_terminals_from_geometries, [18](#), [21](#)
- g2g_terminals_from_gtfs, [19](#), [22](#), [22](#)