multigtfs Documentation

Release 1.1.1

John Whitlock

Aug 27, 2017

Contents

1 Status **3**

2 Example project **5**

3 Development **7**

3.1 Projects using multigtfs 7
3.2 Installation 7
3.3 Usage 8
3.4 Implementation of GTFS 8
3.5 How To Contribute 24
3.6 Authors 24
3.7 Project History 25
3.8 Future 25
3.9 Changelog 25

multigtfs is an Apache 2.0-licensed Django app that supports importing and exporting of GTFS feeds. All features of the June 20, 2012 reference are supported, including all changes up to February 17, 2014. It allows multiple feeds to be stored in the database at once.

It requires a spatial databases compatible with GeoDjango. PostgreSQL 9.x and PostGIS 2.x are recommended for development and production, since these support all the GeoDjango features.

CHAPTER 1

Status

multigtfs is ready for your GTFS project.

Point releases (such as 1.0.0 to 1.0.1) should be safe, only adding features or fixing bugs. Minor updates (1.0.1 to 1.1.0) may include significant changes that will break relying code. In the worst case scenario, you may need to export your GTFS feeds in the original version, update multigtfs and your code, and re-import.

multigtfs works with Django 1.8 (the long-term support, or LTS, release) through 1.11. Support will follow the Django supported releases, as well as the Python versions supported by those releases.

All valid GTFS feeds are supported for import and export. This includes feeds with extra columns not yet included in the GTFS spec, and feeds that omit `calendar.txt` in favor of `calendar_dates.txt` (such as the TriMet archive feeds). If you find a feed that doesn't work, file a bug!

See the issues list for more details on bugs and feature requests.

CHAPTER 2

Example project

Check out the example project.

If you have Docker installed and working, you can run the example project without installing a database.

1. Add one or more feeds to import to the folder `feeds/import`. You can find a feed for download at https://transitfeeds.com, such as Tulsa Transit's Feed.
2. Initialize the containers with `docker-compose up`. After a few minutes, it will display:

```
web_1  | Django version 1.8.18, using settings 'exploreproj.settings'
web_1  | Development server is running at http://0.0.0.0:8000/
web_1  | Using the Werkzeug debugger (http://werkzeug.pocoo.org/)
web_1  | Quit the server with CONTROL-C.
web_1  |  * Debugger is active!
web_1  |  * Debugger PIN: XXX-XXX-XXX
```

3. Visit http://localhost:8000 to view the example project.

See the example project for more details.

CHAPTER 3

Development

Code https://github.com/tulsawebdevs/django-multi-gtfs

Issues https://github.com/tulsawebdevs/django-multi-gtfs/issues

Dev Docs http://multigtfs.readthedocs.io/

IRC irc://irc.freenode.net/tulsawebdevs

Contents:

Projects using multigtfs

Projects using multigtfs include:

- GTFS Explorer - A web application from MRCagney to analyze and visualize public transport data.
- tulsa-transit-google - The original project that spawned multigtfs. Tulsa Transit no longer uses this code, but instead is using the GTFS export feature of their scheduling provider.

Want to see your project here? Send a pull request or open an issue.

Installation

At the command line:

```
$ easy_install multigtfs
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv multigtfs
$ pip install multigtfs
```

In your settings, add `multigtfs` to your `INSTALLED_APPS` and ensure you have a spatial database configured.

Use `./manage.py migrate` to install the tables.

Usage

Management Commands

There are two management commands to get GTFS feeds in and out of the database:

```
./manage.py importgtfs [--name name_of_feed] path/to/gtfsfeed.zip
./manage.py exportgtfs [--name basename_of_file] <feed_id>
```

A third command will update cached geometries, used for making geo-queries at the shape, trip, or route level:

```
./manage.py refreshgeometries --all   # Refresh all geometries
./manage.py refreshgeometries 1 2 3   # Refresh just feeds 1, 2, and 3
```

Note: cached geometries are normally updated whenever the related shape points or stops are updated. This command is useful for refreshing geometries after manual changes or after a bug fix (like the v0.3.3 update).

In Code

multigtfs is composed of Django models that implement GTFS, plus helper methods for importing and exporting to the GTFS format. Where GTFS relates objects through IDs (such as Stop IDs for stops), multigtfs uses ForeignKeys.

multigtfs includes a Feed object, which is not part of GTFS. This is used to include several feeds in the same file without collisions. These can be feeds from different agencies, or different versions of a feed from the same agency. The object has a helper method, `in_feed`, that is sometimes useful in filtering objects by feed. At other times, it is easier to start at the feed and follow relations.

See the next section, Implementation of GTFS, for details on how the GTFS specification is implemented in Django models. Load the app in your Django project, play with the admin, and read the source code to learn more.

Sample Project

The `examples/explore` sample project demonstrates a simple read-only website for viewing one or more GTFS feeds. It include OpenLayers maps for viewing the routes, trips, and shapes. You can use it as is, or as a starting place for your own projects. See the project README (`examples/explore/README.md`) for more information.

Implementation of GTFS

The GTFS specification is fairly stable. Updates occur about once a year, with the latest on Feburary 2nd, 2015. This page describes how multigtfs implements the spec, using text from the spec.

Contents

- *Implementation of GTFS*
  - *GTFS Feeds and general notes*

- *Required files*
  - *agency.txt (Agency)*
  - *stops.txt (Stop)*
  - *route.txt (Route)*
  - *trips.txt (Trip)*
  - *stop_times.txt (StopTime)*
  - *calendar.txt (Service)*
- *Optional Files*
  - *calendar_dates.txt (ServiceDate)*
  - *fare_attributes.txt (Fare)*
  - *fare_rules.txt (FareRule)*
  - *shapes.txt (ShapePoint)*
  - *frequencies.txt (Frequency)*
  - *transfers.txt (Transfer)*
  - *feed_info.txt (FeedInfo)*

GTFS Feeds and general notes

A GTFS feed is a zipfile containing one or more .txt files, containing comma-delimited text. The management commands `importgtfs` and `exportgtfs` can read and create these files (see the docs on Management Commands for more info).

Each .txt file is mapped to a Django model, with additonal models to handle implicit relations in GTFS, and to handle multiple feeds in a single database.

The Feed model doesn't export as a file in the feed, but instead holds feed metadata, including these fields:

- `name` - A human-friendly name for the feed
- `created` - The time the feed was created
- `meta` - A JSON-encoded field with extra data about the feed. For example, if a stops.txt table include an additional column "shelter", then `meta` will contain:

```
{
    "extra_columns": {
        "Stop": [
            "shelter"
        ]
    }
}
```

Models that map to feed records commonly include these fields:

- `feed` - The relation to the Feed
- `id` - The database ID for the record
- `extra_data` - A JSON-encoded field tracking extra data for the record. For example, if a stops.txt file contains an extra column "shelter" with a value "1", `extra_data` will contain:

```
{
    "shelter": "1"
}
```

`Feed.extra_columns` and `<Model>.extra_data` work together to allow importing and exporting of non-standard feeds without losing data.

Required files

The specification marks these as required files for a valid GTFS feed:

- agency.txt - One or more transit agencies that provide the data in this feed.
- stops.txt - Individual locations where vehicles pick up or drop off passengers.
- routes.txt - Transit routes. A route is a group of trips that are displayed to riders as a single service.
- trips.txt - Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
- stop_times.txt - Times that a vehicle arrives at and departs from individual stops for each trip.
- calendar.txt - Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available. *Note - if calendar_dates.txt specifies all the feed dates, then this file can be empty. TriMet in Portland, OR, uses this style for their feed.*

agency.txt (Agency)

`agency.txt` is implemented in the `Agency` model. The fields are:

- agency_id (`Agency.agency_id`) (*optional*): The agency_id field is an ID that uniquely identifies a transit agency. A transit feed may represent data from more than one agency. The agency_id is dataset unique. This field is optional for transit feeds that only contain data for a single agency.
- agency_name (`Agency.name`) (*required*): The agency_name field contains the full name of the transit agency. Google Maps will display this name.
- agency_url (`Agency.url`) (*required*): The agency_url field contains the URL of the transit agency. The value must be a fully qualified URL that includes `http://` or `https://`, and any special characters in the URL must be correctly escaped. See http://www.w3.org/Addressing/URL/4_URI_Recommentations.html for a description of how to create fully qualified URL values.
- agency_timezone (`Agency.timezone`) (*required*): The agency_timezone field contains the timezone where the transit agency is located. Timezone names never contain the space character but may contain an underscore. Please refer to http://en.wikipedia.org/wiki/List_of_tz_zones for a list of valid values. If multiple agencies are specified in the feed, each must have the same agency_timezone.
- agency_lang (`Agency.lang`) (*optional*): The agency_lang field contains a two-letter ISO 639-1 code for the primary language used by this transit agency. The language code is case-insensitive (both en and EN are accepted). This setting defines capitalization rules and other language-specific settings for all text contained in this transit agency's feed. Please refer to http://www.loc.gov/standards/iso639-2/php/code_list.php for a list of valid values.
- agency_phone (`Agency.phone`) (*optional*): The agency_phone field contains a single voice telephone number for the specified agency. This field is a string value that presents the telephone number as typical for the agency's service area. It can and should contain punctuation marks to group the digits of the number. Dialable text (for example, TriMet's "503-238-RIDE") is permitted, but the field must not contain any other descriptive text.

- agency_fare_url (`Agency.fare_url`) (*optional*): The agency_fare_url specifies the URL of a web page that allows a rider to purchase tickets or other fare instruments for that agency online. The value must be a fully qualified URL that includes http:// or https://, and any special characters in the URL must be correctly escaped. See http://www.w3.org/Addressing/URL/4_URI_Recommentations.html for a description of how to create fully qualified URL values.

stops.txt (Stop)

`stops.txt` is implemented in the `Stop` model. The fields are:

- stop_id (`Stop.stop_id`) (*required*): The stop_id field contains an ID that uniquely identifies a stop or station. Multiple routes may use the same stop. The stop_id is dataset unique.
- stop_code (`Stop.code`) (*optional*): The stop_code field contains short text or a number that uniquely identifies the stop for passengers. Stop codes are often used in phone-based transit information systems or printed on stop signage to make it easier for riders to get a stop schedule or real-time arrival information for a particular stop.

  The stop_code field should only be used for stop codes that are displayed to passengers. For internal codes, use stop_id. This field should be left blank for stops without a code.
- stop_name (`Stop.name`) (*required*): The stop_name field contains the name of a stop or station. Please use a name that people will understand in the local and tourist vernacular.
- stop_desc (`Stop.desc`) (*optional*): The stop_desc field contains a description of a stop. Please provide useful, quality information. Do not simply duplicate the name of the stop.
- stop_lat (`Stop.point`) (*required*): The stop_lat field contains the latitude of a stop or station. The field value must be a valid WGS 84 latitude.
- stop_lon (`Stop.point`) (*required*): The stop_lon field contains the longitude of a stop or station. The field value must be a valid WGS 84 longitude value from -180 to 180.
- zone_id (`Stop.zone`) (*optional*): The zone_id field defines the fare zone for a stop ID. Zone IDs are required if you want to provide fare information using fare_rules.txt. If this stop ID represents a station, the zone ID is ignored.
- stop_url (`Stop.url`) (*optional*): The stop_url field contains the URL of a web page about a particular stop. This should be different from the agency_url and the route_url fields.

  The value must be a fully qualified URL that includes `http://` or `https://`, and any special characters in the URL must be correctly escaped. See http://www.w3.org/Addressing/URL/4_URI_Recommentations.html for a description of how to create fully qualified URL values.
- location_type (`Stop.location_type`) (*optional*): The location_type field identifies whether this stop ID represents a stop or station. If no location type is specified, or the location_type is blank, stop IDs are treated as stops. Stations may have different properties from stops when they are represented on a map or used in trip planning.

  The location type field can have the following values:

  - **0 or blank - Stop. A location where passengers board or disembark from a** transit vehicle.
  - 1 - Station. A physical structure or area that contains one or more stop.
- parent_station (`Stop.parent_station`) (*optional*): For stops that are physically located inside stations, the parent_station field identifies the station associated with the stop. To use this field, stops.txt must also contain a row where this stop ID is assigned location type=1.

  If this stop ID represents a stop located inside a station, this entry's location type should be 0 or blank, and the entry's parent_station field contains the stop ID of the station where this stop is located. The stop referenced by parent_station must have location_type=1.

If this stop ID represents a stop located outside a station, this entry's location type should be 0 or blank, and the entry's parent_station field contains a blank value. The parent_station field doesn't apply to this stop.

If this stop ID represents a station, this entry's location type should be 1, and the entry's parent_station field should be a blank value. Stations can't contain other stations.

- stop_timezone (`Stop.timezone`) (*optional*): The stop_timezone field contains the timezone in which this stop or station is located. Please refer to Wikipedia List of Timezones for a list of valid values: http://en.wikipedia.org/wiki/List_of_tz_zones

  If omitted, the stop should be assumed to be located in the timezone specified by agency_timezone in agency.txt.

  When a stop has a parent station, the stop is considered to be in the timezone specified by the parent station's stop_timezone value. If the parent has no stop_timezone value, the stops that belong to that station are assumed to be in the timezone specified by agency_timezone, even if the stops have their own stop_timezone values. In other words, if a given stop has a parent_station value, any stop_timezone value specified for that stop must be ignored.

  Even if stop_timezone values are provided in stops.txt, the times in stop_times.txt should continue to be specified as time since midnight in the timezone specified by agency_timezone in agency.txt. This ensures that the time values in a trip always increase over the course of a trip, regardless of which timezones the trip crosses.

- wheelchair_boarding (`Stop.wheelchair_boarding`) (*optional*): The wheelchair_boarding field identifies whether wheelchair boardings are possible from the specified stop or station. The field can have the following values:
  - **0 (or empty) - indicates that there is no accessibility information for** the stop
  - **1 - indicates that at least some vehicles at this stop can be boarded** by a rider in a wheelchair
  - 2 - wheelchair boarding is not possible at this stop

  When a stop is part of a larger station complex, as indicated by a stop with a parent_station value, the stop's wheelchair_boarding field has the following additional semantics:
  - **0 (or empty) - the stop will inherit its wheelchair_boarding value from** the parent station, if specified in the parent
  - **1 - there exists some accessible path from outside the station to the** specific stop / platform
  - **2 - there exists no accessible path from outside the station to the** specific stop / platform

route.txt (Route)

`route.txt` is implemented in `Route`.

- route_id (`Route.route_id`) (*required*): The route_id field contains an ID that uniquely identifies a route. The route_id is dataset unique.
- agency_id (`Route.agency.agency_id`) (*optional*): The agency_id field defines an agency for the specified route. This value is referenced from the agency.txt file. Use this field when you are providing data for routes from more than one agency.
- route_short_name (`Route.short_name`) (*required*): The route_short_name contains 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, but which doesn't give any indication of what places the route serves. If the route does not have a short name, please specify a route_long_name and use an empty string as the value for this field.

  See a Google Maps screenshot highlighting the route_short_name: http://bit.ly/yIS1sa
- route_long_name (`Route.long_name`) (*required*): The route_long_name contains the full name of a route. This name is generally more descriptive than the route_short_name and will often include the route's destination

or stop. If the route does not have a long name, please specify a route_short_name and use an empty string as the value for this field.

See a Google Maps screenshot highlighting the route_long_name: http://bit.ly/wZw5yH

- route_desc (`Route.desc`) (*optional*): The route_desc field contains a description of a route. Please provide useful, quality information. Do not simply duplicate the name of the route. For example, "A trains operate between Inwood-207 St, Manhattan and Far Rockaway-Mott Avenue, Queens at all times. Also from about 6AM until about midnight, additional A trains operate between Inwood-207 St and Lefferts Boulevard (trains typically alternate between Lefferts Blvd and Far Rockaway)."
- route_type (`Route.rtype`) (*required*): The route_type field describes the type of transportation used on a route. Valid values for this field are:
  - **0 - Tram, Streetcar, Light rail. Any light rail or street level system** within a metropolitan area.
  - **1 - Subway, Metro. Any underground rail system within a metropolitan** area.
  - 2 - Rail. Used for intercity or long-distance travel.
  - 3 - Bus. Used for short- and long-distance bus routes.
  - 4 - Ferry. Used for short- and long-distance boat service.
  - **5 - Cable car. Used for street-level cable cars where the cable runs** beneath the car.
  - **6 - Gondola, Suspended cable car. Typically used for aerial cable cars** where the car is suspended from the cable.
  - 7 - Funicular. Any rail system designed for steep inclines.

  See a Google Maps screenshot highlighting the route_type: http://bit.ly/wSt2h0
- route_url (`Route.url`) (*optional*): The route_url field contains the URL of a web page about that particular route. This should be different from the agency_url.

  The value must be a fully qualified URL that includes http:// or https://, and any special characters in the URL must be correctly escaped. See http://www.w3.org/Addressing/URL/4_URI_Recommentations.html for a description of how to create fully qualified URL values.
- route_color (`Route.color`) (*optional*): In systems that have colors assigned to routes, the route_color field defines a color that corresponds to a route. The color must be provided as a six-character hexadecimal number, for example, 00FFFF. If no color is specified, the default route color is white (FFFFFF).

  The color difference between route_color and route_text_color should provide sufficient contrast when viewed on a black and white screen. The W3C Techniques for Accessibility Evaluation And Repair Tools document offers a useful algorithm for evaluating color contrast: http://www.w3.org/TR/AERT#color-contrast

  There are also helpful online tools for choosing contrasting colors, including the snook.ca Color Contrast Check application: http://snook.ca/technical/colour_contrast/colour.html
- route_text_color (`Route.text_color`) (*optional*): The route_text_color field can be used to specify a legible color to use for text drawn against a background of route_color. The color must be provided as a six-character hexadecimal number, for example, FFD700. If no color is specified, the default text color is black (000000).

  The color difference between route_color and route_text_color should provide sufficient contrast when viewed on a black and white screen.

trips.txt (Trip)

`trips.txt` is implemented in `Trip`

- route_id (`Trip.route.route_id`) (*required*): The route_id field contains an ID that uniquely identifies a route. This value is referenced from the routes.txt file.
- service_id (`Trip.service.service_id`) (*required*): The service_id contains 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.txt or calendar_dates.txt file.
- trip_id (`Trip.trip_id`) (*required*): The trip_id field contains an ID that identifies a trip. The trip_id is dataset unique.
- trip_headsign (`Trip.headsign`) (*optional*): The trip_headsign field contains the text that appears on a sign that identifies the trip's destination to passengers. Use this field to distinguish between different patterns of service in the same route. If the headsign changes during a trip, you can override the trip_headsign by specifying values for the the stop_headsign field in stop_times.txt.

  See a Google Maps screenshot highlighting the headsign: http://bit.ly/A3ot2j
- trip_short_name (`Trip.short_name`) (*optional*): The trip_short_name field contains the text that appears in schedules and sign boards to identify the trip to passengers, for example, to identify train numbers for commuter rail trips. If riders do not commonly rely on trip names, please leave this field blank.

  A trip_short_name value, if provided, should uniquely identify a trip within a service day; it should not be used for destination names or limited/express designations.
- direction_id (`Trip.direction`) (*optional*): The direction_id field contains a binary value that indicates the direction of travel for a trip. Use this field to distinguish between bi-directional trips with the same route_id. This field is not used in routing; it provides a way to separate trips by direction when publishing time tables. You can specify names for each direction with the trip_headsign field.
  - 0 - travel in one direction (e.g. outbound travel)
  - 1 - travel in the opposite direction (e.g. inbound travel)

  For example, you could use the trip_headsign and direction_id fields together to assign a name to travel in each direction on trip "1234", the trips.txt file would contain these rows for use in time tables:

```
trip_id, ... ,trip_headsign,direction_id
1234, ... , to Airport,0
1505, ... , to Downtown,1
```

- block_id (`Trip.block.block_id`) (*optional*): The block_id field identifies the block to which the trip belongs. A block consists of two or more sequential trips made using the same vehicle, where a passenger can transfer from one trip to the next just by staying in the vehicle. The block_id must be referenced by two or more trips in trips.txt.
- shape_id (`Trip.shape.shape_id`) (*optional*): The shape_id field contains an ID that defines a shape for the trip. This value is referenced from the shapes.txt file. The shapes.txt file allows you to define how a line should be drawn on the map to represent a trip.
- **wheelchair_accessible (`Trip.wheelchair_accessible`) (*optional*):**
  - **0 (or empty) - indicates that there is no accessibility information for** the trip
  - **1 - indicates that the vehicle being used on this particular trip can** accommodate at least one rider in a wheelchair
  - **2 - indicates that no riders in wheelchairs can be accommodated on this** trip
- **bikes_allowed (`Trip.bikes_allowed`) (*optional*):**
  - 0 (or empty) - indicates that there is no bike information for the trip

  - **1 - indicates that the vehicle being used on this particular trip can** accommodate at least one bicycle
  - 2 - indicates that no bicycles are allowed on this trip

stop_times.txt (StopTime)

`stop_times.txt` is implemented in `StopTime`.

- trip_id (`StopTime.trip.trip_id`) (*required*): The trip_id field contains an ID that identifies a trip. This value is referenced from the trips.txt file.
- arrival_time (`StopTime.arrival_time`) (*required*): The arrival_time specifies the arrival time at a specific stop for a specific trip on a route. The time is measured from "noon minus 12h" (effectively midnight, except for days on which daylight savings time changes occur) at the beginning of the service date. For times occurring after midnight on the service date, enter the time as a value greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins. If you don't have separate times for arrival and departure at a stop, enter the same value for arrival_time and departure_time.

  You must specify arrival times for the first and last stops in a trip. If this stop isn't a time point, use an empty string value for the arrival_time and departure_time fields. Stops without arrival times will be scheduled based on the nearest preceding timed stop. To ensure accurate routing, please provide arrival and departure times for all stops that are time points. Do not interpolate stops.

  Times must be eight digits in HH:MM:SS format (H:MM:SS is also accepted, if the hour begins with 0). Do not pad times with spaces. The following columns list stop times for a trip and the proper way to express those times in the arrival_time field:

Time	arrival_time value
08:10:00 A.M.	08:10:00 or 8:10:00
01:05:00 P.M.	13:05:00
07:40:00 P.M.	19:40:00
01:55:00 A.M.	25:55:00

  Note: Trips that span multiple dates will have stop times greater than 24:00:00. For example, if a trip begins at 10:30:00 p.m. and ends at 2:15:00 a.m. on the following day, the stop times would be 22:30:00 and 26:15:00. Entering those stop times as 22:30:00 and 02:15:00 would not produce the desired results.

- departure_time (`StopTime.departure_time`) (*required*): The departure_time specifies the departure time from a specific stop for a specific trip on a route. The time is measured from "noon minus 12h" (effectively midnight, except for days on which daylight savings time changes occur) at the beginning of the service date. For times occurring after midnight on the service date, enter the time as a value greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins. If you don't have separate times for arrival and departure at a stop, enter the same value for arrival_time and departure_time.

  You must specify departure times for the first and last stops in a trip. If this stop isn't a time point, use an empty string value for the arrival_time and departure_time fields. Stops without arrival times will be scheduled based on the nearest preceding timed stop. To ensure accurate routing, please provide arrival and departure times for all stops that are time points. Do not interpolate stops.

  Times must be eight digits in HH:MM:SS format (H:MM:SS is also accepted, if the hour begins with 0). Do not pad times with spaces. The following columns list stop times for a trip and the proper way to express those times in the departure_time field:

Time	departure_time value
08:10:00 A.M.	08:10:00 or 8:10:00
01:05:00 P.M.	13:05:00
07:40:00 P.M.	19:40:00
01:55:00 A.M.	25:55:00

Note: Trips that span multiple dates will have stop times greater than 24:00:00. For example, if a trip begins at 10:30:00 p.m. and ends at 2:15:00 a.m. on the following day, the stop times would be 22:30:00 and 26:15:00. Entering those stop times as 22:30:00 and 02:15:00 would not produce the desired results.

- stop_id (`StopTime.stop.stop_id`) (*required*): The stop_id field contains an ID that uniquely identifies a stop. Multiple routes may use the same stop. The stop_id is referenced from the stops.txt file. If location_type is used in stops.txt, all stops referenced in stop_times.txt must have location_type of 0.

  Where possible, stop_id values should remain consistent between feed updates. In other words, stop A with stop_id 1 should have stop_id 1 in all subsequent data updates. If a stop is not a time point, enter blank values for arrival_time and departure_time.

- stop_sequence (`StopTime.stop_sequence`) (*required*): The stop_sequence field identifies the order of the stops for a particular trip. The values for stop_sequence must be non-negative integers, and they must increase along the trip.

  For example, the first stop on the trip could have a stop_sequence of 1, the second stop on the trip could have a stop_sequence of 23, the third stop could have a stop_sequence of 40, and so on.

- stop_headsign (`StopTime.stop_headsign`) (*optional*): The stop_headsign field contains the text that appears on a sign that identifies the trip's destination to passengers. Use this field to override the default trip_headsign when the headsign changes between stops. If this headsign is associated with an entire trip, use trip_headsign instead.

  See a Google Maps screenshot highlighting the headsign: http://bit.ly/y2EO6a

- pickup_type (`StopTime.pickup_type`) (*optional*): The pickup_type field 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. This field also allows the transit agency to indicate that passengers must call the agency or notify the driver to arrange a pickup at a particular stop. Valid values for this field are:

  - 0 - Regularly scheduled pickup
  - 1 - No pickup available
  - 2 - Must phone agency to arrange pickup
  - 3 - Must coordinate with driver to arrange pickup

  The default value for this field is 0.

- drop_off_type (`StopTime.drop_off_type`) (*optional*): The drop_off_type field 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. This field also allows the transit agency to indicate that passengers must call the agency or notify the driver to arrange a drop off at a particular stop. Valid values for this field are:

  - 0 - Regularly scheduled drop off
  - 1 - No drop off available
  - 2 - Must phone agency to arrange drop off
  - 3 - Must coordinate with driver to arrange drop off

  The default value for this field is 0.

- shape_dist_traveled (`StopTime.shape_dist_traveled`) (*optional*): When used in the stop_times.txt file, the shape_dist_traveled field positions a stop as a distance from the first shape point. The shape_dist_traveled field represents a real distance traveled along the route in units such as feet or kilometers. For example, if a bus travels a distance of 5.25 kilometers from the start of the shape to the stop, the shape_dist_traveled for the stop ID would be entered as "5.25". This information allows the trip planner to determine how much of the shape to draw when showing part of a trip on the map. The values used for

shape_dist_traveled must increase along with stop_sequence: they cannot be used to show reverse travel along a route.

The units used for shape_dist_traveled in the stop_times.txt file must match the units that are used for this field in the shapes.txt file.

calendar.txt (Service)

`calendar.txt` is implemented in `Service`.

- service_id (`Service.service_id`) (*required*): The service_id contains an ID that uniquely identifies a set of dates when service is available for one or more routes. Each service_id value can appear at most once in a calendar.txt file. This value is dataset unique. It is referenced by the trips.txt file.
- monday (`Service.monday`) (*required*): The monday field contains a binary value that indicates whether the service is valid for all Mondays.
  - A value of 1 indicates that service is available for all Mondays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Mondays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- tuesday (`Service.tuesday`) (*required*): The tuesday field contains a binary value that indicates whether the service is valid for all Tuesdays.
  - A value of 1 indicates that service is available for all Tuesdays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Tuesdays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- wednesday (`Service.wednesday`) (*required*): The wednesday field contains a binary value that indicates whether the service is valid for all Wednesdays.
  - A value of 1 indicates that service is available for all Wednesdays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Wednesdays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- thursday (`Service.thursday`) (*required*): The thursday field contains a binary value that indicates whether the service is valid for all Thursdays.
  - A value of 1 indicates that service is available for all Thursdays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Thursdays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- friday (`Service.friday`) (*required*): The friday field contains a binary value that indicates whether the service is valid for all Fridays.
  - A value of 1 indicates that service is available for all Fridays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Fridays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.

- saturday (`Service.saturday`) (*required*): The saturday field contains a binary value that indicates whether the service is valid for all Saturdays.
  - A value of 1 indicates that service is available for all Saturdays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Saturdays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- sunday (`Service.sunday`) (*required*): The sunday field contains a binary value that indicates whether the service is valid for all Sundays.
  - A value of 1 indicates that service is available for all Sundays in the date range. (The date range is specified using the start_date and end_date fields.)
  - A value of 0 indicates that service is not available on Sundays in the date range.

  Note: You may list exceptions for particular dates, such as holidays, in the calendar_dates.txt file.
- start_date (`Service.start_date`) (*required*): The start_date field contains the start date for the service.

  The start_date field's value should be in YYYYMMDD format.
- end_date (`Service.end_date`) (*required*): 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.

Optional Files

The specification marks these as optional files for a valid GTFS feed:

- calendar_dates.txt - Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
- fare_attributes.txt - Fare information for a transit organization's routes.
- fare_rules.txt - Rules for applying fare information for a transit organization's routes.
- shapes.txt - Rules for drawing lines on a map to represent a transit organization's routes. *Note: If this data is not included, then routes will be drawn as straight lines between stops.*
- frequencies.txt - Headway (time between trips) for routes with variable frequency of service.
- transfers.txt - Rules for making connections at transfer points between routes.
- feed_info.txt - Additional information about the feed itself, including publisher, version, and expiration information.

calendar_dates.txt (ServiceDate)

`calendar_dates.txt` is implemented in `ServiceDate`

The calendar_dates table allows you to explicitly activate or disable service IDs by date. You can use it in two ways.

Recommended: Use `calendar_dates.txt` in conjunction with `calendar.txt`, where `calendar_dates.txt` defines any exceptions to the default service categories defined in the `calendar.txt` file. If your service is generally regular, with a few changes on explicit dates (for example, to accomodate special event services, or a school schedule), this is a good approach.

Alternate: Omit `calendar.txt`, and include ALL dates of service in `calendar_dates.txt`. If your schedule varies most days of the month, or you want to programmatically output service dates without specifying a normal weekly schedule, this approach may be preferable.

- service_id (`ServiceDate.service.service_id`) (*required*): The service_id contains an ID that uniquely identifies a set of dates when a service exception is available for one or more routes. Each (service_id, date) pair can only appear once in calendar_dates.txt. If the a service_id value appears in both the calendar.txt and calendar_dates.txt files, the information in calendar_dates.txt modifies the service information specified in calendar.txt. This field is referenced by the trips.txt file.
- date (`ServiceDate.date`) (*required*): The date field specifies a particular date when service availability is different than the norm. You can use the exception_type field to indicate whether service is available on the specified date.

  The date field's value should be in YYYYMMDD format.
- exception_type (`ServiceDate.exception_type`) (*required*): The exception_type indicates whether service is available on the date specified in the date field.
  - A value of 1 indicates that service has been added for the specified date.
  - A value of 2 indicates that service has been removed for the specified date.

  For example, suppose a route has one set of trips available on holidays and another set of trips available on all other days. You could have one service_id that corresponds to the regular service schedule and another service_id that corresponds to the holiday schedule. For a particular holiday, you would use the calendar_dates.txt file to add the holiday to the holiday service_id and to remove the holiday from the regular service_id schedule.

fare_attributes.txt (Fare)

`fare_attributes.txt` is implemented in `Fare`.

- fare_id (`Fare.fare_id`) (*required*): The fare_id field contains an ID that uniquely identifies a fare class. The fare_id is dataset unique.
- price (`Fare.price`) (*required*): The price field contains the fare price, in the unit specified by currency_type.
- currency_type (`Fare.currency_type`) (*required*): The currency_type field defines the currency used to pay the fare. Please use the ISO 4217 alphabetical currency codes which can be found at the following URL:

  http://www.iso.org/iso/en/prods-services/popstds/currencycodeslist.html.
- payment_method (`Fare.payment_method`) (*required*): The payment_method field indicates when the fare must be paid. Valid values for this field are:
  - 0 - Fare is paid on board.
  - 1 - Fare must be paid before boarding.
- transfers (`Fare.transfers`) (*required*): The transfers field specifies the number of transfers permitted on this fare. Valid values for this field are:
  - 0 - No transfers permitted on this fare.
  - 1 - Passenger may transfer once.
  - 2 - Passenger may transfer twice.
  - (empty) - If this field is empty, unlimited transfers are permitted.
- transfer_duration (`Fare.transfer_duration`) (*optional*): The transfer_duration field specifies the length of time in seconds before a transfer expires.

When used with a transfers value of 0, the transfer_duration field indicates how long a ticket is valid for a fare where no transfers are allowed. Unless you intend to use this field to indicate ticket validity, transfer_duration should be omitted or empty when transfers is set to 0.

fare_rules.txt (FareRule)

`fare_rules.txt` is implemented in `FareRule`

The fare_rules table allows you to specify how fares in `fare_attributes.txt` apply to an itinerary. Most fare structures use some combination of the following rules:

- Fare depends on origin or destination stations.
- Fare depends on which zones the itinerary passes through.
- Fare depends on which route the itinerary uses.

For examples that demonstrate how to specify a fare structure with fare_rules.txt and fare_attributes.txt, see FareExamples in the GoogleTransitDataFeed open source project wiki.

- fare_id (`FareRule.fare_id`) (*required*): The fare_id field contains an ID that uniquely identifies a fare class. This value is referenced from the fare_attributes.txt file.
- route_id (`FareRule.route.route_id`) (*optional*): The route_id field associates the fare ID with a route. Route IDs are referenced from the routes.txt file. If you have several routes with the same fare attributes, create a row in fare_rules.txt for each route.

  For example, if fare class "b" is valid on route "TSW" and "TSE", the fare_rules.txt file would contain these rows for the fare class:

```
b,TSW
b,TSE
```

- origin_id (`FareRule.origin.zone_id`) (*optional*): The origin_id field associates the fare ID with an origin zone ID. Zone IDs are referenced from the stops.txt file. If you have several origin IDs with the same fare attributes, create a row in fare_rules.txt for each origin ID.

  For example, if fare class "b" is valid for all travel originating from either zone "2" or zone "8", the fare_rules.txt file would contain these rows for the fare class:

```
b, , 2
b, , 8
```

- destination_id (`FareRule.destination.zone_id`) (*optional*): The destination_id field associates the fare ID with a destination zone ID. Zone IDs are referenced from the stops.txt file. If you have several destination IDs with the same fare attributes, create a row in fare_rules.txt for each destination ID.

  For example, you could use the origin_id and destination_id fields together to specify that fare class "b" is valid for travel between zones 3 and 4, and for travel between zones 3 and 5, the fare_rules.txt file would contain these rows for the fare class:

```
b, , 3,4
b, , 3,5
```

- contains_id (`FareRule.contains.zone_id`) (*optional*): The contains_id field associates the fare ID with a zone ID, referenced from the stops.txt file. The fare ID is then associated with itineraries that pass through every contains_id zone.

  For example, if fare class "c" is associated with all travel on the GRT route that passes through zones 5, 6, and 7 the fare_rules.txt would contain these rows:

```
c,GRT,,,5
c,GRT,,,6
c,GRT,,,7
```

Because all contains_id zones must be matched for the fare to apply, an itinerary that passes through zones 5 and 6 but not zone 7 would not have fare class "c". For more detail, see FareExamples in the GoogleTransitDataFeed project wiki.

shapes.txt (ShapePoint)

`shapes.txt` is implemented in `ShapePoint`. It is optional for a valid feed, but without it, routes will be drawn as direct lines between stops (going though buildings, etc.) instead of following the roads.

- shape_id (`ShapePoint.Shape.shape_id`) (*required*): The shape_id field contains an ID that uniquely identifies a shape.
- shape_pt_lat (`ShapePoint.point`) (*required*): The shape_pt_lat field associates a shape point's latitude with a shape ID. The field value must be a valid WGS 84 latitude. Each row in shapes.txt represents a shape point in your shape definition.

  For example, if the shape "A_shp" has three points in its definition, the shapes.txt file might contain these rows to define the shape:

```
A_shp,37.61956,-122.48161,0
A_shp,37.64430,-122.41070,6
A_shp,37.65863,-122.30839,11
```

- shape_pt_lon (`ShapePoint.point`) (*required*): The shape_pt_lon field associates a shape point's longitude with a shape ID. The field value must be a valid WGS 84 longitude value from -180 to 180. Each row in shapes.txt represents a shape point in your shape definition.

  For example, if the shape "A_shp" has three points in its definition, the shapes.txt file might contain these rows to define the shape:

```
A_shp,37.61956,-122.48161,0
A_shp,37.64430,-122.41070,6
A_shp,37.65863,-122.30839,11
```

- shape_pt_sequence (`ShapePoint.sequence`) (*required*): The shape_pt_sequence field associates the latitude and longitude of a shape point with its sequence order along the shape. The values for shape_pt_sequence must be non-negative integers, and they must increase along the trip.

  For example, if the shape "A_shp" has three points in its definition, the shapes.txt file might contain these rows to define the shape:

```
A_shp,37.61956,-122.48161,0
A_shp,37.64430,-122.41070,6
A_shp,37.65863,-122.30839,11
```

- shape_dist_traveled (`ShapePoint.traveled`) (*optional*): When used in the shapes.txt file, the shape_dist_traveled field positions a shape point as a distance traveled along a shape from the first shape point. The shape_dist_traveled field represents a real distance traveled along the route in units such as feet or kilometers. This information allows the trip planner to determine how much of the shape to draw when showing part of a trip on the map. The values used for shape_dist_traveled must increase along with shape_pt_sequence: they cannot be used to show reverse travel along a route.

The units used for shape_dist_traveled in the shapes.txt file must match the units that are used for this field in the stop_times.txt file.

For example, if a bus travels along the three points defined above for A_shp, the additional shape_dist_traveled values (shown here in kilometers) would look like this:

```
A_shp,37.61956,-122.48161,0,0
A_shp,37.64430,-122.41070,6,6.8310
A_shp,37.65863,-122.30839,11,15.8765
```

frequencies.txt (Frequency)

`frequencies.txt` is implemented in `Frequency`

This table is intended to represent schedules that don't have a fixed list of stop times. When trips are defined in frequencies.txt, the trip planner ignores the absolute values of the arrival_time and departure_time fields for those trips in stop_times.txt. Instead, the stop_times table defines the sequence of stops and the time difference between each stop.

- trip_id (`Frequency.trip.trip_id`) (*required*): The trip_id contains an ID that identifies a trip on which the specified frequency of service applies. Trip IDs are referenced from the trips.txt file.
- start_time (`Frequency.start_time`) (*required*): The start_time field specifies the time at which service begins with the specified frequency. The time is measured from "noon minus 12h" (effectively midnight, except for days on which daylight savings time changes occur) at the beginning of the service date. For times occurring after midnight, enter the time as a value greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins. E.g. 25:35:00.
- end_time (`Frequency.end_time`) (*required*): The end_time field indicates the time at which service changes to a different frequency (or ceases) at the first stop in the trip. The time is measured from "noon minus 12h" (effectively midnight, except for days on which daylight savings time changes occur) at the beginning of the service date. For times occurring after midnight, enter the time as a value greater than 24:00:00 in HH:MM:SS local time for the day on which the trip schedule begins. E.g. 25:35:00.
- headway_secs (`Frequency.headway_secs`) (*required*): The headway_secs field indicates the time between departures from the same stop (headway) for this trip type, during the time interval specified by start_time and end_time. The headway value must be entered in seconds.

  Periods in which headways are defined (the rows in frequencies.txt) shouldn't overlap for the same trip, since it's hard to determine what should be inferred from two overlapping headways. However, a headway period may begin at the exact same time that another one ends, for instance:

```
A, 05:00:00, 07:00:00, 600
B, 07:00:00, 12:00:00, 1200
```

- exact_times (`Frequency.exact_times`) (*optional*): The exact_times field determines if frequency-based trips should be exactly scheduled based on the specified headway information. Valid values for this field are:
  - 0 or (empty) - Frequency-based trips are not exactly scheduled. This is the default behavior.
  - 1 - Frequency-based trips are exactly scheduled. For a frequencies.txt row, trips are scheduled starting with trip_start_time = start_time + x * headway_secs for all x in (0, 1, 2, ...) where trip_start_time < end_time.

  The value of exact_times must be the same for all frequencies.txt rows with the same trip_id. If exact_times is 1 and a frequencies.txt row has a start_time equal to end_time, no trip must be scheduled. When exact_times is 1, care must be taken to choose an end_time value that is greater than the last desired trip start time but less than the last desired trip start time + headway_secs.

transfers.txt (Transfer)

`transfer.txt` is implemented in `Transfer`.

Trip planners normally calculate transfer points based on the relative proximity of stops in each route. For potentially ambiguous stop pairs, or transfers where you want to specify a particular choice, use transfers.txt to define additional rules for making connections between routes.

- from_stop_id (`Transfer.from_stop.stop_id`) (*required*): The from_stop_id field contains a stop ID that identifies a stop or station where a connection between routes begins. Stop IDs are referenced from the stops.txt file. If the stop ID refers to a station that contains multiple stops, this transfer rule applies to all stops in that station.
- to_stop_id (`Transfer.to_stop.stop_id`) (*required*): The to_stop_id field contains a stop ID that identifies a stop or station where a connection between routes ends. Stop IDs are referenced from the stops.txt file. If the stop ID refers to a station that contains multiple stops, this transfer rule applies to all stops in that station.
- transfer_type (`Transfer.transfer_type`) (*required*): The transfer_type field specifies the type of connection for the specified (from_stop_id, to_stop_id) pair. Valid values for this field are:
  - 0 or (empty) - This is a recommended transfer point between two routes.
  - **1 - This is a timed transfer point between two routes. The departing** vehicle is expected to wait for the arriving one, with sufficient time for a passenger to transfer between routes.
  - **2 - This transfer requires a minimum amount of time between arrival and** departure to ensure a connection. The time required to transfer is specified by min_transfer_time.
  - 3 - Transfers are not possible between routes at this location.
- min_transfer_time (`Transfer.min_transfer_time`) (*optional*): When a connection between routes requires an amount of time between arrival and departure (transfer_type=2), the min_transfer_time field defines the amount of time that must be available in an itinerary to permit a transfer between routes at these stops. The min_transfer_time must be sufficient to permit a typical rider to move between the two stops, including buffer time to allow for schedule variance on each route.

  The min_transfer_time value must be entered in seconds, and must be a non-negative integer.

feed_info.txt (FeedInfo)

`feed_info.txt` is implemented in `FeedInfo`.

The file contains information about the feed itself, rather than the services that the feed describes. GTFS currently has an agency.txt file to provide information about the agencies that operate the services described by the feed. However, the publisher of the feed is sometimes a different entity than any of the agencies (in the case of regional aggregators). In addition, there are some fields that are really feed-wide settings, rather than agency-wide.

- feed_publisher_name (`FeedInfo.publisher_name`) (*required*): The feed_publisher_name field contains the full name of the organization that publishes the feed. (This may be the same as one of the agency_name values in agency.txt.) GTFS-consuming applications can display this name when giving attribution for a particular feed's data.
- feed_publisher_url (`FeedInfo.publisher_url`) (*required*): The feed_publisher_url field contains the URL of the feed publishing organization's website. (This may be the same as one of the agency_url values in agency.txt.) The value must be a fully qualified URL that includes `http://` or `https://`, and any special characters in the URL must be correctly escaped. See: http://www.w3.org/Addressing/URL/4_URI_Recommentations.html for a description of how to create fully qualified URL values.

- feed_lang (`FeedInfo.lang`) (*required*): The feed_lang field contains a IETF BCP 47 language code specifying the default language used for the text in this feed. This setting helps GTFS consumers choose capitalization rules and other language-specific settings for the feed. For an introduction to IETF BCP 47, please refer to: http://www.rfc-editor.org/rfc/bcp/bcp47.txt http://www.w3.org/International/articles/language-tags/ *DEV NOTE - some historical feeds omit this parameter.*
- feed_start_date (`FeedInfo.start_date`) and feed_end_date (`FeedInfo.end_date`) (*optional*): The feed provides complete and reliable schedule information for service in the period from the beginning of the feed_start_date day to the end of the feed_end_date day. Both days are given as dates in YYYYDDMM format as for calendar.txt, or left empty if unavailable. The feed_end_date date must not precede the feed_start_date date if both are given. Feed providers are encouraged to give schedule data outside this period to advise of likely future service, but feed consumers should treat it mindful of its non-authoritative status. If feed_start_date or feed_end_date extend beyond the active calendar dates defined in calendar.txt and calendar_dates.txt, the feed is making an explicit assertion that there is no service for dates within the feed_start_date or feed_end_date range but not included in the active calendar dates.
- feed_version (`FeedInfo.version`) (*optional*): The feed publisher can specify a string here that indicates the current version of their GTFS feed. GTFS-consuming applications can display this value to help feed publishers determine whether the latest version of their feed has been incorporated.

How To Contribute

We'd love your help in building `multigtfs`. Here's some tips:

- Fork the project on GitHub, clone it locally, and create a feature branch for your work.
- When working with your Django project, use `pip install -e /path/to/multigtfs` to use your modified version.
- Use a seperate virtualenv for development (virtualenvwrapper is helpful as well). Install the recommended requirements (`pip install -r requirements.txt; pip install -r requirements.dev.txt`).
- Test changes with `./run_tests.py`
- Test PEP 8 and code coverage with `./qa_check.sh`
- Add yourself to `AUTHORS.rst`
- When you are happy with the change, publish your branch on GitHub and request a merge to the master branch.

Authors

- John Whitlock (jwhitlock, John-Whitlock@ieee.org)
- Juha Yrjölä (juyrjola, juha.yrjola@iki.fi)
- Kevin Diale (powersurge360, powersurge360@gmail.com)
- Adam Lawrence (alaw005, alaw005@gmail.com)
- Dave Kroondyk (davekaro, davekaro@gmail.com)
- Joshua Goodwin (jclgoodwin, j@joshuagoodw.in)

Project History

multigtfs was first developed for the Tulsa Web Devs' project to get Tulsa's buses into Google Maps. tulsa-transit-google is the Tulsa-specific portion, and multigtfs contains the parts useful for any GTFS effort. Tulsa's bus schedule appeared on Google Maps in July 2013, after a two-year effort. The Tulsa Web Devs founded Code for Tulsa to collaborate on future civic tech projects.

Several features, including GeoDjango support and much faster feed imports, were generously sponsored by MRCagney.

Future

`multigtfs` is production ready. It is likely that future releases will be driven by maintenance needs rather than the release of interesting new features.

Maintenance includes:

- Keeping up with Django releases
- Keeping up with GTFS spec updates

Ideas for future features include:

- Validating the feed against Google's requirements
- Creating a difference report against two feeds

See the issues list for more details.

Changelog

Next (unreleased)

- Handle latitude and longitudes with initial + sign. (issue #70).

1.1.1 (2017-08-02)

- Strip whitespace after commas in CSV files with skipinitialspace (issue #64, PR #65 and #68).
- Discard empty lines in CSV files (issue #66, PR #67)

1.1.0 (2017-07-09)

- Add support for Django 1.10 and 1.11
- Drop support for Django 1.7 and earlier, and for South migrations. If you are using these, upgrade to 1.0.0 first, migrate your codebase to Django 1.8 and Django migrations, then update to 1.1.0.
- Move Python 2 / Python 3 and other compatibilty code to `multigtfs/compat.py`. Exclude this file from the `make qa` coverage report, unless the `COVERAGE_COMPAT` environment variable is set. Because the cross-environment code is now in this file, many lines will be uncovered in a particular environment, while other files should be 100% covered. This file is tested in the supported environments in TravisCI, and a combined coverage report is generated in Coveralls, where `compat.py` should be 100% covered.

- Add a dockerized environment for the explore example app, and run it under Django 1.11.
- Whitespace-only values in import files are treated as empty values (PR #56)

1.0.0 (2016-03-29)

- The project has been production-ready for a while. Updating the version number and the PyPI classifiers to reflect that.
- Add support for Django 1.7 through 1.9, and a compatibility layer to handle future versions.
- Add support for transitioning from South to Django migrations.

0.4.3 (2015-02-24)

- Added documentation (issue #26)
- exportgtfs uses compression if available. Reduces one exported feed from 141MB to 21MB. (issue #27)
- Feeds that omit calendar.txt can be imported and exported. GTFS allows this if all dates are specified in calendar_dates.txt instead. This alternate format is used by the TriMet archive feeds from Portland, OR (issue #28).
- Django 1.7 is *not* supported by multigtfs. Version is limited in setup.py to 1.5 and 1.6.

0.4.2 (2014-07-20)

- importgtfs handles feeds with whitespace strings (issue #36)
- Can update objects with JSON fields in admin (issue #37)
- importgtfs can import an extracted GTFS feed (issue #30)
- importgtfs defaults to a Feed name based on the agency name and start of service (issue #33)

0.4.1 (2014-07-11)

- Import GTFS feeds using BOM (issue #31)
- Export non-ASCII GTFS feeds in Python 2 (issue #34)
- Various admin improvements (issue #29, issue #32)

0.4.0 (2014-06-21)

This release was generously sponsored by MRCagney.

- Import and export are 17-21x faster. Very large feeds (~20MB) can now be imported and exported without running out of memory (4 GB of RAM recommended). When running management commands, increasing verbosity ('-v 1' or -v 2') will print useful status messages.
- Additional columns not in the current GTFS spec are now imported into 'extra_data', a new JSON field. The columns are noted in the Feed's new JSON field, 'meta'. These addition items appear in the example project, and are exported after standard columns in the exported feed.
- Added Python 3 compatibility
- Extend more fields for real-world data (Trip.short_name, Zone.zone_id, and Block.block_id)

- On import, if two rows have duplicate unique ID (trip_id, stop_id, etc.), then only the first will be imported. A warning will printed to stderr. Previously, both may have been imported, with unknown consequences.
- Dropped support for South 0.7.x (not Python 3 compatible)
- Trips now have a single Service. Extra services will be detected by migration 0018, and will have to be manually removed.

0.3.3 (2014-03-28)

- Add new optional fields (issue #23):
  - trip.wheelchair_accessible
  - trip.bikes_allowed
  - stop.wheelchair_boarding
- Route.geometry does not include duplicate Trip.geometry lines (issue #24)
- Fix order of points in Shape.geometry (issue #25)
- Add management command 'refreshgeometries' to refresh cached geometries (useful if you were impacted by issues #24 or #25)

0.3.2 (2014-03-16)

This release was generously sponsored by MRCagney.

- Fix migration 0007 for PostGIS (issue #22)

0.3.1 (2014-03-12)

This release was generously sponsored by MRCagney.

- Add example project 'explore', which represents a feed as linked pages with OpenLayer maps.
- Add cached geometry for Routes, Trips, and Shapes.
- Extend fields for real-world data (FeedInfo.version, Route.short_name).
- Drop support for Points as geography fields.

0.3.0 (2014-02-01)

This release was generously sponsored by MRCagney.

- Convert to GeoDjango: Stops and ShapePoints use Points rather than lat/long, admin shows map of points, and new configuration settings to customize.
- Import south in try/except blocks (so that South really is optional).
- Django 1.5 or above is now required.

0.2.6 (2013-06-07)

- Remove verify_exists from URLField, so it can be used in Django 1.5

0.2.5 (2013-02-13)

- Human-friendly sorting for rest of GTFS output

0.2.4 (2013-02-06)

- Added optional manual sorting of output, used on stop_times.txt

0.2.3 (2012-11-09)

- Added South migrations for applying 0.2.2 changes

0.2.2 (2012-11-09)

- Fixed Fare.transfers for unlimited rides (use None instead of -1)
- First PyPi version