# Package 'SIRItoGTFS'

February 6, 2018

**Type** Package

**Title** Compare SIRI Datasets to GTFS Tables

**Version** 0.2.2

**Date** 2018-02-06

**Description** Allows the user to compare SIRI (Service Interface for Real Time Information) data sets to their GTFS (General Transit Feed Specification) counterparts, a ``Request_id" column us needed for the SIRI data frame in order to subset parts of it for use.

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**Depends** R (>= 3.4.0)

**Imports** dplyr (>= 0.7.2), reshape2 (>= 1.4.2), sp (>= 1.2.5), rgdal (>= 1.2), data.table (>= 1.10.4), rgeos (>= 0.3.23), easycsv (>= 1.0.5)

**URL** https://github.com/bogind/SIRItoGTFS,

http://user47094.vs.easily.co.uk/siri/documentation.htm,

https://developers.google.com/transit/gtfs/

**BugReports** https://github.com/bogind/SIRItoGTFS/issues

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** Dror Bogin [aut, cre]

**Maintainer** Dror Bogin <dror.bogin@gmail.com>

**Repository** CRAN

**Date/Publication** 2018-02-06 08:20:41 UTC

# R topics documented:



---

`SIRItoGTFS-package` *Compare SIRI real time data with GTFS schedules*

---

## Description

SIRItoGTFS allows you to compare the actual arrival time of public transport with the predicted arrival time present in the schedules, represented with the GTFS tables

## Details

The only two functions you're likely to need from **SIRItoGTFS** are `readGTFS` and `STG`.

## Author(s)

**Maintainer**: Dror Bogin <dror.bogin@gmail.com>

## See Also

Useful links:

- https://github.com/bogind/SIRItoGTFS
- http://user47094.vs.easily.co.uk/siri/documentation.htm
- https://developers.google.com/transit/gtfs/
- Report bugs at https://github.com/bogind/SIRItoGTFS/issues

## downloadGTFSil *download the Israeli GTFS .zip file*

### Description

downloads the Israeli GTFS .zip file into the selected directory with a date added to the files name.

### Usage

```
downloadGTFSil(directory = NULL)
```

### Arguments

| | |
|---|---|
| directory | If directory remains NULL the function allows you to choose a directory manually via choose_dir |

### Value

A .zip file in the selected directory, not read into the environment

### Note

Only useful for Israeli users, other users should seek their local GTFS repository

### See Also

choose_dir, https://developers.google.com/transit/gtfs/, download.file

### Examples

```
## Not run:
downloadGTFSil()

## End(Not run)
```

## GTFSagency *GTFS Agency table relevant to SIRI sample*

### Description

Subset of the GTFS agency table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS agency table, extracted to be used with the SIRIsample data. it shows only 1 agency since the SIRIsample data only includes 1 operator in the city of Be'er Sheva. note that GTFS is a global standard and not all operators fill all of the columns.

## Usage

```
GTFSagency
```

## Format

A data frame with 1 observations on the following 7 variables:

**agency_id** An ID that uniquely identifies a transit agency

**agency_name** The agency_name field contains the full name of the transit agency.

**agency_url** The URL of the transit agency.

**agency_timezone** The timezone where the transit agency is located.

**agency_lang** A two-letter ISO 639-1 code for the primary language used by this transit agency.

**agency_phone** A single voice telephone number for the specified agency, NA in this case

**agency_fare_url** The URL of a web page that allows a rider to purchase tickets or other fare instruments for that agency online.

## Source

https://developers.google.com/transit/gtfs/reference/#agencytxt

---

# GTFScalendar *GTFS calendar table relevant to SIRI sample*

---

## Description

Subset of the GTFS calendar table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS calendar table, extracted to be used with the SIRIsample data. it shows the calendar data for only 1 agency since the SIRIsample data only includes 1 operator in the city of Be'er Sheva. note that GTFS is a global standard and not all operators fill all of the columns.

## Usage

```
GTFScalendar
```

## Format

A data frame with 152 observations on the following 10 variables:

**service_id** The service_id contains an ID that uniquely identifies a set of dates when service is available for one or more routes.

**sunday** A binary value that indicates whether the service is valid for all Sundays.

**monday** A binary value that indicates whether the service is valid for all Mondays

**tuesday** A binary value that indicates whether the service is valid for all Tuesdays.

**wednesday** A binary value that indicates whether the service is valid for all Wednesdays.

**thursday** A binary value that indicates whether the service is valid for all Thursdays.

**friday** A binary value that indicates whether the service is valid for all Fridays.

**saturday** A binary value that indicates whether the service is valid for all Saturdays.

**start_date** The start_date field contains the start date for the service. The start_date field's value should be in YYYYMMDD format.

**end_date** The end_date field contains the end date for the service. This date is included in the service interval. The end_date field's value should be in YYYYMMDD format.

## Source

https://developers.google.com/transit/gtfs/reference/#calendartxt

---

`GTFSroutes` *GTFS routes table relevant to SIRI sample*

---

## Description

Subset of the GTFS routes table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS routes table, extracted to be used with the SIRIsample data. it shows the routes data for the SIRIsample data. note that GTFS is a global standard and not all operators fill all of the columns.

## Usage

```
GTFSroutes
```

## Format

A data frame with 57 observations on the following 7 variables:

**route_id** An ID that uniquely identifies a route.

**agency_id** An agency for the specified route. This value is referenced from the agency file.

**route_short_name** The short name of a route. This will often be a short, abstract identifier like "32", "100X", or "Green" that riders use to identify a route, set to NA due it being in UTF-8 Hebrew

**route_long_name** The full name of a route, set to NA due it being in UTF-8 Hebrew

**route_desc** A description of a route, set to NA due it being in UTF-8 Hebrew.

**route_type** The type of transportation used on a route, in this case all values are 3 for bus.

**route_color** In systems that have colors assigned to routes, the route_color field defines a color that corresponds to a route.

## Source

https://developers.google.com/transit/gtfs/reference/#routestxt

## GTFSshapes *GTFS shapes table relevant to SIRI sample*

### Description

Subset of the GTFS shapes table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS shapes table, extracted to be used with the SIRIsample data. it shows the shapes data for the SIRIsample data. note that GTFS is a global standard and not all operators fill all of the columns.

### Usage

```
GTFSshapes
```

### Format

A data frame with 32580 observations on the following 4 variables:

**shape_id** An ID that uniquely identifies a shape.

**shape_pt_lat** A valid WGS 84 latitude. Each row in shapes represents a shape point.

**shape_pt_lon** A valid WGS 84 longitude value from -180 to 180. Each row in shapes represents a shape point.

**shape_pt_sequence** The point's sequence order along the shape.

### Source

https://developers.google.com/transit/gtfs/reference/#shapestxt

## GTFSstops *GTFS stops table relevant to SIRI sample*

### Description

Subset of the GTFS stops table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS stop_times table, extracted to be used with the SIRIsample data. it shows the stops data for the SIRIsample data, all within the city of Be'er Sheva, Israel. note that GTFS is a global standard and not all operators fill all of the columns.

### Usage

```
GTFSstops
```

## Format

A data frame with 551 observations on the following 9 variables:

`stop_id` An ID that uniquely identifies a stop, station, or station entrance.

`stop_code` A short text or a number that uniquely identifies the stop for passengers.

`stop_name` The name of a stop, station, or station entrance, set to NA due it being in UTF-8 Hebrew.

`stop_desc` The description of a stop, set to NA due it being in UTF-8 Hebrew.

`stop_lat` The latitude of a stop, station, or station entrance. The field value must be a valid WGS 84 latitude.

`stop_lon` The longitude of a stop, station, or station entrance. The field value must be a valid WGS 84 longitude value from -180 to 180.

`location_type` Identifies whether this stop ID represents a stop, station, or station entrance. If no location type is specified, or the location_type is blank, stop IDs are treated as stops.

`parent_station` For stops that are physically located inside stations, the field identifies the station associated with the stop.

`zone_id` Defines the fare zone for a stop ID.

## Source

https://developers.google.com/transit/gtfs/reference/#stopstxt

---

# GTFSstop_times *GTFS stop_times table relevant to SIRI sample*

---

## Description

Subset of the GTFS stop_times table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS stop_times table, extracted to be used with the SIRIsample data. it shows the stop times data for the SIRIsample data. note that GTFS is a global standard and not all operators fill all of the columns.

## Usage

```
GTFSstop_times
```

## Format

A data frame with 153021 observations on the following 8 variables:

**trip_id** An ID that identifies a trip.

**arrival_time** The arrival time at a specific stop for a specific trip on a route.

**departure_time** The departure time from a specific stop for a specific trip on a route.

**stop_id** An ID that uniquely identifies a stop. Multiple routes may use the same stop.

**stop_sequence** The order of the stops for a particular trip.

**pickup_type** Indicates whether passengers are picked up at a stop as part of the normal schedule or whether a pickup at the stop is not available. 0 for Regularly scheduled pickup, 1 for No pickup available

**drop_off_type** Indicates whether passengers are dropped off at a stop as part of the normal schedule or whether a drop off at the stop is not available. 0 for Regularly scheduled drop off, 1 for No drop off available.

**shape_dist_traveled** The distance from the first shape point.

## Source

https://developers.google.com/transit/gtfs/reference/#stop_timestxt

---

`GTFStrips` *GTFS trips table relevant to SIRI sample*

---

## Description

Subset of the GTFS trips table for Israel for the SIRIsample data. This is only a small subset of the Israeli GTFS routes table, extracted to be used with the SIRIsample data. it shows the trips data for the SIRIsample data. note that GTFS is a global standard and not all operators fill all of the columns.

## Usage

```
GTFStrips
```

## Format

A data frame with 5172 observations on the following 5 variables:

`route_id` An ID that uniquely identifies a route.

`service_id` An ID that uniquely identifies a set of dates when service is available for one or more routes. This value is referenced from the calendar table.

`trip_id` An ID that identifies a trip.

`direction_id` A binary value that indicates the direction of travel for a trip. 0 for travel in one direction (e.g. outbound travel), 1 for travel in the opposite direction (e.g. inbound travel).

`shape_id` An ID that defines a shape for the trip. This value is referenced from the shapes table.

## Source

https://developers.google.com/transit/gtfs/reference/#tripstxt

readGTFS *read GTFS files from a folder into R's environment*

## Description

reads multiple tables into the environment, adds the "GTFS" prefix by default, based on fread and fread_folder.

## Usage

```
readGTFS(directory = NULL, extension = "BOTH", sep = "auto",
  nrows = -1L, header = "auto", na.strings = "NA",
  stringsAsFactors = FALSE, verbose = getOption("datatable.verbose"),
  autostart = 1L, skip = 0L, drop = NULL, colClasses = NULL,
  integer64 = getOption("datatable.integer64"), dec = if (sep != ".") "."
  else ",", check.names = FALSE, encoding = "unknown", quote = "\"",
  strip.white = TRUE, fill = FALSE, blank.lines.skip = FALSE,
  key = NULL, prefix = "GTFS", minimal = FALSE,
  showProgress = getOption("datatable.showProgress"), data.table = FALSE)
```

## Arguments

| | |
|---|---|
| directory | a directory from which to read the GTFS tables, if NULL then a manual choice is provided on windows, Linux and OSX. |
| extension | "TXT" for tables in '.txt' files, "CSV" for tables in '.csv' files, "BOTH" for both file endings. Default is "BOTH" |
| sep | The separator between columns. Defaults to the first character in the set [,\t \|;:] that exists on line autostart outside quoted ("") regions, and separates the rows above autostart into a consistent number of fields, too. |
| nrows | The number of rows to read, by default -1 means all. Unlike read.table, it doesn't help speed to set this to the number of rows in the file (or an estimate), since the number of rows is automatically determined and is already fast. Only set nrows if you require the first 10 rows, for example. 'nrows=0' is a special case that just returns the column names and types; e.g., a dry run for a large file or to quickly check format consistency of a set of files before starting to read any. |
| header | Does the first data line contain column names? Defaults according to whether every non-empty field on the first data line is type character. If so, or TRUE is supplied, any empty column names are given a default name. |
| na.strings | A character vector of strings which are to be interpreted as NA values. By default ",," for columns read as type character is read as a blank string ("") and ",NA," is read as NA. Typical alternatives might be na.strings=NULL (no coercion to NA at all!) or perhaps na.strings=c("NA","N/A","null") |
| stringsAsFactors | Convert all character columns to factors? |
| verbose | Be chatty and report timings? |

`autostart` Any line number within the region of machine readable delimited text, by default 30. If the file is shorter or this line is empty (e.g. short files with trailing blank lines) then the last non empty line (with a non empty line above that) is used. This line and the lines above it are used to auto detect sep and the number of fields. It's extremely unlikely that autostart should ever need to be changed, we hope.

`skip` If 0 (default) use the procedure described below starting on line autostart to find the first data row. skip>0 means ignore autostart and take line skip+1 as the first data row (or column names according to header="auto"|TRUE|FALSE as usual). skip="string" searches for "string" in the file (e.g. a substring of the column names row) and starts on that line (inspired by read.xls in package gdata).

`drop` Vector of column names or numbers to drop, keep the rest.

`colClasses` A character vector of classes (named or unnamed), as read.csv. Or a named list of vectors of column names or numbers, see examples. colClasses in fread is intended for rare overrides, not for routine use. fread will only promote a column to a higher type if colClasses requests it. It won't downgrade a column to a lower type since NAs would result. You have to coerce such columns afterwards yourself, if you really require data loss.

`integer64` "integer64" (default) reads columns detected as containing integers larger than 2^31 as type bit64::integer64. Alternatively, "double"|"numeric" reads as base::read.csv does; i.e., possibly with loss of precision and if so silently. Or, "character".

`dec` The decimal separator as in base::read.csv. If not "." (default) then usually ",". See details.

`check.names` default is FALSE. If TRUE then the names of the variables in the data.table are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

`encoding` default is "unknown". Other possible options are "UTF-8" and "Latin-1". Note: it is not used to re-encode the input, rather enables handling of encoded strings in their native encoding.

`quote` By default ("\""), if a field starts with a doublequote, fread handles embedded quotes robustly as explained under Details. If it fails, then another attempt is made to read the field as is, i.e., as if quotes are disabled. By setting quote="", the field is always read as if quotes are disabled.

`strip.white` default is TRUE. Strips leading and trailing whitespaces of unquoted fields. If FALSE, only header trailing spaces are removed.

`fill` logical (default is FALSE). If TRUE then in case the rows have unequal length, blank fields are implicitly filled.

`blank.lines.skip` logical, default is FALSE. If TRUE blank lines in the input are ignored.

`key` Character vector of one or more column names which is passed to setkey. It may be a single comma separated string such as key="x,y,z", or a vector of names such as key=c("x","y","z"). Only valid when argument data.table=TRUE

`prefix` A character string to be prefixed to each table name, default is "GTFS".

| | |
|---|---|
| minimal | whether or not to read all the GTFS tables or just those needed for SIRItoGTFS, default is FALSE, meaning all GTFS tables will be read |
| showProgress | TRUE displays progress on the console using \r. It is produced in fread's C code where the very nice (but R level) txtProgressBar and tkProgressBar are not easily available. |
| data.table | logical. TRUE returns a data.table. FALSE returns a data.frame. default for SIRItoGTFS is FALSE, should be kept that way. |

## Value

Multiple data.frame containing a representation of the data in the file with the "GTFS" prefix.

## Warning

Do Not use this function on it's own, it is meant to be used only as part of the STG process

## References

Bogin, D., Levy, N. and Ben-Elia E. (2018) *EEstimation of Public Transportation Service Reliability Using Big Data and Open Source Tools*

## See Also

STG, fread, fread_folder

## Examples

```
require(SIRItoGTFS)
directory = getwd()
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/agency.csv"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/calendar.csv"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/routes.txt"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/shapes.txt"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/stop_times.txt"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/stops.txt"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/translations.txt"))
write.csv(data.frame(matrix(1:9, nrow = 3)), file = file.path(directory,"/trips.txt"))

# now we read just the minimal tables needed for `STG`,
# meaning everything besides shapes and translations
readGTFS(directory, minimal = TRUE, extension = "BOTH")
```

SIRIsample *A SIRI data frame sample*

## Description

A data sample of preprocessed SIRI download, includes the data for the local public transport operator in Be'er Sheva, Israel for the date of 19/07/2017 All columns are intentionally character vectors, this is raw data to be used in the analysis process of STG. The table includes the maximum possible columns a SIRI protocol can produce( with the Israeli server, including wrapper nodes which produce NA columns), The data was collected for the development of the SIRItoGTFS package and contains only 100 calls to server, the "request_id" and "call_time_toServer" columns were added locally and are needed for anyone trying to use the package. the data is provided for anyone wishing to test the methods used during research.

## Usage

```
SIRIsample
```

## Format

A data frame with 2500 observations on the following 22 variables.

`RecordedAtTime` Time stamp provided by the server, can differ within the same call, response rate for stops to server may vary

`ItemIdentifier` The server's ID for each observation

`MonitoringRef` The server's ID for the stop/group of stops

`MonitoredVehicleJourney` A wrapper node, can be ignored

`LineRef` The bus trip's route ID in the GTFS routes table

`DirectionRef` The trip's direction code

`PublishedLineName` The bus line's published name

`OperatorRef` The operator's ID in the GTFS agency table

`DestinationRef` The last stop's ID in the GTFS stops table

`OriginAimedDepartureTime` The scheduled departure time for the trip

`VehicleLocation` A wrapper node, can be ignored

`VehicleRef` A unique vehicle ID

`MonitoredCall` A wrapper node, can be ignored

`Longitude` The observations recorded Longitude on a WGS 84 projection

`Latitude` The observations recorded Latitude on a WGS 84 projection

`StopPointRef` The stops' ID in the GTFS stops table

`VehicleAtStop` Is the vehicle currently at the stop, by default NA which is FALSE

`ExpectedArrivalTime` The predicted time the bus will arrive at the stop

`request_id` A unique identifier for each call to server

`call_time_toServer` Local time stamp for each call, not for each response

`AimedArrivalTime` The scheduled time, only relevant for trips that have not yet begun

`ArrivalStatus` In fully operational systems this should report whether the bus is early or late, in this case it does not report anything

## Source

http://user47094.vs.easily.co.uk/siri/documentation.htm

## References

Bogin, D., Levy, N. and Ben-Elia E. (2018) *Estimation of Public Transportation Service Reliability Using Big Data and Open Source Tools*

---

`STG` *Wrapper function for the SIRItoGTFS library*

---

## Description

Performs a comparison between a SIRI data.frame and GTFS tables, requires the SIRI table as well as the minimal GTFS tables to be in the environment. should be used after `readGTFS`.

## Usage

```
STG(SIRIDF, GTFSstops., GTFSagency., GTFScalendar., GTFSroutes.,
  GTFSstop_times., GTFStrips., linerefs = NULL, epsg = 2039)
```

## Arguments

| | |
|---|---|
| `SIRIDF` | A `data.frame` containing SIRI protocol data downloaded from a public transportation authority. |
| `GTFSstops.` | A GTFS stops table, best load into environment with `readGTFS` |
| `GTFSagency.` | A GTFS agency table, best load into environment with `readGTFS` |
| `GTFScalendar.` | A GTFS calendar table, best load into environment with `readGTFS` |
| `GTFSroutes.` | A GTFS routes table, best load into environment with `readGTFS` |
| `GTFSstop_times.` | A GTFS stop_times table, best load into environment with `readGTFS` |
| `GTFStrips.` | A GTFS trips table, best load into environment with `readGTFS` |
| `linerefs` | Optional, a numeric vector of GTFS route_id numbers to process. if not used all route_id's in the SIRIDF provided will be used. |
| `epsg` | The EPSG code for the projection to be used. |

## Details

The function provides an "easy to use" wrapper for users unfamiliar with the functions in **SIRItoGTFS**. It should be used after a SIRI table has been read into R's environment along with GTFS tables who have a corresponding date. it is best used after `readGTFS`. the SIRI table used should have the minimal columns: "RecordedAtTime", "MonitoringRef", "LineRef", "DirectionRef", "PublishedLineName", "OperatorRef", "DestinationRef", "OriginAimedDepartureTime", "Longitude", "Latitude", "VehicleRef", "StopPointRef" & "ExpectedArrivalTime". The output table will contain a time and distance comparison between the schedule provided in the GTFS tables and the real-time data provided with the SIRI table.

## Value

A `data.frame` containing a comparison between a public transportation mode's schedule and real-time data.

## References

Bogin, D., Levy, N. and Ben-Elia E. (2018) *Estimation of Public Transportation Service Reliability Using Big Data and Open Source Tools*

## See Also

`readGTFS`

## Examples

```
require(SIRItoGTFS)
require(data.table)
# use the sample SIRI data included with the package
data("sirisample")
SIRIsample$Longitude = as.numeric(SIRIsample$Longitude)
SIRIsample$Latitude = as.numeric(SIRIsample$Latitude)
# load your own GTFS data with `readGTFS()`
# or use the subset of GTFS data conformable to the SIRI sample, also included in the package
data("GTFSstops")
data("GTFSstop_times")
data("GTFScalendar")
data("GTFStrips")
data("GTFSagency")
data("GTFSroutes")
busesDF = STG(SIRIsample,
              GTFSstops. = GTFSstops,
              GTFSagency. = GTFSagency,
              GTFScalendar. = GTFScalendar,
              GTFSroutes. = GTFSroutes,
              GTFSstop_times. = GTFSstop_times,
              GTFStrips. = GTFStrips,
              linerefs = unique(SIRIsample$LineRef[1]))
```

# Index