Type Package
Title Read, Validate, Analyze, and Map GTFS Feeds
Version 1.2.0
Description Read General Transit Feed Specification (GTFS) zipfiles into a list of R dataframes. Perform validation of the data structure against the specification. Analyze the headways and frequencies at routes and stops. Create maps and perform spatial analysis on the routes and stops. Please see the GTFS documentation here for more detail: <https://gtfs.org/>.

License GPL
LazyData TRUE
Depends R (>= 3.6.0)
Imports gtfsio (>= 0.1.0), dplyr, data.table (>= 1.12.8), httr, rlang, sf, hms, digest, checkmate, geodist
Suggests testthat, knitr, markdown, rmarkdown, ggplot2, scales, lubridate

RoxygenNote 7.1.1
URL https://github.com/r-transit/tidytransit
BugReports https://github.com/r-transit/tidytransit
VignetteBuilder knitr
Encoding UTF-8

NeedsCompilation no

Author Flavio Poletti [aut, cre],
Daniel Herszenhut [aut] (<https://orcid.org/0000-0001-8066-1105>),
Mark Padgham [aut],
Tom Buckley [aut],
Danton Noriega-Goodwin [aut],
Angela Li [ctb],
Elaine McVey [ctb],
Charles Hans Thompson [ctb],
Michael Sumner [ctb],
Patrick Hausmann [ctb],
Bob Rudis [ctb],
James Lamb [ctb],
Alexandra Kapp [ctb],
Kearey Smith [ctb],
Dave Vautin [ctb],
Kyle Walker [ctb]

Maintainer Flavio Poletti <flavio.poletti@hotmail.ch>
Repository CRAN
Date/Publication 2021-11-23 12:10:02 UTC

R topics documented:

convert_times_to_hms .................................................. 3
feedlist ................................................................. 3
feed_contains .......................................................... 4
filter_feed_by_area ...................................................... 4
filter_feed_by_date ...................................................... 5
filter_feed_by_stops ..................................................... 5
filter_feed_by_trips ..................................................... 6
filter_stops ............................................................. 7
filter_stop_times ......................................................... 7
get_feedlist ............................................................... 8
get_route_frequency ...................................................... 9
get_route_geometry .................................................... 10
get_stop_frequency ..................................................... 10
get_trip_geometry ....................................................... 11
gtfs_as_sf ............................................................... 12
gtfs_duke ................................................................. 13
gtfs_transform .......................................................... 13
plot.tidygtfs ............................................................ 14
print.tidygtfs ............................................................ 14
raptor ...................................................................... 15
read_gtfs ................................................................. 17
route_type_names ........................................................ 18
set_api_key ............................................................... 18
set_servicepattern ...................................................... 19
sf_as_tbl ................................................................. 19
sf_lines_to_df ........................................................... 20
sf_points_to_df .......................................................... 20
shapes_as_sf ............................................................. 21
stops_as_sf .............................................................. 21
summary.tidygtfs ....................................................... 22
travel_times .............................................................. 22
validate_gtfs ............................................................. 24
write_gtfs ................................................................. 25

Index 27
**convert_times_to_hms**  
*Use hms::hms columns in feed*

---

**Description**

Overwrites character columns in stop_times (arrival_time, departure_time) and frequencies (start_time, end_time) with times converted with `hms::hms()`.

**Usage**

```r
convert_times_to_hms(gtfs_obj)
```

**Arguments**

- `gtfs_obj` a gtfs object in which hms times should be set, the modified gtfs_obj is returned

**Value**

- `gtfs_obj` with added hms times columns for stop_times and frequencies

---

**feedlist**  
*Dataframe of source GTFS data from Transitfeeds*

---

**Description**

A dataset containing a list of URLs for GTFS feeds

**Usage**

```r
feedlist
```

**Format**

A data frame with 911 rows and 10 variables:

- `id` the id of the feed on transitfeeds.com
- `t` title of the feed
- `loc_id` location id
- `loc_pid` location placeid of the feed on transitfeeds.com
- `loc_t` the title of the location
- `loc_n` the shortname fo the location
- `loc_lat` the location latitude
- `loc_lng` the location longitude
- `url_d` GTFS feed url
- `url_i` the metadata url for the feed
**feed_contains**

Returns TRUE if the given `gtfs_obj` contains the table. Used to check for tidytransit’s calculated tables in sublist.

**Description**

Returns TRUE if the given `gtfs_obj` contains the table. Used to check for tidytransit’s calculated tables in sublist.

**Usage**

`feed_contains(gtfs_obj, table_name)`

**Arguments**

- `gtfs_obj` : `gtfs` object
- `table_name` : name as string of the table to look for

---

**filter_feed_by_area**

Filter a `gtfs` feed so that it only contains trips that pass a given area.

**Description**

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

**Usage**

`filter_feed_by_area(gtfs_obj, area)`

**Arguments**

- `gtfs_obj` : tidygtfs object
- `area` : all trips passing through this area are kept. Either a bounding box (numeric vector with xmin, ymin, xmax, ymax) or a sf object.

**Value**

tidygtfs object with filtered tables

**See Also**

`filter_feed_by_stops, filter_feed_by_trips, filter_feed_by_date`
**filter_feed_by_date**  
*Filter a gtfs feed so that it only contains trips running on a given date*

---

**Description**

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.

**Usage**

```r
filter_feed_by_date(
  gtfs_obj,  
  extract_date,  
  min_departure_time,  
  max_arrival_time
)
```

**Arguments**

- `gtfs_obj`: a gtfs feed
- `extract_date`: date to extract trips from this day (Date or "YYYY-MM-DD" string)
- `min_departure_time`: (optional) The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.

**Value**

tidygtfs object with filtered tables

**See Also**

`filter_stop_times`, `filter_feed_by_trips`, `filter_feed_by_trips`, `filter_feed_by_date`

---

**filter_feed_by_stops**  
*Filter a gtfs feed so that it only contains trips that pass the given stops*

---

**Description**

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and transfers belonging to one of those trips are kept.
filter_feed_by_trips

Usage

filter_feed_by_stops(gtfs_obj, stop_ids = NULL, stop_names = NULL)

Arguments

gtfs_obj:
tidygtfs object

stop_ids:
vector with stop_ids. You can either provide stop_ids or stop_names

stop_names:
vector with stop_names (will be converted to stop_ids)

Value

tidygtfs object with filtered tables

Note

The returned gtfs_obj likely contains more than just the stops given (i.e. all stops that belong to a
trip passing the initial stop).

See Also

filter_feed_by_trips, filter_feed_by_trips, filter_feed_by_date

filter_feed_by_trips

Filter a gtfs feed so that it only contains a given set of trips

Description

Only stop_times, stops, routes, services (in calendar and calendar_dates), shapes, frequencies and
transfers belonging to one of those trips are kept.

Usage

filter_feed_by_trips(gtfs_obj, trip_ids)

Arguments

gtfs_obj:
tidygtfs object

trip_ids:
vector with trip_ids

Value

tidygtfs object with filtered tables

See Also

filter_feed_by_stops, filter_feed_by_area, filter_feed_by_date
**filter_stops**

*Get a set of stops for a given set of service ids and route ids*

**Description**

Get a set of stops for a given set of service ids and route ids.

**Usage**

```r
filter_stops(gtfs_obj, service_ids, route_ids)
```

**Arguments**

- `gtfs_obj`: as read by read_gtfs()
- `service_ids`: the service for which to get stops
- `route_ids`: the route_ids for which to get stops

**Value**

stops table for a given service

**Examples**

```r
library(dplyr)
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(local_gtfs_path)
select_service_id <- filter(nyc$calendar, monday==1) %>% pull(service_id)
select_route_id <- sample_n(nyc$routes, 1) %>% pull(route_id)
filtered_stops_df <- filter_stops(nyc, select_service_id, select_route_id)
```

**filter_stop_times**

*Filter a stop_times table for a given date and timespan.*

**Description**

Filter a stop_times table for a given date and timespan.

**Usage**

```r
filter_stop_times(gtfs_obj, extract_date, min_departure_time, max_arrival_time)
```
get_feedlist

Arguments

- `gtfs_obj` a gtfs feed
- `extract_date` date to extract trips from this day (Date or "YYYY-MM-DD" string)
- `min_departure_time` (optional) The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.
- `max_arrival_time` (optional) The latest arrival time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.

Value

Filtered stop_times data.table for `travel_times()` and `raptor()`.

Examples

```r
feed_path <- system.file("extdata", "sample-feed-fixed.zip", package = "tidytransit")
g <- read_gtfs(feed_path)

# filter the sample feed
stop_times <- filter_stop_times(g, "2007-01-06", "06:00:00", "08:00:00")
```

Description

Get list of all available feeds from transitfeeds API

Usage

`get_feedlist()`

Value

a data frame with the gtfs feeds on transitfeeds

See Also

`feedlist_df`

Examples

```r
## Not run:
feedlist_df <- get_feedlist()
## End(Not run)
```
Description

Calculate the number of departures and mean headways for routes within a given timespan and for given service_ids.

Usage

```r
get_route_frequency(
  gtfs_obj,
  start_time = "06:00:00",
  end_time = "22:00:00",
  service_ids = NULL
)
```

Arguments

- `gtfs_obj`: a list of gtfs dataframes as read by the `trread` package.
- `start_time`: analysis start time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
- `end_time`: analysis period end time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
- `service_ids`: A set of service_ids from the calendar dataframe identifying a particular service id. If not provided, the service_id with the most departures is used.

Value

A dataframe of routes with variables or headway/frequency in seconds for a route within a given time frame.

Note

Some GTFS feeds contain a frequency data frame already. Consider using this instead, as it will be more accurate than what tidytransit calculates.

Examples

```r
data(gtfs_duke)
routes_frequency <- get_route_frequency(gtfs_duke)
x <- order(routes_frequency$median_headways)
head(routes_frequency[x, ])
```
get_route_geometry  
*Get all trip shapes for a given route and service*

**Description**
Get all trip shapes for a given route and service

**Usage**
```r
get_route_geometry(gtfs_sf_obj, route_ids = NULL, service_ids = NULL)
```

**Arguments**
- `gtfs_sf_obj`: tidytransit gtfs object with sf data frames
- `route_ids`: routes to extract
- `service_ids`: service_ids to extract

**Value**
An sf dataframe for gtfs routes with a row/linestring for each trip

**Examples**
```r
data(gtfs_duke)
gtfs_duke_sf <- gtfs_as_sf(gtfs_duke)
routes_sf <- get_route_geometry(gtfs_duke_sf)
plot(routes_sf[c(1,1350),])
```

get_stop_frequency  
*Get Stop Frequency*

**Description**
Calculate the number of departures and mean headways for all stops within a given timespan and for given service_ids.

**Usage**
```r
get_stop_frequency(
  gtfs_obj,
  start_time = "06:00:00",
  end_time = "22:00:00",
  service_ids = NULL,
  by_route = TRUE
)
```
**get_trip_geometry**

**Arguments**

- `gtfs_obj`: a list of gtfs dataframes as read by `read_gtfs()`.
- `start_time`: analysis start time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
- `end_time`: analysis period end time, can be given as "HH:MM:SS", hms object or numeric value in seconds.
- `service_ids`: A set of service_ids from the calendar dataframe identifying a particular service id. If not provided, the service_id with the most departures is used.
- `by_route`: Default TRUE, if FALSE then calculate headway for any line coming through the stop in the same direction on the same schedule.

**Value**

dataframe of stops with the number of departures and the headway (departures divided by timespan) in seconds as columns

**Note**

Some GTFS feeds contain a frequency data frame already. Consider using this instead, as it will be more accurate than what tidytransit calculates.

**Examples**

```r
data(gtfs_duke)
stop_frequency <- get_stop_frequency(gtfs_duke)
x <- order(stop_frequency$mean_headway)
head(stop_frequency[x,])
```

---

**get_trip_geometry**  
*Get all trip shapes for given trip ids*

**Description**

Get all trip shapes for given trip ids

**Usage**

`get_trip_geometry(gtfs_sf_obj, trip_ids)`

**Arguments**

- `gtfs_sf_obj`: tidytransit gtfs object with sf data frames
- `trip_ids`: trip_ids to extract shapes

**Value**

an sf dataframe for gtfs routes with a row/linestring for each trip
gtfs_as_sf

Convert stops and shapes to Simple Features

Description

Stops are converted to POINT sf data frames. Shapes are created as LINESTRING data frame. Note that this function replaces stops and shapes tables in gtfs_obj.

Usage

gtfs_as_sf(gtfs_obj, skip_shapes = FALSE, crs = NULL, quiet = TRUE)

Arguments

gtfs_obj a standard tidytransit gtfs object
skip_shapes if TRUE, shapes are not converted. Default FALSE.
crs optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates of stops and shapes
quiet boolean whether to print status messages

Value

tidygtfs object with stops and shapes as sf dataframes

See Also

sf_as_tdf
gtfs_duke  Example GTFS data

Description

Usage

gtfs_duke

Format
An object of class tidygtfs (inherits from gtfs) of length 25.

See Also
read_gtfs

gtfs_transform  Transform or convert coordinates of a gtfs feed

Description
Transform or convert coordinates of a gtfs feed

Usage

gtfs_transform(gtfs_obj, crs)

Arguments

  gtfs_obj tidygtfs object
  crs target coordinate reference system, used by sf::st_transform

Value
tidygtfs object with transformed stops and shapes sf dataframes
### plot.tidygtfs

**Plot GTFS stops and trips**

**Description**

Plot GTFS stops and trips

**Usage**

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

**Arguments**

- `x` a gtfs_obj as read by read_gtfs()
- `...` further specifications

**Value**

plot

**Examples**

```r
local_gtfs_path <- system.file("extdata",
    "google_transit_nyc_subway.zip",
    package = "tidytransit")
nyc <- read_gtfs(local_gtfs_path)
plot(nyc)
```

### print.tidygtfs

**Print a GTFS object**

**Description**

Prints a GTFS object suppressing the class attribute.

**Usage**

```r
## S3 method for class 'tidygtfs'
print(x, ...)
```
**raptor**

**Arguments**

- `x` A GTFS object.
- `...` Optional arguments ultimately passed to `format`.

**Value**

The GTFS object that was printed, invisibly

**Examples**

```r
## Not run:
path = system.file("extdata",
                   "google_transit_nyc_subway.zip",
                   package = "tidytransit")

g = read_gtfs(path)
print(g)

## End(Not run)
```

---

**raptor**  
*Calculate travel times from one stop to all reachable stops*

**Description**

`raptor` finds the minimal travel time, earliest or latest arrival time for all stops in `stop_times` with journeys departing from `stop_ids` within `time_range`.

**Usage**

```r
raptor(
  stop_times,
  transfers,
  stop_ids,
  arrival = FALSE,
  time_range = 3600,
  max_transfers = NULL,
  keep = "all"
)
```

**Arguments**

- `stop_times` A (prepared) `stop_times` table from a `gtfs` feed. Prepared means that all stop time rows before the desired journey departure time should be removed. The table should also only include departures happening on one day. Use `filter_stop_times()` for easier preparation.
- `transfers` Transfers table from a `gtfs` feed. In general no preparation is needed.
stop_ids  Character vector with stop_ids from where journeys should start (or end)
arrival   If FALSE (default), all journeys start from stop_ids. If TRUE, all journeys end at stop_ids.
time_range Departure or arrival time range in seconds. All departures from the first departure of stop_times (not necessarily from stop_id in stop_ids) within time_range are considered. If arrival is TRUE, all arrivals within time_range before the latest arrival time of stop_times are considered.
max_transfers Maximum number of transfers allowed, no limit (NULL) as default.
keep      One of c("all", "shortest", "earliest", "latest"). By default, all journeys arriving at a stop are returned. With shortest the journey with shortest travel time is returned. With earliest the journey arriving at a stop the earliest is returned, latest works accordingly.

Details
With a modified Round-Based Public Transit Routing Algorithm (RAPTOR) using data.table, earliest arrival times for all stops are calculated. If two journeys arrive at the same time, the one with the later departure time and thus shorter travel time is kept. By default, all journeys departing within time_range that arrive at a stop are returned in a table. If you want all journeys arriving at stop_ids within the specified time range, set arrival to TRUE.

Journeys are defined by a "from" and "to" stop_id, a departure, arrival and travel time. Note that the exact journeys (with each intermediate stop and route ids for example) is not returned.

For most cases, stop_times needs to be filtered, as it should only contain trips happening on a single day and departures later than a given journey start time, see filter_stop_times(). The algorithm scans all trips until it exceeds max_transfers or all trips in stop_times have been visited.

Value
A data.table with journeys (departure, arrival and travel time) to/from all stop_ids reachable by stop_ids.

See Also
travel_times() for an easier access to travel time calculations via stop_names.

Examples
nyc_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)

# you can use initial walk times to different stops in walking distance (arbitrary example values)
stop_ids_harlem_st <- c("301", "301N", "301S")
walk_times <- data.frame(stop_id = c(stop_ids_harlem_st, stop_ids_155_st),
walk_time = c(rep(600, 3), rep(410, 6)), stringsAsFactors = FALSE)
# Use journeys departing after 7 AM with arrival time before 11 AM on 26th of June
stop_times <- filter_stop_times(nyc, "2018-06-26", 7*3600, 9*3600)

# calculate all journeys departing from Harlem St or 155 St between 7:00 and 7:30
rptr <- raptor(stop_times, nyc$transfers, walk_times$stop_id, time_range = 1800,
keep = "all")

# add walk times to travel times
rptr <- merge(rptr, walk_times, by.x = "from_stop_id", by.y = "stop_id")
rptr$travel_time_incl_walk <- rptr$travel_time + rptr$walk_time

# get minimal travel times (with walk times) for all stop_ids
library(data.table)
shortest_travel_times <- setDT(rptr)[order(travel_time_incl_walk)][, .SD[, by = "to_stop_id"]
hist(shortest_travel_times$travel_time_incl_walk, breaks = 360)

---

**read_gtfs**

*Read and validate GTFS files*

**Description**

Reads GTFS text files from either a local .zip file or an URL and validates them against GTFS specifications.

**Usage**

read_gtfs(path, files = NULL, quiet = TRUE)

**Arguments**

- **path**
  - The path to a GTFS .zip file.

- **files**
  - A character vector containing the text files to be read from the GTFS (without the .txt extension). If NULL (the default) all existing files are read.

- **quiet**
  - Whether to hide log messages and progress bars (defaults to TRUE).

**Value**

A tidygtfs object: a list of tibbles in which each entry represents a GTFS text file. Additional tables are stored in the .sublist.

**See Also**

- validate_gtfs
Examples

```r
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
gtfs <- read_gtfs(local_gtfs_path)
names(gtfs)

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

---

route_type_names

**Dataframe of route type id's and the names of the types (e.g. "Bus")**

---

Description

Extended GTFS Route Types: https://developers.google.com/transit/gtfs/reference/extended-route-types

Usage

```r
route_type_names
```

Format

A data frame with 136 rows and 2 variables:

- **route_type** the id of route type
- **route_type_name** name of the gtfs route type

Source

https://gist.github.com/derhuerst/b0243339e22c310bee2386388151e11e

---

set_api_key

**Set TransitFeeds API key for recall**

Description

Set TransitFeeds API key for recall

Usage

```r
set_api_key()
```

Value

No value returned, function is used for setting environment variables
### set_servicepattern

**Calculate servicepattern ids for a gtfs feed**

**Description**

Each trip has a defined number of dates it runs on. This set of dates is called a service pattern in tidytransit. Trips with the same servicepattern id run on the same dates. In general, service_id can work this way but it is not enforced by the GTFS standard.

**Usage**

```r
set_servicepattern(
  gtfs_obj,
  id_prefix = "s_",
  hash_algo = "md5",
  hash_length = 7
)
```

**Arguments**

- `gtfs_obj`: tidytransit gtfs feed
- `id_prefix`: all servicepattern id will start with this string
- `hash_algo`: hashing algorithm used by digest
- `hash_length`: length the hash should be cut to with substr(). Use -1 if the full hash should be used

**Value**

modified gtfs_obj with added servicepattern list and a table linking trips and pattern (trip_servicepatterns)

### sf_as_tbl

**Convert stops and shapes from sf objects to tibbles**

**Description**

Coordinates are transformed to lon/lat

**Usage**

```r
sf_as_tbl(gtfs_obj)
```

**Arguments**

- `gtfs_obj`: tidygtfs object
Value

tidygtfs object with stops and shapes converted to tibbles

See Also

gtfs_as_sf

---

**sf_lines_to_df**

Adds the coordinates of an sf LINESTRING object as columns and rows

### Description

Adds the coordinates of an sf LINESTRING object as columns and rows

### Usage

```r
sf_lines_to_df(
  lines_sf,
  coord_colnames = c("shape_pt_lon", "shape_pt_lat"),
  remove_geometry = TRUE
)
```

### Arguments

- `lines_sf`: sf object
- `coord_colnames`: names of the new columns (existing columns are overwritten)
- `remove_geometry`: remove sf geometry column?

---

**sf_points_to_df**

Adds the coordinates of an sf POINT object as columns

### Description

Adds the coordinates of an sf POINT object as columns

### Usage

```r
sf_points_to_df(
  pts_sf,
  coord_colnames = c("stop_lon", "stop_lat"),
  remove_geometry = TRUE
)
```
shapes_as_sf

Arguments
pts_sf sf object
coord_colnames names of the new columns (existing columns are overwritten)
remove_geometry remove sf geometry column?

Description
Convert shapes into Simple Features Linestrings

Usage
shapes_as_sf(gtfs_shapes, crs = NULL)

Arguments
gtfs_shapes a gtfs$shapes dataframe
crs optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates

Value
an sf dataframe for gtfs shapes

stops_as_sf

Description
Convert stops into Simple Features Points

Usage
stops_as_sf(stops, crs = NULL)

Arguments
stops a gtfs$stops dataframe
crs optional coordinate reference system (used by sf::st_transform) to transform lon/lat coordinates
Value

an sf dataframe for gtfs routes with a point column

Examples

data(gtfs_duke)
some_stops <- gtfs_duke$stops[sample(nrow(gtfs_duke$stops), 40),]
some_stops_sf <- stops_as_sf(some_stops)
plot(some_stops_sf)

summary.tidygtfs  GTFS feed summary

Description

GTFS feed summary

Usage

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

Arguments

object  a gtfs_obj as read by read_gtfs()
...
    further specifications

Value

the tidygtfs object, invisibly

travel_times  Calculate shortest travel times from a stop to all reachable stops

Description

Function to calculate the shortest travel times from a stop (given by stop_name) to all other stops of a feed. filtered_stop_times needs to be created before with filter_stop_times() or filter_feed_by_date().
travel_times

Usage

travel_times(
    filtered_stop_times,
    stop_name,
    time_range = 3600,
    arrival = FALSE,
    max_transfers = NULL,
    max_departure_time = NULL,
    return_coords = FALSE,
    return_DT = FALSE
)

Arguments

filtered_stop_times
    stop_times data.table (with transfers and stops tables as attributes) created with filter_stop_times() where the departure or arrival time has been set. Alternatively, a filtered feed created by filter_feed_by_date() can be used.

stop_name
    Stop name for which travel times should be calculated. A vector with multiple names is accepted.

time_range
    All departures within this range in seconds after the first departure of filtered_stop_times are considered for journeys. If arrival is TRUE, all journeys arriving within time range before the latest arrival of filtered_stop_times are considered.

arrival
    If FALSE (default), all journeys start from stop_name. If TRUE, all journeys end at stop_name.

max_transfers
    The maximum number of transfers

max_departure_time
    Either set this parameter or time_range. Only departures before max_departure_time are used. Accepts "HH:MM:SS" or seconds as a numerical value. Unused if arrival is TRUE.

return_coords
    Returns stop coordinates as columns. Default is FALSE.

return_DT
    travel_times() returns a data.table if TRUE. Default is FALSE which returns a tibble.tbl_df.

Details

This function allows easier access to raptor() by using stop names instead of ids and returning shortest travel times by default.

Value

A table with travel times to/from all stops reachable by stop_name and their corresponding journey departure and arrival times.
validate_gtfs

Examples

```r
nyc_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(nyc_path)

# Use journeys departing after 7 AM with arrival time before 9 AM on 26th June
stop_times <- filter_stop_times(nyc, "2018-06-26", 7*3600, 9*3600)

stop_times <- travel_times(stop_times, "34 St - Herald Sq", return_coords = TRUE)
library(dplyr)
tts <- tts %>% filter(travel_time <= 3600)

# travel time to Queensboro Plaza is 810 seconds, 13:30 minutes
tts %>% filter(to_stop_name == "Queensboro Plaza") %>% pull(travel_time) %>% hms::hms()

# plot a simple map showing travel times to all reachable stops
# this can be expanded to isochron maps
library(ggplot2)
ggplot(tts) + geom_point(aes(x=to_stop_lon, y=to_stop_lat, color = travel_time))
```

validate_gtfs

Validate GTFS file

**Description**

Validates the GTFS object against GTFS specifications and raises warnings if required files/fields are not found. This function is called in `read_gtfs`.

**Usage**

`validate_gtfs(gtfs_obj, files = NULL, quiet = TRUE, warnings = TRUE)`

**Arguments**

- `gtfs_obj`: A GTFS object.
- `files`: A character vector containing the text files to be validated against the GTFS specification (without the `.txt` extension). If `NULL` (the default) the provided GTFS is validated against all possible GTFS text files.
- `quiet`: Whether to hide log messages (defaults to `TRUE`).
- `warnings`: Whether to display warning messages (defaults to `TRUE`).

**Value**

A `tidygtfs` with a `validation_result` attribute. This attribute is a tibble containing the validation summary of all possible fields from the specified files.
write_gtfs

Details

GTFS object’s files and fields are validated against the GTFS specifications as documented in Google’s Static GTFS Reference:

- GTFS feeds are considered valid if they include all required files and fields. If a required file/field is missing the function (optionally) raises a warning.
- Optional files/fields are listed in the reference above but are not required, thus no warning is raised if they are missing.
- Extra files/fields are those who are not listed in the reference above (either because they refer to a specific GTFS extension or due to any other reason).

Note that some files (calendar.txt, calendar_dates.txt and feed_info.txt) are conditionally required. This means that:

- calendar.txt is initially set as a required file. If it’s not present, however, it becomes optional and calendar_dates.txt (originally set as optional) becomes required.
- feed_info.txt is initially set as an optional file. If translations.txt is present, however, it becomes required.

Examples

```r
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
gtfs <- read_gtfs(local_gtfs_path)
attr(gtfs, "validation_result")

gtfs$shapes <- NULL
validation_result <- validate_gtfs(gtfs)

# should raise a warning
gtfs$stop_times <- NULL
validation_result <- validate_gtfs(gtfs)
```

write_gtfs
Write a tidygtfs object to a zip file

Description

Write a tidygtfs object to a zip file

Usage

```
write_gtfs(gtfs_obj, zipfile, compression_level = 9, as_dir = FALSE)
```
Arguments

- **gtfs_obj**: a tidygtfs object
- **zipfile**: path to the zip file the feed should be written to
- **compression_level**: a number between 1 and 9.9, passed to zip::zip
- **as_dir**: if TRUE, the feed is not zipped and zipfile is used as a directory path. Files within the directory will be overwritten.

Value

Invisibly returns gtfs_obj

Note

Auxilliary tidytransit tables (e.g. dates_services) are not exported.
# Index

<table>
<thead>
<tr>
<th>* datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>feedlist, 3</td>
</tr>
<tr>
<td>gdfs_duke, 13</td>
</tr>
<tr>
<td>route_type_names, 18</td>
</tr>
<tr>
<td>convert_times_to_hms, 3</td>
</tr>
<tr>
<td>feed_contains, 4</td>
</tr>
<tr>
<td>feedlist, 3</td>
</tr>
<tr>
<td>filter_feed_by_area, 4, 6</td>
</tr>
<tr>
<td>filter_feed_by_date, 4, 5, 6</td>
</tr>
<tr>
<td>filter_feed_by_date(), 22, 23</td>
</tr>
<tr>
<td>filter_feed_by_stops, 4, 5, 6</td>
</tr>
<tr>
<td>filter_feed_by_trips, 4–6, 6</td>
</tr>
<tr>
<td>filter_stop_times, 5, 7</td>
</tr>
<tr>
<td>filter_stop_times(), 15, 16, 22, 23</td>
</tr>
<tr>
<td>filter_stops, 7</td>
</tr>
<tr>
<td>get_feedlist, 8</td>
</tr>
<tr>
<td>get_route_frequency, 9</td>
</tr>
<tr>
<td>get_route_geometry, 10</td>
</tr>
<tr>
<td>get_stop_frequency, 10</td>
</tr>
<tr>
<td>get_trip_geometry, 11</td>
</tr>
<tr>
<td>gdfs_as_sf, 12, 20</td>
</tr>
<tr>
<td>gdfs_duke, 13</td>
</tr>
<tr>
<td>gdfs_transform, 13</td>
</tr>
<tr>
<td>hms::hms(), 3</td>
</tr>
<tr>
<td>plot.tidygtfs, 14</td>
</tr>
<tr>
<td>print.tidygtfs, 14</td>
</tr>
<tr>
<td>raptor, 15</td>
</tr>
<tr>
<td>raptor(), 8, 23</td>
</tr>
<tr>
<td>read_gdfs, 17, 24</td>
</tr>
<tr>
<td>read_gdfs(), 17, 22</td>
</tr>
<tr>
<td>route_type_names, 18</td>
</tr>
<tr>
<td>set_api_key, 18</td>
</tr>
<tr>
<td>set_servicepattern, 19</td>
</tr>
<tr>
<td>sf_as_tbl, 12, 19</td>
</tr>
</tbody>
</table>

| sf_lines_to_df, 20 |
| sf_points_to_df, 20 |
| shapes_as_sf, 21 |
| stops_as_sf, 21 |
| summary.tidygtfs, 22 |
| travel_times, 22 |
| travel_times(), 8, 16 |
| validate_gdfs, 17, 24 |
| write_gdfs, 25 |