Processing Open Transport Data: Design and Implementation of an Extension for a Data Pipeline Modeling Language

MASTER THESIS

Johannes Noah Schilling

Submitted on 30 June 2023

Friedrich-Alexander-Universität Erlangen-Nürnberg
Faculty of Engineering, Department Computer Science
Professorship for Open Source Software

Supervisor:
Philip Heltweg, M.Sc.
Prof. Dr. Dirk Riehle, M.B.A.

Friedrich-Alexander-Universität
Faculty of Engineering

Declaration of Originality

I confirm that the submitted thesis is original work and was written by me without further assistance. Appropriate credit has been given where reference has been made to the work of others. The thesis was not examined before, nor has it been published. The submitted electronic version of the thesis matches the printed version.

_____

Erlangen, 30 June 2023

License

_____

Erlangen, 30 June 2023

To Barbara, Arnd, Franzi, and Paul who are always helping me to become the best version of myself

Acknowledgements

I deeply appreciate Philip for his invaluable time, insightful feedback, and unwavering support throughout this thesis journey as well as the entire JValue Team for their consistent encouragement and assistance.

Furthermore, I acknowledge the help of ChatGPT in improving the writing and nothing but the writing of this thesis.

And lastly, I am truly thankful to all the proofreaders for their significant contributions in improving this thesis.

Abstract

Open transport data enables innovation by offering a vast amount of information for developers, researchers, urban planners, and entrepreneurs to create new applications, services, and business models. However, the lack of specific guidelines for making data “open” has led to the existence of diverse proprietary and heterogeneous open data platforms. Open transport data formats, such as static and real-time General Transit Feed Specification (GTFS), provide a standardized way to share information about public transit systems, including schedules and vehicle positions. Since this data is difficult to access and often times volatile, archiving GTFS data has several advantages, including the possibility of passenger routing and traffic flow analysis. The JValue research project aims to democratize collaborative data engineering by providing, among other components, Jayvee, a domain-specific language for data pipeline modeling. This thesis focuses on extending Jayvee to support processing of GTFS static and real-time data. The development process involves defining functional requirements through a Request for Comments (RFC) process and implementing the extension incrementally by introducing new language features such like a data extractor for HTTP content, an interpreter for ZIP-files, or a filesystem component. Providing a demonstrator, an evaluation phase showcases proper system execution and the periodic archival mechanism. As a result, it is now possible for users of Jayvee, to access, process, and archive GTFS static and realtime data periodically. Future improvements include automating optional fields and tables handling, providing user-friendly pre-configured GTFS dataset layouts, and introducing a concept for composite pipelines. This engineering thesis serves as a guide for the open transport data research community, on how to extend open source software like Jayvee to reduce barriers accessing and processing open transport data.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

API Application Programming Interface

AST Abstract Syntax Tree

ETL Extract-Transform-Load

ITS Intelligent Transportation System

GTFS General Transit Feed Specification

GTFS-RT General Transit Feed Specification - Realtime

NAP National Access Point

NFR Non Functional Requirement

PR Pull Request

PSI Public Sector Information

RFC Request for Comments

UAC User Acceptance Criteria

1 Introduction

> *"Without data we can't build information, and without information there is no new knowledge."* (European Commission, 2022)

With this statement the European Commission's official platform — the *European Data Portal* — encapsulates the fundamental purpose of open data. The concept comprises data that is commonly available and can be used and republished without restrictions from copyrights or patents (Braunschweig et al., 2012). As of 2022, the portal hosts over 1.5 million public sector datasets, which represent just one of the many available access points for open data (European Commission, 2022).

The movement towards providing free access to public data is primarily driven by legislative efforts, such as the European Public Sector Information Directive of 2003 (European Commission, 2003) or the United States Open Government Initiative directive (Obama, 2009). These policies establish the right for all individuals to reuse public information. However, there may be a misunderstanding when comparing the terms "open data" and "public data". To clarify, we adopt the Open Knowledge Foundation's recommendation that "public data" may be restricted to non-commercial usage purposes, whereas "open data" has no such limitations on usage (Mahajan et al., 2022; Molloy, 2011).

Open data encompasses various domains, including geographic data, tourist information, statistical and business data, meteorological data, and more. Its original motivation was to promote government accountability and citizen participation in political progress (Janssen, 2011). One of the unintended advantages of this practice is its ability to foster a whole new ecosystem of innovation. Numerous studies have demonstrated that taking a proactive approach to releasing public and private data in open formats not only benefits companies, the research community, citizens, and other stakeholders but also creates fresh business opportunities, enhances innovative capacity, and catalyzes transformative potential (Conradie & Choenni, 2014; Zuiderwijk et al., 2014).

Given the increasing pace of urbanization and mobility demands, significant changes in transportation infrastructure are necessary (Dimitrakopoulos & De-

mestichas, 2010). Hereby, computer systems such as the Intelligent Transportation System (ITS) encompass various fields, including transportation management, traffic control and operations. They provide a reliable platform for addressing transportation-related issues and promoting cooperation within users of the transportation system. In addition to highway traffic, ITS should also provide access to public transport infrastructure (Qureshi & Abdullah, 2013). To harmonize European efforts for a standardized ITS, the European Commission released a set of directives in 2010, based on the Public Sector Information (PSI) directive from 2003, providing a framework for the deployment of such a system (European Commission, 2010). One outcome of the standardization directive is the obligation of each country to establish a dedicated National Access Point (NAP) that publishes static and dynamic mobility data applying the open data approach (European Commission, 2021).

Over the years a vast amount of data has been collected and published by public and governmental institutions, due to the open data directives. However, these directives do not specify the methodology for making data "open", resulting in a diverse set of open data platforms that are often proprietary, heterogeneous, and poorly documented. As a consequence, a majority of these platforms lack proper standards and Application Programming Interfaces (APIs), making it difficult for data scientists and data engineers to collaborate on open data projects (Braunschweig et al., 2012). To address these challenges, the Professorship of Open Source Software at the University of Erlangen-Nuremberg has launched the "JValue" project. This initiative aims at provide a platform solution for Data Scientists and Data Engineers to collaborate on open data projects in an easy, safe, and reliable manner (Professorship of Open Source Software at the University of Erlangen, 2022).

GTFS and General Transit Feed Specification - Realtime (GTFS-RT) are open data standards which enable sharing information about public transit systems, including schedules, routes, and stops (Google, 2022). These formats have gained widespread adoption among transit agencies worldwide and are publicly accessible for utilization by anyone. Consequently, numerous applications and tools have been developed, aiding riders in trip planning, route visualization, and real-time information access. GTFS-RT is a real-time extension of GTFS that provides dynamic information, such as vehicle locations (Koetsier et al., 2017). The establishment of open data standards in the transportation industry demonstrates how open data can improve transparency, planning, management, and accessibility to public transportation information for the benefit of riders and the general public (Antrim, Barbeau et al., 2013).

Since this data is often times difficult to access and voluntarily, archiving GTFS data has several benefits. First, by creating a national centralized transportation data hub, organizations and individuals can access and utilize the data for various

purposes, such as passenger routing and traffic flow analysis (Kujala et al., 2018). Second, timetable data for a city is often fragmented, making it difficult to access complete and accurate information. Archiving the data allows for consolidation of these fragmented feeds, ensuring accessibility even if certain endpoints are down. Third, archiving the data allows for time-series analyses for identifying trends and patterns in transportation usage over time (Kaeoruean et al., 2020). Fourth, when providing GTFS data for large areas, such as entire countries, spatial filtering based on city boundaries is necessary. Archiving the data in this case allows for easier spatial filtering. However, GTFS data may contain logical errors and thus validation is necessary to ensure accuracy (Harding & Davies, 2012). In conclusion, archiving GTFS and GTFS-RT data is crucial for improving transportation planning, management, and decision-making.

The objective of this engineering thesis is to extend the JValue Project's capabilities to process and archive both static and real-time GTFS data. In Chapter 2, the fundamental concepts of a GTFS and an introduction to the JValue project and its components, particularly the domain-specific language Javve, are presented. Chapter 3 outlines the process of requirements engineering and lists the extracted requirements. The software artifact's architecture is described in Chapter 4, and Chapter 5 presents details on the design and implementation. Introducing a demonstrator, Chapter 6 evaluates the developed artifact against the functional requirements from the previous chapter, highlighting the limitations of the implementation. The evaluation process exemplary utilizes open transport data from the NAP of France. As GTFS and its dynamic counterpart GTFS-RT are globally established static and dynamic transport feed specifications, the results obtained can be potentially applied to any GTFS data source. The work's conclusion is presented in Chapter 7.

2 Fundamentals

Section 2.1 provides an overview of the essential characteristics of data models for both static GTFS files and their dynamic real-time counterpart, GTFS-RT. Furthermore, Section 2.2 introduces the context of the JValue project, with a particular emphasis on Jayvee, where the relevant system components and design paradigms are emphasized.

2.1 GTFS: Static and Realtime Datasets

GTFS has gained widespread popularity over the past decade as an open-source industry standard. It is published under the Create Commons Attribution 3.0 License for describing and publishing fixed- and dynamic-route transit operations (Google, 2022b). It is a data standard that defines how public transit agencies should provide schedule information to developers. The specification includes information about stops, routes, and schedules for various forms of transportation like buses and trains (Wu et al., 2022). The open design and clear documentation of GTFS, along with input and feedback from the user community, have contributed to its evolution into a robust and widely-used data format (Koetsier et al., 2017).

The specification was originally introduced in 2005 as a collaboration between Google and TriMet to create a web-based transit trip planner application. In 2007, it was made public for general use (Goldstein & Dyson, 2013). In 2011, Google released an extension to the static GTFS schedule called GTFS-RT, which aimed to standardize real-time information feeds, such as real-time vehicle positions or service alerts. In 2017, a revised version of GTFS-RT, known as GTFS-RT v2.0, was released in order to address limitations of the previous version, particularly the lack of proper documentation and open source validation tools (Lim et al., 2019).

GTFS is a collection of CSV files that are packaged within a ZIP file. Each CSV file can be viewed as a table of data. The primary purpose of GTFS is to facilitate passenger routing for public transportation. However, it can also

be utilized for research purposes, such as modeling the accessibility provided by public transportation (Kujala et al., 2018). GTFS establishes its own terminology for data modeling, drawing from widely accepted data modeling languages (Google, 2022b):

- *Dataset* - A collection of files, wrapped in an archive file (ZIP) describing a transit system. The granularity is not further defined which can be for example on a city-level, on a community-level, or on a country-level.
- *Record* - Multiple field values that describe a single entity such as a transit agency, a stop, or a route (represented in a table as row).
- *Field* - Property of an entity (represented in a table as column).
- *Field Value* - Individual entry in a field (represented in a table as cell).

The specification differs between the status of files, records and fields:

- *required* - must be provided
- *optional* - may be ommitted
- *conditionally optional* - required under certain conditions

Required files for a GTFS dataset consist of six dimensions: agency.txt, stops.txt, routes.txt, trips.txt, stop_times.txt and one of two possible representations of the calendar. Table 2.1 provides a detailed description of the contents of a GTFS dataset.

Filename	Required	Definition
agency.txt	Required	Transit agencies with service.
stops.txt	Required	Defines stations and entrances.
routes.txt	Required	A group of trips displayed to riders as a single service.
trips.txt	Required	Trips for each route.
stop_times.txt	Required	Times that a vehicle arrives at and departs from stops for each trip.
calendar.txt	Conditionally required	Service dates using a weekly schedule (timetable).
calendar_dates.txt	Conditionally required	Exceptions for the services defined in the calendar.txt (date oriented).
fare_attributes.txt	Optional	Fare information for a transit agency's routes.
fare_rules.txt	Optional	Rules to apply fares for itineraries.
shapes.txt	Optional	Rules for mapping vehicle travel paths.
frequencies.txt	Optional	Time between trips for headway-based service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
pathways.txt	Optional	Pathways linking together locations within stations.
levels.txt	Optional	Levels within stations.
feed_info.txt	Conditionally required	Metadata, including publisher, version, and expiration information.
translations.txt	Optional	Translated information of a transit agency.
attributions.txt	Optional	Specifies the attributions, applied to the dataset.

Table 2.1: GTFS static specification adapted from developer reference (Google, 2022b)

As an example of conditionally required dimensions, we consider the representation of the calendar. An GTFS endpoint developer can choose to publish a

timetable in a weekly format using calendar.txt, exemplary depicted in Table 2.2. The column service_id uniquely identifies a set of dates when a service is available for one or more routes. The column for each day of a week indicates whether the service operates on that weekday in the date range specified by the columns start_date and end_date (Google, 2022b).

service_id	monday	...	sunday	start_date	end_date
1	0	...	0	20230510	20230614
2	1	...	0	20230509	20230616
3	1	...	0	20230509	20230616
4	0	...	0	20230512	20230616
5	1	...	1	20230519	20230619
...	...	...	...	...	...

Table 2.2: Exemplary content of calendar.txt representing a weekly timetable of services

Further, calendar_dates.txt can be used in conjunction with calendar.txt to define exceptions to the default service pattern. In Table 2.3 the exception type indicates whether service is available on the date specified (1 = Service has been added for the specified date, 2 = Service has been removed for the specified date). Alternatively, if calendar.txt is omitted, calendar_dates.txt can be used to define a service for each date, accommodating services without normal weekly schedules. Therefore, both files are classified as conditionally optional (Google, 2022b).

service_id	date	exception_type
2	20230518	2
2	20230519	2
3	20230518	2
3	20230519	2
4	20230519	2
...	...	...

Table 2.3: Exemplary content of calendar_dates.txt used in conjunction with calendar.txt

In addition to GTFS, Google introduced the GTFS-RT specification specifically for real-time updates. This extension enables developers to access real-time information regarding vehicle location, status, and any service disruptions or delays (Barbeau, 2018). Typically, GTFS-RT data is provided through streaming data feeds that are continuously updated in real-time as events occur. It is important to note that the real-time feed is always accompanied by its corresponding static feed, which defines the schedule and essential dimensions such as agency.txt or

routes.txt in relation to the live updates (Koetsier et al., 2016). GTFS-RT specification encompasses three types of additional information, which can be combined into a single feed to enhance the static GTFS data (Google, 2022a):

- *Trip updates* - cancellations, delays and changed routes
- *Service alerts* - unforeseen events with impact on the transportation network
- *Vehicle positions* - real-time information on vehicles position in coordinates

Unlike static GTFS data, which changes only when new schedules are released manually, real-time feeds necessitate frequent updates at a high rate (typically in the range of seconds) due to the involvement of live locations. Consequently, GTFS-RT is specifically designed to be streamed using the protocol buffer format, which offers an efficient binary representation of the data (Wu et al., 2022).

So, consuming and processing a GTFS-RT feed entails an additional decoding step to convert the messages into human-readable plain text as well as a mapping logic to extract the corresponding information from the static GTFS schedule. The combined data model resulting from the integration of both specifications, GTFS and GTFS-RT, is illustrated in Figure 2.1. Due to space limitations, only the required dimensions and fields are included in the data model[1].

Furthermore, each protobuf-encoded file is required to specify the structure of its elements and their corresponding type definitions in a text file known as gtfs-real-time.proto. This file is provided by the GTFS reference and serves as a means to parse the protocol buffer data into a programming language specific class representation. An example of a decoded feed message, which includes details about a vehicle's current position at the time the request was sent to the endpoint, is illustrated in Listing 2.1.

GTFS and GTFS-RT are used by a wide range of applications, including trip planning and navigation apps, as well as tools for managing and optimizing transit systems. These standards have greatly improved the availability and accessibility of public transit information, making it easier for people to use public transportation and leading to increased ridership and reduced congestion in cities around the world (Goliszek & Połom, 2016).

One significant benefit of GTFS and GTFS-RT is their ability to support the development of applications that can seamlessly work with multiple transit agencies, even if those agencies utilize different systems or technologies. This interoperability is possible because GTFS and GTFS-RT establish a standardized data structure and format, as described earlier, that can be utilized to represent information from any agency. As a result, developers are empowered to create applications

[1] A complete reference can be accessed at https://developers.google.com/transit/site-map

that can seamlessly connect with any transport agency that has adopted these standards. This facilitates easier access and usage of public transportation for individuals (Kujala et al., 2018).

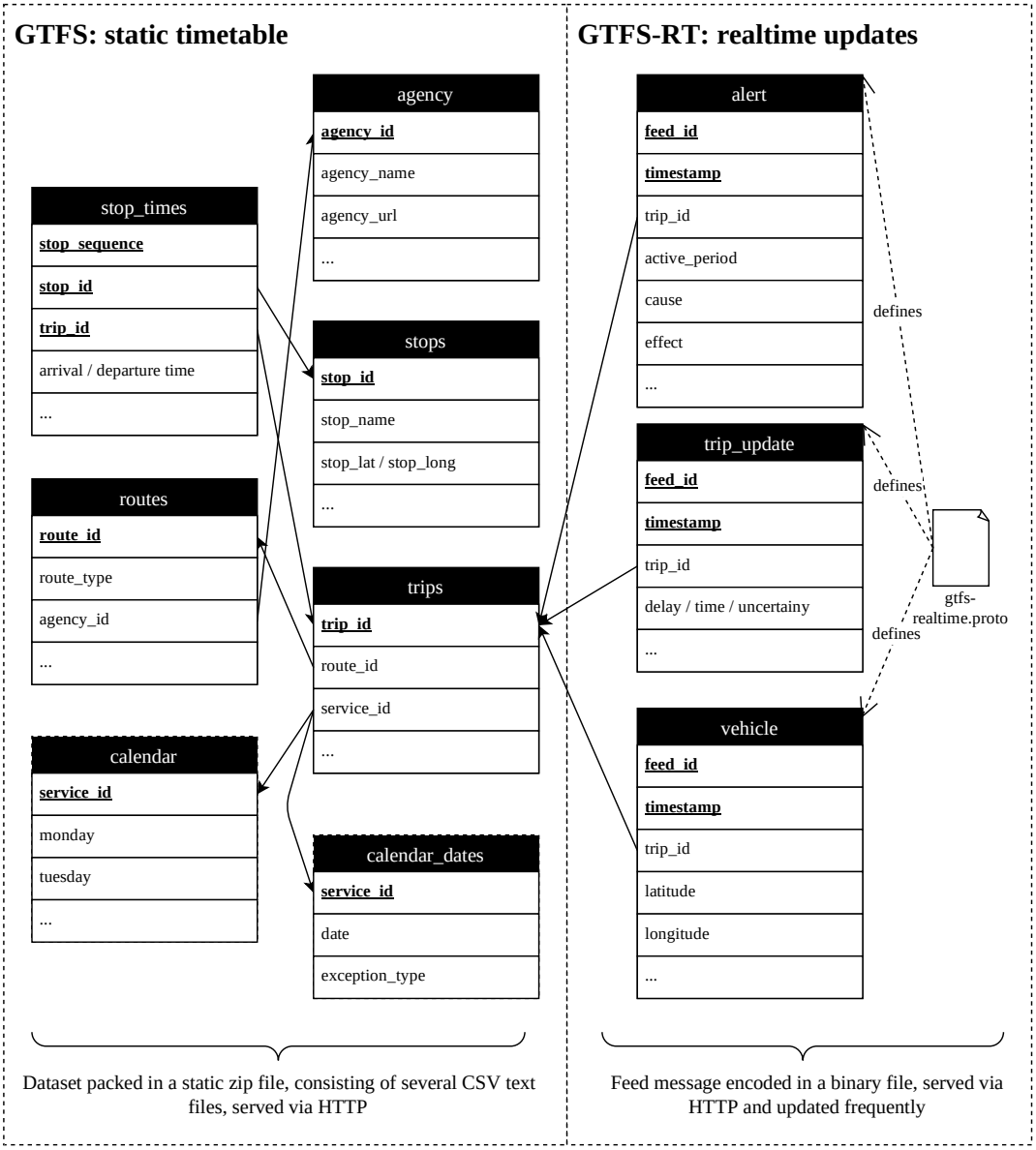


Figure 2.1: Data model of a GTFS file collection including GTFS-RT

```
{
  " entity ":    [
     {
        " id ":   " vehicle :268435809" ,
        " vehicle ":    {
           " current_status ":       " IN_TRANSIT_TO ",
           " position ":     {
              " bearing ":    184.0 ,
              " latitude ":      48.39114761352539 ,
              " longitude ":     -4.436010360717773
           },
           " trip ":    {
              " route_id ":     "03" ,
              " trip_id ":      "15797257"
           },
           " vehicle ":    {
              " id ":   "268435809"
           },
            //    ...
        }
     },
     //    ...
     ]
}
```

Listing 2.1 Exemplary content of a decoded feed message of type vehicle

2.2 JValue Tooling Ecosystem

Data engineering is the process of transforming and preparing raw data into a format that is ready for analysis, modeling, or other downstream uses (Ramamoorthy & Wah, 1989). In open data, data engineering is even more crucial due to the varying quality of data provided by publishers. Open collaboration, which is similar to open-source development in software engineering context, can help reduce individual costs by allowing separate parties to collaborate on shared artifacts. By collaborating to improve data quality, open data users can avoid relying solely on slow or poorly-structured data publishers (Heltweg & Riehle, 2022).

The primary objective of JValue is to achieve "[...] a world that utilizes open data to its fullest." (Professorship of Open Source Software at the University of Erlangen, 2022). JValue is a project initiated by the Professorship of Open Source Software at Friedrich-Alexander University Erlangen-Nürnberg. It offers a collaborative open-source software solution for Extract-Transform-Load (ETL) pipelining. The distinguishing and integral feature of JValue lies in its capacity to facilitate collaborative data engineering projects, marking a vital step towards democratizing open data utilization (Professorship of Open Source Software at the University of Erlangen, 2022).

JValue's Components Overview

Since the project is currently under continuous development, the state described in this thesis relates to the beginning of 2023. Back then, the project consisted of three main parts:

Jayvee A domain specific language for declarative description of ETL-pipelines stored in human-readable Jayvee files (file-extension .jv). The project also includes an interpreter for processing and executing Jayvee files.

JValue Hub A web frontend, which enables users to define and collaborate on data pipeline models using Jayvee as a modeling language. The Hub communicates with the backend and controls the execution of the defined pipelines.

JValue Pipeline Service A backend service, which uses a runtime for executing Jayvee pipeline descriptions coming from the Hub by interpreting the Jayvee models using a Jayvee-interpreter.

Interactions between the overall JValue tooling ecosystem are visualized as a sequence diagram in Figure 2.2. After the user has defined and configured a pipeline, the execution of the pipeline can be triggered. Therefore, the Hub initiates a pipeline run by calling the Pipeline Service. The Pipeline Service then sends a request to the Runtime for executing the run using the pipeline model. The Jayvee Interpreter starts executing the pipeline definition. The current status of the pipeline execution (e.g., “running” or “success”) is shown in the Hub.

The project’s source code is managed via a GitHub Organisation containing repositories for Jayvee[2] as well as for the Hub and Pipeline Service. The whole project is written in TypeScript, each repository consisting of multiple packages which are organized in a monorepository approach. This improves code consistency, reusability, developer experience, and code quality while supporting the JValue team working on complex projects. TypeScript provides tooling and static typing which can help catch errors early and make the code more robust and maintainable. Additionally, it allows for the creation of reusable types and interfaces which can be shared across multiple packages in the monorepository, leading to more efficient development and fewer bugs.

[2] GitHub Repository https://github.com/jvalue/jayvee

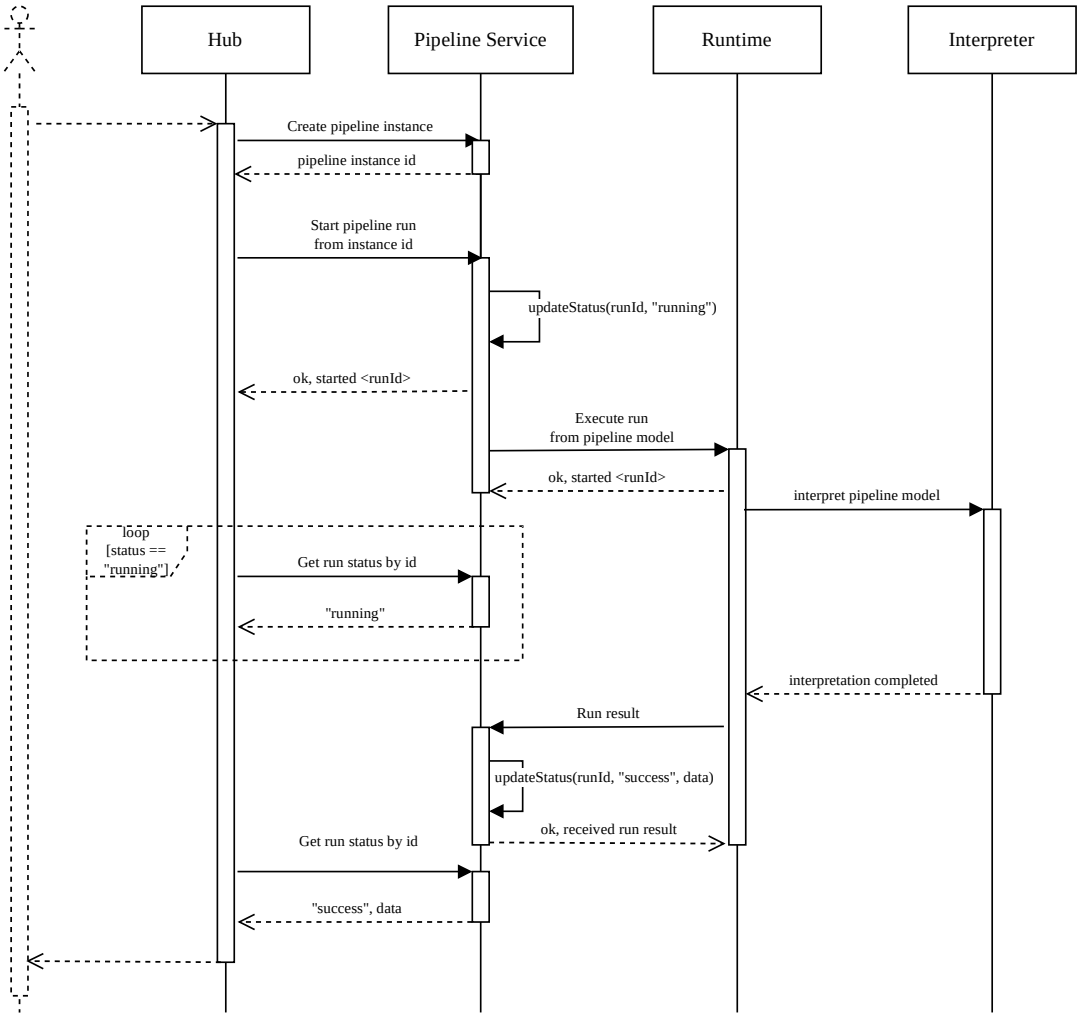


Figure 2.2 Sequence diagram of the JValue tooling ecosystem (extracted and adapted from the JValue documentation)

Jayvee's Conceptual Foundation

This Section serves as a foundation for the architecture draft and design decisions discussed in Chapters 4 and 5. Specifically, it provides an overview of Jayvee, a domain-specific language designed for the declarative description of ETL-pipelines.

The codebase of Jayvee consists mainly of these packages relevant for this research:

- *language-server* - defines and implements the domain specific language us-

ing Langium[3]

- *interpreter* - Command line tool for interpreting Jayvee files)
- *extension* - Implementations of language features
  - *std* - Standard functionality
  - *rdbms* - Extension for support of relational databases
  - *tabular* - Extension for support of tabluar data like CSV files
- *execution* - Execution related code used by the interpreter and by the extensions

The Jayvee language's grammar primarily comprises three, basic entities:

- *Pipeline* - holds a collection of Block and Pipe entities
- *Block* - performs an ETL-related task, such as data extraction, transformation, or loading
- *Pipe* - specifies the execution order of multiple Blocks within a pipeline

A Block has a key-value map syntax that is used to configure the Block. The keys and their corresponding ValueTypes depend on the type of the Block, which is indicated by the identifier that comes after the keyword *a* or *an*. A ValueType is a data type that is used to define the type of the data that a Block processes (e.g., primitive ValueTypes like integer or text).

Further, Jayvee differs between three basic block types:

- *Extractor* - represents a data source and has only a default output, no input
- *Transformator* - represents a transformation and has both a default input and output
- *Loader* - represents a data sink and has only a default input

Data is conveyed through the pipeline as an io-type (e.g., Sheet, Table, or void). To illustrate the use of Jayvee in describing an ETL pipeline, a simple pipeline is presented in Listing 2.2. The pipeline is designed to extract information about cars from a CSV file, interpret it as a table, and then load it into an SQLite database.

[3] https://langium.org

```
pipeline CarsPipeline {
  block CarsExtractor oftype CSVFileExtractor {
    url : " https :// example . org / cars . csv ";
  }
  block CarsTableInterpreter oftype TableInterpreter {
    header : true ;
    columns : [
      " name "typed text ,
      " mpg "typed decimal ,
    ];
  }
  block CarsLoader oftype SQLiteLoader {
    file : " ./ cars . db ";
  }
  CarsExtractor
        -> CarsTableInterpreter
            -> CarsLoader ;
}
```

Listing 2.2 Example of a Jayvee pipeline

For transforming an input string (e.g., the example pipeline defined in Listing 2.2) into a semantic model, Jayvee uses Langium. It constructs an Abstract Syntax Tree (AST) by generating a lexer, analyzer and parser based on the grammar rules defined by Jayvee. Validation is performed to ensure compliance with the language rules, encompassing type checking, name resolution, and scoping. An AST is a tree-like data structure that represents the syntactic structure of source code while abstracting away details such as formatting, punctuation, and comments. The nodes of the tree represent constructs such as expressions, statements, and declarations, while the edges of the tree represent the relationships between them (Noonan, 1985).

3 Requirements

Requirements in software engineering are the specifications that define what a software system should do to meet the needs of its stakeholders (Nuseibeh & Easterbrook, 2000). They guide our design, development and testing of the artefact implemented later on.

3.1 Definition Procedure and Representation

Contributing to Jayvee follows a common RFC process to propose and develop new features, typically established in open-source communities. This process allows anyone to submit a proposal for a new feature or to suggest changes to an existing standard. Once a proposal is submitted, it is reviewed and discussed by the community, and if approved, it is implemented as an official part of the project. For that, JValue uses well-established features for collaborative software development in GitHub. This includes branching for parallel development, issues for bug tracking, and Pull Requests (PRs) for proposing and merging changes. The RFC process in JValue covers the following steps:

1. *Drafting an RFC* - The person proposing the change or standard drafts an RFC document that outlines the proposal. The document must include a clear description of the problem the proposal is trying to solve, the proposed solution, and the potential impact of the change.
2. *Community review* - The RFC document is then committed onto a new git branch in the RFC-folder of the project and feedback from the community is requested by opening a PR referencing the branch. The community will typically discuss the proposal, provide feedback, and suggest changes.
3. *Revision* - Based on the feedback received, the proposer may revise the RFC document and resubmit it for further review.
4. *Final approval* - Once the RFC document has been revised and reviewed, it may be approved by the community.

5. *Implementation* - The RFC and its requirements can now be implemented and shipped to the project, using GitHub Issues and further PRs.

The RFC process ensures that proposals are thoroughly reviewed and discussed by the community before being adopted as standards within the project and also provides an historic perspective on the evolution of the project itself. Figure 3.1 visualizes the process applied in this thesis, consisting of requirement engineering and implementation phase.

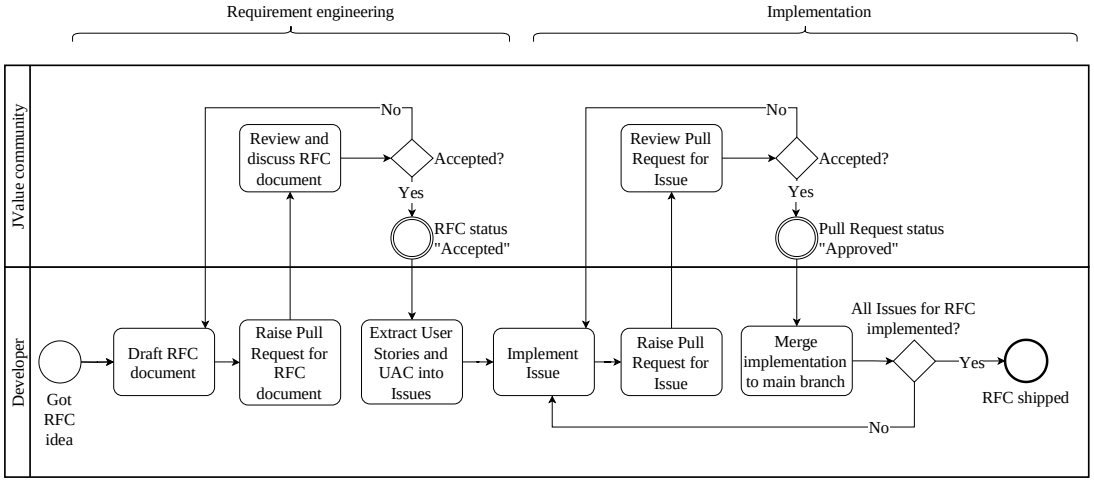


Figure 3.1: Requirement engineering and implementation process applied in this thesis

By following the iterative evolution and detailed description process outlined earlier, a comprehensive understanding of the implementation requirements is achieved. In the context of this thesis, the upstream RFC serves as the foundation for extracting all the necessary requirements. In the next step, User Stories are formulated, encompassing a set of User Acceptance Criteria (UAC) that directly incorporate the requirements derived from the corresponding RFC.

User Stories are a technique applied in agile software development to capture requirements from the perspective of end-users or customers. A User Story is a short, simple statement that describes a specific feature or function that a user needs or wants to perform a particular task or achieve a particular goal (Dalpiaz & Brinkkemper, 2018).

In JValue, User Stories follow a common format *“As a ⟨user/persona⟩, I want ⟨to perform this action⟩, so that ⟨I can accomplish this goal⟩.”*.

UAC are a set of conditions or requirements that must be met for a single User Story to be accepted by the end-users or customers. These are used to validate that the outcome meets their expectations and requirements. UAC are typically

written in plain language and should be testable and measurable, so that it is clear when they have been met (Pandit & Tahiliani, 2015). Following this definitions, an RFC can be seen as finally shipped to the project once the complete set of its User Stories (including their UAC) are implemented and accepted.

3.2 Functional Requirements for GTFS Support

The capability to process GTFS files using the Jayvee interpreter, as seen in the components in Figure 2.2, was introduced into the project through the RFC-0002 Mobility Extension[1].

This document underwent five iterations, with community and developer discussions and reviews. Table 3.1 displays the scope and changes of each iteration. In the digital version of this paper, the column labeled PR contains a hyperlink to the relevant details on GitHub. Otherwise, details can be accessed manually using the URL pattern https://github.com/jvalue/jayvee/pull/<PR>. Further details on significant design decisions and specific implementation approaches are discussed in Chapter 4 and Chapter 5.

Iteration	Package	Stage	Scope	PR
1	Jayvee	Initial concept	File processing using collections	111
2	Jayvee	Refinement	File processing using collections	115
3	Jayvee	Change of concept	File processing using file-pickers	116
4	Jayvee	Refinement	File processing using file-pickers	117
5	Jayvee	ACCEPTED	File processing using file-pickers	119

Table 3.1: Stage, Scope, and Pull Request of different iterations for RFC0002 GTFS support

All requirements extracted from RFC-0002 result in one User Story[2]:

1. As a *user of Jayvee*,
2. I want to *archive GTFS-data from an http-endpoint*,
3. so that *this kind of domain specific data gets stored in a SQLite database file according to the GTFS-relational model*

The requirement to store data in SQLite-format is based on the fact that SQLite is a recommended format for long term archival of structured data according to

[1] RFC document can be found in Appendix Section A and on GitHub: https://github.com/jvalue/jayvee/tree/main/rfc/0002-mobility-extension

[2] User Story document can be found in Appendix Section B and on GitHub: https://github.com/jvalue/jayvee/issues/123

the US Libary of congress (SQLite Consortium, 2018). Further, archiving GTFS-data in an tabular way offers the possibility to validate the underlying schema right away during the load. Lastly, SQLite-sinks are already supported by Jayvee, which integrates well with our overall goal to archive GTFS-data using the JValue tooling ecosystem.

To fulfill the expectation of that User Story, following UAC must be met:

- ☐ *UAC-1 Implement io-type File* - A new io-type File is implemented.
- ☐ *UAC-2 Implement io-type FileSystem* - A new io-type FileSystem is implemented.
- ☐ *UAC-3 Implement io-type None* - A new io-type None is implemented.
- ☐ *UAC-4 Extend io-type Table* - The io-type Table stores the table's name.
- ☐ *UAC-5 Process table name* - The block LayoutValidator processes the new table name coming from Table.
- ☐ *UAC-6 Refactor Jayvee-examples using table name* - The exsiting Jayvee-examples-files are storing the table's name in Table-Block.
- ☐ *UAC-7 Abort execution* - If a precessor of a block outputs None, the execution of the current pipeline aborts.
- ☐ *UAC-8 Introduce Folderstructure* - A folderstructure for io-types is introduced.
- ☐ *UAC-9 Implement HTTPExtractor* - A new blocktype HTTPExtractor is implemented in the standard-extension of Jayvee.
- ☐ *UAC-10 Implement ArchiveInterpreter* - A new blocktype ArchiveInterpreter is implemented in the standard-extension of Jayvee.
- ☐ *UAC-11 Implement FilePicker* - A new blocktype FilePicker is implemented in the standard-extension of Jayvee.
- ☐ *UAC-12 Implement CSVInterpreter* - The current blocktype CSVFileExtractor is refactored to an CSVInterpreter.
- ☐ *UAC-13 Refactor CSVFileExtractor* - The former extractor-functionality of CSVFileExtractor is covered by the new HTTPExtractor and ArchiveInterpreter.
- ☐ *UAC-14 Refactor Jayvee-examples to new blocks* - The Jayvee-examples-files are adapted using the new blocks HTTPExtractor and ArchiveInterpreter.
- ☐ *UAC-15 Conditional GTFS-columns* - All conditional required columns of the GTFS schema are considered as required.

- *UAC-16 Multiple block inputs* - The current block SQLiteSink accepts multiple inputs (For a proof of concept, multiple sinks are accepted, rather than multiple inputs for one sink).
- *UAC-17 Processing table name* - The current block SQLiteSink processes the new table's name.
- *UAC-18 Database creation* - The current block SQLiteSink does not recreate a database each call.
- *UAC-19 Parallel processing* - Parallel processing of independent blocks does not interfere the overall execution of a pipeline.
- *UAC-20 Sucessful execution* - Jayvee successfully processes a GTFS pipeline.

3.3 Functional Requirements for GTFS-RT Support

The concept of Jayvee to processes real-time is outlined in RFC-0006 GTFS-RT support[3]. This document underwent three iterations, which were reviewed and discussed by both the community and developers. Table 3.2 provides a summary of the scope and changes made during each iteration. In the digital version of this paper, the column PR contains a hyperlink to the relevant details on GitHub. Otherwise, the details can be accessed manually using the URL pattern https://github.com/jvalue/jayvee/pull/<PR>. Further details on major design decisions and specific implementation approaches are provided in Chapters 4 and 5.

Iteration	Package	Stage	Scope	PR
1	Jayvee	Initial concept	File processing using GtfsRTInterpreter	200
2	Jayvee	Refinement	File processing using GtfsRTInterpreter	201
3	Jayvee	ACCEPTED	File processing using GtfsRTInterpreter	311

Table 3.2: Stage, Scope, and Pull Request of different iterations for RFC0006 GTFS-RT support

[3] RFC document can be found in Appendix Section C and on GitHub: https://github.com/jvalue/jayvee/tree/main/rfc/0006-gtfs-rt-support

The User Story[4] pertains to process both, static and real-time GTFS data using Jayvee, and is stated as followed:

1. As a *user of Jayvee*,
2. I want to *archive GTFS-RT-data from an http-endpoint*,
3. so that *multiple executions of a pipeline containing both, GTFS and GTFS-RT sections, demonstrate an archiving process of static as well as real-time GTFS data*

To fulfill the expectation of that User Story, following UAC must be met:

- ☐ *UAC-1 Implement GtfsRTInterpreter* - A new blocktype GtfsRTInterpreter is implemented in std-extension.
- ☐ *UAC-1.1 Define simple GTFS-RT pipeline* - A new demo pipeline gtfs-rt-simple.jv is implemented.
- ☐ *UAC-2 Implement DropTable attribute* - The current blocktype SQLiteSink is configurable by an attribute dropTable indicating to drop data before loading to the sink.
- ☐ *UAC-3 Showcase GTFS and GTFS-RT data processing* - A new pipeline gtfs-static-and-rt.jv is added to showcase the processing of real world GTFS as well as GTFS-RT data.
- ☐ *UAC-4 Create/Update SQLite file* - Every run of gtfs-static-and-rt.jv creates/updates one single SQLite database.
- ☐ *UAC-5 Overwrite GTFS data* - Every run of gtfs-static-and-rt.jv downloads GTFS data and overwrites GTFS tables.
- ☐ *UAC-6 Append GTFS-RT data* - Every run of gtfs-static-and-rt.jv downloads GTFS-RT data and appends to GTFS-RT tables.
- ☐ *UAC-7 Sucessful execution* - Jayvee processes gtfs-static-and-rt.jv successfully.

[4] User Story document can be found in Appendix Section D and on Github: https://github.com/jvalue/jayvee/issues/219

3.4 Non Functional Requirements

To ensure Jayvee's robust performance it's essential to examine Non Functional Requirements (NFRs), as they shape system constraints like reliability, usability and scalability (Chung et al., 2012). Both extensions, GTFS and GTFS-RT, encompass the following:

- *NFR-1 Using compositions* - To increase the reliability and reusability of the implemented block types and artifacts, a generic approach should be adopted. Specifically, utilizing a composition of blocks should be applied, as this approach involves breaking down the functionality into smaller packages rather than relying on domain-specific blocks. It enables the seamless integration of the artifacts and block types into any ETL-pipeline, thereby enhancing their overall applicability.
- *NFR-2 Seamless integration* - The extension should integrate seamlessly with the existing Jayvee implementation and should not introduce significant logical modifications to the grammar.
- *NFR-3 High execution performance* - The implementation must exhibit high execution performance and should not give rise to any significant runtime issues.

These non-functional requirements for both extensions, including using compositions, seamless integration, and high execution performance, guide the development process towards achieving a reliable, versatile, and efficient system.

4 Architecture

Architecture refers to the high-level structure of a software system. It defines the overall organization of the system, including the major components and their relationships, as well as the constraints and principles that guide the design and implementation of the system. The architecture of a system defines the fundamental decisions that shape the system and its ability to meet the needs of its users (Garlan & Shaw, 1993).

The development of JValue and Jayvee is still in its early stages, which means that changes to the architecture and core principles can be expected during the course of this thesis and after its publication. Consequently, every concept described here refers to the point in time when the corresponding PR was merged into the main repository branch, unless explicitly stated otherwise. However, in order to maintain coherence in our arguments, we will also discuss the core concepts and approaches that have undergone minor modifications when necessary. It is important to note that these changes are a natural part of the software development process and are aimed at enhancing the overall functionality and performance of the platform.

4.1 GTFS Support

In the domain of processing GTFS files, we utilize the pipeline design pattern, which is thoroughly explained in Section 2.2. This pattern consists of multiple stages, namely an Extractor, several Interpreters, some Validators, and finally a Loader that directs the processed data to a designated sink. The main goal is to load a GTFS dataset into a tabular representation, as illustrated in Figure 2.1.

In the initial stages of the project, the proposed plan outlined in RFC-0002 iteration 1[1] was to develop a GtfsInterpreter block type capable of handling collections of io-type to process the GTFS archive file. The idea was to unpack the GTFS archive and process a set of CSV files within the GtfsInterpreter block. However, this implementation was found to be impractical as it required

[1] RFC-0002 iteration 1: https://github.com/jvalue/jayvee/pull/111

substantial modifications to the grammar to support the concept of collections of io-type.

Instead of following the idea of io-type collections, a concept of a filesystem containing multiple files accessed by file pickers was proposed and finally accepted[2]. This approach avoids a fundamental change in the grammar of Jayvee which would have been caused by supporting collections. To achieve this goal, all underlying CSV files that comprise the GTFS archive file are processed using a separate track in the pipeline, with each CSV file processed with its dedicated blocks.

After unarchiving the GTFS archive file, the individual CSV files containing the dimension's data are selected based on their filename and processed independently in parallel[3]. In Figure 4.1 a model of the GTFS pipeline discussed is shown, including all necessary blocks highlighted by their implementation status (Implementation from scratch / Changes to existing artifacts). However, if a file is not found in the GTFS file collection as expected, the processing of that file is immediately halted, and no table is generated from it. This prevents the accumulation of incomplete or erroneous data in the database, reducing the risk of errors or inconsistencies in downstream applications that depend on the processed data. Finally, each successfully generated table is loaded into the same SQLite sink, consolidating all the relevant data into a single database. This approach ensures the capture of all crucial data from the GTFS file, while excluding any missing or incomplete data, thereby guaranteeing the integrity of the processed data.

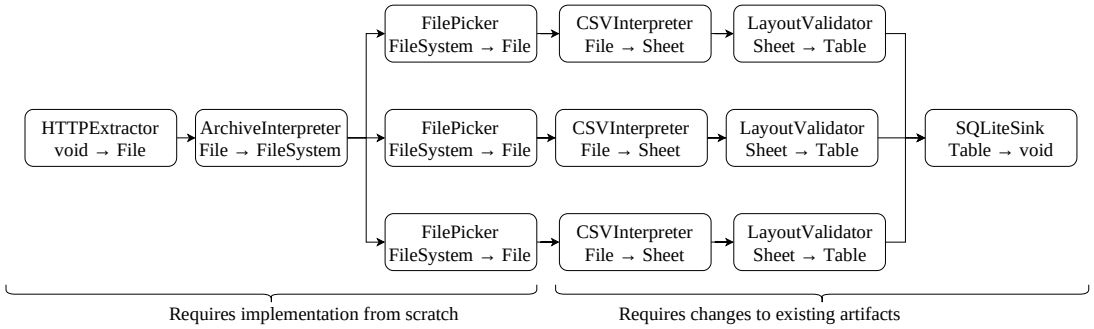


Figure 4.1: GTFS pipeline model

In more detail, the pipeline begins with a HttpExtractor that sends a HTTP-GET request to an endpoint, which provides a GTFS dataset. The output of this block is the HTTP response, which is stored as a binary file. Next, an instance of the ArchiveInterpreter is employed to parse the downloaded file, assuming it

[2] RFC-0002 iteration 5: https://github.com/jvalue/jayvee/pull/119

[3] From an execution perspective, the Jayvee interpreter still executes each pipeline sequentially based on a total order of all blocks

follows an archive file format. This interpreter initializes an in-memory filesystem based on the content of the input file.

After the preparation phases have been successful, the following steps are executed in parallel. A FilePicker is responsible for selecting a specific File from the incoming FileSystem and forwarding it to the downstream CSVInterpreter. The CSVInterpreter then interprets the selected File as a tabular-like Sheet. This Sheet is then evaluated against an expected schema by the subsequent LayoutValidator.

Finally, the Table parsed from the input GTFS archive is loaded into a user-named table in a SQLite database. This ensures that each Table is stored separately, maintaining organization and avoiding data overlap.

With this setup, the user can control which dimensions of the the GTFS-schema should be loaded to the sink by adding or removing accordingly the parallel streams of FilePicker, CSVInterpreter and LayoutValidator, since every dimension needs their own stream. Overall, this pipeline design pattern provides a reliable and scalable solution for processing large GTFS datasets. Additionally, the newly introduced concepts of files, filesystems and file pickers can be used generically with every kind of folder structures, since this concept is not GTFS specific.

4.2 GTFS-RT Support

The integration of GTFS-RT support can be viewed as a further extension of the GTFS foundation in the Jayvee architecture enabling real-time update processing. By incorporating GTFS-RT, Jayvee can process pipelines that not only extract static public transportation schedules and associated geographic information but also real-time updates about associated fleets such as delays, cancellations, and vehicle positions, among others. As described in Chapter 2, GTFS-RT data is typically provided in the form of streaming data feeds that are updated in real-time as events occur. It is specified that GTFS-RT is streamed using the protocol buffer format. So, to store human-readable plain text in the SQLite sink, an additional decoding stage is required to convert the feed's messages.

Figure 4.2 illustrates a simplified model of a GTFS-RT pipeline capable of processing all possible entities within a GTFS-RT feed. This concept, proposed in the accepted RFC-0006[4], utilizes artifacts implemented in the former GTFS architecture proposal.

The pipeline begins with an HTTPExtractor responsible for downloading the protobuf file from an endpoint. Next, a parallel configuration of GtfsRTInterp

[4] RFC-0006 iteration 2: https://github.com/jvalue/jayvee/pull/201

reters is employed to extract entities such as trip updates, alerts, and vehicle positions, as defined in the GTFS-RT reference. These interpreters are designed to work in parallel, enabling efficient processing of the data.

Following the extraction of entities, the TableInterpreter block is introduced to match the extracted data against an expected schema. Due to parallel changes by the JValue team, the TableInterpreter replaces the previous LayoutValida tor block. However, it is important to note that there have been no functional changes, only differences in syntax.

Finally, the SQLiteSink block saves the processed data as tables within a SQLite database. This ensures the persistence of the extracted information for further analysis and utilization.

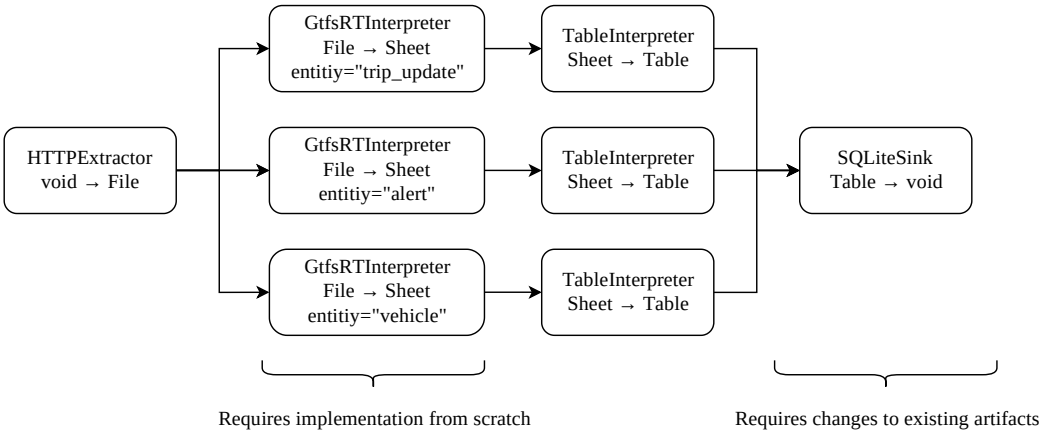


Figure 4.2: GTFS-RT pipeline model

4.3 Periodical Archival Mechanism

Since the overall goal of this thesis is to enable a periodical load of GTFS- and related GTFS-RT-data, the concept for the archival mechanism is:

1. One pipeline containing both, GTFS and GTFS-RT sections.
2. Periodical execution of the whole pipeline.
3. An additional attribute for SQLite-sink indicates whether tables should be dropped before load start. GTFS-tables are dropped every run, GTFS-RT-tables not, which leads to a dataset containing the static information as well as the incrementally growing real-time data tables.

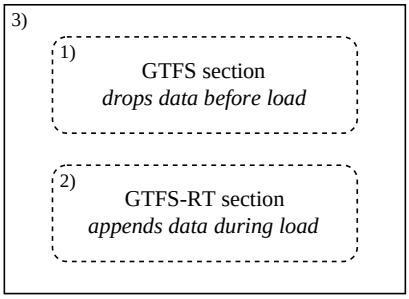


Figure 4.3: Schematic pipeline (3), which executes segments for GTFS (1) and GTFS-RT (2) periodically

The proposed concept results in a pipeline that contains two sections, as illustrated in Figure 4.3. By including both static GTFS-information and real-time updates from GTFS-RT in the same pipeline, this approach provides a comprehensive and up-to-date archive of transportation data that can be accessed at any time. Additionally, the periodic execution of the pipeline ensures that the data remains current, and any updates to the transportation system are captured in a timely manner.

5 Design and Implementation

Design refers to the detailed implementation of the system. It includes the specific choices made about data structures, algorithms, and interfaces, as well as the organization of the codebase. Design decisions are based on the architecture and they determine how the system will function (Perry & Wolf, 1992).

The subsequent sections provide a detailed description of the design and implementation phase, which encompasses interfaces, design decisions, and implementation approaches. The complete source code can be accessed through the GitHub references provided in the footnotes. A bill of materials, which includes all newly introduced libraries and their corresponding licenses, is available in Appendix Section E.

It is important to note, as the development of Jayvee is progressing rapidly, some of the fundamental concepts and approaches have been subject to minor adjustments during the design and implementation phase of this thesis. Also, minor improvements of a concept were introduced, after the initial PR has already been merged. Hence, any explanation of a concept presented herein refers to the time when the corresponding PR was merged into the main branch of the repository, unless specified otherwise.

5.1 GTFS Support

To enable Jayvee to process GTFS data, the following components need to be added:

- New io-types: The introduction of File and FileSystem io-types is necessary. These will handle file-related operations and manage the file system within Jayvee.
- Enhancing the Table io-type: The Table io-type needs to be modified to include the ability to store a table's name. This enhancement will allow for the handling of multiple tables as input within the framework.
- New blocktypes: The addition of new blocktypes is required to support

GTFS data processing. These blocktypes include HTTPExtractor, which handles downloading data from HTTP endpoints, ArchiveInterpreter, which interprets archive file formats, and FilePicker, responsible for selecting specific files from a file system for further processing.

- Implementation of an abort mechanism: To incorporate an abort mechanism, a new io-type called None must be introduced. This io-type will be utilized to halt the execution of subsequent blocks if a predecessor block outputs None.

IO-Types for GTFS Support

In a pipeline, data is encapsulated within an io-type, with each block expecting a specific type. Typically, the output type of a preceding block must correspond to the input type of the subsequent block. However, in the case of extractors/-loaders, the input and output types are naturally defined as void.

Filesystem Design[1]

To simplify the retrieval and storage of files within directories, we propose the implementation of a new class hierarchy that adheres to the commonly-used composite design pattern, as discussed by Gamma et al. (1996) among others. A UML class diagram depicting the implemented classes and interfaces is illustrated in Figure 5.1. Implementing a filesystem using this pattern offers several advantages. Firstly, it provides a unified interface for accessing both files and directories, which simplifies the code and reduces complexity. This unified interface allows clients to treat both files and directories in the same way, making it easier to work with the filesystem. In our case, we treat the InMemoryFileSystem as the client using the abstract FileSystemNode class. Secondly, the composite pattern allows for the creation of complex hierarchical structures, with directories containing subdirectories and files. This enables the creation of a more organized and efficient filesystem, with a clear hierarchy that can be easily navigated. Thirdly, the composite pattern allows for the implementation of recursive algorithms that can be applied to the entire filesystem.

In the initial version, the implementation differed from the current version, as the composite pattern was incorporated later for optimization purposes. As a result, some of the code artifacts are associated with more than one PR.

[1] PR on GitHub: https://github.com/jvalue/jayvee/pull/256

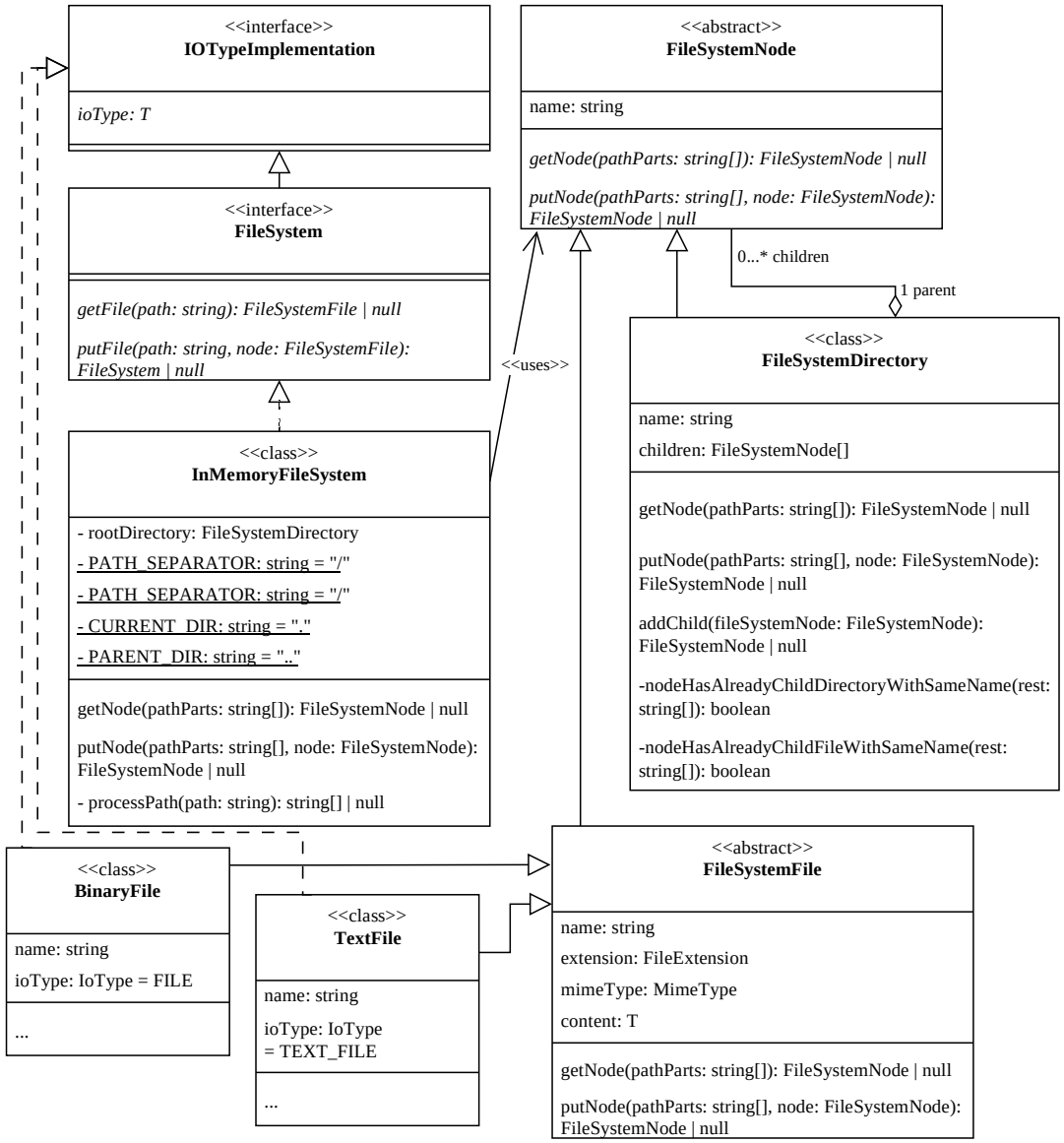


Figure 5.1: Class diagram of FileSystem and File using the composite design pattern

For instance, in our implementation, the process of retrieving a file (Listing 5.1) and storing a file (Listing 5.2) is executed recursively by traversing through the directories specified in the filepath across the entire file system. This is achieved by considering each node as a composite of itself and its children. Furthermore, when using the putNode() method, new directories are dynamically created as needed. Therefore, if a file is to be stored in a directory path where some parts of this path do not yet exist, the method generates new directories. Also, the composite pattern allows for the creation of abstract base classes that can be extended to

implement specific functionality for different types of nodes. In our case this is represented by the abstract FileSystemFile where BinaryFile and TextFile extend from.

```
override getNode(pathParts: string[]): FileSystemNode | null {
  const [firstPart, ...rest] = pathParts;
  // Base case: Called a wrong node
  if (firstPart !== this.name) {
    return null;
  }
  // Base case: Called the right node
  if (rest.length === 0) {
    return this;
  }
  // Recursion case: Traverse child nodes
  for (const child of this.children) {
    const f = child.getNode(rest);
    if (f) {
      return f;
    }
  }
  return null;
}
```

Listing 5.1 Retrieving a node recursively in FileSystemDirectory (filesystem-node-directory.ts)

```
override putNode(pathParts: string[], node: FileSystemNode): FileSystemNode | null {
  const [firstPart, ...rest] = pathParts;
  // Base case: Called a wrong node
  if (firstPart !== this.name) {
    return null;
  }
  // Base case: One path part left (which is the filename)
  if (rest.length === 1) {
    if (
      !this.nodeHasAlreadyChildFileWithSameName(rest) &&
      node.name === rest[0]
    ) {
      this.addChild(node);
      return node;
    }
    return null;
  }
  // Case: Add directory, because it does not exist
  if (
    !this.nodeHasAlreadyChildDirectoryWithSameName(rest) &&
    rest[0] != null
  ) {
    const newdir = new FileSystemDirectory(rest[0]);
    this.addChild(newdir);
    return newdir.putNode(rest, node);
  }
  // Recursion case: Traverse child nodes
  for (const child of this.children) {
    const f = child.putNode(rest, node);
    if (f) {
      return f;
    }
  }
  return null;
}
```

Listing 5.2 Storing a node recursively while creating new directories (filesystem-node-directory.ts)

IO-Type FileSystem[2]

An io-type FileSystem (Listing 5.3) allows for a hierarchical representation of multiple files through the unpacking of, for instance, a GTFS ZIP file containing multiple text files. The interface defines generic methods for accessing and storing files. The getFile-method retrieves a file by its absolute path starting at the root of the file system, returning a null value if the file does not exist. The putFile-method stores a file using its absolute path, returning the FileSystem if the file is stored successfully. This return value enables method chaining, which increases the usability (e.g., fs.putFile("/file1")?.putFile("/file2")).

```
1  export interface FileSystem extends IOTypeImplementation < IOType . FILE_SYSTEM > {
2    getFile ( path : string ): FileSystemFile < unknown > | null ;
3    putFile ( path : string , file : FileSystemFile < unknown >) : FileSystem | null ;
4  }
```

Listing 5.3: Interface FileSystem (filesystem-io-type.ts)

We present an implementation of a FileSystem in shape of FileSystemInMemory. The implementing methods (Listings 5.4 and 5.5) process the incoming path and subsequently delegate the call to the dedicated methods described earlier. Processing a path involves validating if it starts with a PATH_SEPARATOR, removing empty path components, and handling current and parent directory references. The processed path components are then returned as an array (Listing 5.6).

```
1  getFile ( path : string ) : FileSystemFile < unknown > | null {
2      const processedParts = this . processPath ( path );
3      if ( processedParts != null ) {
4        const node = this . rootDirectory . getNode ( processedParts );
5        if ( node instanceof FileSystemFile ) {
6          return node ;
7        }
8      }
9      return null ;
10 }
```

Listing 5.4: Getting a file from the InMemoryFileSystem (filesystem-inmemory.ts)

```
1  putFile ( path : string , file : FileSystemFile < unknown >) : FileSystem | null {
2      const processedParts = this . processPath ( path );
3      if ( processedParts != null ) {
4        const node = this . rootDirectory . putNode ( processedParts , file );
5        if ( node instanceof FileSystemFile ) {
6          return this ;
7        }
8      }
9      return null ;
10 }
```

Listing 5.5: Storing a file into the InMemoryFileSystem (filesystem-inmemory.ts)

[2] PRs on GitHub: https://github.com/jvalue/jayvee/pull/126 and https://github.com/jvalue/jayvee/pull/256

```
private processPath ( path : string ) : string [] | null {
    if (! path . startsWith ( InMemoryFileSystem . PATH_SEPARATOR )) {
      return null ;
    }
    const parts = path
      . split ( InMemoryFileSystem . PATH_SEPARATOR )
      // Process paths like " folder1 // folder1 " to " folder1 / folder2 "
      . filter (( p) => p !== "");
    const processedParts : string [] = [];
    for ( const part of parts ) {
      if ( part ===InMemoryFileSystem . CURRENT_DIR ) {
        continue ; // Skip current dirs in path
      }
      if ( part ===InMemoryFileSystem . PARENT_DIR ) {
        // Go level up in folder hierarchy , max level up is root dir
        const poppedPath = processedParts . pop () ;
        // If Path ascend beyond root , error
        if ( poppedPath ===undefined ) {
          return null ;
        }
      } else {
        processedParts . push ( part );
      }
    }
    return ["" , ... processedParts ];
}
```

Listing 5.6: Processing a path into its parts, resolving current and parent directory indicators (filesystem-inmemory.ts)

IO-Type FileSystemFile[3]

The io-type FileSystemFile (Listing 5.7) and its implementations BinaryFile and TextFile is used for managing the GTFS-archive file and later unpacked CSV files. It defines a data type for an object that represents a file, which has four properties. The first property, name, represents the filename as a string. The second property, extension, stores common extensions, including extensions like ZIP or TXT, in an enumeration of type FileExtension. The third property, mimeType, stores common MIME types, such as APPLICATION_OCTET_STREAM, in an enumeration of type MimeType. Finally, the fourth property, content, is capable of storing various types of data, including binary data, and gets initialized by its implementing child classes.

```
export abstract class FileSystemFile <T > extends FileSystemNode {
  public override readonly name : string ;
  public readonly extension : FileExtension ;
  public readonly mimeType : MimeType ;
  public readonly content : T;
}
```

Listing 5.7: Abstract class FileSystemFile (file-io-type.ts)

[3] PRs on GitHub: https://github.com/jvalue/jayvee/pull/125 and https://github.com/jvalue/jayvee/pull/256

IO-Type None[4]

In order to indicate skipping of downstream block execution in the interpreter, the io-type None (Listing 5.8) is utilized. If a GTFS endpoint fails to provide a table expected by a defined pipeline, the execution of that particular pipeline section[5] should be halted. This is because GTFS serves as a reference that outlines both mandatory and optional tables. In a future scenario, the inclusion of GTFS support will facilitate data archiving from various endpoints. The intention is to develop a pipeline that encompasses all possible tables and fields, ensuring that execution is terminated if a file is not present.

```
1 export interface None {
2
3 }
```

Listing 5.8: Interface None (none-io-type.ts)

IO-Type Table[6]

During the discussion regarding the addition of the table name to the io-type, these proposed changes were already implemented by different team members.

Block Types for GTFS Support

In order to comply with the proposed concept (Section 4.1), it is necessary to implement new block types, namely HttpExtractor, ArchiveInterpreter, and FilePicker from the ground up. Conversely, pre-existing block types such as CSVExtractor and SQLiteSink require only minor refactoring or modification to conform to the proposed specifications.

Block Type HttpExtractor[7]

Input: void → Output: File

A HttpExtractor block (Listing 5.9) is designed to retrieve data from an HTTP endpoint by sending an HTTP GET request to a specified URL, and then outputting the response as a File. Such a block is versatile and can be used to obtain a wide range of data types via HTTP(S). As a result, it has been implemented in the std-extension for general use.

[4] PR on GitHub: https://github.com/jvalue/jayvee/pull/126

[5] Pipeline section refers back to the parallel sections shown in the demo pipeline in Figure 4.1

[6] PRs on GitHub: https://github.com/jvalue/jayvee/pull/164 and https://github.com/jvalue/jayvee/pull/165

[7] PR on GitHub: https://github.com/jvalue/jayvee/pull/134

```
block MyHttpExtractor oftype HttpExtractor {
    url: "https://developers.google.com/static/transit/gtfs/examples/sample-feed.zip";
}
```

Listing 5.9 Block of type HttpExtractor (example)

Listing 5.10 presents a simplified implementation of the core functionality of the HttpExtractor block for fetching data from an HTTP endpoint. After successfully fetching the data, this method endeavors to extract metadata, such as the file name, extension, and MIME type of the output file. In situations where the metadata cannot be inferred, fallback values are utilized to ensure valid output. A new file of type BinaryFile is then instantiated using the previously inferred metadata. It is worth noting that BinaryFile is a child class that implements the introduced abstrac class FileSystemNode .

```
private fetchRawDataAsFile(
  url: string,
  context: ExecutionContext,
): Promise<R.Result<BinaryFile>> {
    // Logging
    // ..
  return new Promise((resolve) => {
    https.get(url, (response) => {
      // Catch errors
      // ..
      // Get chunked data and store to ArrayBuffer
      let rawData = new Uint8Array(0);
      response.on('data', (chunk: Buffer) => { /* ... */ });
      // When all data is downloaded
      response.on('end', () => {
        // Infer metadata (name, extension, MIME type) and create file
        // ...
          const file = new BinaryFile(/* ... */);
          resolve(R.ok(file));
      });
      response.on('error', (errorObj) => { /* ... */ });
  });
}
```

Listing 5.10: Simplified version of method for fetching HTTP data as File (http-extractor-executor.ts)

Block Type ArchiveInterpreter[8]

Input: File → Output: FileSystem

An ArchiveInterpreter block (Listing 5.11) accepts a File as input and interprets it as an archive file. It then proceeds to unpack the archive and initialize a FileSystem with the contents of the unpacked files. This block is designed to work with any archive file and has therefore been implemented in the std-extension.

[8] PR on GitHub: https://github.com/jvalue/jayvee/pull/135

```
block ZipArchiveInterpreter oftype ArchiveInterpreter {
    archiveType: "zip"
}
```

Listing 5.11: Block of type ArchiveInterpreter (example)

To extract zipped files, we utilize an additional library namely JSZip[9] which is published under the MIT license (Bill of Materials in Appendix Section E). JSZip offers a straightforward API for creating, reading, and modifying zipped files. The code snippet shown in Listing 5.12 demonstrates how the library is employed to initialize an InMemoryFileSystem with the content of the zipped file. Prior to adding a File object to the InMemoryFileSystem, the method attempts to deduce the metadata. In cases where the metadata cannot be inferred, default values are used to guarantee accurate output.

```
private async loadZipFileToInMemoryFileSystem(
    archiveFile: BinaryFile,
    context: ExecutionContext,
): Promise<R.Result<FileSystem>> {
    // Logging
    // ..
    const jszip = JSZip();
    const root = new InMemoryFileSystem();
    const archivedObjects = await jszip.loadAsync(archiveFile.content);
    for (const [relPath, archivedObject] of Object.entries(
        archivedObjects.files,
    )) {
        // Infer metadata (name, extension, MIME type) and create file ...
        // ...
        root.putFile(relPath, file);
        // Assert correct result
        // ...
    }
    return R.ok(root);
}
```

Listing 5.12: Simplified version of method for unpacking zip archives (archive-interpreter-executor.ts)

Block Type FilePicker[10]

Input: FileSystem → Output: File

A FilePicker block (Listing 5.13) takes a FileSystem as input and uses the path specified in the path attribute to navigate to the file. If the file is found, the block outputs an initialized File object.

```
block AgencyFilePicker oftype FilePicker {
    path: "/agency.txt";
}
```

Listing 5.13: Block of type FilePicker (example)

[9] https://github.com/Stuk/jszip

[10] PR on GitHub: https://github.com/jvalue/jayvee/pull/136

Block Type CSVInterpreter

Input: File → Output: Sheet

In the Jayvee extension tabular, a CSVFileExtractor is implemented to load a CSV file from a URL and output a Sheet. However, this violates the separation of block types into Extractor and Transformers. To address this, we refactored the CSV-related functionality from CSVFileExtractor into a new dedicated block namely CSVInterpreter (Listing 5.14). It takes in a File assumed to be a CSV file and outputs the file as a Sheet. The delimiter can theoretically be any string value and is set to a comma by default for convenience reasons. For extractor functionality is then provided by the HttpExtractor.[11]

```
block MyCSVInterpreter oftype CSVInterpreter {
    delimiter: ",";
}
```

Listing 5.14: Block of type CSVInterpreter (example)

Block Type LayoutValidator replaced by TableInterpreter

Input: Sheet → Output: Table

The LayoutValidator has already been implemented prior to this thesis and hence there is no need for further changes. Since JValue is under heavy development, during the implementation phase, the LayoutValidator got replaced by a TableInterpreter, which has the same functionality but slightly different syntax. It is important to note that we consider conditional columns of an GTFS dimension in a first draft as required (e.g., column agency_id in Listing 5.15).

```
block AgencyTableInterpreter oftype TableInterpreter {
  header: true;
  columns: [
    "agency_id" oftype text,
    "agency_name" oftype text,
    "agency_url" oftype text,
    "agency_timezone" oftype text
  ];
}
```

Listing 5.15: Block of type TableInterpreter for validating dimension agency of a GTFS dataset (example)

[11] PRs on GitHub: https://github.com/jvalue/jayvee/pull/168 and https://github.com/jvalue/jayvee/pull/169

Block Type SQLiteSink[12]

Input: Table → Output: void

Based on the requirements defined in Section 3.2, it is necessary to modify this block's specification to accommodate multiple inputs. This modification would entail a change in the execution logic. Therefore, for an initial demonstration, we have made the decision to create a separate SQLiteSink (Listing 5.16) for each dimension of the GTFS dataset, all using the same database for storing data. This approach allows us to assess the project's results more quickly.

```
block GtfsLoader oftype SQLiteLoader {
    file: " ./ gtfs . sqlite ";
}
```

Listing 5.16: Block of type SQLiteSink (example)

Resulting GTFS Pipeline

After implementing aforementioned components, it becomes feasible to process a complete GTFS dataset. This results in a GTFS-static pipeline (Listing 5.17, complete excerpt in Appendix Section F). As elucidated in Chapter 4, a pipeline executes an HttpExtractor once and allocates distinct sections for each dimension. Each section consumes data from the identical ZipArchiveInterpreter instance.

```
pipeline GtfsPipeline {
  // GTFS related blocks
  // ...
  // GTFS related pipes
  MyHttpExtractor
    -> ZipArchiveInterpreter
      -> AgencyFilePicker
        -> AgencyCSVInterpreter
          -> AgencyTableInterpreter
            -> AgencyLoader ;
    ZipArchiveInterpreter
      -> CalendarDatesFilePicker
        -> CalendarDatesCSVInterpreter
          -> CalendarDatesTableInterpreter
            -> CalendarDatesLoader ;
  // ...
}
```

Listing 5.17: Simplified version of a GTFS-static pipeline which loads agencies and calendar_dates (gtfs-static.jv)

[12] PR on GitHub: https://github.com/jvalue/jayvee/pull/164

5.2 GTFS-RT Support

As the former extension builds the foundation for real-time support, Jayvee needs to be extented by following parts to be able to process GTFS-RT-data:

- New blocktype GtfsRTInterpreter
- The blocktype SQLiteSink needs a mechanism to append data during load

Block Types for GTFS-RT Support

Given that the data format and structure of GTFS-RT is highly specific to its use case, we have decided to provide a dedicated GtfsRTInterpreter that performs the entire processing of real-time data instead of splitting the functionality into a composition of blocks. For retrieving a GTFS-RT protobuf file, we can make use of the previously implemented HttpExtractor.

Block Type GtfsRTInterpreter[13]

Input: File → Output: Sheet

A block GtfsRTInterpreter (Listing 5.18) is designed to receive a binary protobuf-encoded File as input from an upstream HttpExtractor. A specific GTFS-RT entity to be processed from a incoming protobuf file, such as vehicle, trip_update, or alert, is specified in a block parameter. Furthermore, a block decodes a protobuf file and outputs the respective entity as a Sheet. In the initial version, only the required columns of the entity, as defined in gtfs-realtime.proto, are considered. Since there is no dedicated mobility-extension folder in Jayvee, the new block type is implemented in the std-extension.

```
1 block MyGtfsRTInterpreter oftype GtfsRTInterpreter {
2     entity : " vehicle ";
3 }
```

Listing 5.18: Block of type GtfsRTInterpreter (example)

The block implementation employs the gtfs-realtime-bindings library[14] (licensed under Apache 2.0), which supplies language bindings generated from the real-time protocol buffer. These classes facilitate the construction of data model objects for GTFS-RT, which can then be serialized into binary data or conversely, parsed from binary data and converted back into data model objects.

The process of transforming an object containing nested objects and arrays into

[13] PR on GitHub: https://github.com/jvalue/jayvee/pull/223
[14] https://github.com/MobilityData/gtfs-realtime-bindings

a flattened table representation is referred to as “flattening” a object. This involves converting the hierarchical structure of a object into a two-dimensional table format, where each row corresponds to a unique combination of the original object’s properties and values.

To ensure a static data schema in the resulting Sheet, which can be processed by the subsequent TableInterpreter, we opted to employ entity-specific parsing methods. These methods navigate through the entity based on a predefined expected output schema. They unfold and flatten the nested data structure into a row. To achieve consistent entity flattening, it is crucial to identify the index of the collection type (in this case, an Array) with the highest depth. The depth determines the number of unfolds that must be applied during the flattening process.

The overall methodology of the entity-specific parsing methods is similar:

1. Select a specific entity collection to parse.
2. Create an empty Sheet and add the entity header.
3. Iterate over the collection of the selected entity:
   3.1. Select the nested collection of the current iteration.
   3.2. Iterate through the selected nested collection:
      3.2.1. Create a row containing all information, including that of its parent, and add it to the Sheet.
4. Return the resulting Sheet.

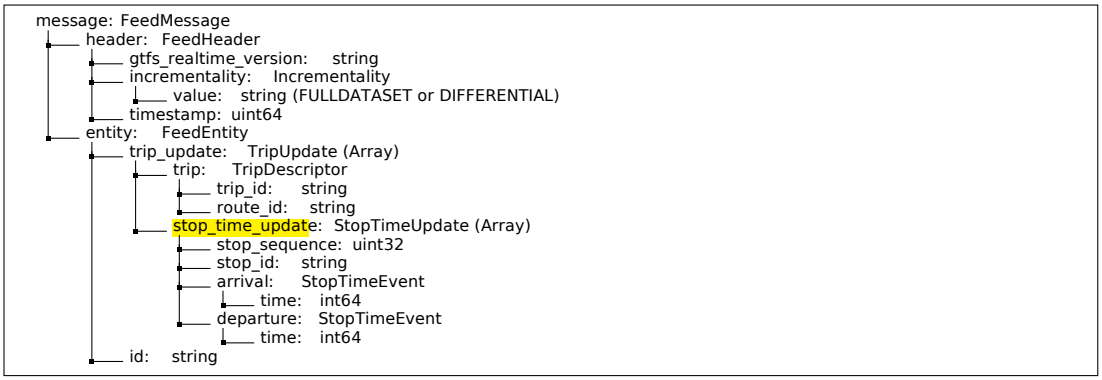


Figure 5.2: GTFS-RT element index of TripUpdate-Entity having stop_time_update as collection with maximum depth (simplified version)

Figure 5.2 illustrates an simplified element index containing the required and necessary conditional required fields for the TripUpdate message entity of

GTFS-RT feed. In this example, the field stop_time_update is the collection with the maximum depth and therefore defines the granularity of a row. This information is explicitly outlined in the GTFS-RT reference and is therefore incorporated as a type definition in the GtfsRTInterpreter.

In order to enhance the understanding of the implementation's behavior, we conducted JSON decoding on a TripUpdate feed message, as illustrated in Listing 5.19. The resulting row is presented as CSV in Listing 5.20. This demonstration effectively showcases the functionality of the flattening mechanism, which proficiently converts the hierarchical JSON structure into a two-dimensional row representation.

```
1  {
2    " header ":  {  " gtfsRealtimeVersion ":      "2.0" ,   " incrementality ":     " FULL_DATASET " , " timestamp ":  "1685198944"  },
3    " entity ":   [
4      {
5        " id ":  " trip :15833110" ,
6        " tripUpdate ":   {
7          " trip ":   {  " tripId ":    "15833110" ,  " routeId ":   "17"  } ,
8          " stopTimeUpdate ":  [{  " stopSequence ":  0 ,  " arrival ":    {  " time ":  "1685199579"  },  " departure ":   {  " time ":
                 "1685199579"  } ,  " stopId ":   " ST_JJ "   }]
9        }
10     },
11     //   ...
12   ]
13 }
```

Listing 5.19: Feed message of type TripUpdate decoded as JSON (example)

```
1  header . gtfs_realtime_version , header . timestamp , header . incrementality , entity . id , entity . trip_update . trip .
       trip_id , entity . trip_update . trip . route_id , entity . trip_update . stop_time_update . stop_sequence , entity .
       trip_update . stop_time_update . stop_id , entity . trip_update . stop_time_update . arrival . time , entity .
       trip_update . stop_time_update . departure . time
2  2.0 ,1685198944 , FULL_DATASET , trip :15833110 ,15833110 ,17 ,0 , ST_JJ ,1685199579 ,1685199579
```

Listing 5.20: Resulting extracted TripUpdate row as CSV including header

Block Type SQLiteSink[15]

Input: Table → Output: void

The SQLite (Listing 5.21) sink is augmented with an extra attribute to indicate whether a table should be dropped before a load operation begins. Given that blocks presently support only one input, a boolean data type suffices for dropping a table. The existing implementation of this block is modified to verify whether to drop any existing tables prior to the load operation.

[15] PR on GitHub: https://github.com/jvalue/jayvee/pull/254

```
block VehicleLoader oftype SQLiteLoader {
    file: "./gtfs.db";
    dropTable: false;
}
```

Listing 5.21: Block of type SQLiteLoader (example)

Resulting GTFS-RT Pipeline

With these contributions, we are now able to utilize the append-data functionality of the SQLiteSink and seamlessly process GTFS-RT data (Listing 5.22, complete excerpt in Appendix Section G).

```
pipeline GtfsRTSimplePipeline {
    // GTFS-RT related blocks
    // ...
    // GTFS-RT related pipes
    GTFSRTTripUpdateFeedExtractor
        -> GtfsRTTripUpdateInterpreter
            -> TripUpdateTableInterpreter
                -> TripUpdateLoader;
    // ...
}
```

Listing 5.22: Simplified version of a GTFS-RT pipeline which loads entities of type TripUpdate (gtfs-rt-simple.jv)

5.3 Combining GTFS with GTFS-RT

Placing the entire implementation within its context, Jayvee now has the ability to load an entire GTFS dataset, including real-time data. To accommodate both types of data, a single pipeline is established with separate sections dedicated to each (Listing 5.23, complete excerpt in Appendix Section H). By periodically executing this pipeline, we can now archive static and real-time GTFS data in a SQLite database. In the pipeline defined, the loader-blocks of the GTFS part do not have an attribute dropTable as it defaults to true. However, the GTFS-RT loader has explicitly set this attribute to false. The JValue team is currently working on "composite blocks" that allow users to combine multiple existing blocks into a new block type, simplifying the pipeline.

```
pipeline     GtfsStaticAndRealtimePipeline          {
      //   GTFS  related    blocks
      //   ...
      block   TripsLoader    oftype   SQLiteLoader    {
            table :    " static_trips ";
            file :     " ./ gtfs - static - and - rt . sqlite ";
      }
      //   ..
      //   GTFS  related    pipes
      GTFSExtractor    ->  ZipArchiveInterpreter ;
      //   ...
      ZipArchiveInterpreter
            ->  TripsFilePicker
            ->  TripsTextFileInterpreter
            ->  TripsCSVInterpreter
            ->  TripsTableInterpreter
            ->  TripsLoader ;
      //   ...
      //   GTFS - RT related    blocks
      //   ...
      block   VehicleLoader    oftype   SQLiteLoader    {
            table :    " rt_vehicle_position " ;
            file :     " ./ gtfs - static - and - rt . sqlite ";
            dropTable :    false ;
      }
      // ...
      //   GTFS - RT related    pipes
      //   ...
      GTFSRTVehiclePositionFeedExtractor
            -> GtfsRTVehiclePositionInterpreter
            -> VehiclePositionTableInterpreter
            -> VehicleLoader ;
      //   ...
}
```

Listing 5.23 Simplified version of a pipeline loading both GTFS and GTFS-RT data (gtfs-static-and-rt.jv)

6 Evaluation

In this Chapter, we present a demonstrator to showcase the functional requirements of the system. This demonstrator will provide a real-world example of how the system can be used to address the problem statement. Overall, the evaluation will provide insights into the effectiveness and usability of the system, as well as highlight any areas where improvements can be made. Furthermore, in reference to the demonstrator, we will evaluate our system against the requirements defined in Chapter 3.

6.1 Demonstrator

To demonstrate and validate the functionality of the implemented system, we have undertaken two actions. First, we have designed a pipeline (complete excerpt in Appendix Section H) that retrieves both static and real-time data from a specific area provided by the NAP of France. Second, we have implemented a dedicated GtfsDemonstrator[1] that utilizes the aforementioned pipeline, validates the output and demonstrates the usage.

The city of Brest[2] and its surrounding region serve as an ideal illustration for showcasing the implemented artifacts detailed in Chapter 4 and 5. With a population of approximately 140,000 residents and 1062 bus stops, Brest provides a compelling case study. It offers four distinct endpoints from which GTFS data can be obtained, as outlined in Table 6.1. The pipeline utilized in this context features dedicated sections for each static file and each of the three real-time entities, effectively querying these endpoints. Upon execution, the pipeline generates a comprehensive GTFS dataset that encompasses both static and real-time data for the Brest region.

[1] Repository on GitHub: https://github.com/schlingling/jayvee-gtfs-demonstrator

[2] French NAP site for Brest: https://transport.data.gouv.fr/datasets/horaires-theoriques-et-temps-reel-des-bus-et-tramways-circulant-sur-le-territoire-de-brest-metropole

Kind of data offered	Endpoint
GTFS static	https://ratpdev-mosaic-prod-bucket-raw.s3-eu-west-1.amazonaws.com/11/exports/1/gtfs.zip
GTFS-RT trip_update	https://proxy.transport.data.gouv.fr/resource/bibus-brest-gtfs-rt-trip-update
GTFS-RT alert	https://proxy.transport.data.gouv.fr/resource/bibus-brest-gtfs-rt-alerts
GTFS-RT vehicle	https://proxy.transport.data.gouv.fr/resource/bibus-brest-gtfs-rt-vehicle-position

Table 6.1: GTFS related endpoint of metropolis region around the city of Brest used for validation

The implemented pipeline comprises a total of 60 blocks and incorporates 12 distinct instances of the SQLiteLoader. The average execution time of the pipeline is 9194ms, as determined by 10 runs carried out on an Apple M1-Pro machine. An exemplary execution output displayed in Listing 6.1 demonstrates a successful extraction, interpretation, and loading of a GTFS file into an SQLite database (complete excerpt in Appendix Section A). The raw data is fetched, parsed, and inserted into a new table named static_trips. Detailed execution logs for all dimensions are available in the appendix.

```
1  [ GTFSExtractor ]  Fetching  raw  data  from  https :// ratpdev - mosaic - prod - bucket - raw . s3 -eu - west -1. amazonaws . com
       /11/ exports /1/ gtfs . zip
2  [ GTFSExtractor ]  Successfully  fetched  raw  data
3  [ GTFSExtractor ]  Execution  duration :  540  ms .
4  [ ZipArchiveInterpreter ]  Loading  zip  file  from  binary  content
5  [ ZipArchiveInterpreter ]  Execution  duration :  95  ms .
6  [ TripsFilePicker ]  Execution  duration :  0  ms .
7  [ TripsTextFileInterpreter ]  Decoding  file  content  using  encoding  " utf -8"
8  [ TripsTextFileInterpreter ]  Splitting  lines  using  line  break  /\ r ?\ n/
9  [ TripsTextFileInterpreter ]  Lines  were  split  successfully ,  the  resulting  text  file  has  5901  lines
10 [ TripsTextFileInterpreter ]  Execution  duration :  2  ms .
11 [ TripsCSVInterpreter ]  Parsing  raw  data  as  CSV using  delimiter  " ,"
12 [ TripsCSVInterpreter ]  Parsing  raw  data  as  CSV - sheet  successful
13 [ TripsCSVInterpreter ]  Execution  duration :  258  ms .
14 [ TripsTableInterpreter ]  Matching  header  with  provided  column  names
15 [ TripsTableInterpreter ]  Validating  6471  row (s)  according  to  the  column  types
16 [ TripsTableInterpreter ]  Validation  completed ,  the  resulting  table  has  6471  row (s )  and  10  column (s)
17 [ TripsTableInterpreter ]  Execution  duration :  2  ms .
18 [ TripsLoader ]  Opening  database  file  ./ gtfs - static - and - rt . sqlite
19 [ TripsLoader ]  Dropping  previous  table  " static_trips "  if  it  exists
20 [ TripsLoader ]  Creating  table  " static_trips "
21 [ TripsLoader ]  Inserting  6471  row (s )  into  table  " static_trips "
22 [ TripsLoader ]  The  data  was  successfully  loaded  into  the  database
23 [ TripsLoader ]  Execution  duration :  35  ms .
```

Listing 6.1 Exemplary execution output for section trips

To assess the performance of the implemented system, the following assertions are be verified:

1. All records from the raw source files which are provided by the endpoints listed in Table 6.1 have been saved to the database.
2. There are no extraneous records present in the sink database.

Assuming the conditions hold true for each dimension, it can be inferred that the pipeline has successfully processed the data. To facilitate easy verification, the GtfsDemonstrator provides dedicated methods for validating both the GTFS and GTFS-RT data. This involves comparing the raw data with the output generated by the pipeline, as described previously. Finally, the pipeline was

run once to generate the SQLite database, which served as the sink reference for verification purposes. At the same point in time, all the endpoints consumed by the pipeline were queried manually to obtain the raw data for comparison. By using this approach, both the raw files (GTFS and GTFS-RT) and the processed output of the pipeline are available, enabling the verification of assertions made earlier.

GTFS Validation

In the domain of the GtfsDemonstrator, the validation of GTFS involves several steps for each raw GTFS CSV file[3]:

1. Retrieval of the content of the raw CSV file, such as the agency dimension.
2. Retrieval of the content of the corresponding table from the SQLite sink, such as the static_agency table.
3. Comparison of the contents obtained in 1) and 2). If both contents are semantically equivalent, then both assertions are deemed to be fulfilled.

The logs generated during the validation process (Figure 6.1) provide evidence that the number of rows in each dimension of the raw data matches the number of rows in the corresponding sink, as well as semantic equivalence between both. These results serve to verify the correctness of the GTFS extension in handling the data. With that, we have demonstrated that our system loads static GTFS data to a relational database.

```
------------------started validation of GTFS------------------
```

(index)	dimension	#rows_src	#rows_sink	valid
0	'agency'	2	2	true
1	'calendar'	39	39	true
2	'calendar_dates'	164	164	true
3	'feed_info'	1	1	true
4	'routes'	65	65	true
5	'shapes'	80412	80412	true
6	'stop_times'	133464	133464	true
7	'stops'	1058	1058	true
8	'trips'	6471	6471	true

```
------------------finished validation of GTFS------------------
```

Figure 6.1: Validation console output for GTFS data

[3] Method validateGtfs() of GtfsDemonstrator: https://github.com/schlingling/jayve-gtfs-demonstrator

GTFS-RT Validation

The procedure for verifying the output of the GTFS-RT component operates as follows for each entity[4]:

1. Decoding the protobuf files acquired from the designated endpoint.
2. For each event contained within the entity, the validation process retrieves the row of the corresponding table from the SQLite sink utilizing a composite key comprised of the current event's attributes.
3. If the single row precisely matches the event, it is inferred that the raw record has been stored in the sink.

For instance, in the case of the entity trip, a distinct composite key can be constructed by combining the following columns:

- entity.id
- entity.vehicle_position.vehicle_descriptor.id
- entity.vehicle_position.trip.trip_id
- entity.vehicle_position.trip.route_id

The validation output logs (Figure 6.2) demonstrate that for each entity, the number of events in the raw data matches the number of rows in the sink, and the rows match each other. Therefore, it can be concluded that the GTFS-RT extension accurately processes the data.

```
------------------started validation of GTFS-RT------------------
```

(index)	entity	#events_src	#rows_sink	valid
0	'trip_update'	12168	12168	true
1	'alert'	123	123	true
2	'vehicle'	33	33	true

```
------------------finished validation of GTFS-RT-----------------
```

Figure 6.2: Validation console output for GTFS-RT data

Periodical Archival Mechanism

The GtfsDemonstrator proposes a methodology for executing a defined pipeline periodically. The pipeline is executed periodically and the output of each run is logged in a dedicated logging folder, while the increase in sink file size is tracked. Since the real-time endpoints offer a refresh rate of 30 seconds, the demonstrator is configured to operate at the same rate to not miss any updates. To observe the system's behavior and the increase in archived data over time, the demonstrator

[4] Method validateGtfsRT() of GtfsDemonstrator: https://github.com/schlingling/jayvee-gtfs-demonstrator

is run for a duration of 2 hours (Figure 6.3). After the first run, the database file has an initial size of 14.48 MB, where 13,7 MB are allocated by GTFS and 0.78 MB by GTFS-RT data, increasing by around 0.78 MB per execution as new real-time data is appended. It sums up to a total of 169.9 MB after 2 hours. We observe a nearly linear growth of data over time, with minor deviations that could be related to a fluctuation of streamed vehicle positions or service alerts.

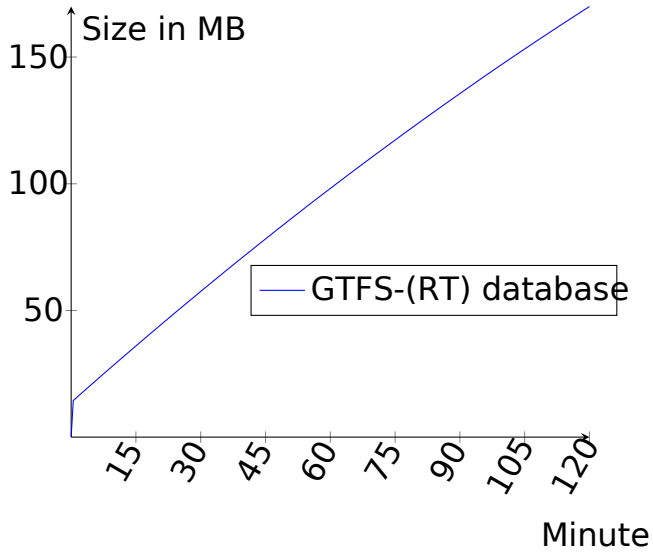


Figure 6.3: Increase of SQLite database file size over a period of 2 hours when archiving data periodically

In Table 6.2, we have extrapolated the estimated size of the SQLite file based on a one-year period. This extrapolation is performed by utilizing the observed growth rate in the demonstrator.

Period	Extrapolated SQLite file size in GB (precision of MB)
2 hours	0.17
1 day (24 hours)	2.0392
1 week (7 days)	14.274
1 month (30 days)	61.175
1 year (365 days)	744.293

Table 6.2: File size extrapolated from the growth rate observed in the demonstrator by different periods

Assuming a consistent growth rate throughout the year, the projected file size would be 744.29 GB. While this size may initially appear substantial, it is important to consider the context of the Brest region, which consists of over 1000 bus stops and captures real-time data at a granularity of 30 seconds over the

course of a year. When compared within this context, the file size becomes more relative. It is worth noting that in a real-world scenario, the growth rate is unlikely to remain strictly consistent due to various factors such as events, seasons, weekends, or holidays, which can influence the data growth.

By leveraging the capabilities of relational databases, optimization techniques such as indexing can be applied to rapidly retrieve data based on specific criteria, such as location, time, or service provider. Moreover, downstream processes can enforce data consistency and integrity via relationships and constraints between tables, ensuring accurate and complete GTFS data, which is critical for scheduling and planning transit services. This implementation is especially beneficial as it enables "everyone as", stated by Professorship of Open Source Software at the University of Erlangen (2022), to handle open transport data using common relational database operations instead of raw CSV or protobuf encoded files. These insights gained through relational data can then be utilized to optimize transit services, improve route planning, and enhance the passenger experience.

One potential drawback of using the provided implementation to archive data is the substantial amount of data that is stored, particularly when handling real-time updates. This considerable volume of data is primarily attributed to two factors:

1. The real-time data is stored in plain text format in the sink
2. The real-time data is flattened into a relational data model, leading to a large number of rows with redundant data

Multiple approaches exist to mitigate the aforementioned drawback, each of which involves a trade-off between reducing storage requirements for archiving and increasing computational power for analysis, or vice versa. One potential solution is to store real-time data in an encoded format rather than plain text, which would require an additional decoding stage prior to analysis. Depending on the use case, another approach is to keep relatively recent data in the relational format discussed but handle long term archiving with encoded files. Alternatively, normalization of real-time data can be leveraged either during the load process into the sink or in a downstream computation step. However, these approaches may introduce additional complexity into the pipeline and its underlying implementation.

6.2 Functional Requirements

In the context of this thesis, a User Story is considered accepted only when it satisfies the agreed upon UAC by both the team and the product owner, who in this case is the JValue team. To provide an overview of the functional complete-

ness of the discussed implementation, we have created two tables, Table 6.3 and Table 6.4. These tables map the UAC defined in Section 3.2 and Section 3.3 to their respective statuses after the completion of the implementation. A status shown in parentheses indicates that the corresponding UAC became obsolete due to parallel progress made in the project. In the digital version of this paper, the column labeled PR contains a hyperlink to the relevant details on GitHub[5]. Finally, the comment column in the tables provides additional context regarding the status.

No.	Name	Accepted	PR	Comment
UAC-1	Implement io-type File	✓	125, 256	Approach described in Chapter 5.1
UAC-2	Implement io-type FileSystem	✓	126, 256	Approach described in Chapter 5.1
UAC-3	Implement io-type None	✓	126	Approach described in Chapter 5.1
UAC-4	Extend io-type Table	(✓)	165	Has already been implemented
UAC-5	Process table name in LayoutValidator	(✓)	164	Has already been implemented
UAC-6	Refactor Jayvee-examples using table name	✓	166	Approach described in Chapter 5.1
UAC-7	Abort execution	✓	136	Approach described in Chapter 5.1
UAC-8	Introduce Folderstructure	✓	126, 256	Approach described in Chapter 5.1
UAC-9	Implement HTTPExtractor	✓	134	Approach described in Chapter 5.1
UAC-10	ImplementArchiveInterpreter	✓	135	Approach described in Chapter 5.1
UAC-11	Implement FilePicker	✓	136	Approach described in Chapter 5.1
UAC-12	Implement CSVInterpreter	✓	168	Approach described in Chapter 5.1
UAC-13	Refactor CSVFileExtractor	✓	169	Approach described in Chapter 5.1
UAC-14	Refactor Jayvee-examples to new blocks	✓	169	Validated with demonstrator
UAC-15	Conditional GTFS-columns	(✓)	85	Covered by assumption
UAC-16	Multiple block inputs	(✓)	na	Has already been implemented
UAC-17	Process table name in SQLiteSink	(✓)	164	Has already been implemented
UAC-18	Database creation	(✓)	na	Has already been implemented
UAC-19	Parallel processing	(✓)	na	Has already been implemented
UAC-20	Sucessful execution	✓	180	Validated with demonstrator

Table 6.3: User Acceptance criteria of GTFS User Story[6] by their acceptance status and corresponding Pull Request in GitHub

No.	Name	Accepted	PR	Comment
UAC-1	Implement GtfsRTInterpreter	✓	223	Approach described in Chapter 5.2
UAC-1.1	Define simple GTFS-RT pipeline	✓	223	Approach described in Chapter 5.2
UAC-2	Implement DropTable attribute	✓	254	Approach described in Chapter 5.2
UAC-3	Showcase GTFS and GTFS-RT data processing	✓	255	Validated with demonstrator
UAC-4	Create/Update SQLite file	(✓)	255	Validated with demonstrator
UAC-5	Overwrite GTFS data	(✓)	255	Validated with demonstrator
UAC-6	Append GTFS-RT data	✓	255	Validated with demonstrator
UAC-7	Sucessfull execution	✓	255	Validated with demonstrator

Table 6.4: User Acceptance criteria of GTFS-RT User Story[7] by their acceptance status and corresponding Pull Request in GitHub

[5] The details can be accessed manually using the URL pattern https://github.com/jvalue/jayvee/pull/<PR>

[6] GTFS User Story on GitHub https://github.com/jvalue/jayvee/pull/123

[7] GTFS-RT User Story on GitHub https://github.com/jvalue/jayvee/pull/219

Moreover, the complete integration of both GTFS support and GTFS-RT support RFCs into the project's main branch, following the requirement definition process outlined in Section 3.1, signifies the acceptance of the associated User Stories by the JValue community. The satisfaction of all UAC demonstrates that both User Stories have been accepted, confirming their functional suitability.

6.3 Non-Functional Requirements

As we stated in Section 3.4 both extensions should satisfy the same non-functional requirements. This section validates the fulfillment of the three prior defined ones.

NFR-1 Using Compositions

The functional requirement is fully met by the GTFS component, as it has been broken down into various block types, such as HttpExtractor, ArchiveInterpreter, and FilePicker. These block types have already been utilized by other pipelines in the project.

However, the GTFS-RT extension requires the utilization of a domain-specific block type, GtfsRTInterpreter, which fails to satisfy the non-functional requirement. This decision was necessitated by the current limitation of Jayvee to process only tabular data structures. At the time of implementation, document-oriented data structures like JSON files were not supported by the language's features, and adopting them would have necessitated significant changes to the language that would have violated NFR-2.

NFR-2 Seamless Integration

Both implementations integrate seamlessly into Jayvee without necessitating any significant logical changes to the language's grammar.

NFR-3 High Execution Performance

Both implementations have demonstrated effective performance, as evidenced by the example pipeline consisting of 60 blocks and processing approximately 260,000 rows per run. The average execution time of the pipeline, based on 10 runs conducted on an Apple M1-Pro machine, is determined to be 9194ms.

6.4 Limitations

However, it must be noted that the current implementation is merely a proof of concept, and as such, there exist several limitations that need to be addressed in the future:

1. The current implementation treats conditional optional fields within the table as mandatory, lacking a mechanism to handle them dynamically at runtime. This limitation arises from Jayvee's inability to handle optional values. Consequently, in our schema definitions, we have assumed that conditional optional fields defined by the GTFS reference are required.

2. The system does not support optional tables, necessitating the creation of specific pipeline definitions for each endpoint.

3. The example pipeline was designed to treat all fields as text, resulting in a decrease in data quality during the ETL process, as numeric and date data types are converted into text format. It is important to note that this limitation is specific to the defined pipeline and does not reflect any inherent issues with the implementation itself. This simplification was implemented to prevent potential parsing errors during execution. In essence, Jayvee's interpreter would discard rows containing parsing errors.

4. The periodic archival mechanism employed for static data, as demonstrated, discards the previous version of the static dataset before loading the updated one. This can introduce inconsistencies in the data set, as real-time data may reference dropped static entities. To address this issue, Jayvee requires a history mechanism to track and flag outdated data.

5. The implementation utilized a tabular sink, but alternative sink types should be explored for improved performance, such as a flat file sink (which needs to be implemented).

6. Due to the loading of data into a tabular sink, the flattening of nested event objects in real-time data resulted in a significant increase in the number of rows stored in the sink. To mitigate this issue, a normalization step should be introduced or different sink types should be evaluated.

7. The GtfsRTInterpreter currently functions as a black box. It would be beneficial to decompose its functionality into a composition of blocks, such as a JsonInterpreter, to enhance modularity and flexibility. However, supporting these types of data would require major changes to the Jayvee grammar, as they are not yet supported.

In the long run, it is desirable to offer users a generic schema for both GTFS and GTFS-RT, which would be accessible and allow for data extraction from any GTFS endpoint, regardless of the specific tables provided. Such an approach would greatly enhance usability, particularly for automated data archiving from diverse endpoints. The JValue team is currently exploring the concept of "composite blocks", which would enable users to combine multiple existing blocks to create a new block type. This could also be employed to streamline existing extensive pipelines. Nonetheless, these advanced concepts proposed can be de-

veloped based on the groundwork laid by the current implementation.

7 Conclusion

Open Transport Data enables innovation by providing abundant information for developers, researchers, urban planners, and entrepreneurs to create applications, services, and business models. However, the lack of specific guidelines for open data has resulted in the proliferation of proprietary data platforms. Standardized references like GTFS and GTFS-RT facilitate the sharing of public transit information, but processing and archiving this data can be challenging. This thesis introduces an extension to the open source domain specific language for data pipelining, namely Jayvee, to process and archive GTFS(-RT) data.

We outline the requirement definition phase, where we extract functional requirements using the research project's RFC process. We incrementally carry out the architecture, design, and implementation phases, including GTFS static support, GTFS-RT support, and a concept for the periodic archival mechanism. For each extension, we provide a high-level architecture of the resulting pipeline's components, which are then implemented according to the defined UACs. To ensure a valid evaluation basis, we introduce a demonstrator that validates the system's correct behavior. To this end, we design an example pipeline that processes static and real-time GTFS data from a real-world GTFS scenario using the French metropolitan region of Brest. The demonstrator confirms the expected functionality of the implementation and successfully archives static and real-time data for a time-boxed experiment period, allowing us to validate the data growth efficiency. Finally, we highlight aspects for future work to enhance the implementation's user-friendliness and efficiency, such as handling optional fields and tables, providing generic GTFS schemata out of the box, and introducing a concept for block compositions.

Our approach not only facilitates the seamless integration of static and real-time GTFS data but also serves as a valuable guide for the open transport data research community in extending open-source software like Jayvee. By leveraging the wealth of information embedded within GTFS datasets, we contribute to the broader goal of reducing barriers and fulfilling the mission of making the use of open data easy, safe, and reliable, as stated by the Professorship of Open Source Software at the University of Erlangen (2022).

Appendices

A RFC Document for GTFS Support (RFC-0002)

<!--
SPDX-FileCopyrightText: 2023 Friedrich-Alexander-Universitat Erlangen-Nurnberg

SPDX-License-Identifier: AGPL-3.0-only
-->

Feature Tag mobility-extension

Status ACCEPTED <!-- Possible values: DRAFT, DISCUSSION, ACCEPTED, REJECTED -->

Responsible @schlingling
Implemented via #123 (https://github.com/jvalue/jayvee/issues/123)

Disclaimer: This RFC is part of my master-thesis "Archiving open transport data using the JValue tooling ecosystem" supervised by @rhazn.

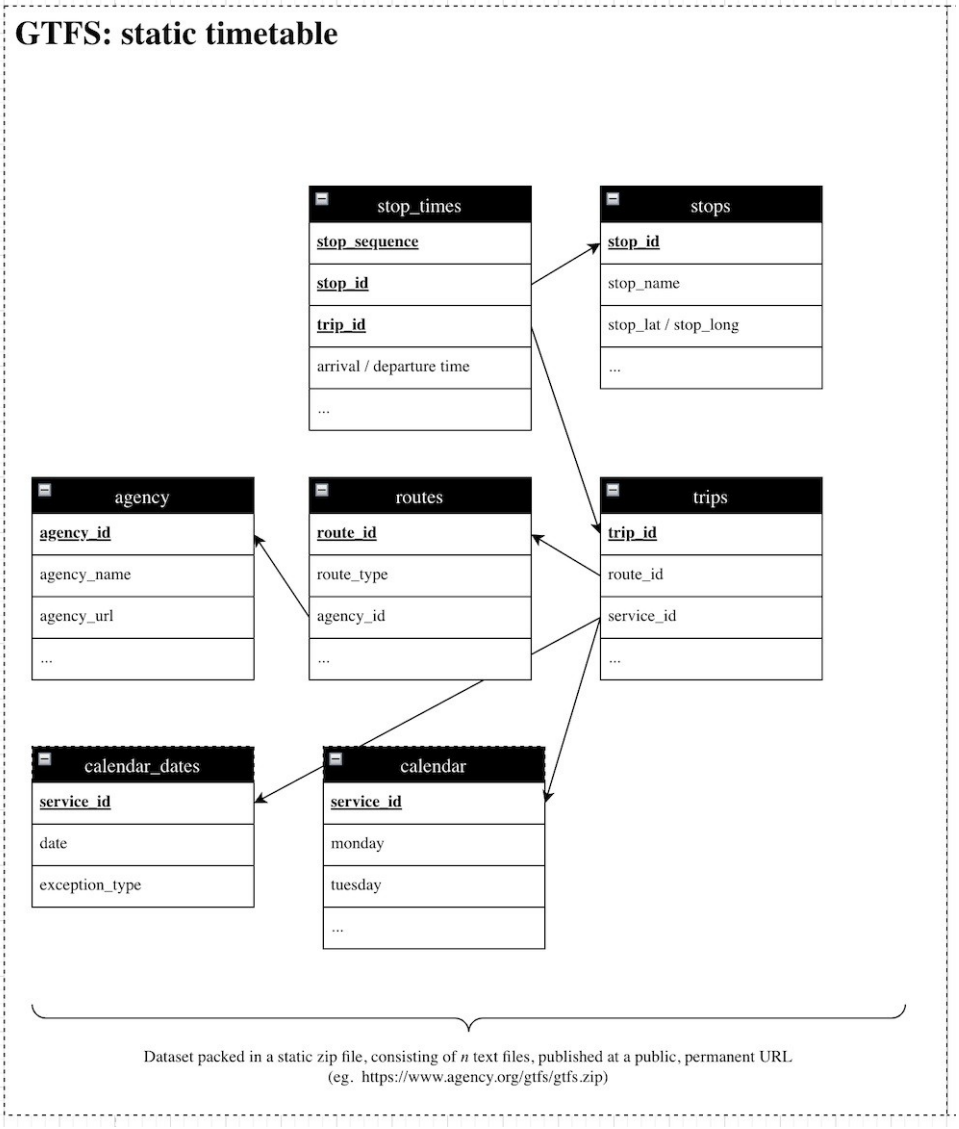
Summary

This RFC enables a pipeline extracting, validating and loading GTFS-data (part of domain mobility-data) by providing an GTFS-endpoint under consideration of the GTFS-specification (https://developers.google.com/transit/gtfs/reference). For that reason some changes and extensions of Jayvee have to be made. The overall goal of this RFC is processing GTFS-data with a minimum of changes/extensions in Jayvee.

Motivation

GTFS has gained widespread popularity over the past decade as an open-source industry standard for describing and publishing fixed- and dynamic route transit operations. It is a data standard that defines how public transit agencies can provide schedule information to developers. It is used by agencies around the world to publish their transit data in a common format, allowing developers to create applications that can access and use this data. GTFS data includes information about stops, routes, and schedules for buses, trains, and other forms of public transportation. GTFS-data is provided by an endpoint, which publishes a zip-

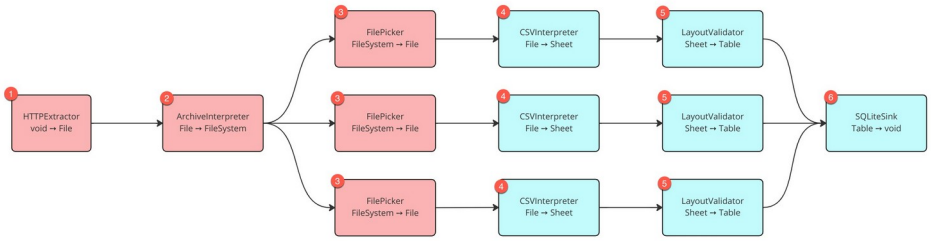
file, consisting of a collection of comma-separated-values in plain text files. A example of a gtfs-zip-file could result in this datamodel (this visualization just includes required dimensions).



Explanation

The following picture is a visualization of the corresponding example.jv (example.jv)-file. A GTFS-pipeline follows the overall pipeline pattern, consisting of an Extractor, some Interpreters, some Validators and finally a Loader to a sink (in our case all gtfs-csv-files are loaded into a SQLite database, each csv-file into its corresponding table). The individual GTFS files are picked out using their filename and further processed independently in parallel using block types that already exist (or at least in a similar form). In the image, there

are three such parallel processing steps as an example. In practice, there would be one for each GTFS file in the ZIP archive. In case a file are not present, the further processing of that file is aborted and hence no table is created from that file. At the end, each successfully created table is loaded into the same SQLiteSink.



The red block types need to be created from scratch whereas the blue block types are either already present or only require minor changes (this classification is also reflected in the following chapter titles).

Jayvee needs to be extented by following parts to be able to process GTFS-data:

- New io-datatypes called File and FileSystem
- New blocktypes HTTPExtractor, ArchiveInterpreter and FilePicker
- The io-datatype Table needs to store its name to be able to handle multiple tables as input
- An abort-mechanism must be implemented, for that we need a new io-type None to abort, if a precessor outputs None

Each of the following subchapters explains the idea behind.

New io-datatypes

io-datatype File *(Requires implementation from scratch)*

A File datatype could look like this and should be added to io-datatypes.ts.

```
export interface File {
  name: string // The name of the file, excluding its file extension

  extension: string //The file extension

  filetype: string //The MIME type of the file taken from the Content-Type header (for HTTP requests only) Otherwise inferred from the file extension, default application/octet-stream for unknown or missing file extensions

  content: ArrayBuffer //The content of the file as a ArrayBuffer
}
```

io-datatype FileSystem *(Requires implementation from scratch)*

A FileSystem could look like this and should be added to io-datatypes.ts. Provides generic methods for navigating in the file system using paths and for accessing files. In order to implement the interface, we create a class which provides the attributes / methods demanded by the interface.

```
export interface FileSystem {
   //Methods as needed
}
```

io-datatype None *(Requires implementation from scratch)*

A None type could look like this and should be added to io-datatypes.ts. If a block output emits a None value, downstream blocks are not executed for that value.

```
export interface None {
   //Methods as needed
}
```

io-datatype Table *(Requires minor change)*

The io-datatype Table should be adapted, to store its name to be able to handle multiple tables as input later in an DB-Loader. This leads also to a minor change in the LayoutValidator and the example to process the additional attribute tableName.

```
export interface Table {
  tableName: string;
  columnNames: string[];
  columnTypes: Array<AbstractDataType | undefined>;
  data: string[][];
}
```

Change of folderstructure

Since we are introducing multiple new io-datatypes and some implemenations of them, we move the file io-datatype.ts to a new folder, holding all types and implemenations.

New Block Types

1) HttpExtractor *(Requires implementation from scratch)*

Input: void, Output: File

A HttpExtractors gets an Url, sends an HTTP-GET-REQUEST to that URL and outputs the response as File. This block can be used for getting any kind of data via a HTTP-Endpoint. It should be implemented in the std-extension.

```
block MyHttpExtractor oftype HttpExtractor {
   url: "https://www.data.gouv.fr/fr/datasets/r/c4d9326f-9f41-4dfb-9746-31bc97a31fc6";
}
```

2) ArchiveInterpreter *(Requires implementation from scratch)*

Input: File, Output: FileSystem

A ArchiveInterpreter gets a File, and initializes an FileSystem ontop of the file (open filestream etc.). Provides generic methods for navigating in the file system using paths and for accessing files. As it is not clear, what the file contains. It should be implemented in the std-extension. The ArchiveInterpreter needs to be able to instantiate a FileSystem instance in order to output it as a result.

```
block ZipArchiveInterpreter oftype ArchiveInterpreter{
    archiveType: string //now only accepting the string "zip"
}
```

3) FilePicker *(Requires implementation from scratch)*

Input: FileSystem, Output: File

A FilePicker gets an FileSystem, navigates to the file, and initializes an file via the path. The FilePicker needs to work with methods provided by a FileSystem instance in order to read the file according to the provided path.

```
block MyFilePicker oftype FilePicker{
    path: string // Absolute path to file (from the root folder) eg. /agency.txt
}
```

4) CSVInterpreter *(Requires minor change)*

Input: File, Output: Sheet

In the package tabular a CSVFileExtractor is already implemented, which loads a CSV from an URL and outputs a Sheet. We need to rewrite the existing example pipelines (gas and cars), to use the new HTTPExtractor as well, then we just need one CSVFileInterpreter and now longer an CSVFileExtractor.

5) LayoutValidator and Layouts *(Requires minor change)*

The following description is out of scope for this RFC, will be considered in future but is important for understanding the gtfs-specification:

- Some columns in GTFS-csv-files are optional and conditional optional
- For an implementation of an optional mechanism, we need to present the optional columns with their datatype, e.g. saying their datatype is text or undefined.
- So, we'd need an undefined datatype in Jayvee and a way to combine these types using an or-expression (see chapter datatypes).

For a first draft, we only consider required columns and reuse the existing language features for that.

```
layout agencyLayout {
        header row 1: text;
        column agency_id: text; //Conditional columns are considered as required
        column agency_name: text;
        column agency_url: text;
        column agency_timezone: text;
    }
```

A vision is, that the GTFS-pipeline later on processes a list of GTFS-Endpoints. Because every endpoint has at least the required-columns, we need to have the optional-mechanism in our layout.

In a GTFS-Validator, some conditional (aka logical) checks could possibly be applied during the validation (not just a static header/datatype validaton). This could be done by checking required columns. If in future an conditional required is not longer considered as required, we could also implement conditional logical validation. An example for that is the columns agency_id in table agencies. The specification states, that the agency_id is optional, when the whole dataset just contains data from one agency.

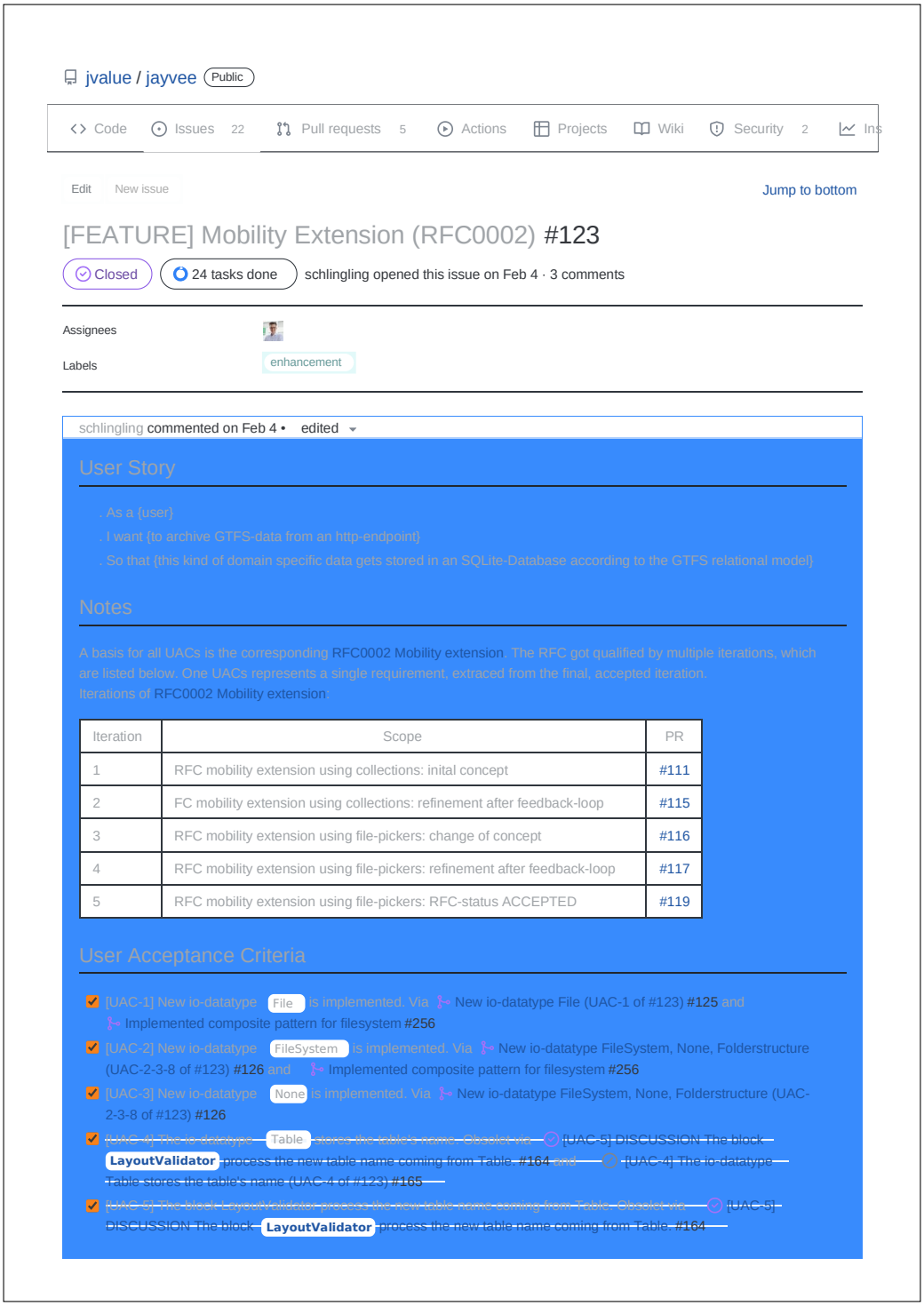
IMPORTANT: In the GTFS specifiation, the order of the columns is not defined, so we need to access the columns by their names, not their index as every GTFS-endpoint could possibly have a different order!!

6) SQLiteSink *(Requires minor change)*

Input: Table, Output: void
This Block needs to be adapted, to handle multiple Inputs. As the parallel processing of the differnt files does not depend on each other, for an inital demonstration if the PoC we use each own SQLiteSinks without the effort of modifying the execution logic (here we have to change the SQLite sink, to not recreate the DB each call). We also change the SQLiteSink to receive the table name via the io-datatype Table itself.

```
block GtfsLoader oftype SQLiteTablesLoader {
        file: "./gtfs.db";
    }
```

B GitHub Issue for RFC-0002 GTFS Support



jvalue / jayvee Public

Code Issues 22 Pull requests 5 Actions Projects Wiki Security 2 Ins

Edit New issue Jump to bottom

[FEATURE] Mobility Extension (RFC0002) #123

Closed 24 tasks done schlingling opened this issue on Feb 4 · 3 comments

Assignees

Labels enhancement

schlingling commented on Feb 4 • edited

User Story

. As a {user}
. I want {to archive GTFS-data from an http-endpoint}
. So that {this kind of domain specific data gets stored in an SQLite-Database according to the GTFS relational model}

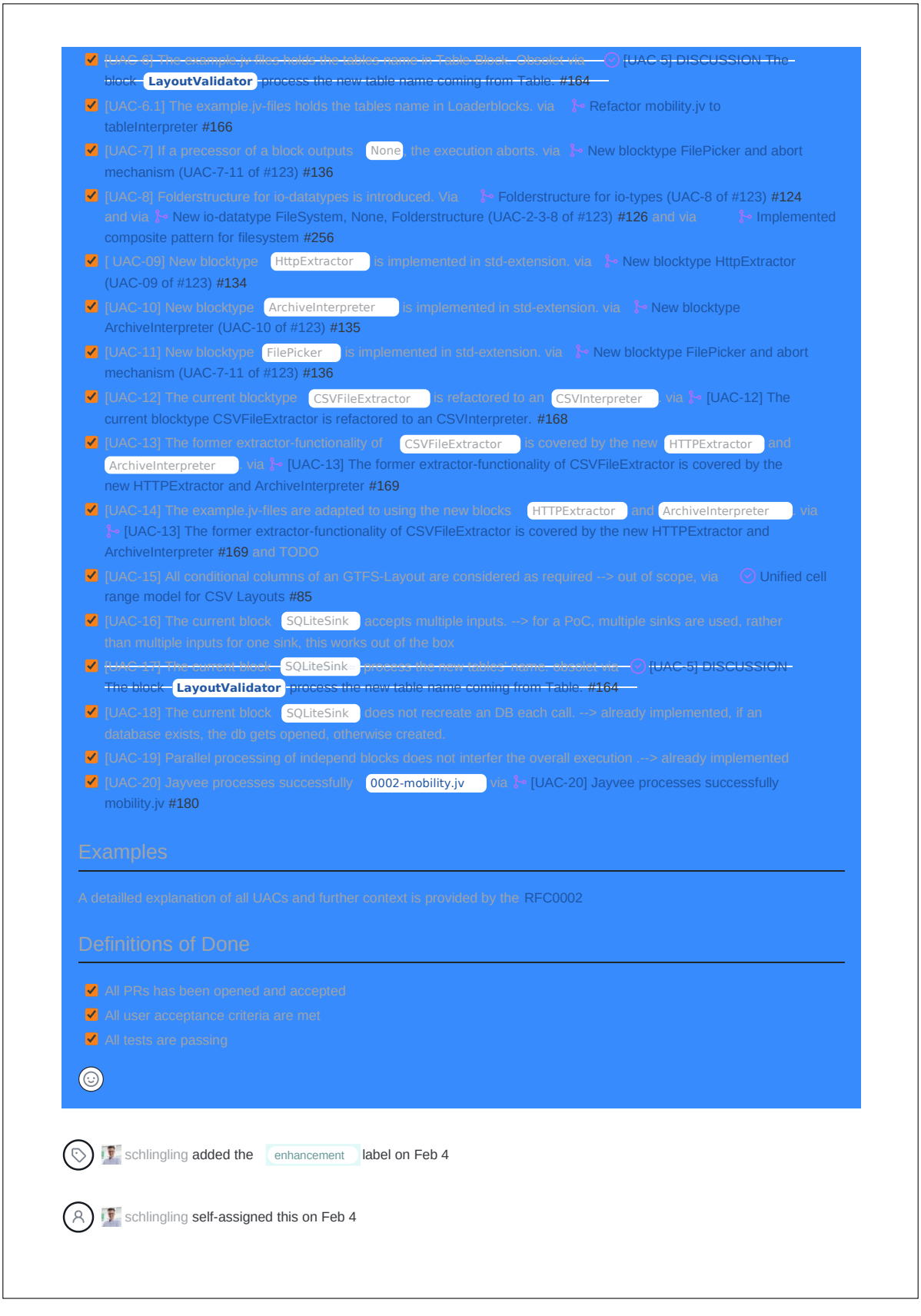
Notes

A basis for all UACs is the corresponding RFC0002 Mobility extension. The RFC got qualified by multiple iterations, which are listed below. One UACs represents a single requirement, extraced from the final, accepted iteration.
Iterations of RFC0002 Mobility extension

Iteration	Scope	PR
1	RFC mobility extension using collections: inital concept	#111
2	FC mobility extension using collections: refinement after feedback-loop	#115
3	RFC mobility extension using file-pickers: change of concept	#116
4	RFC mobility extension using file-pickers: refinement after feedback-loop	#117
5	RFC mobility extension using file-pickers: RFC-status ACCEPTED	#119

User Acceptance Criteria

- [x] [UAC-1] New io-datatype File is implemented. Via New io-datatype File (UAC-1 of #123) #125 and Implemented composite pattern for filesystem #256
- [x] [UAC-2] New io-datatype FileSystem is implemented. Via New io-datatype FileSystem, None, Folderstructure (UAC-2-3-8 of #123) #126 and Implemented composite pattern for filesystem #256
- [x] [UAC-3] New io-datatype None is implemented. Via New io-datatype FileSystem, None, Folderstructure (UAC-2-3-8 of #123) #126
- [x] [UAC-4] The io-datatype Table stores the table's name. Blocked via [UAC-5] DISCUSSION The block LayoutValidator process the new table name coming from Table. #164 and [UAC-4] The io-datatype Table stores the table's name (UAC-4 of #123) #165
- [x] [UAC-5] The block LayoutValidator process the new table name coming from Table. Closed via [UAC-5] DISCUSSION The block LayoutValidator process the new table name coming from Table. #164



- [x] ~~[UAC-6] The example.jv-files holds the tables name in Table Block. Obsolet via [UAC-5] DISCUSSION The block LayoutValidator process the new table name coming from Table. #164~~
- [x] [UAC-6.1] The example.jv-files holds the tables name in Loaderblocks. via Refactor mobility.jv to tableInterpreter #166
- [x] [UAC-7] If a precessor of a block outputs None the execution aborts. via New blocktype FilePicker and abort mechanism (UAC-7-11 of #123) #136
- [x] [UAC-8] Folderstructure for io-datatypes is introduced. Via Folderstructure for io-types (UAC-8 of #123) #124 and via New io-datatype FileSystem, None, Folderstructure (UAC-2-3-8 of #123) #126 and via Implemented composite pattern for filesystem #256
- [x] [UAC-09] New blocktype HttpExtractor is implemented in std-extension. via New blocktype HttpExtractor (UAC-09 of #123) #134
- [x] [UAC-10] New blocktype ArchiveInterpreter is implemented in std-extension. via New blocktype ArchiveInterpreter (UAC-10 of #123) #135
- [x] [UAC-11] New blocktype FilePicker is implemented in std-extension. via New blocktype FilePicker and abort mechanism (UAC-7-11 of #123) #136
- [x] [UAC-12] The current blocktype CSVFileExtractor is refactored to an CSVInterpreter via [UAC-12] The current blocktype CSVFileExtractor is refactored to an CSVInterpreter. #168
- [x] [UAC-13] The former extractor-functionality of CSVFileExtractor is covered by the new HTTPExtractor and ArchiveInterpreter via [UAC-13] The former extractor-functionality of CSVFileExtractor is covered by the new HTTPExtractor and ArchiveInterpreter #169
- [x] [UAC-14] The example.jv-files are adapted to using the new blocks HTTPExtractor and ArchiveInterpreter via [UAC-13] The former extractor-functionality of CSVFileExtractor is covered by the new HTTPExtractor and ArchiveInterpreter #169 and TODO
- [x] [UAC-15] All conditional columns of an GTFS-Layout are considered as required --> out of scope, via Unified cell range model for CSV Layouts #85
- [x] [UAC-16] The current block SQLiteSink accepts multiple inputs. --> for a PoC, multiple sinks are used, rather than multiple inputs for one sink, this works out of the box
- [x] ~~[UAC-17] The current block SQLiteSink process the new tables name. obsolet via [UAC-5] DISCUSSION The block LayoutValidator proceiss the new table name coming from Table. #164~~
- [x] [UAC-18] The current block SQLiteSink does not recreate an DB each call. --> already implemented, if an database exists, the db gets opened, otherwise created.
- [x] [UAC-19] Parallel processing of independ blocks does not interfer the overall execution .--> already implemented
- [x] [UAC-20] Jayvee processes successfully 0002-mobility.jv via [UAC-20] Jayvee processes successfully mobility.jv #180

Examples

A detailled explanation of all UACs and further context is provided by the RFC0002

Definitions of Done

- [x] All PRs has been opened and accepted
- [x] All user acceptance criteria are met
- [x] All tests are passing

schlingling added the enhancement label on Feb 4

schlingling self-assigned this on Feb 4

schlingling changed the title [~~DRAFT] [FEATURE] Mobility Extension (coming from RFC0002) [FEA~~TURE] Mobility Extension (RFC0002) on Feb 4

schlingling commented on Feb 4 • edited

@rhazn May you have a short look, if this issue fits your expectation? Then i would start the implementation.

rhazn commented on Feb 4

Sounds good , also FYI @felix-oq

1 1

schlingling mentioned this issue on Feb 5

Folderstructure for io-types (UAC-8 of #123) #124

Merged

1 task

schlingling linked a pull request on Feb 5 that will close this issue

Folderstructure for io-types (UAC-8 of #123) #124 Merged

1 task

schlingling removed a link to a pull request on Feb 5

Folderstructure for io-types (UAC-8 of #123) #124 Merged

1 task

schlingling mentioned this issue on Feb 5

New io-datatype File (UAC-1 of #123) #125

Merged

1 task

schlingling added a commit that referenced this issue on Feb 5

Merge pull request #124 from jvalue/feat-mobility-extension-uac-8 ··· b307423

schlingling added a commit that referenced this issue on Feb 5

Merge pull request #125 from jvalue/feat-mobility-extension-uac-1 ··· 1706112

schlingling mentioned this issue on Feb 5

New io-datatype FileSystem, None, Folderstructure (UAC-2-3-8 of #123) #126

Merged

3 tasks

schlingling added a commit that referenced this issue on Feb 9

Merge pull request #126 from jvalue/feat-mobility-extension-uac-2-fil... ··· f418f69

This was referenced on Feb 12

New blocktype HttpExtractor (UAC-09 of #123) #134

Merged

New blocktype ArchiveInterpreter (UAC-10 of #123) #135

Merged

New blocktype FilePicker and abort mechanism (UAC-7-11 of #123) #136

Merged

schlingling added a commit that referenced this issue on Feb 13

Merge pull request #135 from jvalue/feat-archive-interpreter-uac-10 ··· 0b40c01

schlingling added a commit that referenced this issue on Feb 13

Merge pull request #134 from jvalue/feat-mobility-ext-uac-9 ··· ead1df5

This was referenced on Feb 18

[UAC-5] DISCUSSION The block **LayoutValidator** process the new table name coming from Table. #164

Closed

[UAC-4] The io-datatype Table stores the table's name (UAC-4 of #123) #165

Closed

Refactor mobility.jv to tableInterpreter #166

Merged

closed #167

Closed

[UAC-12] The current blocktype CSVFileExtractor is refactored to an CSVInterpreter. #168

Merged

[UAC-13] The former extractor-functionality of CSVFileExtractor is covered by the new HTTPExtractor and ArchiveInterpreter #169

Merged

schlingling added a commit that referenced this issue on Feb 20

Merge pull request #136 from jvalue/feat-mobility-ext-uac-11 ··· 76febcf

schlingling mentioned this issue on Feb 25

[UAC-20] Jayvee processes successfully mobility.jv #180

Merged

1 task

schlingling commented on Feb 26

Successfully extended Jayvee by GTFS-Support as proposed in RFC0002. See #180 for an detailed perspective on the result. RFC0002 is therefore fully implemented.

FYI @georg-schwarz @felix-oq @rhazn

schlingling closed this as completed on Feb 26

This was referenced on Feb 26

RFC for gtfs extension #111

Closed

RFC for mobility extension #115

Merged

This was referenced on Feb 26

RFC 0002 for mobility extension #116

Merged

RFC 0002 for mobility extension #117

Merged

RFC 0002 for mobility extension #119

Merged

schlingling mentioned this issue 3 weeks ago

Implemented composite pattern for filesystem #256

Merged

Assignees

schlingling

Labels

enhancement

Projects

None yet

Milestone

No milestone

Appendix B: GitHub Issue for RFC-0002 GTFS Support

Development

Create a branch for this issue or link a pull request.

2 participants

Pin issue

C RFC Document for GTFS-RT Support (RFC-0006)

<!--
SPDX-FileCopyrightText: 2023 Friedrich-Alexander-Universitat Erlangen-Nurnberg

SPDX-License-Identifier: AGPL-3.0-only
-->

RFC 0006: GTFS-RT Support

Feature Tag gtfs-rt-support

Status ACCEPTED <!-- Possible values: DRAFT, DISCUSSION, ACCEPTED, REJECTED -->

Responsible @schlingling
Implemented via #219 (https://github.com/jvalue/jayvee/issues/219)

<!--
Status Overview:

- DRAFT: The RFC is not ready for a review and currently under change. Feel free to already ask for feedback on the structure and contents at this stage.
- DISCUSSION: The RFC is open for discussion. Usually, we open a PR to trigger discussions.
- ACCEPTED: The RFC was accepted. Create issues to prepare implementation of the RFC.
- REJECTED: The RFC was rejected. If another revision emerges, switch to status DRAFT.
  -->
  Disclaimer: This RFC is part of my master-thesis "Archiving open transport data using the JValue tooling ecosystem" supervised by @rhazn.

Summary

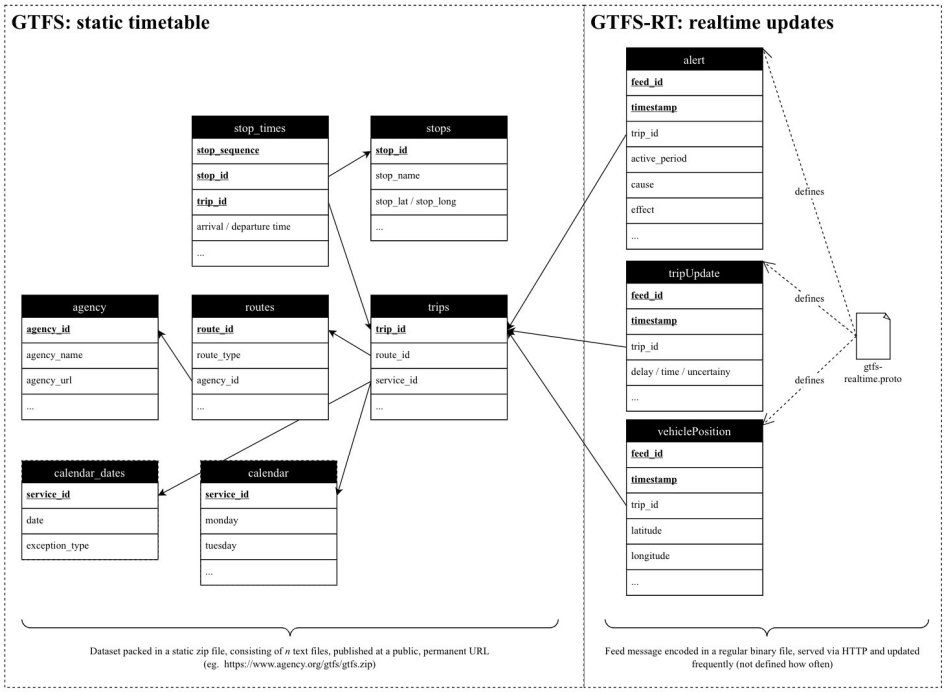
Introduces support for GTFS-RT (https://developers.google.com/transit/gtfs-realtime) (realtime) endpoints and extends therefore functionality of 0002-mobility-extension (https://github.com/jvalue/jayvee/tree/main/rfc/0002-mobility-extension). With this RFC, Jayvee can then process pipelines, which are extracting static public transportation schedules and associated geographic information and on top realtime updates about associated fleets like delays, cancellations, vehicle positions, etc.

Motivation

When it comes to nearly realtime updates, Google introduced an additional specification GTFS-RT on top of GTFS. This specification provides real-time up-dates to transit schedules and locations. It allows developers to access real-time
information about the location and status of vehicles, as well as any disruptions
or delays in service. GTFS-RT data is typically provided in
shape of streaming data feeds that are updated in real-time as events occur.
This realtime-feed always needs its corresponding static feed, which defines the
schedule and dimensions like agency.txt or routes.txt around live updates. The realtime specification can be divided into three types
of additional information, which enriches the static GTFS-feed:

- Trip updates - cancellations, delays and changed routes

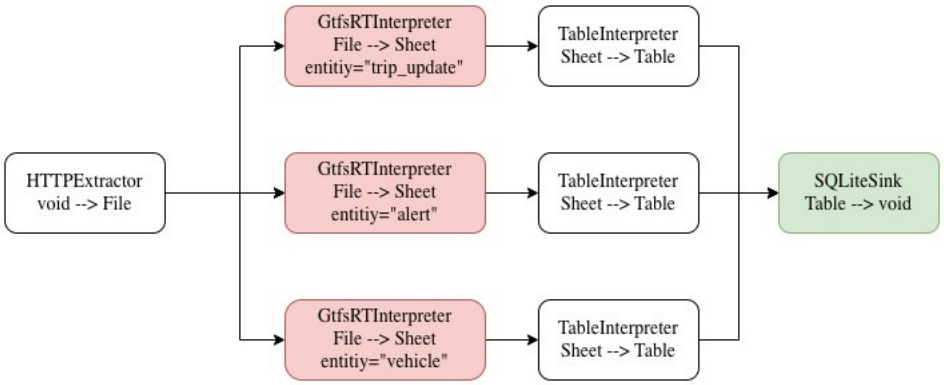
- Service alerts - unforeseen events with impact on the transportation network
- Vehicle positions - realtime information on vehicles position in coordinates



Explanation

In contrast to static GTFS, which only changes manually, when new schedules are released, realtime feeds require a frequent update rate (in the range of seconds), since live locations are played out. For this reason, it is specified that GTFS-RT is streamed using the protocol buffer format, which corresponds to a very efficient binary representation of the data. As a result, consuming and processing a GTFS-RT-feed needs an additional encoding stage to convert the messages to human readable plain text. The gtfs-realtime.proto textfile is used for parsing the protocol buffer into an JSON-like representation.

A simple GTFS-RT-Pipeline is shown in the picture below.

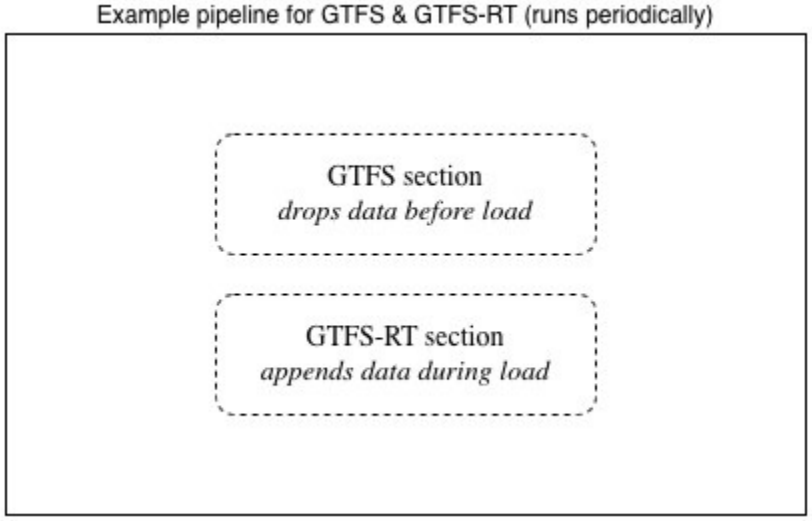


The red block types need to be created from scratch whereas the green block types are either already present or only require minor changes (this classification is also reflected in the following chapter titles).

Since realtime feeds require a frequent update rate, we enable a periodical load from GTFS **and** GTFS-RT data by following concept already discussed with @rhazn:

1. One pipeline containing both, GTFS and GTFS-RT sections.
2. Periodical execution of the pipeline.
3. An additional attribute for SQLite-sink indicates whether tables should be dropped before load starts. GTFS-tables are dropped every run, GTFS-RT-tables not, which leads to the dataset visualized above in the Output-SQLite-Database containing the static GTFS-information as well as the incrementally growing RT-tables.

This concept results in following pipeline.



Block Types

1) GtfsRTInterpreter *(Requires implementation from scratch)*

Input: File, Output: Sheet

A GtfsRTInterpreter gets an entity ("vehicle", "trip_update" or "alert") to process from the incoming protobuf-file, decodes the protobuf-file and outputs the entity as a sheet. In a first step, just required columns (defined in gtfs-realtime.proto) are considered. As we dont have a dedicated mobility-extension folder in Jayvee this should be implemented in the std-extension (already discussed with @rhazn).

```
block MyGtfsRTInterpreter oftype GtfsRTInterpreter {
    entity: "vehiclePosition"; // TEXT: "vehiclePosition", "tripUpdate" or "alert"
}
```

2) SQLiteSink *(Requires minor change)*

The SQLite sink needs an additional attribute indicating whether table should be dropped before load starts. Since blocks currently just support one input, a boolean data type for dropping a table is enough.

```
block VehicleLoader oftype SQLiteLoader {
        file: "./gtfs.db";
    dropTable: false // BOOLEAN
}
```

Drawbacks

The proposed concept is functional, but it could be more efficient if it would not involve dropping static data with each run. However, since addressing this issue would make the RFC more complex, we have decided to implement this optimization at a later stage, and focus on creating a first proof of concept for now.

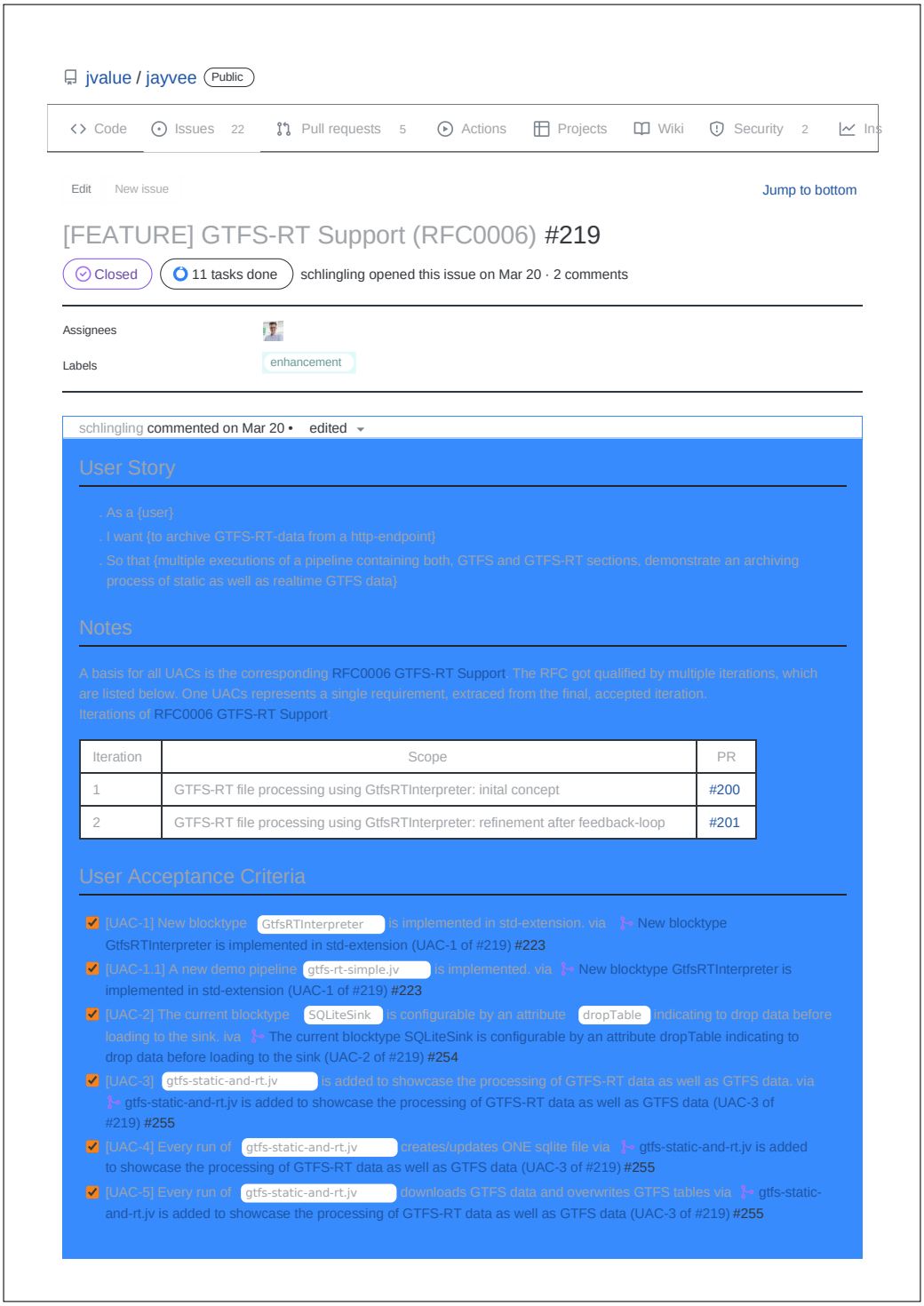
Alternatives

An alternative approach could involve using a generic block type called JsonInterpreter instead of the proposed GtfsRTInterpreter. This block type would parse incoming files into a new io-datatype called JSON. A downstream block type JsonFlattener would then flatten the JSON into a tabular sheet representation (eg. by having a file map to some form of tree structure for JSON which maps to sheets). Since the flattening of generic JSON files into tabular representation is a fundamental topic for ETL systems, this approach should be discussed in a separate RFC as it falls outside the scope of the master thesis.

Outlook

Once we want to logically validate the data model during load, we have the dependency to first load the static data and afterwards the realtime data because realtime depends on static. We just want to load realtime data to the sink, if the data is conform to the static data which was loaded in advance. Unfortunally state now we cannot model this sequential dependency in one pipeline(eg. by connecting the GTFS-sink with the GTFS-RT-Extractor) but to shed light on this the dependcy this is mentioned as an outlook.

D GitHub Issue for RFC-0006 GTFS-RT Support



jvalue / jayvee Public

Code | Issues 22 | Pull requests 5 | Actions | Projects | Wiki | Security 2 | Ins

Edit | New issue | Jump to bottom

[FEATURE] GTFS-RT Support (RFC0006) #219

Closed | 11 tasks done | schlingling opened this issue on Mar 20 · 2 comments

Assignees

Labels: enhancement

schlingling commented on Mar 20 • edited

User Story

. As a {user}
. I want {to archive GTFS-RT-data from a http-endpoint}
. So that {multiple executions of a pipeline containing both, GTFS and GTFS-RT sections, demonstrate an archiving process of static as well as realtime GTFS data}

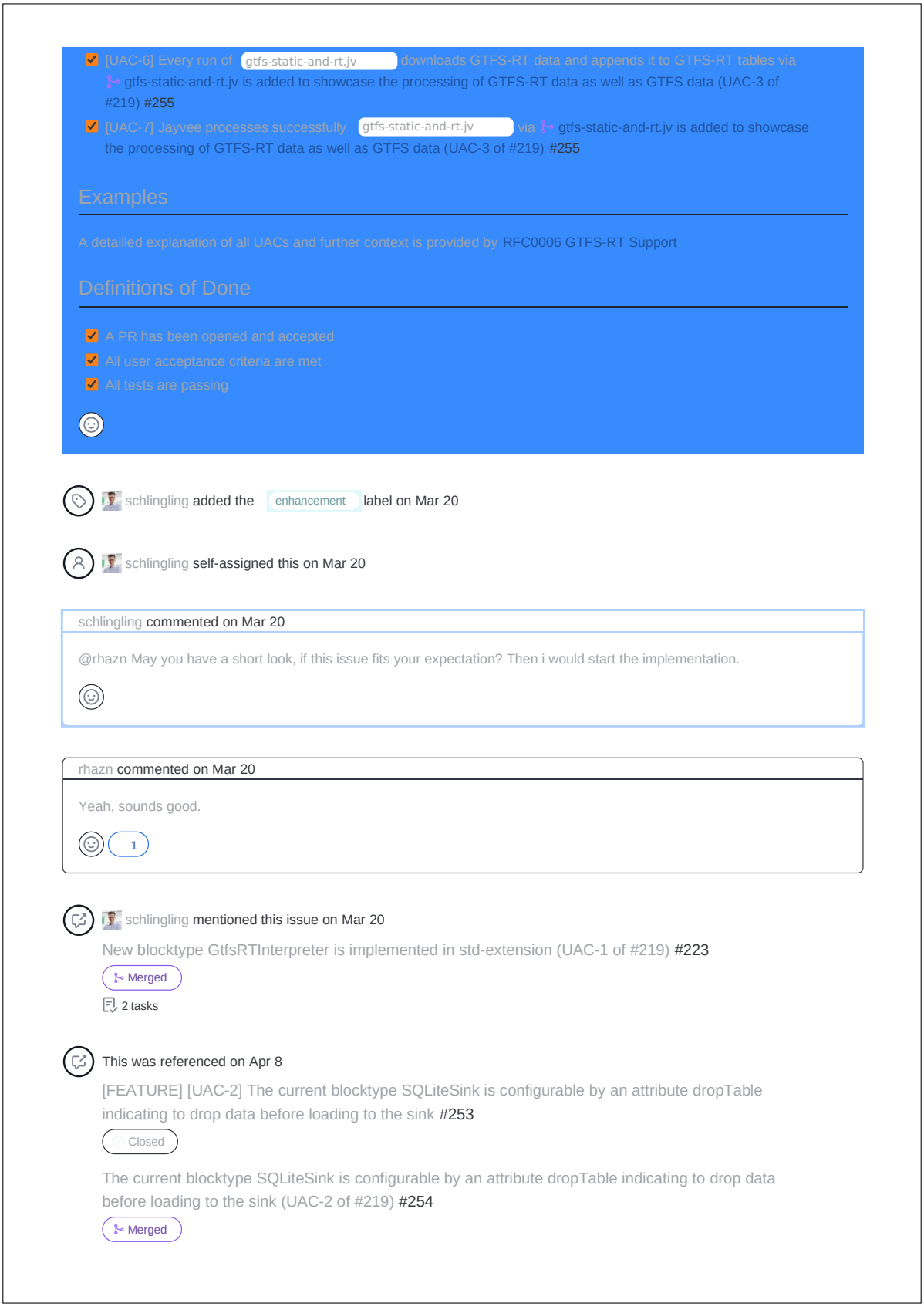
Notes

A basis for all UACs is the corresponding RFC0006 GTFS-RT Support. The RFC got qualified by multiple iterations, which are listed below. One UACs represents a single requirement, extraced from the final, accepted iteration.
Iterations of RFC0006 GTFS-RT Support:

Iteration	Scope	PR
1	GTFS-RT file processing using GtfsRTInterpreter: inital concept	#200
2	GTFS-RT file processing using GtfsRTInterpreter: refinement after feedback-loop	#201

User Acceptance Criteria

- [x] [UAC-1] New blocktype `GtfsRTInterpreter` is implemented in std-extension. via New blocktype GtfsRTInterpreter is implemented in std-extension (UAC-1 of #219) #223
- [x] [UAC-1.1] A new demo pipeline `gtfs-rt-simple.jv` is implemented. via New blocktype GtfsRTInterpreter is implemented in std-extension (UAC-1 of #219) #223
- [x] [UAC-2] The current blocktype `SQLiteSink` is configurable by an attribute `dropTable` indicating to drop data before loading to the sink. iva The current blocktype SQLiteSink is configurable by an attribute dropTable indicating to drop data before loading to the sink (UAC-2 of #219) #254
- [x] [UAC-3] `gtfs-static-and-rt.jv` is added to showcase the processing of GTFS-RT data as well as GTFS data. via gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255
- [x] [UAC-4] Every run of `gtfs-static-and-rt.jv` creates/updates ONE sqlite file via gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255
- [x] [UAC-5] Every run of `gtfs-static-and-rt.jv` downloads GTFS data and overwrites GTFS tables via gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255



- [x] [UAC-6] Every run of `gtfs-static-and-rt.jv` downloads GTFS-RT data and appends it to GTFS-RT tables via gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255
- [x] [UAC-7] Jayvee processes successfully `gtfs-static-and-rt.jv` via gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255

Examples

A detailled explanation of all UACs and further context is provided by RFC0006 GTFS-RT Support

Definitions of Done

- [x] A PR has been opened and accepted
- [x] All user acceptance criteria are met
- [x] All tests are passing

schlingling added the `enhancement` label on Mar 20

schlingling self-assigned this on Mar 20

schlingling commented on Mar 20

@rhazn May you have a short look, if this issue fits your expectation? Then i would start the implementation.

rhazn commented on Mar 20

Yeah, sounds good.

1

schlingling mentioned this issue on Mar 20

New blocktype GtfsRTInterpreter is implemented in std-extension (UAC-1 of #219) #223

Merged

2 tasks

This was referenced on Apr 8

[FEATURE] [UAC-2] The current blocktype SQLiteSink is configurable by an attribute dropTable indicating to drop data before loading to the sink #253

Closed

The current blocktype SQLiteSink is configurable by an attribute dropTable indicating to drop data before loading to the sink (UAC-2 of #219) #254

Merged

gtfs-static-and-rt.jv is added to showcase the processing of GTFS-RT data as well as GTFS data (UAC-3 of #219) #255

Merged

schlingling added a commit that referenced this issue on Apr 10

Merge pull request #255 from jvalue/feat-gtfs-rt-uac-3-gtfsjv-both ... 8d90c2b

schlingling added a commit that referenced this issue on Apr 10

Merge pull request #254 from jvalue/fet-gtfs-rt-uac2-droptable-indicator ... 2d17b70

schlingling added a commit that referenced this issue on Apr 12

Merge pull request #223 from jvalue/feat-gtfs-rt-uac1-gtfsrtinterpreter ... d1db034

schlingling closed this as completed 3 weeks ago

schlingling mentioned this issue 3 weeks ago

Implemented composite pattern for filesystem #256

Merged

Assignees

schlingling

Labels

enhancement

Projects

None yet

Milestone

No milestone

Development

Create a branch for this issue or link a pull request.

2 participants

Pin issue

E Bill of Materials

Bill of materials with packages which were additionally introduced by the implementations of this thesis:

Name	Version	License	Used in	Comment
gtfs-realtime-bindings	1.1.1	Apache 2.0	gtfs-rt extension, demonstrator	Bindings for gtfs realtime
jszip	3.10.1	MIT	gtfs extension	Unarchiver for zip files
mime-types	2.1.35	MIT	gtfs and gtfs-rt extension	Maps file extensions to common mimetypes
csv-parser	3.0.0	MIT	demonstrator	API for CSV files
sqlite	4.1.2	MIT	demonstrator	API for sqlite databases

F GTFS Static Pipeline

GitHub: https://github.com/jvalue/jayvee/blob/98fedbef2b717b1bb586593502804fa2ad3dba06/example/gtfs-static.jv

```
pipeline GtfsPipeline {
  block GTFSSampleFeedExtractor oftype HttpExtractor {
    url : " https :// developers . google . com / static / transit / gtfs / examples / sample - feed . zip ";
  }

  block ZipArchiveInterpreter oftype ArchiveInterpreter {
    archiveType : " zip ";
  }

  block AgencyFilePicker oftype FilePicker {
    path : "/ agency . txt ";
  }

  block CalendarDatesFilePicker oftype FilePicker {
    path : "/ calendar_dates . txt ";
  }

  block CalendarFilePicker oftype FilePicker {
    path : "/ calendar . txt ";
  }

  block FareAttributesFilePicker oftype FilePicker {
    path : "/ fare_attributes . txt ";
  }

  block FareRulesFilePicker oftype FilePicker {
    path : "/ fare_rules . txt ";
  }

  block FrequenciesFilePicker oftype FilePicker {
    path : "/ frequencies . txt ";
  }

  block RoutesFilePicker oftype FilePicker {
    path : "/ routes . txt ";
  }

  block ShapesFilePicker oftype FilePicker {
    path : "/ shapes . txt ";
  }

  block StopTimesFilePicker oftype FilePicker {
    path : "/ stop_times . txt ";
  }

  block StopsFilePicker oftype FilePicker {
    path : "/ stops . txt ";
  }

  block TripsFilePicker oftype FilePicker {
    path : "/ trips . txt ";
  }

  block AgencyTextFileInterpreter oftype TextFileInterpreter {
  }

  block CalendarDatesTextFileInterpreter oftype TextFileInterpreter {
  }

  block CalendarTextFileInterpreter oftype TextFileInterpreter {
  }

  block FareAttributesTextFileInterpreter oftype TextFileInterpreter {
  }

  block FareRulesTextFileInterpreter oftype TextFileInterpreter {
  }

  block FrequenciesTextFileInterpreter oftype TextFileInterpreter {
  }

  block RoutesTextFileInterpreter oftype TextFileInterpreter {
  }

  block ShapesTextFileInterpreter oftype TextFileInterpreter {
  }

```

```
78  block StopTimesTextFileInterpreter oftype TextFileInterpreter {
79  }
80
81  block StopsTextFileInterpreter oftype TextFileInterpreter {
82  }
83
84  block TripsTextFileInterpreter oftype TextFileInterpreter {
85  }
86
87  block AgencyCSVInterpreter oftype CSVInterpreter {
88  }
89
90  block CalendarDatesCSVInterpreter oftype CSVInterpreter {
91  }
92
93  block CalendarCSVInterpreter oftype CSVInterpreter {
94  }
95
96  block FareAttributesCSVInterpreter oftype CSVInterpreter {
97  }
98
99  block FareRulesCSVInterpreter oftype CSVInterpreter {
100 }
101
102 block FrequenciesCSVInterpreter oftype CSVInterpreter {
103 }
104
105 block RoutesCSVInterpreter oftype CSVInterpreter {
106 }
107
108 block ShapesCSVInterpreter oftype CSVInterpreter {
109 }
110
111 block StopTimesCSVInterpreter oftype CSVInterpreter {
112 }
113
114 block StopsCSVInterpreter oftype CSVInterpreter {
115 }
116
117 block TripsCSVInterpreter oftype CSVInterpreter {
118 }
119
120 block AgencyTableInterpreter oftype TableInterpreter {
121   header : true ;
122   columns :[
123     " agency_id " oftype text , // Conditional columns are considered as required
124     " agency_name " oftype text ,
125     " agency_url " oftype text ,
126     " agency_timezone " oftype text
127   ];
128 }
129
130 block CalendarDatesTableInterpreter oftype TableInterpreter {
131   header : true ;
132   columns : [
133     " service_id " oftype text ,
134     " date " oftype text ,
135     " exception_type " oftype text
136   ];
137 }
138
139 block CalendarTableInterpreter oftype TableInterpreter {
140   header : true ;
141   columns : [
142     " service_id " oftype text ,
143     " monday " oftype text ,
144     " tuesday " oftype text ,
145     " wednesday " oftype text ,
146     " thursday " oftype text ,
147     " friday " oftype text ,
148     " saturday " oftype text ,
149     " sunday " oftype text ,
150     " start_date " oftype text ,
151     " end_date " oftype text
152   ];
153 }
154
155 block FareAttributesTableInterpreter oftype TableInterpreter {
156   header : true ;
157   columns : [
158     " fare_id " oftype text ,
159     " price " oftype text ,
160     " currency_type " oftype text ,
161     " payment_method " oftype text ,
162     " transfers " oftype text ,
163     " transfer_duration " oftype text
164   ];
165 }
```

```
166
167   block   FareRulesTableInterpreter       oftype   TableInterpreter     {
168      header :  true ;
169      columns :  [
170         " fare_id "    oftype   text ,
171         " route_id "    oftype   text ,
172         " origin_id "     oftype   text ,
173         " destination_id "      oftype   text ,
174         " contains_id "     oftype   text
175      ];
176   }
177
178   block   FrequenciesTableInterpreter        oftype   TableInterpreter     {
179      header :  true ;
180      columns :  [
181         " trip_id "     oftype   text ,
182         " start_time "    oftype   text ,
183         " end_time "  oftype   text ,
184         " headway_secs "  oftype   text
185      ];
186   }
187
188   block   RoutesTableInterpreter      oftype   TableInterpreter     {
189      header :  true ;
190      columns :  [
191         " route_id "    oftype   text ,
192         " agency_id "   oftype   text ,
193         " route_short_name "    oftype   text ,
194         " route_long_name "   oftype   text ,
195         " route_desc "   oftype   text ,
196         " route_type "   oftype   text ,
197         " route_url "    oftype   text ,
198         " route_color "     oftype   text ,
199         " route_text_color "      oftype   text
200      ];
201   }
202
203   block   ShapesTableInterpreter      oftype   TableInterpreter     {
204      header :  true ;
205      columns :  [
206         " shape_id "   oftype   text ,
207         " shape_pt_lat "    oftype   text ,
208         " shape_pt_lon "    oftype   text ,
209         " shape_pt_sequence "   oftype   text ,
210         " shape_dist_traveled "      oftype   text
211      ];
212   }
213
214   block   StopTimesTableInterpreter       oftype   TableInterpreter     {
215      header :  true ;
216      columns :  [
217         " trip_id "     oftype   text ,
218         " arrival_time "     oftype   text ,
219         " departure_time "    oftype   text ,
220         " stop_id "    oftype   text ,
221         " stop_sequence "   oftype   text ,
222         " stop_headsign "    oftype   text ,
223         " pickup_type "    oftype   text ,
224         " drop_off_time "     oftype   text ,
225         " shape_dist_traveled "      oftype   text
226      ];
227   }
228
229   block   StopsTableInterpreter       oftype   TableInterpreter     {
230      header :  true ;
231      columns :[
232         " stop_id "   oftype   text ,
233         " stop_name "  oftype   text ,
234         " stop_desc "   oftype   text ,
235         " stop_lat "    oftype   text ,
236         " stop_lon "    oftype   text ,
237         " zone_id "   oftype   text ,
238         " stop_url "    oftype   text
239      ];
240   }
241
242   block   TripsTableInterpreter       oftype   TableInterpreter     {
243      header :  true ;
244      columns :  [
245         " route_id "    oftype   text ,
246         " service_id "     oftype   text ,
247         " trip_id "     oftype   text ,
248         " trip_headsign "     oftype   text ,
249         " direction_id "     oftype   text ,
250         " block_id "    oftype   text ,
251         " shape_id "   oftype   text
252      ];
253   }
```

```
254
255     block   AgencyLoader   oftype   SQLiteLoader   {
256       table :   " agency ";
257       file :    "./ gtfs . sqlite ";
258     }
259
260     block   CalendarDatesLoader    oftype   SQLiteLoader   {
261       table :   " calendar_dates ";
262       file :    "./ gtfs . sqlite ";
263     }
264
265     block   CalendarLoader    oftype   SQLiteLoader   {
266       table :   " calendar ";
267       file :    "./ gtfs . sqlite ";
268     }
269
270     block   FareAttributesLoader      oftype   SQLiteLoader   {
271       table :   " fare_attributes ";
272       file :    "./ gtfs . sqlite ";
273     }
274
275     block   FareRulesLoader    oftype   SQLiteLoader   {
276       table :   " fare_rules ";
277       file :    "./ gtfs . sqlite ";
278     }
279
280     block   FrequenciesLoader    oftype   SQLiteLoader   {
281       table :   " frequencies ";
282       file :    "./ gtfs . sqlite ";
283     }
284
285     block   RoutesLoader   oftype   SQLiteLoader   {
286       table :   " routes ";
287       file :    "./ gtfs . sqlite ";
288     }
289
290     block   ShapesLoader   oftype   SQLiteLoader   {
291       table :   " shapes ";
292       file :    "./ gtfs . sqlite ";
293     }
294
295     block   StopTimesLoader   oftype   SQLiteLoader   {
296       table :   " stop_times ";
297       file :    "./ gtfs . sqlite ";
298     }
299
300     block   StopsLoader   oftype   SQLiteLoader   {
301       table :   " stops ";
302       file :    "./ gtfs . sqlite ";
303     }
304
305     block   TripsLoader    oftype   SQLiteLoader   {
306       table :   " trips ";
307       file :    "./ gtfs . sqlite ";
308     }
309
310     GTFSSampleFeedExtractor    ->  ZipArchiveInterpreter ;
311
312     ZipArchiveInterpreter
313       ->  AgencyFilePicker
314       ->  AgencyTextFileInterpreter
315       ->  AgencyCSVInterpreter
316       ->  AgencyTableInterpreter
317       ->  AgencyLoader ;
318
319     ZipArchiveInterpreter
320       ->  CalendarDatesFilePicker
321       ->  CalendarDatesTextFileInterpreter
322       ->  CalendarDatesCSVInterpreter
323       ->  CalendarDatesTableInterpreter
324       ->  CalendarDatesLoader ;
325
326     ZipArchiveInterpreter
327       ->  CalendarFilePicker
328       ->  CalendarTextFileInterpreter
329       ->  CalendarCSVInterpreter
330       ->  CalendarTableInterpreter
331       ->  CalendarLoader ;
332
333     ZipArchiveInterpreter
334       ->  FareAttributesFilePicker
335       ->  FareAttributesTextFileInterpreter
336       ->  FareAttributesCSVInterpreter
337       ->  FareAttributesTableInterpreter
338       ->  FareAttributesLoader ;
339
340     ZipArchiveInterpreter
341       ->  FareRulesFilePicker
```

```
342         ->  FareRulesTextFileInterpreter
343         ->  FareRulesCSVInterpreter
344         ->  FareRulesTableInterpreter
345         ->  FareRulesLoader ;
346
347     ZipArchiveInterpreter
348         ->  FrequenciesFilePicker
349         ->  FrequenciesTextFileInterpreter
350         ->  FrequenciesCSVInterpreter
351         ->  FrequenciesTableInterpreter
352         ->  FrequenciesLoader ;
353
354     ZipArchiveInterpreter
355         ->  RoutesFilePicker
356         ->  RoutesTextFileInterpreter
357         ->  RoutesCSVInterpreter
358         ->  RoutesTableInterpreter
359         ->  RoutesLoader ;
360
361     ZipArchiveInterpreter
362         ->  ShapesFilePicker
363         ->  ShapesTextFileInterpreter
364         ->  ShapesCSVInterpreter
365         ->  ShapesTableInterpreter
366         ->  ShapesLoader ;
367
368     ZipArchiveInterpreter
369         ->  StopTimesFilePicker
370         ->  StopTimesTextFileInterpreter
371         ->  StopTimesCSVInterpreter
372         ->  StopTimesTableInterpreter
373         ->  StopTimesLoader ;
374
375     ZipArchiveInterpreter
376         ->  StopsFilePicker
377         ->  StopsTextFileInterpreter
378         ->  StopsCSVInterpreter
379         ->  StopsTableInterpreter
380         ->  StopsLoader ;
381
382     ZipArchiveInterpreter
383         ->  TripsFilePicker
384         ->  TripsTextFileInterpreter
385         ->  TripsCSVInterpreter
386         ->  TripsTableInterpreter
387         ->  TripsLoader ;
388
389 }
390 }
```

G GTFS Realtime Pipeline

GitHub: https://github.com/jvalue/jayvee/blob/98fedbef2b717b1bb586593502804fa2ad3dba06/example/gtfs-rt-simple.jv

```
pipeline     GtfsRTSimplePipeline       {
   block    GTFSRTTripUpdateFeedExtractor       oftype    HttpExtractor      {
      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - trip - update ";
   }

      block    GTFSRTVehiclePositionFeedExtractor          oftype    HttpExtractor      {
      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - vehicle - position ";
   }

      block    GTFSRTAlertFeedExtractor       oftype    HttpExtractor      {
      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - alerts ";
   }

      block    GtfsRTTripUpdateInterpreter         oftype    GtfsRTInterpreter       {
            entity :     " trip_update ";
      }

      block    GtfsRTAlertInterpreter         oftype    GtfsRTInterpreter       {
            entity :     " alert ";
      }

      block    GtfsRTVehiclePositionInterpreter            oftype    GtfsRTInterpreter       {
            entity :     " vehicle ";
      }
      block    TripUpdateTableInterpreter         oftype    TableInterpreter       {
      header :   true ;
      columns :[
         " header . gtfs_realtime_version "        oftype    text ,
        " header . timestamp "  oftype    text ,
                  " header . incrementality "      oftype    text ,
                  " entity . id "     oftype    text ,
                  " entity . trip_update . trip . trip_id "          oftype    text ,
                  " entity . trip_update . trip . route_id "         oftype    text ,
                  " entity . trip_update . stop_time_update . stop_sequence "         oftype    text ,
                  " entity . trip_update . stop_time_update . stop_id "          oftype    text ,
                  " entity . trip_update . stop_time_update . arrival . time "          oftype    text ,
                  " entity . trip_update . stop_time_update . departure . time "         oftype    text ,
      ];
   }

      block    VehiclePositionTableInterpreter            oftype    TableInterpreter       {
      header :   true ;
      columns :[
                  " header . gtfs_realtime_version "        oftype    text ,
                  " header . timestamp "  oftype    text ,
                  " header . incrementality "      oftype    text ,
                  " entity . id "     oftype    text ,
                  " entity . vehicle_position . vehicle_descriptor . id "             oftype    text ,
                  " entity . vehicle_position . trip . trip_id "             oftype    text ,
                  " entity . vehicle_position . trip . route_id "           oftype    text ,
                  " entity . vehicle_position . position . latitude "            oftype    text ,
                  " entity . vehicle_position . position . longitude "           oftype    text ,
                  " entity . vehicle_position . timestamp "         oftype    text
            ];
   }

      block    AlertTableInterpreter         oftype    TableInterpreter       {
      header :   true ;
      columns :[
                  ' header . gtfs_realtime_version '         oftype    text ,
                  ' header . timestamp '   oftype    text ,
                  ' header . incrementality '        oftype    text ,
                  ' entity .id '      oftype    text ,
                  ' entity . alert . informed_entity . route_id '          oftype    text ,
                  ' entity . alert . header_text '       oftype    text ,
                  ' entity . alert . description_text '          oftype    text ,
            ];
   }

      block    TripUpdateLoader     oftype    SQLiteLoader     {
      table :    " gtfs -rt - trip_update ";
      file :     "./ gtfs . sqlite ";
            dropTable :    false ;
   }

      block    VehicleLoader     oftype    SQLiteLoader     {
      table :    " gtfs -rt - vehicle_position ";
```

```
78          file :     "./ gtfs . sqlite ";
79                dropTable :    false ;
80      }
81
82          block    AlertLoader     oftype    SQLiteLoader    {
83          table :    " gtfs -rt - alert ";
84          file :     "./ gtfs . sqlite ";
85                dropTable :    false ;
86      }
87
88          GTFSRTTripUpdateFeedExtractor
89                -> GtfsRTTripUpdateInterpreter
90                -> TripUpdateTableInterpreter
91                -> TripUpdateLoader ;
92
93          GTFSRTVehiclePositionFeedExtractor
94                -> GtfsRTVehiclePositionInterpreter
95                -> VehiclePositionTableInterpreter
96                -> VehicleLoader ;
97
98          GTFSRTAlertFeedExtractor
99                -> GtfsRTAlertInterpreter
100               -> AlertTableInterpreter
101               -> AlertLoader ;
102 }
```

H GTFS Static and Realtime Pipeline

GitHub: https://github.com/jvalue/jayvee/blob/98fedbef2b717b1bb586593502804fa2ad3dba06/example/gtfs-static-and-rt.jv

```
pipeline GtfsStaticAndRealtimePipeline {

  block GTFSExtractor oftype HttpExtractor {
    url : " https :// ratpdev - mosaic - prod - bucket - raw . s3 -eu - west -1. amazonaws . com /11/ exports /1/ gtfs . zip ";
  }

  block ZipArchiveInterpreter oftype ArchiveInterpreter {
    archiveType : " zip ";
  }

  block AgencyFilePicker oftype FilePicker {
    path : "/ agency . txt ";
  }

  block CalendarDatesFilePicker oftype FilePicker {
    path : "/ calendar_dates . txt ";
  }

  block FeedInfoFilePicker oftype FilePicker {
    path : "/ feed_info . txt ";
  }


  block CalendarFilePicker oftype FilePicker {
    path : "/ calendar . txt ";
  }

  block RoutesFilePicker oftype FilePicker {
    path : "/ routes . txt ";
  }

  block ShapesFilePicker oftype FilePicker {
    path : "/ shapes . txt ";
  }

  block StopTimesFilePicker oftype FilePicker {
    path : "/ stop_times . txt ";
  }

  block StopsFilePicker oftype FilePicker {
    path : "/ stops . txt ";
  }

  block TripsFilePicker oftype FilePicker {
    path : "/ trips . txt ";
  }


  block AgencyTextFileInterpreter oftype TextFileInterpreter {
  }

  block CalendarDatesTextFileInterpreter oftype TextFileInterpreter {
  }

  block CalendarTextFileInterpreter oftype TextFileInterpreter {
  }

  block FeedInfoTextFileInterpreter oftype TextFileInterpreter {
  }


  block RoutesTextFileInterpreter oftype TextFileInterpreter {
  }

  block ShapesTextFileInterpreter oftype TextFileInterpreter {
  }

  block StopTimesTextFileInterpreter oftype TextFileInterpreter {
  }

  block StopsTextFileInterpreter oftype TextFileInterpreter {
  }

  block TripsTextFileInterpreter oftype TextFileInterpreter {
  }

  block AgencyCSVInterpreter oftype CSVInterpreter {
```

```
}

block   CalendarDatesCSVInterpreter       oftype    CSVInterpreter     {
}

block   CalendarCSVInterpreter      oftype    CSVInterpreter     {
}

block   FeedInfoCSVInterpreter       oftype    CSVInterpreter     {
}

block   RoutesCSVInterpreter      oftype    CSVInterpreter     {
}

block   ShapesCSVInterpreter      oftype    CSVInterpreter     {
}

block   StopTimesCSVInterpreter      oftype    CSVInterpreter     {
}

block   StopsCSVInterpreter      oftype    CSVInterpreter     {
}

block   TripsCSVInterpreter       oftype    CSVInterpreter     {
}

block   AgencyTableInterpreter       oftype    TableInterpreter      {
   header :  true ;
   columns :[
      " agency_id "   oftype   text ,
      " agency_name " oftype   text ,
      " agency_url "   oftype   text ,
      " agency_timezone "   oftype   text ,
      " agency_lang "   oftype   text ,
      " agency_phone "  oftype   text ,
      " agency_fare_url "     oftype   text ,
      " agency_email "   oftype   text
   ];
}

block   CalendarDatesTableInterpreter        oftype    TableInterpreter      {
   header :  true ;
   columns :  [
      " service_id "    oftype   text ,
      " date "  oftype   text ,
      " exception_type "    oftype   text
   ];
}

block   CalendarTableInterpreter       oftype    TableInterpreter      {
   header :  true ;
   columns :  [
      " service_id "    oftype   text ,
      " monday "oftype   text ,
      " tuesday "  oftype   text ,
      " wednesday " oftype   text ,
      " thursday "   oftype   text ,
      " friday "    oftype   text ,
      " saturday "   oftype   text ,
      " sunday "  oftype   text ,
      " start_date "    oftype   text ,
      " end_date "  oftype   text
   ];
}

block   FeedInfoTableInterpreter       oftype    TableInterpreter      {
   header :  true ;
   columns :  [
      " feed_publisher_name "    oftype   text ,
      " feed_publisher_url "      oftype   text ,
      " feed_lang "    oftype   text ,
      " feed_start_date "     oftype   text ,
      " feed_end_date "    oftype   text ,
      " feed_version "    oftype   text ,
      " feed_contact_email "     oftype   text ,
      " feed_contact_url "      oftype   text ,
   ];
}

block   RoutesTableInterpreter       oftype    TableInterpreter      {
   header :  true ;
   columns :  [
      " route_id "    oftype   text ,
      " agency_id "   oftype   text ,
      " route_short_name "    oftype   text ,
      " route_long_name "   oftype   text ,
      " route_desc "    oftype   text ,
      " route_type "    oftype   text ,
```

```
        " route_url "    oftype   text ,
        " route_color "    oftype   text ,
        " route_text_color "      oftype   text ,
        " route_sort_order "      oftype   text
    ];
}

block   ShapesTableInterpreter      oftype   TableInterpreter     {
    header :  true ;
    columns :  [
        " shape_id "   oftype   text ,
        " shape_pt_lat "    oftype   text ,
        " shape_pt_lon "    oftype   text ,
        " shape_pt_sequence "    oftype   text ,
    ];
}

block   StopTimesTableInterpreter       oftype   TableInterpreter     {
    header :  true ;
    columns :  [
        " trip_id "    oftype   text ,
        " arrival_time "     oftype   text ,
        " departure_time "    oftype   text ,
        " stop_id "    oftype   text ,
        " stop_sequence "   oftype   text ,
        " stop_headsign "   oftype   text ,
        " pickup_type "   oftype   text ,
        " drop_off_type "    oftype   text ,
    ];
}

block   StopsTableInterpreter       oftype   TableInterpreter     {
    header :  true ;
    columns :[
        " stop_id "   oftype   text ,
        " stop_code "  oftype   text ,
        " stop_name "  oftype   text ,
        " stop_desc "   oftype   text ,
        " stop_lat "    oftype   text ,
        " stop_lon "    oftype   text ,
        " zone_id "   oftype   text ,
        " stop_url "    oftype   text ,
        " location_type "     oftype   text ,
        " parent_station "     oftype   text ,
        " stop_timezone "   oftype   text ,
        " wheelchair_boarding "     oftype   text ,
        " level_id "    oftype   text ,
        " platform_code "    oftype   text
    ];
}

block   TripsTableInterpreter       oftype   TableInterpreter     {
    header :  true ;
    columns :  [
        " route_id "   oftype   text ,
        " service_id "     oftype   text ,
        " trip_id "    oftype   text ,
        " trip_headsign "     oftype   text ,
        " trip_short_name "     oftype   text ,
        " direction_id "      oftype   text ,
        " block_id "    oftype   text ,
        " shape_id "   oftype   text ,
        " wheelchair_accessible "      oftype   text ,
        " bikes_allowed "    oftype   text
    ];
}

block   AgencyLoader   oftype   SQLiteLoader   {
    table :   " static_agency ";
    file :    "./ gtfs - static - and - rt . sqlite ";
}

block   CalendarDatesLoader    oftype   SQLiteLoader   {
    table :   " static_calendar_dates ";
    file :    "./ gtfs - static - and - rt . sqlite ";
}

block   CalendarLoader    oftype   SQLiteLoader   {
    table :   " static_calendar ";
    file :    "./ gtfs - static - and - rt . sqlite ";
}

block   FeedInfoLoader    oftype   SQLiteLoader   {
    table :   " static_feed_info ";
    file :    "./ gtfs - static - and - rt . sqlite ";
}

block   RoutesLoader   oftype   SQLiteLoader   {
```

```
254         table :    " static_routes ";
255         file :     "./ gtfs - static - and - rt . sqlite ";
256     }
257
258     block   ShapesLoader   oftype   SQLiteLoader    {
259         table :    " static_shapes ";
260         file :     "./ gtfs - static - and - rt . sqlite ";
261     }
262
263     block   StopTimesLoader   oftype   SQLiteLoader    {
264         table :    " static_stop_times ";
265         file :     "./ gtfs - static - and - rt . sqlite ";
266     }
267
268     block   StopsLoader   oftype   SQLiteLoader    {
269         table :    " static_stops ";
270         file :     "./ gtfs - static - and - rt . sqlite ";
271     }
272
273     block   TripsLoader    oftype   SQLiteLoader    {
274         table :    " static_trips ";
275         file :     "./ gtfs - static - and - rt . sqlite ";
276     }
277
278     GTFSExtractor    ->  ZipArchiveInterpreter ;
279
280     ZipArchiveInterpreter
281         ->  AgencyFilePicker
282         ->  AgencyTextFileInterpreter
283         ->  AgencyCSVInterpreter
284         ->  AgencyTableInterpreter
285         ->  AgencyLoader ;
286
287     ZipArchiveInterpreter
288         ->  CalendarDatesFilePicker
289         ->  CalendarDatesTextFileInterpreter
290         ->  CalendarDatesCSVInterpreter
291         ->  CalendarDatesTableInterpreter
292         ->  CalendarDatesLoader ;
293
294     ZipArchiveInterpreter
295         ->  CalendarFilePicker
296         ->  CalendarTextFileInterpreter
297         ->  CalendarCSVInterpreter
298         ->  CalendarTableInterpreter
299         ->  CalendarLoader ;
300
301     ZipArchiveInterpreter
302         ->  FeedInfoFilePicker
303         ->  FeedInfoTextFileInterpreter
304         ->  FeedInfoCSVInterpreter
305         ->  FeedInfoTableInterpreter
306         ->  FeedInfoLoader ;
307
308     ZipArchiveInterpreter
309         ->  RoutesFilePicker
310         ->  RoutesTextFileInterpreter
311         ->  RoutesCSVInterpreter
312         ->  RoutesTableInterpreter
313         ->  RoutesLoader ;
314
315     ZipArchiveInterpreter
316         ->  ShapesFilePicker
317         ->  ShapesTextFileInterpreter
318         ->  ShapesCSVInterpreter
319         ->  ShapesTableInterpreter
320         ->  ShapesLoader ;
321
322     ZipArchiveInterpreter
323         ->  StopTimesFilePicker
324         ->  StopTimesTextFileInterpreter
325         ->  StopTimesCSVInterpreter
326         ->  StopTimesTableInterpreter
327         ->  StopTimesLoader ;
328
329     ZipArchiveInterpreter
330         ->  StopsFilePicker
331         ->  StopsTextFileInterpreter
332         ->  StopsCSVInterpreter
333         ->  StopsTableInterpreter
334         ->  StopsLoader ;
335
336     ZipArchiveInterpreter
337         ->  TripsFilePicker
338         ->  TripsTextFileInterpreter
339         ->  TripsCSVInterpreter
340         ->  TripsTableInterpreter
341         ->  TripsLoader ;
```

```
342
343
344   //   GTFS -RT - Part
345
346
347   block   GTFSRTTripUpdateFeedExtractor      oftype   HttpExtractor     {
348      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - trip - update ";
349   }
350
351      block   GTFSRTVehiclePositionFeedExtractor         oftype   HttpExtractor     {
352      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - vehicle - position ";
353   }
354
355      block   GTFSRTAlertFeedExtractor      oftype   HttpExtractor     {
356      url :    " https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs - rt - alerts ";
357   }
358
359      block   GtfsRTTripUpdateInterpreter        oftype   GtfsRTInterpreter {
360            entity :     " trip_update ";
361      }
362
363      block   GtfsRTAlertInterpreter         oftype   GtfsRTInterpreter {
364            entity :     " alert ";
365      }
366
367      block   GtfsRTVehiclePositionInterpreter            oftype   GtfsRTInterpreter {
368            entity :     " vehicle ";
369      }
370      block   TripUpdateTableInterpreter        oftype   TableInterpreter      {
371      header :  true ;
372      columns :[
373         " header . gtfs_realtime_version "       oftype   text ,
374         " header . timestamp "  oftype   text ,
375                  " header . incrementality "     oftype   text ,
376                  " entity . id "    oftype   text ,
377                  " entity . trip_update . trip . trip_id "         oftype   text ,
378                  " entity . trip_update . trip . route_id "        oftype   text ,
379                  " entity . trip_update . stop_time_update . stop_sequence "        oftype   text ,
380                  " entity . trip_update . stop_time_update . stop_id "         oftype   text ,
381                  " entity . trip_update . stop_time_update . arrival . time "         oftype   text ,
382                  " entity . trip_update . stop_time_update . departure . time "        oftype   text ,
383      ];
384   }
385
386      block   VehiclePositionTableInterpreter           oftype   TableInterpreter      {
387      header :  true ;
388      columns :[
389                  " header . gtfs_realtime_version "       oftype   text ,
390                  " header . timestamp "  oftype   text ,
391                  " header . incrementality "     oftype   text ,
392                  " entity . id "    oftype   text ,
393                  " entity . vehicle_position . vehicle_descriptor . id "             oftype   text ,
394                  " entity . vehicle_position . trip . trip_id "           oftype   text ,
395                  " entity . vehicle_position . trip . route_id "          oftype   text ,
396                  " entity . vehicle_position . position . latitude "           oftype   text ,
397                  " entity . vehicle_position . position . longitude "          oftype   text ,
398                  " entity . vehicle_position . timestamp "       oftype   text
399            ];
400   }
401
402      block   AlertTableInterpreter        oftype   TableInterpreter      {
403      header :  true ;
404      columns :[
405                  ' header . gtfs_realtime_version '        oftype   text ,
406                  ' header . timestamp '  oftype   text ,
407                  ' header . incrementality '       oftype   text ,
408                  ' entity .id '     oftype   text ,
409                  ' entity . alert . informed_entity . route_id '          oftype   text ,
410                  ' entity . alert . header_text '      oftype   text ,
411                  ' entity . alert . description_text '          oftype   text ,
412            ];
413   }
414
415      block   TripUpdateLoader    oftype   SQLiteLoader    {
416      table :    " rt_trip_update ";
417      file :     "./ gtfs - static - and - rt . sqlite ";
418            dropTable :    false ;
419   }
420
421      block   VehicleLoader    oftype   SQLiteLoader    {
422      table :    " rt_vehicle_position ";
423      file :     "./ gtfs - static - and - rt . sqlite ";
424            dropTable :    false ;
425   }
426
427      block   AlertLoader    oftype   SQLiteLoader    {
428      table :    " rt_alert ";
429      file :     "./ gtfs - static - and - rt . sqlite ";
```

```
            dropTable :    false ;
    }

        GTFSRTTripUpdateFeedExtractor
            -> GtfsRTTripUpdateInterpreter
                -> TripUpdateTableInterpreter
                    -> TripUpdateLoader ;

        GTFSRTVehiclePositionFeedExtractor
            -> GtfsRTVehiclePositionInterpreter
                -> VehiclePositionTableInterpreter
                    -> VehicleLoader ;

        GTFSRTAlertFeedExtractor
            -> GtfsRTAlertInterpreter
                -> AlertTableInterpreter
                    -> AlertLoader ;

}
```

I Execution Output Logs

Logs for pipeline gtfs-static-and-rt.jv:

```
[ GtfsStaticAndRealtimePipeline ]  Overview :
        Blocks  (59  blocks  with  13  pipes ) :
->  GTFSExtractor  ( HttpExtractor )
->  ZipArchiveInterpreter  ( ArchiveInterpreter )
->  AgencyFilePicker  ( FilePicker )
->  AgencyTextFileInterpreter  ( TextFileInterpreter )
->  AgencyCSVInterpreter  ( CSVInterpreter )
->  AgencyTableInterpreter  ( TableInterpreter )
->  AgencyLoader  ( SQLiteLoader )
->  CalendarDatesFilePicker  ( FilePicker )
->  CalendarDatesTextFileInterpreter  ( TextFileInterpreter )
->  CalendarDatesCSVInterpreter  ( CSVInterpreter )
->  CalendarDatesTableInterpreter  ( TableInterpreter )
->  CalendarDatesLoader  ( SQLiteLoader )
->  CalendarFilePicker  ( FilePicker )
->  CalendarTextFileInterpreter  ( TextFileInterpreter )
->  CalendarCSVInterpreter  ( CSVInterpreter )
->  CalendarTableInterpreter  ( TableInterpreter )
->  CalendarLoader  ( SQLiteLoader )
->  FeedInfoFilePicker  ( FilePicker )
->  FeedInfoTextFileInterpreter  ( TextFileInterpreter )
->  FeedInfoCSVInterpreter  ( CSVInterpreter )
->  FeedInfoTableInterpreter  ( TableInterpreter )
->  FeedInfoLoader  ( SQLiteLoader )
->  RoutesFilePicker  ( FilePicker )
->  RoutesTextFileInterpreter  ( TextFileInterpreter )
->  RoutesCSVInterpreter  ( CSVInterpreter )
->  RoutesTableInterpreter  ( TableInterpreter )
->  RoutesLoader  ( SQLiteLoader )
->  ShapesFilePicker  ( FilePicker )
->  ShapesTextFileInterpreter  ( TextFileInterpreter )
->  ShapesCSVInterpreter  ( CSVInterpreter )
->  ShapesTableInterpreter  ( TableInterpreter )
->  ShapesLoader  ( SQLiteLoader )
->  StopTimesFilePicker  ( FilePicker )
->  StopTimesTextFileInterpreter  ( TextFileInterpreter )
->  StopTimesCSVInterpreter  ( CSVInterpreter )
->  StopTimesTableInterpreter  ( TableInterpreter )
->  StopTimesLoader  ( SQLiteLoader )
->  StopsFilePicker  ( FilePicker )
->  StopsTextFileInterpreter  ( TextFileInterpreter )
->  StopsCSVInterpreter  ( CSVInterpreter )
->  StopsTableInterpreter  ( TableInterpreter )
->  StopsLoader  ( SQLiteLoader )
->  TripsFilePicker  ( FilePicker )
->  TripsTextFileInterpreter  ( TextFileInterpreter )
->  TripsCSVInterpreter  ( CSVInterpreter )
->  TripsTableInterpreter  ( TableInterpreter )
->  TripsLoader  ( SQLiteLoader )
->  GTFSRTTripUpdateFeedExtractor  ( HttpExtractor )
->  GtfsRTTripUpdateInterpreter  ( GtfsRTInterpreter )
->  TripUpdateTableInterpreter  ( TableInterpreter )
->  TripUpdateLoader  ( SQLiteLoader )
->  GTFSRTVehiclePositionFeedExtractor  ( HttpExtractor )
->  GtfsRTVehiclePositionInterpreter  ( GtfsRTInterpreter )
->  VehiclePositionTableInterpreter  ( TableInterpreter )
->  VehicleLoader  ( SQLiteLoader )
->  GTFSRTAlertFeedExtractor  ( HttpExtractor )
->  GtfsRTAlertInterpreter  ( GtfsRTInterpreter )
->  AlertTableInterpreter  ( TableInterpreter )
->  AlertLoader  ( SQLiteLoader )
[ GTFSRTAlertFeedExtractor ]  Fetching  raw  data  from  https :// proxy . transport . data . gouv . fr / resource / bibus - brest
      - gtfs - rt - alerts
[ GTFSRTAlertFeedExtractor ]  Successfully  fetched  raw  data
[ GTFSRTAlertFeedExtractor ]  Execution  duration :  115  ms .
[ GtfsRTAlertInterpreter ]  Parsing  raw  gtfs - rt  feed  data  as  Alerts "
[ GtfsRTAlertInterpreter ]  Execution  duration :  3  ms .
[ AlertTableInterpreter ]  Matching  header  with  provided  column  names
[ AlertTableInterpreter ]  Validating  47  row ( s)  according  to  the  column  types
[ AlertTableInterpreter ]  Validation  completed ,  the  resulting  table  has  47  row (s)  and  7  column (s)
[ AlertTableInterpreter ]  Execution  duration :  1  ms .
[ AlertLoader ]  Opening  database  file  ./ gtfs - static - and - rt . sqlite
[ AlertLoader ]  Creating  table  " rt_alert "
[ AlertLoader ]  Inserting  47  row (s)  into  table  " rt_alert "
[ AlertLoader ]  The  data  was  successfully  loaded  into  the  database
[ AlertLoader ]  Execution  duration :  7  ms .
[ GTFSRTVehiclePositionFeedExtractor ]  Fetching  raw  data  from  https :// proxy . transport . data . gouv . fr / resource /
      bibus - brest - gtfs -rt - vehicle - position
[ GTFSRTVehiclePositionFeedExtractor ]  Successfully  fetched  raw  data
[ GTFSRTVehiclePositionFeedExtractor ]  Execution  duration :  88  ms .
```

```
[ GtfsRTVehiclePositionInterpreter ]  Parsing raw gtfs - rt feed data as VehiclePosition "
[ GtfsRTVehiclePositionInterpreter ]  Execution duration : 1 ms .
[ VehiclePositionTableInterpreter ]  Matching header with provided column names
[ VehiclePositionTableInterpreter ]  Validating 33 row (s) according to the column types
[ VehiclePositionTableInterpreter ]  Validation completed , the resulting table has 33 row (s ) and 10 column (s)
[ VehiclePositionTableInterpreter ]  Execution duration : 0 ms .
[ VehicleLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ VehicleLoader ] Creating table " rt_vehicle_position "
[ VehicleLoader ] Inserting 33 row (s ) into table " rt_vehicle_position "
[ VehicleLoader ] The data was successfully loaded into the database
[ VehicleLoader ] Execution duration : 2 ms .
[ GTFSRTTripUpdateFeedExtractor ] Fetching raw data from https :// proxy . transport . data . gouv . fr / resource / bibus - brest - gtfs -rt - trip - update
[ GTFSRTTripUpdateFeedExtractor ] Successfully fetched raw data
[ GTFSRTTripUpdateFeedExtractor ] Execution duration : 168 ms .
[ GtfsRTTripUpdateInterpreter ]  Parsing raw gtfs - rt feed data as TripUpdate "
[ GtfsRTTripUpdateInterpreter ]  Execution duration : 29 ms .
[ TripUpdateTableInterpreter ]  Matching header with provided column names
[ TripUpdateTableInterpreter ]  Validating 8917 row (s ) according to the column types
[ TripUpdateTableInterpreter ]  Validation completed , the resulting table has 8917 row ( s) and 10 column (s)
[ TripUpdateTableInterpreter ]  Execution duration : 6 ms .
[ TripUpdateLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ TripUpdateLoader ] Creating table " rt_trip_update "
[ TripUpdateLoader ] Inserting 8917 row ( s) into table " rt_trip_update "
[ TripUpdateLoader ] The data was successfully loaded into the database
[ TripUpdateLoader ] Execution duration : 42 ms .
[ GTFSExtractor ] Fetching raw data from https :// ratpdev - mosaic - prod - bucket - raw .s3 - eu - west -1. amazonaws . com /11/ exports /1/ gtfs . zip
[ GTFSExtractor ] Successfully fetched raw data
[ GTFSExtractor ] Execution duration : 464 ms .
[ ZipArchiveInterpreter ]  Loading zip file from binary content
[ ZipArchiveInterpreter ]  Execution duration : 93 ms .
[ TripsFilePicker ]  Execution duration : 0 ms .
[ TripsTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ TripsTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ TripsTextFileInterpreter ]  Lines were split successfully , the resulting text file has 7078 lines
[ TripsTextFileInterpreter ]  Execution duration : 3 ms .
[ TripsCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ TripsCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ TripsCSVInterpreter ]  Execution duration : 327 ms .
[ TripsTableInterpreter ]  Matching header with provided column names
[ TripsTableInterpreter ]  Validating 7077 row (s) according to the column types
[ TripsTableInterpreter ]  Validation completed , the resulting table has 7077 row (s ) and 10 column (s)
[ TripsTableInterpreter ]  Execution duration : 2 ms .
[ TripsLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ TripsLoader ] Dropping previous table " static_trips " if it exists
[ TripsLoader ] Creating table " static_trips "
[ TripsLoader ] Inserting 7077 row (s ) into table " static_trips "
[ TripsLoader ] The data was successfully loaded into the database
[ TripsLoader ] Execution duration : 41 ms .
[ StopsFilePicker ]  Execution duration : 0 ms .
[ StopsTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ StopsTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ StopsTextFileInterpreter ]  Lines were split successfully , the resulting text file has 1063 lines
[ StopsTextFileInterpreter ]  Execution duration : 0 ms .
[ StopsCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ StopsCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ StopsCSVInterpreter ]  Execution duration : 36 ms .
[ StopsTableInterpreter ]  Matching header with provided column names
[ StopsTableInterpreter ]  Validating 1062 row (s) according to the column types
[ StopsTableInterpreter ]  Validation completed , the resulting table has 1062 row (s ) and 14 column (s)
[ StopsTableInterpreter ]  Execution duration : 0 ms .
[ StopsLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ StopsLoader ] Dropping previous table " static_stops " if it exists
[ StopsLoader ] Creating table " static_stops "
[ StopsLoader ] Inserting 1062 row (s ) into table " static_stops "
[ StopsLoader ] The data was successfully loaded into the database
[ StopsLoader ] Execution duration : 9 ms .
[ StopTimesFilePicker ]  Execution duration : 0 ms .
[ StopTimesTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ StopTimesTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ StopTimesTextFileInterpreter ]  Lines were split successfully , the resulting text file has 159967 lines
[ StopTimesTextFileInterpreter ]  Execution duration : 17 ms .
[ StopTimesCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ StopTimesCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ StopTimesCSVInterpreter ]  Execution duration : 4363 ms .
[ StopTimesTableInterpreter ]  Matching header with provided column names
[ StopTimesTableInterpreter ]  Validating 159966 row ( s) according to the column types
[ StopTimesTableInterpreter ]  Validation completed , the resulting table has 159966 row (s) and 8 column (s)
[ StopTimesTableInterpreter ]  Execution duration : 19 ms .
[ StopTimesLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ StopTimesLoader ] Dropping previous table " static_stop_times " if it exists
[ StopTimesLoader ] Creating table " static_stop_times "
[ StopTimesLoader ] Inserting 159966 row (s) into table " static_stop_times "
[ StopTimesLoader ] The data was successfully loaded into the database
[ StopTimesLoader ] Execution duration : 591 ms .
[ ShapesFilePicker ]  Execution duration : 0 ms .
[ ShapesTextFileInterpreter ]  Decoding file content using encoding " utf -8"
```

```
[ ShapesTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ ShapesTextFileInterpreter ]  Lines were split successfully , the resulting text file has 81398 lines
[ ShapesTextFileInterpreter ]  Execution duration : 7 ms .
[ ShapesCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ ShapesCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ ShapesCSVInterpreter ]  Execution duration : 2452 ms .
[ ShapesTableInterpreter ]  Matching header with provided column names
[ ShapesTableInterpreter ]  Validating 81397 row ( s) according to the column types
[ ShapesTableInterpreter ]  Validation completed , the resulting table has 81397 row (s) and 4 column (s)
[ ShapesTableInterpreter ]  Execution duration : 9 ms .
[ ShapesLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ ShapesLoader ] Dropping previous table " static_shapes " if it exists
[ ShapesLoader ] Creating table " static_shapes "
[ ShapesLoader ] Inserting 81397 row (s) into table " static_shapes "
[ ShapesLoader ] The data was successfully loaded into the database
[ ShapesLoader ] Execution duration : 210 ms .
[ RoutesFilePicker ]  Execution duration : 0 ms .
[ RoutesTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ RoutesTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ RoutesTextFileInterpreter ]  Lines were split successfully , the resulting text file has 66 lines
[ RoutesTextFileInterpreter ]  Execution duration : 0 ms .
[ RoutesCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ RoutesCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ RoutesCSVInterpreter ]  Execution duration : 3 ms .
[ RoutesTableInterpreter ]  Matching header with provided column names
[ RoutesTableInterpreter ]  Validating 65 row (s) according to the column types
[ RoutesTableInterpreter ]  Validation completed , the resulting table has 65 row (s) and 10 column ( s)
[ RoutesTableInterpreter ]  Execution duration : 0 ms .
[ RoutesLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ RoutesLoader ] Dropping previous table " static_routes " if it exists
[ RoutesLoader ] Creating table " static_routes "
[ RoutesLoader ] Inserting 65 row (s) into table " static_routes "
[ RoutesLoader ] The data was successfully loaded into the database
[ RoutesLoader ] Execution duration : 3 ms .
[ FeedInfoFilePicker ]  Execution duration : 0 ms .
[ FeedInfoTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ FeedInfoTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n /
[ FeedInfoTextFileInterpreter ]  Lines were split successfully , the resulting text file has 2 lines
[ FeedInfoTextFileInterpreter ]  Execution duration : 0 ms .
[ FeedInfoCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ FeedInfoCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ FeedInfoCSVInterpreter ]  Execution duration : 0 ms .
[ FeedInfoTableInterpreter ]  Matching header with provided column names
[ FeedInfoTableInterpreter ]  Validating 1 row (s) according to the column types
[ FeedInfoTableInterpreter ]  Validation completed , the resulting table has 1 row (s ) and 8 column ( s)
[ FeedInfoTableInterpreter ]  Execution duration : 0 ms .
[ FeedInfoLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ FeedInfoLoader ] Dropping previous table " static_feed_info " if it exists
[ FeedInfoLoader ] Creating table " static_feed_info "
[ FeedInfoLoader ] Inserting 1 row (s ) into table " static_feed_info "
[ FeedInfoLoader ] The data was successfully loaded into the database
[ FeedInfoLoader ] Execution duration : 2 ms .
[ CalendarFilePicker ]  Execution duration : 1 ms .
[ CalendarTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ CalendarTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n /
[ CalendarTextFileInterpreter ]  Lines were split successfully , the resulting text file has 43 lines
[ CalendarTextFileInterpreter ]  Execution duration : 0 ms .
[ CalendarCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ CalendarCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ CalendarCSVInterpreter ]  Execution duration : 1 ms .
[ CalendarTableInterpreter ]  Matching header with provided column names
[ CalendarTableInterpreter ]  Validating 42 row (s ) according to the column types
[ CalendarTableInterpreter ]  Validation completed , the resulting table has 42 row ( s) and 10 column (s)
[ CalendarTableInterpreter ]  Execution duration : 0 ms .
[ CalendarLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ CalendarLoader ] Dropping previous table " static_calendar " if it exists
[ CalendarLoader ] Creating table " static_calendar "
[ CalendarLoader ] Inserting 42 row ( s) into table " static_calendar "
[ CalendarLoader ] The data was successfully loaded into the database
[ CalendarLoader ] Execution duration : 3 ms .
[ CalendarDatesFilePicker ]  Execution duration : 0 ms .
[ CalendarDatesTextFileInterpreter ]  Decoding file content using encoding " utf -8"
[ CalendarDatesTextFileInterpreter ]  Splitting lines using line break /\ r ?\ n/
[ CalendarDatesTextFileInterpreter ]  Lines were split successfully , the resulting text file has 153 lines
[ CalendarDatesTextFileInterpreter ]  Execution duration : 0 ms .
[ CalendarDatesCSVInterpreter ]  Parsing raw data as CSV using delimiter " ,"
[ CalendarDatesCSVInterpreter ]  Parsing raw data as CSV - sheet successful
[ CalendarDatesCSVInterpreter ]  Execution duration : 5 ms .
[ CalendarDatesTableInterpreter ]  Matching header with provided column names
[ CalendarDatesTableInterpreter ]  Validating 152 row (s) according to the column types
[ CalendarDatesTableInterpreter ]  Validation completed , the resulting table has 152 row (s) and 3 column (s )
[ CalendarDatesTableInterpreter ]  Execution duration : 0 ms .
[ CalendarDatesLoader ] Opening database file ./ gtfs - static - and - rt . sqlite
[ CalendarDatesLoader ] Dropping previous table " static_calendar_dates " if it exists
[ CalendarDatesLoader ] Creating table " static_calendar_dates "
[ CalendarDatesLoader ] Inserting 152 row (s) into table " static_calendar_dates "
[ CalendarDatesLoader ] The data was successfully loaded into the database
[ CalendarDatesLoader ] Execution duration : 2 ms .
```

```
[ AgencyFilePicker ]      Execution    duration :    0  ms .
[ AgencyTextFileInterpreter ]        Decoding   file    content   using   encoding   " utf -8"
[ AgencyTextFileInterpreter ]        Splitting      lines    using   line    break   /\ r ?\ n/
[ AgencyTextFileInterpreter ]        Lines   were  split    successfully ,      the   resulting     text   file    has  3  lines
[ AgencyTextFileInterpreter ]        Execution    duration :    0  ms .
[ AgencyCSVInterpreter ]      Parsing    raw  data   as  CSV using   delimiter     " ,"
[ AgencyCSVInterpreter ]      Parsing    raw  data   as  CSV - sheet  successful
[ AgencyCSVInterpreter ]      Execution    duration :    1  ms .
[ AgencyTableInterpreter ]       Matching    header   with   provided    column  names
[ AgencyTableInterpreter ]       Validating      2  row ( s)  according    to   the   column  types
[ AgencyTableInterpreter ]       Validation      completed ,   the   resulting     table    has  2  row (s)   and  8  column (s)
[ AgencyTableInterpreter ]       Execution    duration :    0  ms .
[ AgencyLoader ]   Opening   database   file     ./ gtfs - static - and - rt . sqlite
[ AgencyLoader ]   Dropping   previous    table    " static_agency "     if    it    exists
[ AgencyLoader ]   Creating    table    " static_agency "
[ AgencyLoader ]   Inserting      2  row (s)   into    table    " static_agency "
[ AgencyLoader ]   The  data   was  successfully      loaded   into    the   database
[ AgencyLoader ]   Execution    duration :    2  ms .
[ GtfsStaticAndRealtimePipeline ]          Execution    duration :    9136  ms .
```

References

Antrim, A., Barbeau, S. J., et al. (2013). The many uses of gtfs data–opening the door to transit and multimodal applications. *Location-Aware Information Systems Laboratory at the University of South Florida*, *4*.

Barbeau, S. J. (2018). Quality control-lessons learned from the deployment and evaluation of GTFS-realtime feeds. *97th Annual Meeting of the Transportation Research Board, Washington, DC*.

Braunschweig, K., Eberius, J., Thiele, M., & Lehner, W. (2012). The state of open data. *Limits of current open data platforms*, *1*, 72–72.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.

Conradie, P., & Choenni, S. (2014). On the barriers for local government releasing open data. *Government Information Quarterly*, *31*, 10–17. https://doi.org/10.1016/j.giq.2014.01.003

Dalpiaz, F., & Brinkkemper, S. (2018). Agile requirements engineering with user stories. *2018 IEEE 26th International Requirements Engineering Conference (RE)*, 506–507. https://doi.org/10.1109/re.2018.00075

Dimitrakopoulos, G., & Demestichas, P. (2010). Intelligent transportation systems. *IEEE Vehicular Technology Magazine*, *5*(1), 77–84. https://doi.org/10.1109/mvt.2009.935537

European Commission. (2003, December 31). DIRECTIVE 2003/98/EC of the European Parliament and of the Council of 17 november 2003 on the re-use of public sector information. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32003L0098&from=en

European Commission. (2010, July 7). Directive 2010/40/EU of the European Parliament and of the Council of 7 july 2010 on the framework for the deployment of intelligent transport systems in the field of road transport and for interfaces with other modes of transport. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32010L0040&from=EN

European Commission. (2021). *Monitoring and harmonisation of national access points* [European ITS platform]. Retrieved December 28, 2022, from https://www.its-platform.eu/achievement/monitoring-harmonisation-of-naps/

European Commission. (2022). *What is open data?* Retrieved December 28, 2022, from https://data.europa.eu/elearning/en/module1/#/id/co-01

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1996). Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, Deutsche Übersetzung von Dirk Riehle. In *Programmers choice* (pp. 239–253). Addison–Wesley–Longman.

Garlan, D., & Shaw, M. (1993). An Introduction To Software Architecture. In *Serieson software engineering and knowledge engineering* (pp. 1–39). World Scientific. https://doi.org/10.1142/9789812798039_0001

Goldstein, B., & Dyson, L. (Eds.). (2013). *Beyond transparency: Open data and the future of civic innovation*. Code for America Press.

Goliszek, S., & Połom, M. (2016). The use of general transit feed specification (GTFS) application to identify deviations in the operation of public transport at morning peak hours on the example of szczecin. *Europa XXI*, *31*, 51–60. https://doi.org/10.7163/Eu21.2016.31.4

Google. (2022a). *GTFS realtime reference | realtime transit*. Retrieved December 29, 2022, from https : / / developers . google . com / transit / gtfs - realtime / reference

Google. (2022b). *GTFS static reference | static transit*. Retrieved December 29, 2022, from https://developers.google.com/transit/gtfs/reference

Harding, M., & Davies, N. (2012). Citadel: A community platform for archiving travel data. *Proceedings of the 6th ACM workshop on Next generation mobile computing for dynamic personalised travel planning - Sense Transport '12*, 7. https://doi.org/10.1145/2307874.2307881

Heltweg, P., & Riehle, D. (2022). Challenges to open collaborative data engineering. https://doi.org/10.5281/zenodo.6598447

Janssen, K. (2011). The influence of the PSI directive on open government data: An overview of recent developments. *Government Information Quarterly*, *28* (4), 446–456. https://doi.org/10.1016/j.giq.2011.01.004

Kaeoruean, K., Phithakkitnukoon, S., Demissie, M. G., Kattan, L., & Ratti, C. (2020). Analysis of demand–supply gaps in public transit systems based on census and GTFS data: A case study of calgary, canada. *Public Transport*, *12* (3), 483–516. https://doi.org/10.1007/s12469-020-00252-y

Koetsier, S., van Heerden, Q., & Maditse, N. K. (2017). Using the GTFS Format to Improve Public Transport Data Accessibility In Gauteng. *South Africa*.

Kujala, R., Weckström, C., Darst, R. K., Mladenović, M. N., & Saramäki, J. (2018). A collection of public transport network data sets for 25 cities. *Scientific Data*, *5* (1), 180089. https://doi.org/10.1038/sdata.2018.89

Lim, A., Sharma, S., Bhaskar, A., & Arkatkar, S. (2019). An open source framework for GTFS data analytics: Case study using the Brisbane TransLink network.

Mahajan, V., Kuehnel, N., Intzevidou, A., Cantelmo, G., Moeckel, R., & Antoniou, C. (2022). Data to the people: A review of public and propriet-

ary data for transport models. *Transport Reviews*, *42* (4), 415–440. https://doi.org/10.1080/01441647.2021.1977414

Molloy, J. C. (2011). The open knowledge foundation: Open data means better science. *PLoS Biology*, *9* (12), e1001195. https://doi.org/10.1371/journal.pbio.1001195

Noonan, R. E. (1985). An algorithm for generating abstract syntax trees. *Computer Languages*, *10* (3), 225–236. https://doi.org/10.1016/0096-0551(85)90018-9

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: A roadmap. *Proceedings of the Conference on The Future of Software Engineering*, 35–46. https://doi.org/10.1145/336512.336523

Obama, B. (2009, January 21). *Transparency and open government*. Retrieved December 28, 2022, from https://obamawhitehouse.archives.gov/the-press-office/transparency-and-open-government

Pandit, P., & Tahiliani, S. (2015). AgileUAT: A framework for user acceptance testing based on user stories and acceptance criteria. *International Journal of Computer Applications*, *120* (10), 16–21. https://doi.org/10.5120/21262-3533

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, *17* (4), 40–52. https://doi.org/10.1145/141874.141884

Professorship of Open Source Software at the University of Erlangen. (2022). *The JValue project – on a mission to make using open data easy, safe, and reliable*. Retrieved December 30, 2022, from https://jvalue.org/

Qureshi, K. N., & Abdullah, A. H. (2013). A survey on intelligent transportation systems. *Middle-East Journal of Scientific Research*, *15* (5), 629–642.

Ramamoorthy, C., & Wah, B. (1989). Knowledge and data engineering. *IEEE Transactions on Knowledge and Data Engineering*, *1* (1), 9–16. https://doi.org/10.1109/69.43400

SQLite Consortium. (2018). *Recommended storage format*. Retrieved March 5, 2023, from https://www.sqlite.org/locrsf.html

Wu, J., Du, B., Gong, Z., Wu, Q., Shen, J., Zhou, L., & Cai, C. (2022). A GTFS data acquisition and processing framework and its application to train delay prediction. *International Journal of Transportation Science and Technology*, S2046043022000090. https://doi.org/10.1016/j.ijtst.2022.01.005

Zuiderwijk, A., Janssen, M., & Davis, C. (2014). Innovation with open data: Essential elements of open data ecosystems. *Information Polity*, *19* (1), 17–33. https://doi.org/10.3233/ip-140329