

# Package ‘tidytransit’

February 28, 2021

**Type** Package

**Title** Read, Validate, Analyze, and Map Files in the General Transit Feed Specification

**Version** 0.7.3

**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: <http://gtfs.org/>.

**License** GPL

**LazyData** TRUE

**Depends** R (>= 3.6.0)

**Imports** dplyr, zip (>= 2.0.1), tibble, readr, data.table (>= 1.12.8), httr, assertthat, rlang, sf, lubridate, hms, tidyr, tools, digest, checkmate

**Suggests** testthat, knitr, rmarkdown, ggplot2, scales

**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],  
Daniel Herszenhut [aut] (<<https://orcid.org/0000-0001-8066-1105>>),  
Mark Padgham [aut],  
Tom Buckley [aut, cre],  
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** Tom Buckley <tom@tbuck1.com>

**Repository** CRAN

**Date/Publication** 2021-02-28 00:00:02 UTC

## R topics documented:

feedlist . . . . .	3
feed_contains . . . . .	3
filter_stops . . . . .	4
filter_stop_times . . . . .	5
get_feedlist . . . . .	5
get_route_frequency . . . . .	6
get_route_geometry . . . . .	7
get_stop_frequency . . . . .	7
get_trip_geometry . . . . .	8
gtfs_as_sf . . . . .	9
gtfs_duke . . . . .	10
plot.gtfs . . . . .	10
raptor . . . . .	11
read_gtfs . . . . .	13
route_type_names . . . . .	14
set_api_key . . . . .	14
set_date_service_table . . . . .	15
set_hms_times . . . . .	15
shapes_as_sf . . . . .	16
stops_as_sf . . . . .	16
summary.gtfs . . . . .	17
travel_times . . . . .	17
validate_gtfs . . . . .	19
write_gtfs . . . . .	20

<b>Index</b>	<b>21</b>
--------------	-----------

---

feedlist	<i>Dataframe of source GTFS data from Transitfeeds</i>
----------	--

---

**Description**

A dataset containing a list of URLs for GTFS feeds

**Usage**

```
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

**Source**

<http://www.transitfeeds.com/>

---

feed_contains	<i>Returns TRUE if the given gtfs_obj contains the table. Used to check for tidytransit's calculated tables in sublist</i>
---------------	--

---

**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_stops	<i>Get a set of stops for a given set of service ids and route ids</i>
--------------	--

---

**Description**

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

**Usage**

```
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 for a given service

**Examples**

```
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	<i>Filter a stop_times table for a given date and timespan.</i>
-------------------	---

---

**Description**

Filter a stop\_times table for a given date and timespan.

**Usage**

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

**Arguments**

gtfs_obj	a gtfs feed
extract_date	date to extract trips from in YYYY-MM-DD format
min_departure_time	The earliest departure time. Can be given as "HH:MM:SS", hms object or numeric value in seconds.
max_arrival_time	The latest arrival time. Can be given as "HH:MM:SS", hms object or numeric value in seconds This function creates filtered stop_times for <a href="#">travel_times()</a> and <a href="#">raptor()</a> . If you want to filter a feed multiple times it is faster to precalculate date_service_table with <a href="#">set_date_service_table()</a> .

**Examples**

```
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")
```

---

get_feedlist	<i>Get list of all available feeds from transitfeeds API</i>
--------------	--

---

**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**

```
## Not run:
feedlist_df <- get_feedlist()

## End(Not run)
```

---

get\_route\_frequency    *Get Route Frequency*

---

**Description**

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

**Usage**

```
get_route_frequency(
  gtfs_obj,
  start_hour = 6,
  end_hour = 22,
  service_ids = c(),
  dow = c(1, 1, 1, 1, 1, 0, 0)
)
```

**Arguments**

gtfs_obj	a list of gtfs dataframes as read by the tread package.
start_hour	(optional) an integer, default 6 (6 am)
end_hour	(optional) an integer, default 22 (10 pm)
service_ids	(optional) a string from the calendar dataframe identifying a particular service schedule.
dow	(optional) an integer vector with days of week. monday=1. default: c(1,1,1,1,1,0,0)

**Value**

a dataframe of routes with variables (gtfs\_obj\$.routes\_frequency) for headway/frequency for a route within a given time frame

**Examples**

```
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**

```
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**

```
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**

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

**Usage**

```
get_stop_frequency(
  gtfs_obj,
  start_hour = 6,
  end_hour = 22,
  service_ids = c(),
  dow = c(1, 1, 1, 1, 1, 0, 0),
  by_route = TRUE,
  wide = FALSE
)
```

**Arguments**

gtfs_obj	a list of gtfs dataframes as read by <code>read_gtfs()</code> .
start_hour	(optional) an integer indicating the start hour (default 6)
end_hour	(optional) an integer indicating the end hour (default 22)
service_ids	(optional) a set of service_ids from the calendar dataframe identifying a particular service id
dow	(optional) integer vector indicating which days of week to calculate for. default is weekday, e.g. <code>c(1,1,1,1,1,0,0)</code>
by_route	default TRUE, if FALSE then calculate headway for any line coming through the stop in the same direction on the same schedule.
wide	(optional) if true, then return a wide rather than tidy data frame

**Value**

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

**Examples**

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

---

get_trip_geometry	<i>Get all trip shapes for a given route and service.</i>
-------------------	---

---

**Description**

Get all trip shapes for a given route and service.

**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

**Examples**

```
data(gtfs_duke)  
gtfs_duke <- gtfs_as_sf(gtfs_duke)  
trips_sf <- get_trip_geometry(gtfs_duke, c("t_726295_b_19493_tn_41", "t_726295_b_19493_tn_40"))  
plot(trips_sf[1,])
```

---

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, quiet = TRUE)
```

**Arguments**

gtfs\_obj        a standard tidytransit gtfs object  
skip\_shapes    if TRUE, shapes are not converted. Default FALSE.  
quiet          boolean whether to print status messages

**Value**

gtfs\_obj a tidytransit gtfs object with stops and shapes as sf data frames

---

`gtfs_duke`*Example GTFS data*

---

**Description**

Data obtained from <http://data.trilliumtransit.com/gtfs/duke-nc-us/duke-nc-us.zip>.

**Usage**

```
gtfs_duke
```

**Format**

An object of class `gtfs` of length 24.

**See Also**

```
read_gtfs
```

---

`plot.gtfs`*Plot GTFS object routes and their frequencies*

---

**Description**

Plot GTFS object routes and their frequencies

**Usage**

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

**Arguments**

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

**Examples**

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

---

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

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

### Arguments

<code>stop_times</code>	A (prepared) <code>stop_times</code> 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 <a href="#">filter_stop_times()</a> for easier preparation.
<code>transfers</code>	Transfers table from a gtfs feed. In general no preparation is needed.
<code>stop_ids</code>	Character vector with <code>stop_ids</code> from where journeys should start (or end)
<code>arrival</code>	If FALSE (default), all journeys <i>start</i> from <code>stop_ids</code> . If TRUE, all journeys <i>end</i> at <code>stop_ids</code> .
<code>time_range</code>	Departure or arrival time range in seconds. All departures from the first departure of <code>stop_times</code> (not necessarily from <code>stop_id</code> in <code>stop_ids</code> ) within <code>time_range</code> are considered. If <code>arrival</code> is TRUE, all arrivals within <code>time_range</code> before the latest arrival time of <code>stop_times</code> are considered.
<code>max_transfers</code>	Maximum number of transfers allowed, no limit (NULL) as default.
<code>keep</code>	One of c("all", "shortest", "earliest", "latest"). By default, all journeys arriving at a stop are returned. With <code>shortest</code> the journey with shortest travel time is returned. With <code>earliest</code> the journey arriving at a stop the earliest is returned, <code>latest</code> 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")
stop_ids_155_st <- c("A11", "A11N", "A11S", "D12", "D12N", "D12S")
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[1], by = "to_stop_id"]
hist(shortest_travel_times$travel_time, breaks = 360)
```

---

read_gtfs	<i>Read and validate GTFS files</i>
-----------	-------------------------------------

---

### 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,  
  warnings = TRUE,  
  parse_dates = 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).
warnings	Whether to display warning messages (defaults to TRUE).
parse_dates	date columns in calendar.txt, calendar_dates.txt and feed_info.txt are converted to a Date object.

### Value

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

### See Also

[validate\\_gtfs](#)

### Examples

```
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	<i>Dataframe of route type id's and the names of the types (e.g. "Cable Car")</i>
------------------	---

---

**Description**

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

**Usage**

```
route_type_names
```

**Format**

A data frame with 132 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	<i>Set TransitFeeds API key for recall</i>
-------------	--

---

**Description**

Set TransitFeeds API key for recall

**Usage**

```
set_api_key()
```

---

`set_date_service_table`*Returns all possible date/service\_id combinations as a data frame*

---

**Description**

Use it to summarise service. For example, get a count of the number of services for a date. See [example](#).

**Usage**

```
set_date_service_table(gtfs_obj)
```

**Arguments**

`gtfs_obj` a `gtfs_object` as read by [read\\_gtfs\(\)](#)

**Value**

a `date_service` data frame

**Examples**

```
library(dplyr)
local_gtfs_path <- system.file("extdata", "google_transit_nyc_subway.zip", package = "tidytransit")
nyc <- read_gtfs(local_gtfs_path) %>% set_date_service_table()
nyc_services_by_date <- nyc$.date_service_table
# count the number of services running on each date
nyc_services_by_date %>% group_by(date) %>% count()
```

---

`set_hms_times`*Add hms::hms columns to feed*

---

**Description**

Adds columns to `stop_times` (`arrival_time_hms`, `departure_time_hms`) and frequencies (`start_time_hms`, `end_time_hms`) with times converted with [hms::hms\(\)](#).

**Usage**

```
set_hms_times(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

---

shapes\_as\_sf                      *Convert shapes into Simple Features Linestrings*

---

**Description**

Convert shapes into Simple Features Linestrings

**Usage**

```
shapes_as_sf(gtfs_shapes)
```

**Arguments**

gtfs\_shapes      a gtfs\$shapes dataframe

**Value**

an sf dataframe for gtfs shapes

---

stops\_as\_sf                      *Convert stops into Simple Features Points*

---

**Description**

Convert stops into Simple Features Points

**Usage**

```
stops_as_sf(stops)
```

**Arguments**

stops                      a gtfs\$stops dataframe

**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.gtfs	<i>GTFS feed summary</i>
--------------	--------------------------

---

**Description**

GTFS feed summary

**Usage**

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

**Arguments**

object	a gtfs_obj as read by <a href="#">read_gtfs()</a>
...	further specifications

---

travel_times	<i>Calculate shortest travel times from a stop to all reachable stops</i>
--------------	---

---

**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\(\)](#).

**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 <a href="#">filter_stop_times()</a> where the departure or arrival time has been set.
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 <code>filtered_stop_times</code> are considered for journeys. If arrival is TRUE, all journeys arriving within time range before the latest arrival of <code>filtered_stop_times</code> are considered.
arrival	If FALSE (default), all journeys <i>start</i> from <code>stop_name</code> . If TRUE, all journeys <i>end</i> at <code>stop_name</code> .
max_transfers	The maximum number of transfers
max_departure_time	Either set this parameter or <code>time_range</code> . Only departures before <code>max_departure_time</code> 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	<code>travel_times()</code> returns a <code>data.table</code> if TRUE. Default is FALSE which returns a <code>tibble/tbl_df</code> .

### 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.

### Examples

```
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)

tts <- 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	<i>Validate GTFS file</i>
---------------	---------------------------

---

### Description

Validates the GTFS object against GTFS specifications and raises warnings if required files/fields are not found.

### Usage

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

### Arguments

gtfs	A GTFS object as created by <a href="#">read_gtfs</a> .
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 GTFS object with a `validation_result` attribute. This attribute is a `data.table` containing the validation summary of all possible fields from the specified files.

### 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**

```

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	<i>Writes a gtfs object to a zip file. Calculated tidytransit tables and columns are not exported.</i>
------------	--

---

**Description**

Writes a gtfs object to a zip file. Calculated tidytransit tables and columns are not exported.

**Usage**

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

**Arguments**

gtfs_obj	a gtfs feed 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.

# Index

- \* **datasets**
  - feedlist, 3
  - gtfs\_duke, 10
  - route\_type\_names, 14
- feed\_contains, 3
- feedlist, 3
- filter\_stop\_times, 5
- filter\_stop\_times(), 11, 12, 17
- filter\_stops, 4
- get\_feedlist, 5
- get\_route\_frequency, 6
- get\_route\_geometry, 7
- get\_stop\_frequency, 7
- get\_trip\_geometry, 8
- gtfs\_as\_sf, 9
- gtfs\_duke, 10
- hms::hms(), 15
- plot.gtfs, 10
- raptor, 11
- raptor(), 5, 18
- read\_gtfs, 13, 19
- read\_gtfs(), 8, 15, 17
- route\_type\_names, 14
- set\_api\_key, 14
- set\_date\_service\_table, 15
- set\_date\_service\_table(), 5
- set\_hms\_times, 15
- shapes\_as\_sf, 16
- stops\_as\_sf, 16
- summary.gtfs, 17
- travel\_times, 17
- travel\_times(), 5, 12
- validate\_gtfs, 13, 19
- write\_gtfs, 20