



GTFSync User Guide

**A comprehensive guide for detecting, updating, and
merging GTFS feeds**

Prepared by:

Harshal Bawale and Dr. J. David Porter

**School of Mechanical, Industrial, and Manufacturing Engineering
Oregon State University**

TABLE OF CONTENTS

1.0 INTRODUCTION	4
1.1 GTFSYNC CORE FEATURES	4
2.0 SYSTEM REQUIREMENTS AND SETUP	5
2.1 REQUIRED PYTHON LIBRARIES	5
2.2 INSTALLATION INSTRUCTIONS	5
3.0 TRANSIT DETECT MODULE	5
3.1 RUNNING THE TRANSIT DETECT MODULE	6
4.0 FLOW UPDATE MODULE.....	7
4.1 RUNNING THE FLOW UPDATE MODULE	7
5.0 SEAMLESS MERGE MODULE.....	9
5.1 RUNNING THE SEAMLESS MERGE MODULE	9
6.0 CONCLUSION	10

TABLE OF FIGURES

Figure 1. Main Modules of GTFSync.	4
Figure 2. Input for Code 1.	6
Figure 3. Summary Granularity Level.	6
Figure 4. Specific File Changes.	6
Figure 5. Change Log Option.	7
Figure 6. File Name for Change Log.	7
Figure 7. GTFS Filename Requirement.	8
Figure 8. Input Folder Paths.	8
Figure 9. Output folder and Custom File Names.	8
Figure 10. Console Output for Trips Verification.	8
Figure 11. User Decision Step to Proceed.	9
Figure 12. Input Folders and Custom Suffixes.	9

1.0 INTRODUCTION

GTFSync is a Python-based tool designed to streamline the process of merging General Transit Feed Specification (GTFS) feeds. Public transit agencies often face challenges in maintaining consistency in GTFS feeds across different time periods. GTFSync addresses these challenges by providing a structured workflow for detecting changes, updating critical identifiers, and merging GTFS feeds.

GTFSync identifies discrepancies between GTFS feeds to ensure consistency and improve data accuracy. It enhances efficiency by automating tedious manual updates to reduce errors and save time. Merging GTFS feeds while preserving unique identifiers across multiple time periods allows the user to produce a valid GTFS feed that is also compliant with GTFS standards.

1.1 GTFSYNC CORE FEATURES

GTFSync consists of three main modules that must be run in sequence, as shown in Figure 1:

1. **Transit Detect Module.** This module compares two GTFS feeds, file by file, to identify added, removed, or modified records.
2. **Flow Update Module.** This module detects changes in route IDs, trip IDs, stop IDs, service IDs, and shape IDs, and updates their corresponding files to ensure that all files remain synchronized.
3. **Seamless Merge Module.** This module merges multiple GTFS feeds while preserving unique identifiers to avoid generating duplicate entries. Therefore, this module helps to preserve data integrity.

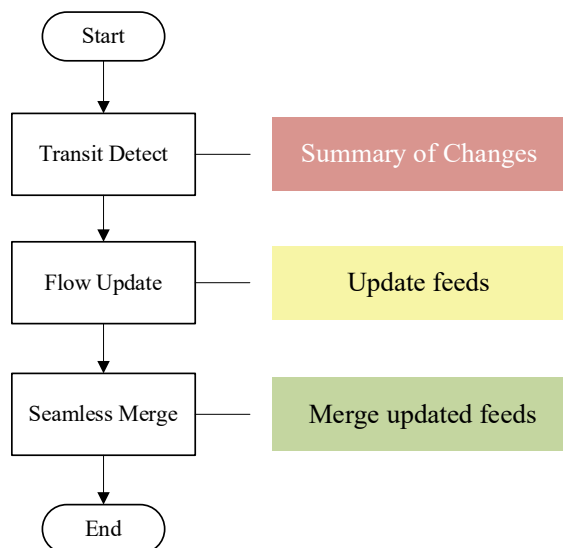


Figure 1. Main Modules of GTFSync.

2.0 SYSTEM REQUIREMENTS AND SETUP

GTFSync is a Python-based tool that does not require a separate software installation. However, users must have Python (version 3.7 or higher) installed on their computer, along with a few essential libraries.

2.1 REQUIRED PYTHON LIBRARIES

The **DiffLib** and **Collections** libraries are a part of Python's standard library and do not require a separate installation. The **Pandas** library is required for data manipulation and must be installed separately.

2.2 INSTALLATION INSTRUCTIONS

For MacOS:

1. Open the Terminal.
2. To check if Python is installed, type the command `"python --version"`. This will also show the current version of Python installed.
3. If Python is not installed, download it from [here](#) and follow the installation instructions.
4. After Python is successfully installed, open the Terminal and type the command `"pip install pandas"`. If installed successfully, a message will be displayed in the Terminal.

For Windows:

1. Open the Command Prompt.
2. To check if Python is installed type the command `"python --version"`. This will also show the current version of Python installed.
3. If Python is not installed, download it from [here](#) and follow the installation instructions. Select the checkbox labeled "Add Python to PATH" during installation".
4. After Python is successfully installed, open a Command Prompt window and type the command `"pip install pandas"`. If installed successfully, a message will be shown in the Command Prompt.

GTFSync does not require any special integrated development environment (IDE) and can be run using any Python-compatible IDE, including Visual Studio Code (VS Code), Spyder, and PyCharm.

3.0 TRANSIT DETECT MODULE

The Transit Detect module compares two sets of GTFS directories, file by file, and generates a structured report highlighting all the modifications detected. The process categorizes changes into three main types: "Added", "Removed", and "Modified Entries". The Transit Detect module gives the user the following three choices regarding the detail level of the structured report:

- **High-Level Summary.** Provides a high-level overview indicating which files were added (present only in the newer feed), removed (present only in the older feed), modified (present in both feeds with content differences), or unchanged.

- **Mid-Level Summary.** Reports how many lines were added or removed in each modified file, how many modified sections (diff blocks) were detected, and a count of attribute-level changes without listing exact values.
- **Detailed Summary.** Provides the exact number of added and removed lines, the number of modified sections, and a breakdown of attribute-level changes including specific fields, row indices, and original versus updated values.

3.1 RUNNING THE TRANSIT DETECT MODULE

Execute the following steps to run the Transit Detect module:

1. Specify the full folder path to the GTFS feed you wish to process, as shown in Figure 2. Make sure the folder path you enter points to the directory containing the files, and **NOT** to a zipped folder. Figure 2

```
14 input_folders = [
15     ' ',
16     ' ',
17 ]
```

Figure 2. Input for Code 1.

2. Run the Transit Detect module by pressing the “Run” button provided in the Python’s IDE.
3. The console will prompt you to enter the desired level for the structured report. Enter ‘1’, ‘2’, or ‘3’ to choose High-level, Mid-level, or Detailed summary, as shown in Figure 3.

```
Select granularity level:
1. High-Level Summary
2. Mid-Level Summary
3. Detailed View
Enter your choice (1/2/3):
```

Figure 3. Summary of Granularity Level.

4. After selecting the desired level for the structured report, the console will display the question shown in Figure 4. This is an optional step to choose if you want to view changes for a specific file. The Transit Detect module will continue to the next step regardless of the answer provided (i.e., “yes” or “no”).

```
Do you want to view detailed changes for a
specific file? (yes/no): |
```

Figure 4. Specific File Changes.

5. The Transit Detect module will prompt you to choose whether you want to save the structured report type selected in Step 3 to a change log file in *.txt* format (see Figure 5). If you wish to save a change log, you must enter a file name in the code first (see Figure 6). The change log file will be saved in the same folder from which the module is run. The

summary chosen in Step 3 will be saved to the change log file. If you do not wish to save a change log, input 'no' when prompted and the module will terminate.

```
Do you want to save the change log to a file
(yes/no):
```

Figure 5. Change Log Option.

```
209 output_file = input("Enter the output file name (e.g., changelog.txt): ").strip()
210 save_change_log(changes_summary, output_file)
```




Figure 6. File Name for Change Log.

Once Steps 1 through 5 are done, the Flow Update module can be run next.

4.0 FLOW UPDATE MODULE

The Flow Update module:

- Systematically scans all GTFS files, detects inconsistencies, and updates the necessary identifiers.
- Maps old service IDs to new ones, ensuring a smooth transition between GTFS feed versions.
- Reconciles changes in route IDs, trips IDs, stop IDs, and shape IDs ensuring that records stay aligned.
- Compares the file *trips.txt* exclusively to scan for changes in the actual trips. This is achieved by comparing its corresponding *stop_times.txt* file to look for changes in arrival times, departure times, and stop sequences. By aligning trips in *trips.txt* with their stop-level details in *stop_times.txt*, the Flow Update module can detect whether a trip has been structurally altered, merely renamed, or remains unchanged.

To facilitate different levels of verification, the Flow Update module provides an interactive step to allow users to review the detected changes in trips, ensuring transparency and preventing unintended modifications.

4.1 RUNNING THE FLOW UPDATE MODULE

Execute the following steps to run the Flow Update module:

1. The Flow Update module requires that the GTFS files to be processed are named **exactly** as shown in Figure 7. If the GTFS files to be processed have different names, you can change them here. For example, if the *calendar_dates.txt* file in your feed is named as *calendar_attributes.txt*, make sure that this file name is updated in the code in Figure 7. Otherwise, the Flow Update module will not run.

```

24 def load_data(folder_name):
25     stop_times_path = os.path.join(folder_name, 'stop_times.txt')
26     trips_path = os.path.join(folder_name, 'trips.txt')
27     stops_path = os.path.join(folder_name, 'stops.txt')
28     calD_path = os.path.join(folder_name, 'calendar_dates.txt')
29     cal_path = os.path.join(folder_name, 'calendar.txt')
30     agency_path = os.path.join(folder_name, 'agency.txt')
31     routes_path = os.path.join(folder_name, 'routes.txt')
32     shapes_path = os.path.join(folder_name, 'shapes.txt')

```

Figure 7. GTFS Filename Requirement.

- Specify the full folder path to the GTFS feed you wish to process, as shown in Figure 8.

```

11 import os
12 import pandas as pd
13
14 # Define Input Folders
15 input_folders = [
16     ' ',
17     ' '

```

Enter your paths here

Figure 8. Input Folder Paths.

- Specify the path to the folder in which the results of running the Flow Update module will be saved. In addition, specify any custom file names (if required) in the part of the code shown in Figure 9.

```

193 output_folder = '/Users/harshalbawale/Desktop/Research Work/CTS GTFS Data/Testing/JCT Testing/jct-gtfs-ride/November/2-3 Output'
194 calendar_dates_list[1].to_csv(os.path.join(output_folder, 'calendar_dates.txt'), index=False)
195 calendars_list[1].to_csv(os.path.join(output_folder, 'calendar.txt'), index=False)
196 stops_list[1].to_csv(os.path.join(output_folder, 'stops.txt'), index=False)
197 trips_list[1].to_csv(os.path.join(output_folder, 'trips.txt'), index=False)
198 stop_times_list[1].to_csv(os.path.join(output_folder, 'stop_times.txt'), index=False)
199 routes_list[1].to_csv(os.path.join(output_folder, 'routes.txt'), index=False)
200 shapes_list[1].to_csv(os.path.join(output_folder, 'shapes.txt'), index=False)
201 agency_list[1].to_csv(os.path.join(output_folder, 'agency.txt'), index=False)

```

Figure 9. Output folder and Custom File Names.

- Run the Flow Update module by pressing the “Run” button provided in the Python’s IDE.
- The Flow Update module compares the files *trips.txt* and their corresponding stop times data to check whether the trips are identical, different, or only have updated trip IDs and displays an output in the console to indicate the same, as shown in Figure 10. The Flow Update module will then prompt you to choose whether to proceed with the rest of the updating process (see Figure 11).

```

Trip with trip_id '614' is identical in both feeds.
Trip with trip_id '615' is identical in both feeds.

Trips with trip_id changes:

Actually different trips:
Trip with trip_id '100' in Feed 1 is different.

```

Figure 10. Console Output for Trips Verification.


```
Verification complete. Do you want to proceed with the rest
of the process? (yes/no):
```

Figure 11. User Decision Step to Proceed.

6. If you select “yes” in Step 5, the Flow Update module will update all specified identifiers and save the files in *.txt* format. If you select “no”, the Flow Update module will terminate and display the message “**Process terminated by user**”.

5.0 SEAMLESS MERGE MODULE

The Seamless Merge module combines two GTFS feeds into one unified dataset while preserving the integrity of unique identifiers. This module builds upon the outputs of the Transit Detect and the Flow Update modules and ensures that no data is overwritten or lost during the merging process.

The Seamless Merge module appends the data from Feed 2 to Feed 1 for each GTFS file and automatically handles naming conflicts by adding suffixes or updating identifiers. This is necessary because GTFS datasets often contain overlapping *trip_id*, *route_id*, *stop_id* and other identifiers. Without suffixes, merging multiple feeds could result in duplicates, leading to incorrect mappings and data integrity issues. This allows the final merged dataset to be both complete and compliant with GTFS standards.

5.1 RUNNING THE SEAMLESS MERGE MODULE

Execute the following steps to run the Seamless Merge module:

1. Input your GTFS feed and the corresponding suffixes, as shown in Figure 12. Ensure that the number of input paths and the number of suffixes are the same to correctly tag each GTFS feed data with its suffix.

```
12 input_folders = [
13     ' ',
14     ' ',
15 ]
16
17 # Corresponding suffixes for each feed (Ensure same number as input_folders)
18 suffixes = ['Feed_1', 'Feed_2']
```

Figure 12. Input Folders and Custom Suffixes.

2. Define the output folder on **line 99** of the Seamless Merge module.
3. Run the Seamless Merge module by pressing the “Run” button provided in the Python’s IDE.
4. The merged GTFS feeds will be saved in the specified output folder.

6.0 CONCLUSION

GTFSync offers a streamlined and reliable solution for managing and merging GTFS datasets across different time periods. Whether applied for routine updates or long-term maintenance, GTFSync is designed to support a consistent and scalable GTFS data management process. Users are encouraged to customize the tool to suit their unique datasets and workflows and to refer to this guide as a reference for correct implementation.