Prague GTFS

Release 0.1.0

Nils Wüstefeld, Adam Pasálek

Jun 11, 2025

CONTENTS

Welcome to the Prague GTFS documentation!

CHAPTER

ONE

API REFERENCE

client.py: Defines the Client class to refresh local GTFS databases.

The Client class loads API credentials from Streamlit session state, and orchestrates fetching static routes, stops, and trips data into a local SQLite database.

`class client.Client`

Orchestrates GTFS data refresh by calling individual managers.

Attributes:
api_key (str): API key loaded from Streamlit session state. api_url (str): Base URL for GTFS API. headers (dict): HTTP headers for API requests. db_path (str): Local path to the SQLite database file. routemanager (RouteManager): Manager for GTFS routes. stopmanager (StopManager): Manager for GTFS stops. tripmanager (TripManager): Manager for GTFS trips.

`run()`

Run all data managers to refresh local GTFS tables.

Calls set_routes, set_stops, and set_trips in order to update the SQLite database with the latest static GTFS data.

`class managers.route_manager.RouteManager(`*api_url*, *db_path*, *headers*`)`

Manages GTFS route data: fetches from the API and loads into a local SQLite database.

Attributes:
api_url (str): Base URL of the GTFS API. db_path (str): File path to the SQLite database. headers (dict): HTTP headers containing the API authentication token.

`create_route_table()`

Create the 'routes' table in the SQLite database if it does not already exist.

The table includes fields for route ID, names, type flags, colors, and last modification timestamp.

`get_routes()`

Fetch all routes from the GTFS API.

Returns:
list: A list of route dictionaries on success.

Raises:
Exception: If the HTTP response status is not 200.

`set_routes()`

Fetch route data and populate the local database.

Skips any routes with 'route_type' == 2 (trains) and inserts or replaces remaining routes into the 'routes' table.

class `managers.trip_manager.`**`TripManager`**

Manage GTFS trip data: initialize API settings and load trip information into a local SQLite database.

Attributes:
api_key (str): The API key for GTFS requests. api_url (str): Base URL for the GTFS API. headers (dict): HTTP headers including the API key. db_path (str): Path to the local SQLite database file.

`create_trip_table``()`

Create the 'trips' table in the local SQLite database if it does not exist.

The table stores information about trip IDs, route IDs, service IDs, headsigns, directions, block IDs, shape IDs, accessibility flags, bike flags, exceptions, and last modification.

`get_infos_by_trip_id``(`*trip_ids*`)`

Retrieve additional route information for a list of trip IDs.

Args:
trip_ids (list[str]): List of trip IDs to look up.

Returns:
pandas.DataFrame: DataFrame containing trip_id, shape_id, route_short_name, route_long_name, and route_color.

`get_trips``()`

Fetch all trip records from the GTFS API.

Returns:
list: A list of trip dictionaries from the API.

Raises:
Exception: If the HTTP response status is not 200.

`set_trips``()`

Populate the local database with trip information from the GTFS API.

Creates the trips table if needed, fetches trip data, and inserts or replaces each trip record in the SQLite database.

class `managers.request_manager.`**`RequestManager`**

Manages SSH connection and SQL queries to the remote vehicle_positions database.

This class loads environment variables, opens an SSH tunnel to the remote SQLite database, and provides methods to execute SQL queries and return results as pandas DataFrames.

`connect``()`

Establish an SSH connection to the remote database.

Reads SSH_USER, SERVER_ADRESS, and PEM key path from environment/session state, then connects using Paramiko SSHClient.

Raises:
paramiko.SSHException: If SSH authentication or connection fails.

`load_env``()`

Load environment variables from a .env file.

`server_request``(`*sql_query*, *columns=None*`)`

Execute a SQL query on the remote SQLite database via SSH.

Args:
sql_query (str): The SQL query to run on the remote database. columns (list[str], optional): Column names for the returned DataFrame.

Returns:
pandas.DataFrame or None: A DataFrame with query results (empty if no data), or None if an error occurred.

shape.py: Extract tariff zone polygons from a PID shapefile and save as WKT.

Reads 'DOP_PID_TARIFPASMA_P.shp' from a zip archive, filters for zone 'P', creates a unified polygon, and writes the result to 'tariff_zones.wkt'.

`shape.main()`

Extract tariff zone polygons from a PID shapefile and save as WKT.

PYTHON MODULE INDEX

INDEX