Contents

Before Reading

If you see the sign 🚧, this chapter is a work in progress!

Text boxes that look like this stand for keyboard shortcuts. For example: Ctrl Alt F2

Text that **looks like this** are links. You can click them to open the link in your default web browser.

Text boxes that look like this stand for file paths or directory paths. For example: ~ .local .config Paiagram

You can also read this manual in this language:

- **Simplified Chinese**

1. Introduction

Paiagram is an application for visualizing and managing vehicle timetable and routes. Paiagram organizes stations and intervals between stations in a graph, and organize different services using vehicles. Similar products include:

- **OuDiaSecond**, developed by DiagramMania and take-okm
- **pyETRC** (deprecated) and **qETRC**, developed by x.e.p. (CDK6182CHR)
- **ETRC**, a timetabling software written in Java.

Paiagram is designed with usability, portability, versatility, and internationalization in mind. In fact, it is possible to manage and visualize an entire nation's railway network with Paiagram. For this purpose, Paigram's used terminologies that are neutral, i.e. not related to a specific method of transportation.

1.1. Who Can Use? Use For What?

Transport enthusiasts can use Paiagram to track down vehicles and check schedules.

Transport simulation game players can use Paiagram to timetable vehicles.

Transport Managers can use Paiagram to schedule services.

Anyone can use Paiagram to kill some time :-)

1.2. Origin of the Name

Paiagram is the mix of two words with different roots. The first being "□", which means to make or to organize, and its Pinyin is "pái". The second word is, of course, diagram. Combined they form the word *Paiagram*.

2. Getting Started

You can download Paiagram from **Github releases** or **try it online**. The online version's performance would be worse compared to native versions, but the difference is usually acceptable. You can also download the webpage and run it locally in your browser.

2.1. The Model

Paiagram uses a graph-based system to store all stations and intervals between stations. All stations are represented as nodes in the graph, and each interval is an edge between two nodes. Vehicles could traverse via edges, and visit nodes along the way.

For performance sake, internally, the edges in Paiagram are *not* directional, instead they are bi-directional. However, you can manually specify the edge's direction.

For performance sake, internally, there could be *only one* edge betwee two nodes. However, you can specify extra nodes and connect from those nodes. Those extra nodes *would* instead be of "waypoint" type.

The graph model allows easy manipulation of network, and allows for managing much larger networks. This also means that Paiagram won't have the *Nobori/Kudari* settings in OuDiaSecond, or *Shangxing/Xiaxing* properties in qETRC/pyETRC & friends.

In order to display a track section, you must specify a "Displayed Line". A displayed line holds a set of stations. The distance between stations depends on the interval's length. Yet, it could always be manually adjsuted.

Labels, annotations, and lines on the diagram would automatically avoid colliding into each other, for better display.

Paiagram doesn't support directly exporting to PDF or printing, but you can export each page to Typst, and further render that to SVG/PNG/PDF. The spacing on diagram pages would be reserved. Exporting *directly* to PDF, and exporting to LaTeX, docx, odt, etc. would never be supported.

2.2. Vehicle-based model

Paiagram uses a vehicle-based system, that is, services are not the basic unit of the network, rather they are properties attached to a vehicle's timetable entries.

The vehicle-based system cannot 100% emulate real-world systems. Besides the freight train network in Canada and U.S., which doesn't even have a schedule hence it is impossible to represent it in Paiagram, public transit networks such as bus networks might go with a dispatch

based system, i.e., which vehicle is running the service is not important, and it is fine as long as there is a vehicle running. Take the TransLink's bus lines 15 and 50 as an example. This Metro Vancouver transit company has a dispatch based bus system.

(TODO)

2.3. Graphs, Diagrams, Vehicles

The graph is the top-level representation of the network. Each `.paiagram` file can hold one graph and one graph only.

The diagram is 1. a collection of vehicle times and 2. a visual representation of the graph. Each `.paiagram` file can hold multiple diagrams. It is not recommended to have multiple diagrams for different types of services (e.g. local, express, freight). Instead, keep all services that runs at the same time in the same diagram, and use filtering to only show the services you want. The multi-diagram feature is mainly for representing the network's status in different operation periods (e.g. normal operation, holiday schedule, special event schedule).

Vehicles are entities that traverse the graph. Each vehicle belongs to a diagram.

2.3.1. Interdiagram Vehicle Links and Patches

Vehicles can be linked across diagrams. This is useful when you have multiple diagrams for different operation periods, and you want to represent a vehicle that runs across multiple operation periods. Sometimes a vehicle may have a very small change in its timetable for one diagram, while other parts of the timetable remain the same. In this case, you can use vehicle patches to only specify the changed parts, and link it to the original vehicle.

2.4. Command Line Arguments

On top of using the graphical user interface, you can also use the command line interface on the desktop version for quick opening,

2.5. Arrival and Departure

Paiagram features 3 arrival types and 4 departure types:

- Arrival:

At	The vehicle would arrive at the station AT the given time.
For	The vehicle would travel FOR the given amount of time between the current and the previous station.
Flexible	The arrival time is FLEXIBLE.

- Departure:

At	The vehicle would depart from the station AT the given time.
For	The vehicle would stay at the station FOR the given amount of time
Flexible	The departure time is FLEXIBLE
Non-stop	The vehicle DOES NOT STOP at this station

The "At" type is the most common type you'd see on real-world timetables. If you have a train arriving at Shinagawa at 09:35:21, and departing at 09:35:41, you can specify its arrival and departure types as "At: 09:35:21" and "At: 09:35:41", respectively. Alternatively, if the stopping time is more important, and you don't want to manually calculate the stopping time, you can specify the stopping time to be "For: 00:20". If you have a train departing from Stuttgart, and you're not sure when it would depart, you can set the departure type as "Flexible", to avoid confusion.

When only the departure station and terminal stations' times are important, you can set the stations in between's time to be "Flexible". Paiagram would handle them automatically.

2.6. The Diagram

The diagram is the most important part in the program. You can open the diagram via the Ctrl D shortcut.

2.7. Vehicle Events

Vehicles can have events bound to a timetable entry. There are currently these events available:

- Composition of vehicles
- Decomposition of vehicles
- Loading/unloading passenger (or freight)

3. Interface

The interface is designed to be intuitive and user-friendly. The workspace is split into two parts: the properties panel on the left and the main canvas on the right.

The properties panel contains information and setting for the currently selected object. You can also multi-select objects to edit their properties in bulk. Nevertheless, objects that don't share common properties cannot be edited in bulk.

The main canvas is where you can visualize and edit your timetable graph. You can dock and undock, add or remove tabs as needed. However, you cannot add or remove tabs from the properties panel.

3.1. Overview

The overview tab provides statistics and general information about the current file opened, including:

- Amount of vehicles.
- Amount of services.
- Amount of stations.
- Amount of intervals.

3.2. Searching

Each table would feature a search bar at the top. You can use it to filter the table entries based on your input.

The search bar currently supports matching Pinyin, Double Pinyin (Microsoft scheme) and Romaji for Chinese and Japanese text.

3.3. Diagnostics

The diagnostics tab provides information about potential issues in the current file opened. You can also access the tab from the **status bar**

3.4. Status Bar

The status bar is located at the bottom of the window. It provides quick access to various functions and information, including:

- Tooltips.
- Diagnostics.

3.5. Vehicle

The vehicle view tab is opened upon selecting a vehicle from the main canvas. You can edit the vehicle's timetable entries, services, stops, and other properties here.

3.6. All Vehicles

3.7. All Services

3.8. Global Search

The global search window allows you to search for various objects in the current file opened. You can use the Ctrl Shift F shortcut to open the global search window.

3.9. Station

3.10. Route

3.11. Route Diagram

3.12. Shortcuts

Shortcuts are what makes power users. Here are all shortcuts listed out:

Action	Shortcut
Close tab	Ctrl W
Close app	Alt F4
Open file	Ctrl O
Save file	Ctrl S
Save file as	Ctrl Shift S
Undo	Ctrl Z
Redo	Ctrl Y
Cut	Ctrl X
Copy	Ctrl C
Paste	Ctrl V
Select all	Ctrl A
Find	Ctrl F
Find next	F3
Find previous	Shift F3
Global search	Ctrl Shift F

Table 1: Keyboard Shortcuts

4. Importing Foreign Files 🚧

Paiagram supports importing diagram data from foreign formats including:

- qETRC
- OuDiaSecond

Data specific to one application, e.g. window placement settings in OuDiaSecond, will not be respected.

4.1. qETRC

Importing qETRC data is straightforward: import and it should just work. Train classes, services, trains (⊠⊠), stations, and station intervals are all handled correctly in Paiagram.

4.2. OuDiaSecond

OuDiaSecond's internal structure is very different, and it misses some basic features in Paiagram.

- Stations

  Importing stations is fine

- Intervals

  OuDiaSecond provides minimal information about intervals connecting stations, only the stations the interval connects. Due to this lack of information, intervals imported would always have a length of 1.00km.

- Services

  Most services should work fine.

4.3. OpenTTD JGRPP Orders Exports

Importing JGRPP orders exports is fairly limited, since it could contain *conditional* orders that cannot be known by Paiagram. The only condition Paiagram understands and can adapt from the export is the time.

5. Saving and Exporting

Paiagram supports saving and exporting to various formats

5.1. Saving as `.paiagram`

The `.paiagram` format is the default saving format for Paiagram.

5.2. Exporting to `.csv`

You can export single vehicles, single services, stations info, and interval info as `.csv` files. You cannot export the entire graph as a `.csv` file. For information on exporting graphs, see Section 5.3.

Subjects that can be exported:

- Vehicle schedule
  - With or without calculated times
- Service schedule
  - With or without calculated times
- Vehicle services
  - With or without calculated times
- List of stations
- List of intervals
- Tracks of a station

5.3. Exporting to `.dot`

Graphs can be exported to GraphViz `.dot` files.

5.4. Exporting to Typst code

You can export diagrams to Typst code, which can be then further processed and rendered via the Typst program.

5.5. Exporting to JGRPP Orders

5.5.1. Notes and Tags

Notes attached to a timetable entry would be translated to labels in JGRPP. You can also specify the colour by adding a `<colour>:(content)` prefix. For example:

- `red: Return to Paddington Station`
- `purple: Go to Shinkansen Centre`

all contain a valid colour tag.

5.5.2. Scheduled Dispatch

JGRPP features scheduled dispatch, a way to specify when a vehicle would depart from a station. Scheduled dispatch slots could optionally have a tag. If they do have a tag, their

6. Coming from qETRC/pyETRC 🚧

You may be overwhelmed with the number of arrival and departure types in Paiagram compared to qETRC/pyETRC.

7. Coming from OuDiaSecond ////////////

8. Building Paiagram

8.1. Compiling

Paiagram is built using Rust. If you're from NixOS, you can skip the next chapter!

8.2. Dependencies

Paiagram requires the following dependencies:

- Rust toolchain (1.80 or later)

8.3. Translating

Paiagram uses the **Fluent** standard for translatons. You can submit pull requests for translations on GitHub.

Please be in mind that this is a community project, so please try to avoid submitting unreviewed machine or LLM based translation.

9. External Resources

There aren't many external resources for Paiagram yet.

10. Credits

- Developer: Jeremy Gao
- Documentation: Jeremy Gao
- Special thanks to:
  - x.e.p. for their wonderful work on qETRC/pyETRC, which inspired Paiagram.
  - Random people I spoke to on the internet for their feedback and suggestions.
  - Jason.

Work in Progress Chapters