California
Road Charge

Public Private Roads Project

APPENDIX
E

Pilot System Report

Appendix was prepared to follow industry guidelines and standards for accessibility and usability to the extent feasible. If you have difficulty accessing content in this document, please contact Caltrans for assistance.

California Road Charge

Public/Private Roads Project

PRIME Batch Updates thru Pilot Design and Testing

Version: 2.0
Date: October 4, 2023
Task: 3.b.1 & 3.b.2, Agreement 82A0002

Prepared By:

WSP

Document Control

File Name:	Task3.b.1-3.b.2_PRIME-Batch-Updates-Thru-Pilot-Design-and-Testing_10042023.docx	
Version Number:	2.0	
Created By:	WSP	August 28, 2023
Reviewed By:	Caltrans	September 1, 2023
Modified By:	WSP	October 2, 2023
Approved By:	Caltrans	October 4, 2023

CONTENTS

1 Introduction 7
2 System Scope 8
3 Database Configuration 15
3.1 Agents 15
3.2 BitBrew_2023PILOT 17
3.3 Finance 19
3.4 Security 19
3.5 Schema and Database Configuration 19
4 Data Exchanges with Partners 21
4.1 BitBrew 21
4.2 myMiles 23
4.3 TCA 25
5 Reporting 26
5.1 Transaction Summary Message 26
5.2 Monthly Reporting 26
6 PRIME Web Portal 28
6.1 Administrative Portal 28
6.2 Analytics Engine 30
6.3 Umbraco Web Portal 30
7 Amazon Web Services 33
7.1 Data Sharing (S3) 33
7.1.1 BitBrew Backups 33
7.2 Web Hosting (Elastic Beanstalk) 33
7.3 Domain Registration and HTTPS (Route 53) 34
7.4 BitBrew Real-Time Consumer (DynamoDB) 34
7.5 BitBrew Processing (Lambda Services) 34
8 Security 36
8.1 Amazon Web Services 36
8.2 BitBrew 37
8.3 PRIME Web Portal 37
8.4 Snowflake 38

9 Maintenance 40
9.1 GitHub Version Control 40
9.1.1 Core Package 41
9.1.2 Client Package 42
9.1.3 Web Package 42
9.1.4 Working Package 42
9.2 System Environments 42
Appendix A: Database Schema and Definitions 44
A.1 Agents 44
A.2 BitBrew_2023Pilot 47
A.3 Finance 56
A.4 Security 57
A.5 Web 57
Appendix B: Schema Definition File Layout 59

Figures

Figure 1: California Road Charge Pilot Functional Architecture Diagram 8
Figure 2: California Road Charge Pilot Physical Architecture Diagram 9
Figure 3: California Road Charge Pilot Functional Architecture Diagram/ Data Collection 10
Figure 4: California Road Charge Pilot Functional Architecture Diagram/ Transaction Processing 11
Figure 5: California Road Charge Pilot Functional Architecture Diagram/ Account Management 12
Figure 6: California Road Charge Pilot Functional Architecture Diagram/ Administration System 13
Figure 7: California Road Charge Pilot Functional Architecture Diagram/ Data Clearinghouse 14
Figure 8: Pilot Snowflake Database AGENTS Schema and Relationships 16
Figure 9: Pilot Snowflake Database BITBREW_2023PILOT Schema and Relationships 17
Figure 10: BitBew Data Pipeline 22
Figure 11: Transaction Summary Message Generator in PRIME Administrative Portal 26
Figure 12: PRIME administrative portal - Participant Page (sample) 29
Figure 13: PRIME administrative portal - MRO management page (redacted) 29
Figure 14: Umbraco backend management site - Content section 30
Figure 15: TCA sub-pilot participant portal 31

Figure 16: Pilot Incentives Tracker Admin Member/Management Page 32
Figure 17: Pilot Incentives Tracker Admin Page: Editing a Participant's Incentives 32
Figure 18: Credential YAML Layout (sample) 36
Figure 19 Snowflake IDE 38
Figure 20 GitHub Repository 40
Figure 21: Python Package Hierarchy 41
Figure 22: Excerpt of YAML Schema 59

Tables

Table 1: Schema Summary and PII Content 15
Table 2: Business Partners 44
Table 3: Certification 44
Table 4: Participant-Business Partner Client Map 45
Table 5: MRO 45
Table 6: Participant 46
Table 7: Recorder (reconciliation table) 46
Table 8: Vehicle 47
Table 9: Connect Messages 47
Table 10: Device Connect 48
Table 11: Event Load Log 48
Table 12: GPS Event 49
Table 13: Heartbeat 49
Table 14; Lost Device 50
Table 15: Trip Event 50
Table 16: Trip Segment Event 51
Table 17: Events (View) 52
Table 18: Fuel by Trip (View) 52
Table 19: Fuel by Trip Seg (View) 53
Table 20: Road_User_Charge_Rates 56
Table 21: Users 57
Table 22: MRO Last Seen (View) 57

Acronyms & Terms

Acronym / Term	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
AWS	Amazon Web Services
BRD	Business Requirements Document
Caltrans	California Department of Transportation
ConOps	Concept of Operations
CSP	Customer Support Plan
CSV	Comma Separated Variable
EPA	Environmental Protection Agency
FAQ	Frequently Asked Question
FIPS	Federal Information Processing Standards
GPS	Global Positioning System
HTTPS	Hypertext Transfer Protocol Secure
JSON	Java Script Object Notation
MPG	Miles per Gallon
MRO	Mileage Reporting Option
OAuth	Open Authorization
OBD	On-board Diagnostics
OpsPlan	Operations Plan
PII	Personally Identifiable Information
PRIME	Platform for Road Charge Innovation and Mobility Evolution
SQL	Structured Query Language
SSL	Secure Socket Layer
TCA	Transportation Corridor Agencies (or “the Toll Roads’)
TSM	Transaction Summary Messages
URL	Uniform Resource Locator
VIN	Vehicle Identification Number
VMT	Vehicle Miles Traveled
YAML	YAML A’int Markup Language

1 Introduction

The California Road Charge Public/Private Roads Pilot is a 6-month road charge pilot evaluating the feasibility of geolocation technologies to accurately and consistently capture and differentiate travel on California public, private, and tribal roads, as well as travel that occurs out of state. The pilot officially launched on March 29th, 2023, and operates through September 30, 2023.

The main pilot consists of 254 rural and tribal participants, with "myMiles" serving as the pilot account manager. Additionally, 35 participants who are active customers of California's Transportation Corridor Agencies (TCA) are participating in a sub-pilot with TCA to evaluate the potential for a tolling entity to act as a road charge account manager. Each account manager, myMiles and TCA, provides a distinct Participant Portal that serves as the frontline interface with the participant, providing participants access to their pilot account, monthly road charge (simulated) statements, information on their eligible, earned, and paid incentive activities, and general information about the pilot and their participation.

This document includes a complete system overview of the overall system used to manage the California Road Charge Public/Private Roads Project pilot. The document provides an overview of the system architecture, database structure, third-party tools, security protocols, and communication protocols used to capture and report pilot data during pilot operations.

2 System Scope

The pilot system is logically separated into subsystems, each made up of components that perform specific functionality. Interfaces between subsystems are standardized to support flexibility and scalability.

The architecture for the pilot includes five key subsystems: Data Collection, Transaction Processing, Account Management, Administration, and the Clearinghouse. The functional and physical architectures are described in Figure 1 and Figure 2 below.

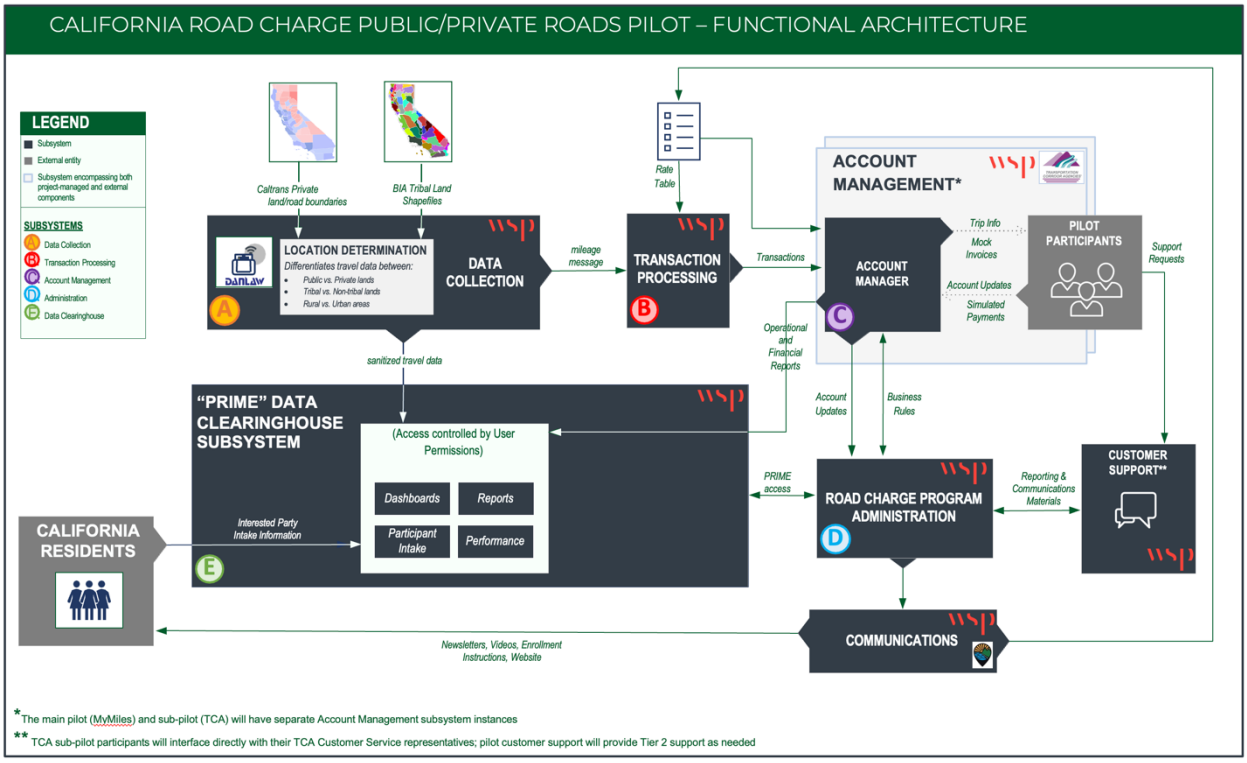


Figure 1: California Road Charge Pilot Functional Architecture Diagram

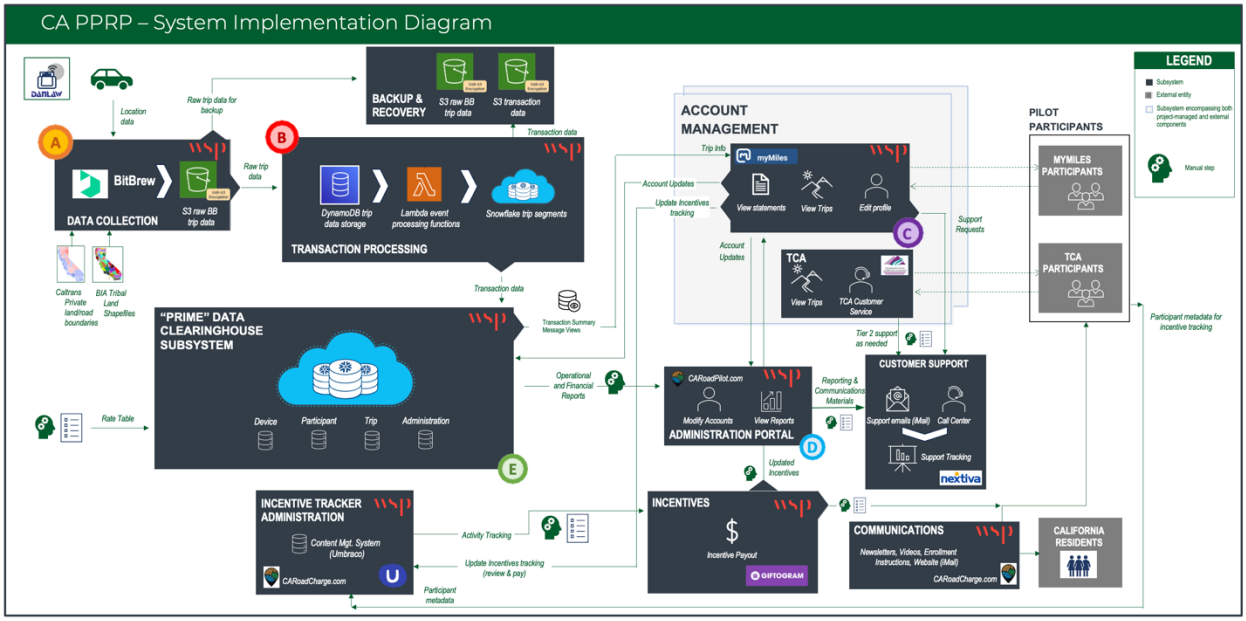


Figure 2: California Road Charge Pilot Physical Architecture Diagram

Each subsystem performs a series of functions to support the pilot, as defined in the project's requirements suite. Communications between subsystems are standardized as defined in the project's Interface Control Document. The architecture diagram, description, project team operator, and where data is stored in the system for each subsystem is described below.

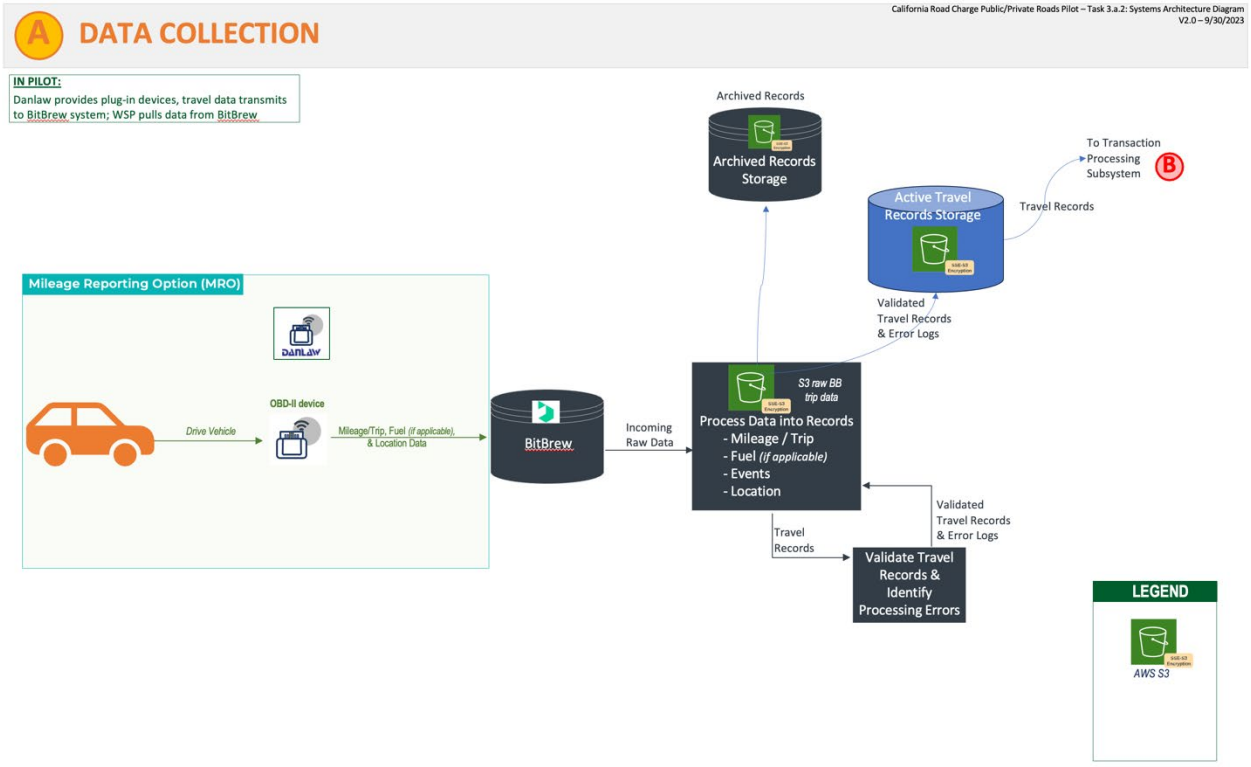


Figure 3: California Road Charge Pilot Functional Architecture Diagram/ Data Collection

- **DATA COLLECTION SUBSYSTEM:** The Data Collection Subsystem interfaces with the participating vehicles to collect travel/trip data. Some of the common data elements captured from the Data Collection Subsystem include vehicle unique identifier, date/time, distance traveled, fuel consumption (if applicable), and detailed vehicle travel location information. Vehicle travel data will be differentiated based on the type of roadway (or non-roadway) the vehicle traveled on during a trip, including public roadways, non-public roadways, off roads, out of state, and federally recognized tribal lands within California. This information is then routed to the Transaction Processor for computation of the vehicle's assessed road charge and fuels tax credit (if applicable).
    - **Operator:** WSP Technical Team manages data collection functionality, including device asset management, collecting participating vehicle travel data (from devices) through contracted Danlaw/BitBrew system, and processing data into system.
    - **Data Storage:** Data collected from devices is first stored in the device itself and in the Danlaw/BitBrew system throughout a trip until the trip is completed. WSP pulls data from the Danlaw/BitBrew system multiple times per day to collect detailed trip data, including trip start/end parameters to evaluate when a trip data set is complete. WSP stores this raw trip data in AWS, then processes and applies road type information to the data in AWS, stores a copy of the processed trip record in AWS, and finally sends to Snowflake for formal pilot database storage and continued processing for transactions.

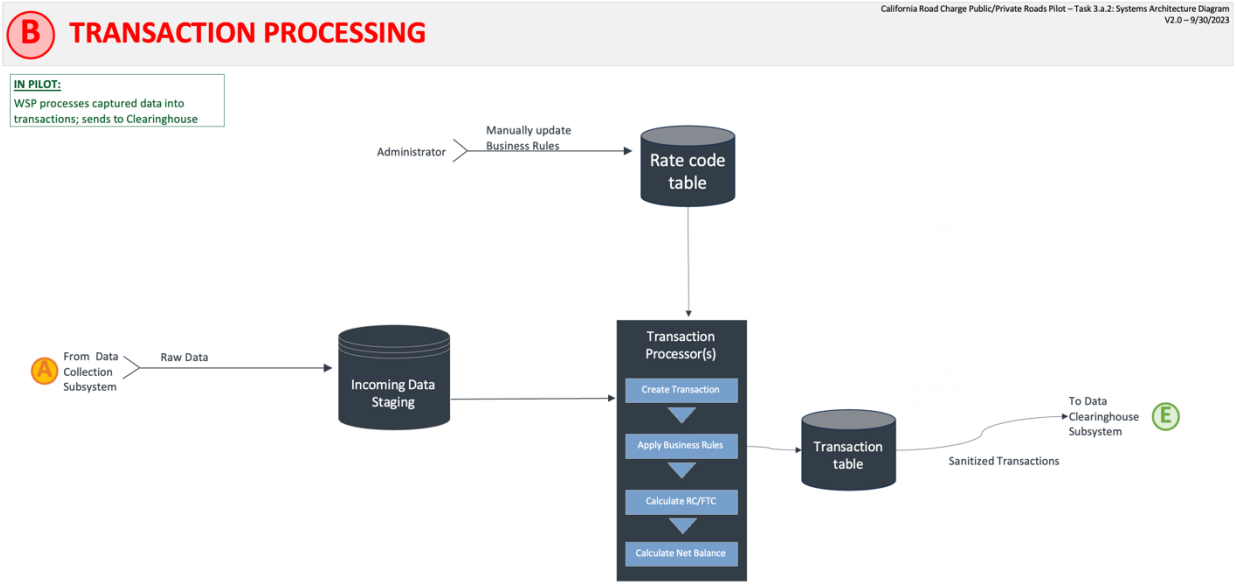


Figure 4: California Road Charge Pilot Functional Architecture Diagram/ Transaction Processing

- **TRANSACTION PROCESSING SUBSYSTEM:** The Transaction Processing Subsystem collects vehicle travel data from the Data Collection Subsystem, processes the data into transactions, applies road charge and fuels tax credit rates based on the location differentiation (using a pre-defined rate table), and calculates the net road charge owed.
    - **Operator:** WSP Technical Team takes data from data collection subsystem and processes into transactions, applying calculations and applicable rates as needed.
    - **Data Storage:** WSP stores all transactions in Snowflake database, as well as the rate table(s) used for calculations.

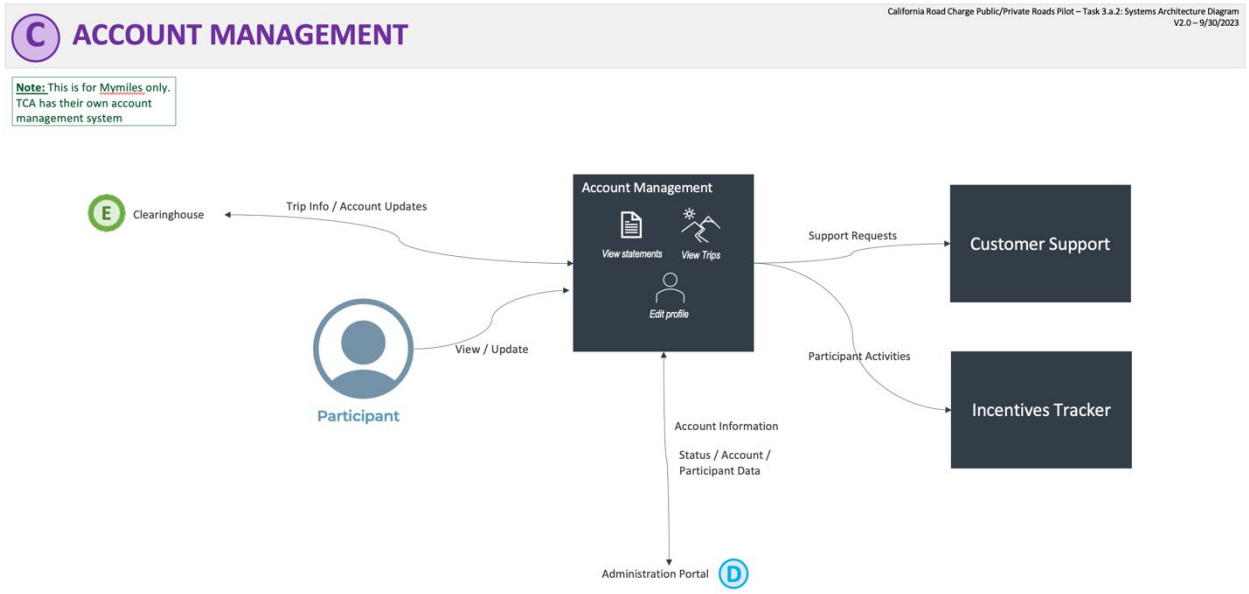
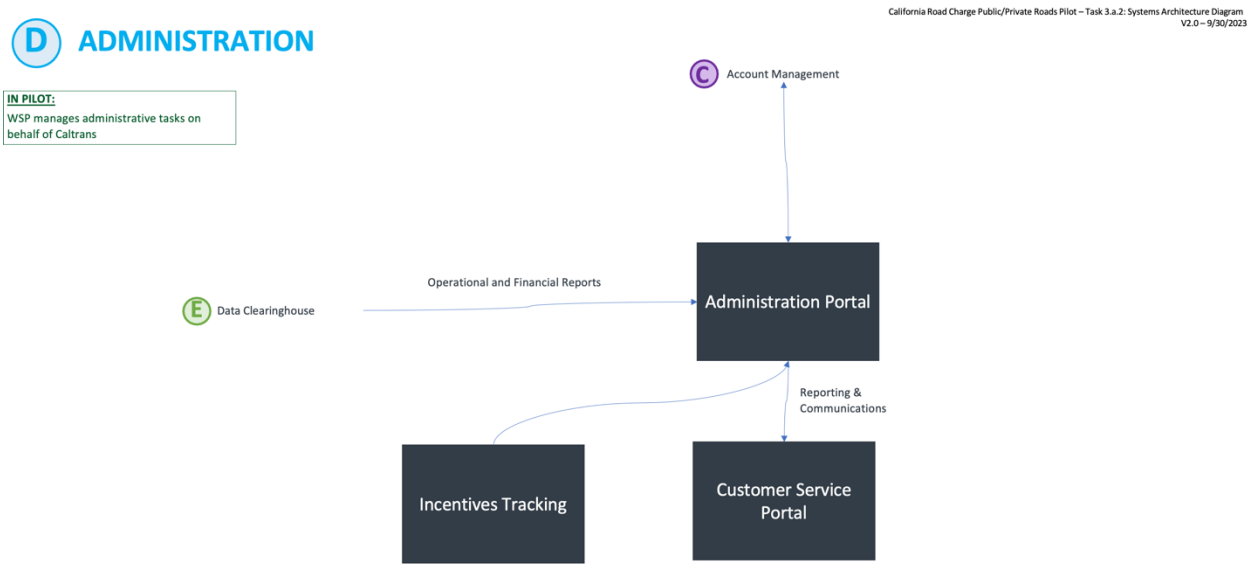


Figure 5: California Road Charge Pilot Functional Architecture Diagram/ Account Management

- **ACCOUNT MANAGEMENT SUBSYSTEM:** The Account Management Subsystem is the frontline interface to the participant. The Account Management Subsystem provides information to participants on their respective pilot accounts and monthly simulated road charge invoices. The Account Management Subsystem provides convenient account access through web portals, mobile apps, and customer support services for participants to interact with the subsystem.
    - **Operator:**
        - WSP manages and hosts the "myMiles" account management system and operations for main pilot participants.
        - TCA manages and hosts the "TCA" account management functionality for TCA sub-pilot participants.
    - **Data Storage:**
        - myMiles pulls transaction information from Snowflake to generate monthly statements, and pulls live account information from Snowflake to populate other aspects of the myMiles account management platform as needed. myMiles also pulls data from WSP-hosted Umbraco database for participant incentive management and tracking information.
        - WSP posts daily transaction summary messages to a WSP-hosted Amazon Web Services (AWS) S3 bucket for TCA to pull to generate monthly statements based on their customers' monthly invoicing/statement generation cadence.
        - TCA sub-pilot participants' general account information, participant agreement and privacy policy, and incentive tracking is stored in WSP-

hosted Umbraco participant portal instance, no interaction/interface with TCA required. Participants have unique logins to access this portal.

Figure 6: California Road Charge Pilot Functional Architecture Diagram/ Administration System



- **ADMINISTRATION SUBSYSTEM:** The Administration Subsystem provides an interface to those managing the pilot to access pilot information, results, and reports.
    - **Operator:** WSP hosts and manages all pilot operations administrative functionality, including participation status, vehicle and device information, pilot reporting, business rules, and general pilot compliance.
    - **Data Storage:** WSP stores all administration subsystem data in Snowflake database for all participants and pilot operations data.

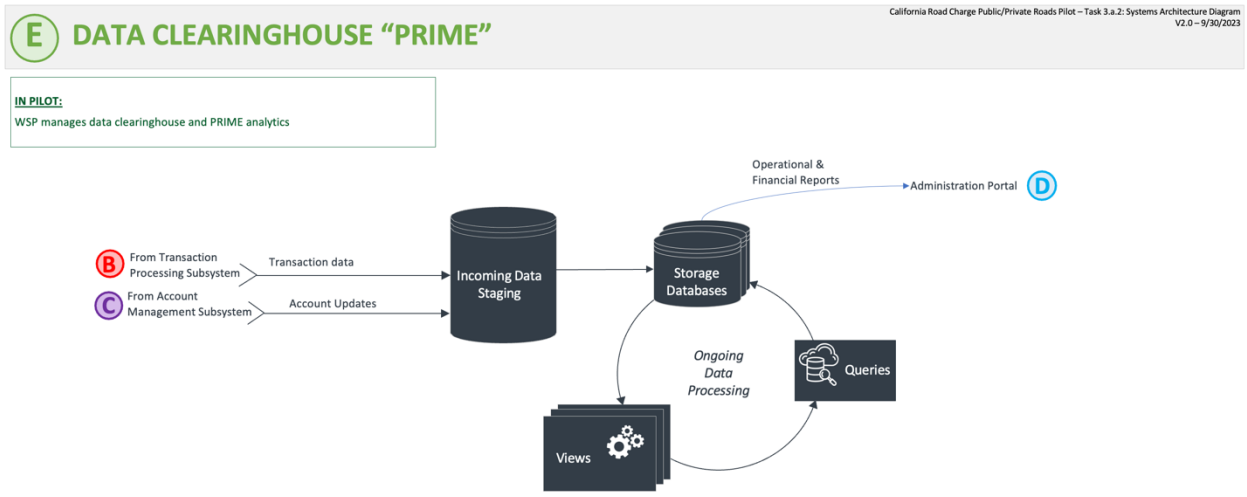


Figure 7: California Road Charge Pilot Functional Architecture Diagram/ Data Clearinghouse

- **DATA CLEARINGHOUSE "PRIME" SUBSYSTEM:** The Data Clearinghouse Subsystem provides the central data repository for collecting aggregate pilot data and was used in a previous pilot but suspended for this pilot.
    - **Operator:** WSP hosts and manages PRIME clearinghouse functionality.
    - **Data Storage:** WSP stores all PRIME clearinghouse data in Snowflake database.

3 Database Configuration

This section provides a summary of the data schemas used by the PRIME system in Snowflake, a cloud-based data warehouse and storage platform. In general, data related to each business partner is contained within its own schema for organizational purposes as well as the potential to more easily accommodate potential future security restrictions on certain portions of the data.

The subsections below provide specific information about the purpose and structure of each schema. In addition, schemas containing PII have been noted in Table 1. If a schema contains PII, the relevant subsection below contains more specific information about the type of PII contained in the schema.

Table 1: Schema Summary and PII Content

Schema	Description	Contains PII
AGENTS	Administrative Information	Yes
BITBREW_2023PILOT	Travel Information from Danlaw Devices	Yes
FINANCE	Road Charge Rate Code Table(s)	No
SECURITY	Web and API Credentials	Yes

3.1 AGENTS

The **AGENTS** schema contains seven tables related to participants, business partners, vehicles, mileage reporting options (MRO), MRO certifications, the relationship between participants and business partners, and the relationship between participants, vehicles, and MROs. The two relationship tables (RECORDER and CLIENT_MAP) exist as reconciliation tables to allow many-to-many relationships between participants, vehicles, and MROs, while maintaining database normalization. The primary purpose of this schema and its tables is to track participant information, their vehicles and MROs, and their relationship with business partners. Figure 8 provides an overview of the table relationships within the schema, and Appendix A provides a schema definition and description for each table.

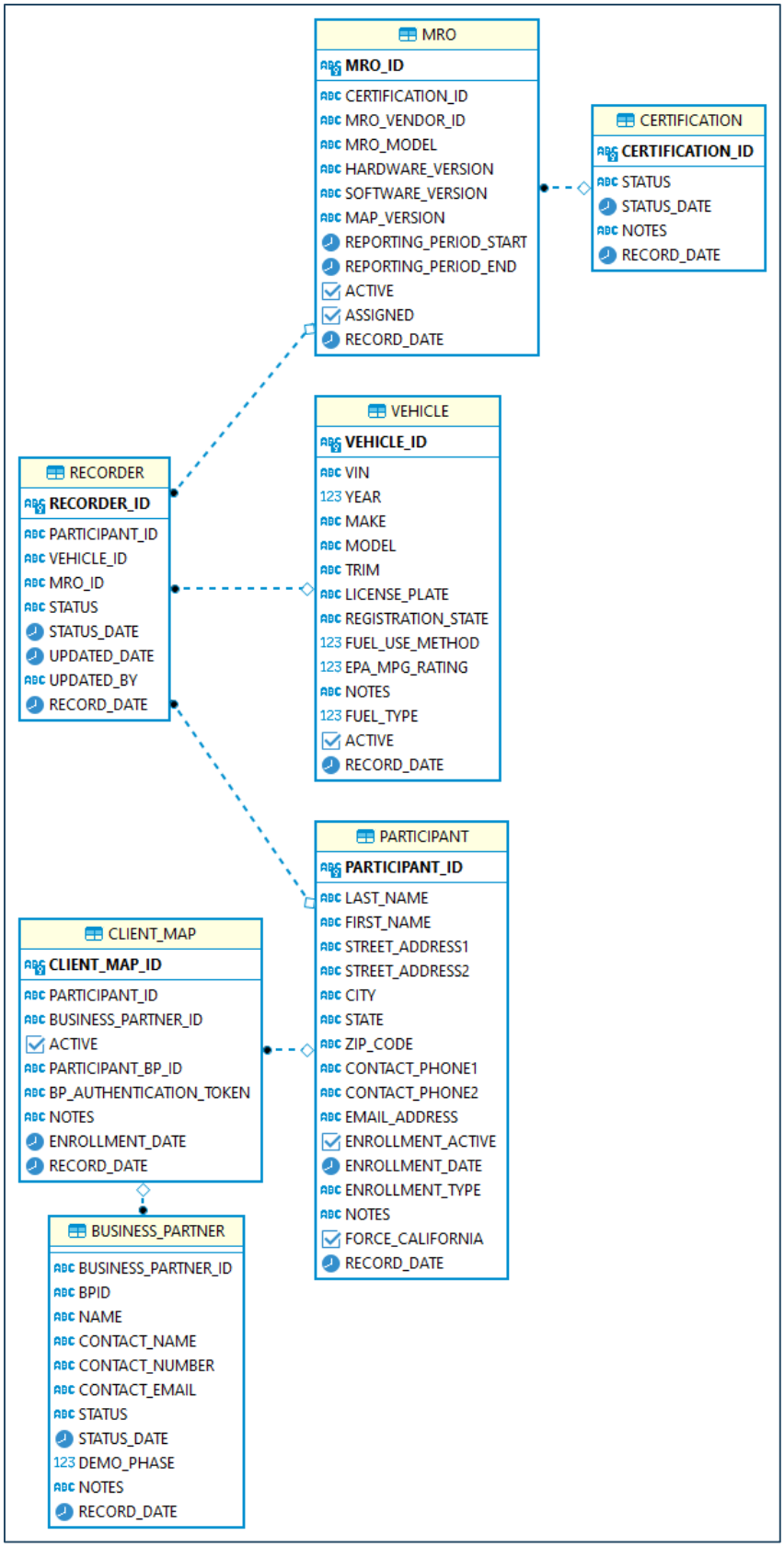


Figure 8: Pilot Snowflake Database AGENTS Schema and Relationships

3.2 BITBREW_2023PILOT

The **BITBREW_2023PILOT** schema contains several tables related to the reporting of MRO device information. The **BITBREW_2023PILOT** schema's primary purpose is to store collected and processed trip information from the Danlaw/BitBrew MRO devices, store processed transaction records, and monitor the connectivity of those devices. Figure 9 provides an overview of the table relationships within the schema, and Appendix A provides a schema definition and description for each table.

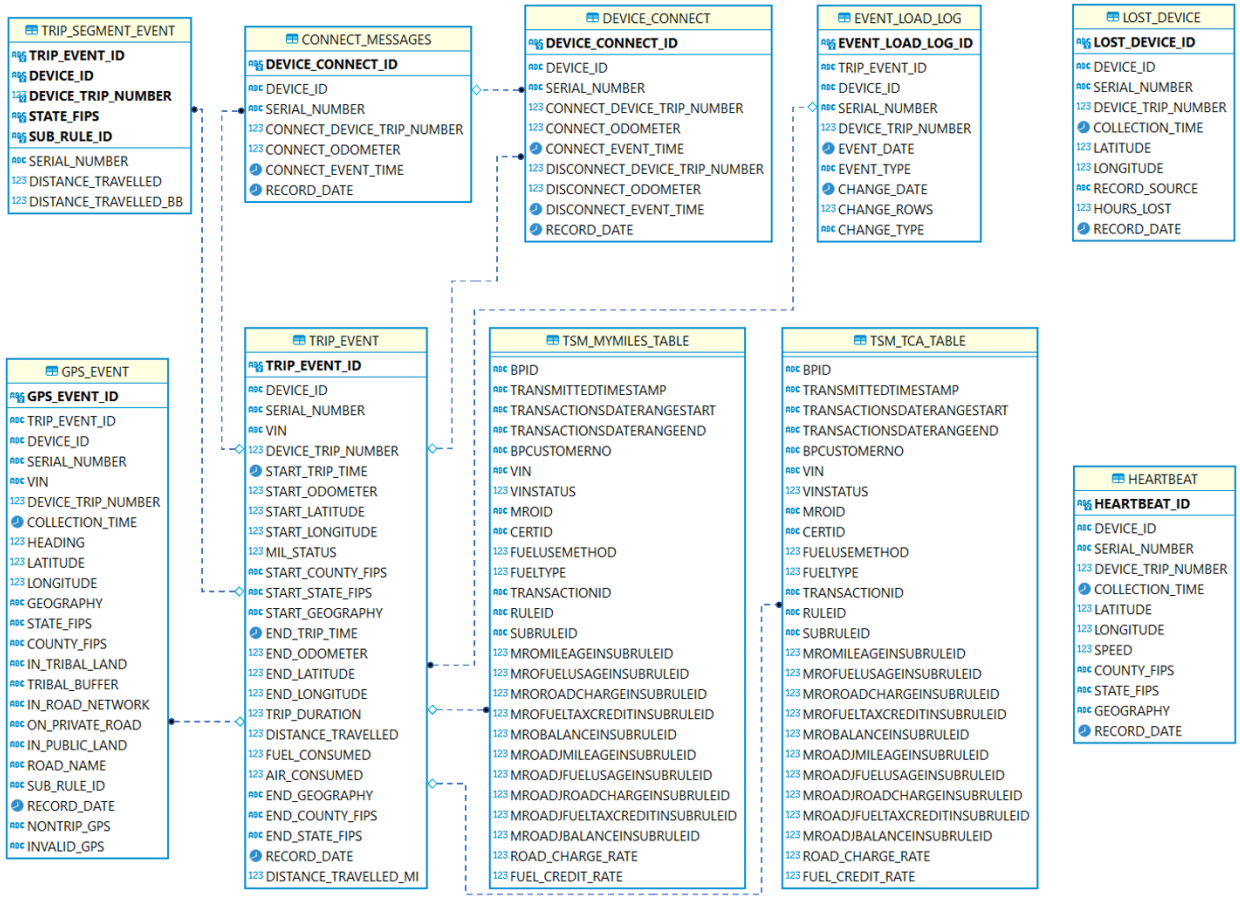


Figure 9: Pilot Snowflake Database BITBREW_2023PILOT Schema and Relationships

The **BITBREW_2023PILOT** schema includes secondary PII related to the travel behavior of pilot participants. With enough data, home and other primary travel locations could be inferred from the Global Positioning System (GPS) data included in this schema. These detailed GPS data are used to calculate miles traveled and identify where participants are traveling to apply the correct road charge rates (simulated) based on state and road type. These detail data are only available to pilot administrators and account managers to operate the pilot and are not shared with other third-parties or Caltrans.

The **BITBREW_2023PILOT** schema also includes several views that leverage one or more tables via joins and data analysis to produce table-like results for various purposes.

- **EVENTS** view: Takes data from the TRIP_EVENT, HEARTBEAT, GPS_EVENT, and DEVICE_CONNECT tables to generate various device event types, such as TRIP_START, TRIP_END, HEARTBEAT, GPS, CONNECT, and DISCONNECT events.
- **LAST_SEEN** view: Takes the most recent event type from the EVENTS view for each pilot device to determine when the device was last seen (had last reported data to the system), which provides context for the PRIME Admin Portal's MRO page to determine when a device last reported to evaluate if the device has been disconnected or otherwise not reporting for long enough that action should be taken with the participant (e.g., request the participant to plug the device back in, inform the participant the device has not reported in 3 days and it may be malfunctioning or not securely plugged into the OBD port, etc.)
- **FUEL_BY_TRIP** view: Takes data from the TRIP_EVENT table (joining with the MRO, RECORDER, and VEHICLE table to associate device serial number with the assigned VIN) to calculate distance traveled in miles (converted from kilometers from the Danlaw devices) as well as fuel consumed from the device directly and the estimated fuel consumed based on miles traveled divided by EPA MPG rating. This view focuses on one record per trip.
- **FUEL_BY_TRIP_SEG** view: Takes data from the TRIP_SEGMENT_EVENT table (joining with the MRO, RECORDER, and VEHICLE table to associate device serial number with the assigned VIN) to calculate distance traveled in miles (converted from kilometers from the Danlaw devices) as well as fuel consumed from the device directly and the estimated fuel consumed based on miles traveled divided by EPA MPG rating. This view focuses on one record for each trip segment (total miles traveled in each RuleID/SubRuleID combination, differentiating California travel by road type [public, private, tribal]).
- **PARTICIPANT_TRAVEL_BY_MONTH** view: Takes data from the TRIP_EVENT table (joining with the MRO, RECORDER, and PARTICIPANT table to associate each trip with the pilot participant) to calculate distance traveled in miles (converted from kilometers from the Danlaw devices) by participant by each month.
- **TRIP_BY_BP** view: Takes data from the TRIP_EVENT table (joining with the MRO, RECORDER, PARTICIPANT, CLIENT_MAP, and BUSINESS_PARTNER table to associate each trip with the business partner) to allow for further analysis.
- **TSM_TCA** view: View created for generating transaction summary messages from multiple tables for TCA participants/devices.
- **TSM_myMiles** view: View created for generating transaction summary messages from multiple tables for myMiles participants/devices.
- **VIN_MISMATCH** view: View created for checking records where there is a mismatch between the vehicle VIN reported by the Danlaw device and the vehicle VIN reported by the participant during the enrollment process.

3.3 FINANCE

The **FINANCE** schema includes one table relevant to this pilot, ROAD_USER_CHARGE_RATES_2023PILOT, providing road charge rates and fuel tax credit rates for each Rule ID (U.S. State, Canadian Province, or Mexican State) and SubRule ID (area within a Rule ID further differentiated for rate applicability; for this pilot – California public, private, and tribal travel is differentiated). Fuel tax credits are also separated in this rate table between gasoline and diesel rates. This table is used to calculate for participants' transactions, monthly statements, and pilot reporting. Appendix A provides a description for the applicable table included in this schema.

3.4 SECURITY

The **SECURITY** schema includes one table, USER, that manages the usernames and passwords (hashed) for the PRIME Admin Portal. Snowflake users with ACCOUNTADMIN granted access can create, edit, and disable users in this table to manage PRIME Admin Portal access. There are two main user roles in this table – ADMIN and VIEWER. The ADMIN role has full access to the Admin Portal, whereas the VIEWER role only has access to certain areas within the Admin Portal that do not contain PII. The management of these roles is done in the GitHub code base.

While the USER table controls access to the admin portal, it can also be updated with information input from the admin portal. Any user (ADMIN or View users) can update their email addresses and passwords themselves through the admin portal using the "forgot password" functionality (An email with token for resetting the password will be sent to the registered email address for the participant), and the updates are reflected and stored in the USER table.

3.5 SCHEMA AND DATABASE CONFIGURATION

The PRIME system relies on a compilation of schema definition YAML (recursive acronym for "YAML Ain't Markup Language") files to build and operate on the Snowflake database system. The schema definition files are mostly maintained in the Core Python repository (See Section 9.1.1), and one is maintained in the Web Python repository (See Section 9.1.3). The schema definitions serve three primary purposes.

First, the schema definitions provide a structured mapping to translate JavaScript Object Notation (JSON) data into flattened comma-separated values (CSV) files. Because many of the JSON definitions used by the PRIME system have a nested structure, the schema definitions include information about parent-child (or primary key / foreign key constraints) relevant to flattening JSON into CSV files. These same structures are used to build appropriate primary key and foreign key relationships as the data moves to the Snowflake database.

Second, the schema definition files provide the necessary information to build the database from scratch in terms of table names, column names, column data types, and foreign keys. Within the Core Python repository, a build_database.py script was created that is used regularly to refresh or rebuild components of the database or quickly set up staging and test environments. Using the metadata included in these YAML definitions, the Core Python package can build the appropriate

Structured Query Language (SQL) syntax to create databases, schemas, tables, and views efficiently and reproducibly.

Finally, the table metadata in YAML definitions also includes special markers to identify table data that is developed on the fly with the Python code or special instructions for geoprocessing tables. For example, the geoprocessing labels identify the latitude and longitude columns as well as the location to store state and county FIPS codes within the table. These special markers relate the Core Python package with other components that need to be called to fulfill the corresponding functionalities.

A description of the Schema Definitions and File Layout is provided in Appendix B.

4 Data Exchanges with Partners

Data is exchanged between the main system and pilot partners, such as Danlaw/BitBrew for device/trip information (vehicle travel) and myMiles/TCA for account management. The data exchange with each partner is described in the subsections below.

4.1 BITBREW

How data flows from BitBrew through the main system to its destination in Snowflake is illustrated in Figure 10. The system processes individual trips as the trip is complete and all trip-related data are available using a "serverless" AWS Lambda function. Data processing is triggered on delay by each trip end event. BitBrew streams trip end records to the cloud storage on AWS S3 to archive existing records. Once a trip end record is received on AWS S3, a delay timer would trigger a Lambda function after 30 minutes to copy the trip end record to a lightweight table on AWS DynamoDB, where the records are temporarily held. Each new trip end event received by DynamoDB triggers a new invocation of the core Lambda function (process_trip_data), which successively queries the trip start event, GPS events, and trip end event for a single trip from other DynamoDB tables that temporarily hold the data. Each of these record sets—start, end, and GPS—are processed (i.e., to assign RuleID and SubRuleID to each GPS point and calculate travel distance) by the Lambda function and uploaded to Snowflake using the same staging process previously developed for bulk uploading. Finally, the streamed records are deleted from DynamoDB tables.

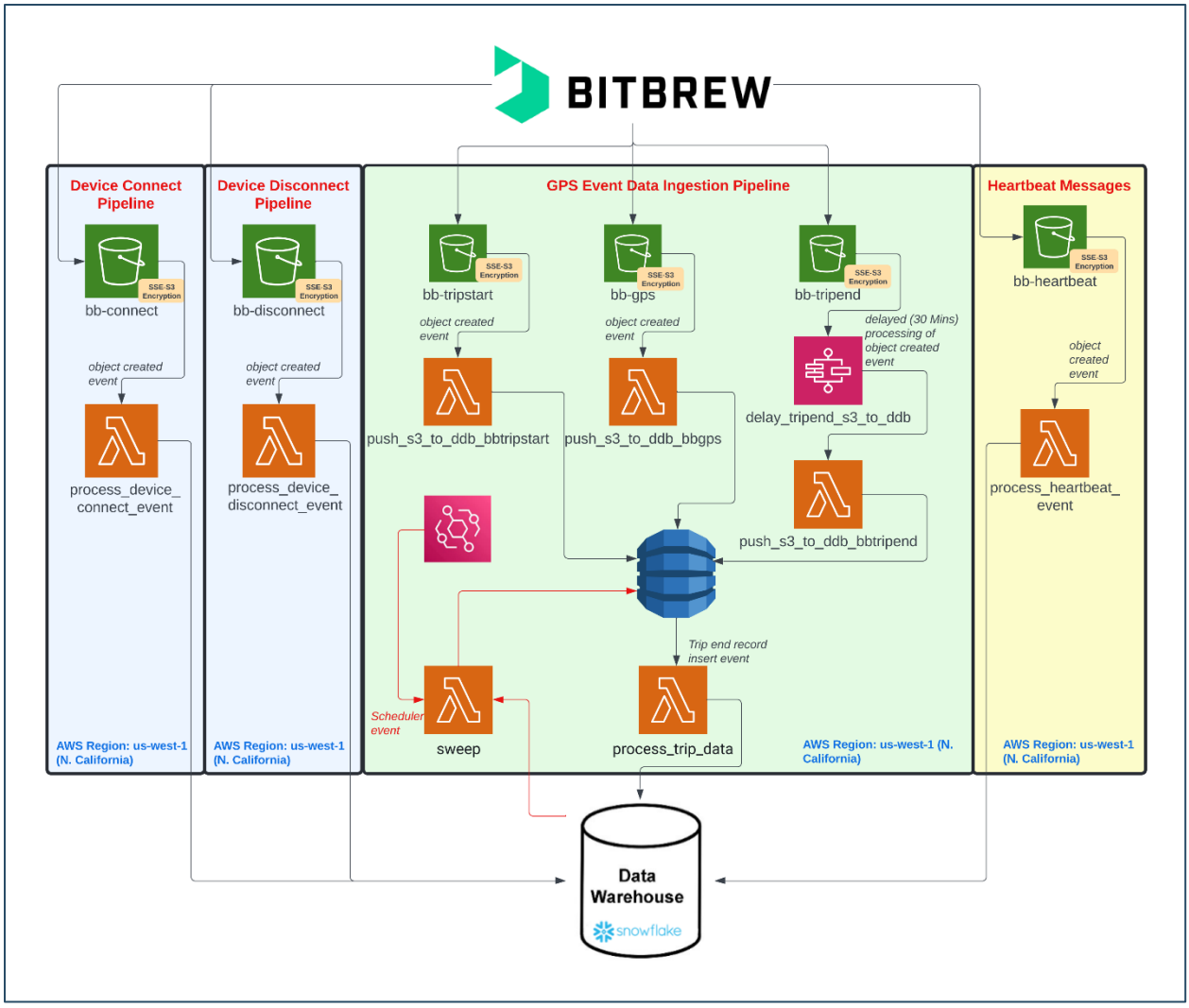


Figure 10: BitBrew Data Pipeline

Several aspects of this loading configuration improve scalability and efficiency for system management. Firstly, BitBrew records are automatically streamed into DynamoDB as they are generated. So no processing time is needed to retrieve and download records from BitBrew. By the time the Lambda function is invoked, all data that it consumes are available within the holding tables in DynamoDB. Secondly, because Lambda functions can run in parallel, there is essentially no limit to the rate at which trip data may theoretically be processed. Hundreds of trips can be streamed, processed and pushed to Snowflake simultaneously if they happen at the same time. Finally, near-real-time processing of trip data improves the currency of summary statistics available on the PRIME administrative portal.

Besides the records associated with trips (i.e., trip start, trip end and GPS), BitBrew also streams three other types of records, including device connect, device disconnect and heartbeat, to AWS S3. Receiving those records by the S3 buckets triggers corresponding Lambda functions that

process the records and push the processed records to Snowflake. These three types of records are critical for monitoring the device status and inferring how long each device stays active.

4.2 MYMILES

The myMiles platform hosts account management functionality for main pilot participants. The myMiles platform communicates new enrollment and account change information to the overarching system database and retrieves data from the overarching system database regularly to populate participant portal viewing of account, statement, incentive, and trip information for participants.

FROM MYMILES BACKEND SYSTEM TO OVERALL SYSTEM DATABASE (data collection, transaction processing, and/or administration subsystem data):

- When a new myMiles participant account is created, the myMiles backend creates new records in the Snowflake database tables:
    - AGENTS.PARTICIPANT: first name, last name, email address, phone number, street address (first and second line), city, state, ZIP code, enrollment date, active status, and enrollment type ("app", a.k.a. myMiles).
    - AGENTS.VEHICLE: vehicle identification number (VIN), vehicle year, vehicle make, vehicle model, vehicle MPG rating, vehicle fuel type, vehicle fuel use method, and vehicle active status.
    - AGENTS.RECORDER: Creates table primary key reconciliation record across participant, vehicle, and MRO tables via those tables' primary key IDs, as well as record status and status date.
    - AGENTS.CLIENT_MAP: Creates relationship and active status between participant ID and associated business partner ID from PARTICIPANT and BUSINESS_PARTNER table, respectively.
- In myMiles, a logged in participant can update their first name, last name, and phone number, in which case the myMiles backend updates the AGENTS.PARTICIPANT table appropriately.
- In myMiles, when a logged in participant goes to the Statements tab and clicks the "Make Simulated Payment" button on the Current statement, the myMiles backend queries the Umbraco-hosted incentive tracker database for the appropriate user and updates that month's "Month #XX: Review and 'Pay'" incentive activity as "IsCompleted" with today's date.

FROM OVERALL SYSTEM DATABASE (data collection, transaction processing, and/or administration subsystem data) TO MYMILES BACKEND SYSTEM:

Once an account is created, myMiles grabs static data from Snowflake database to populate information about the participant's account, trips, statements, and incentives.

- **Statements tab:**
    - First and Last Name: AGENTS.PARTICIPANT

  - VIN, Fuel Type, and MPG: AGENTS.VEHICLE
  - Miles, gross road charges, road charge rate, gross fuel tax credits, appliable fuel tax credit rate (based on fuel type), and total net road charge balance: BITBREW_2023PILOT.TSM_MYMILES_TABLE
- **Trips tab:**
  - BITBREW_2023PILOT.TRIP_EVENT_TCP:
    - Date/time trip start and end:
    - Trip start and end locations
  - Miles and fuel gallons displayed on each trip's summary line: BITBREW_2023PILOT.FUEL_BY_TRIP_SEG
- **Trip Details tab:**
  - BITBREW_2023PILOT.TRIP_EVENT_TCP:
    - Trip start and end date/time
    - Trip start and end locations
  - BITBREW_2023PILOT.GPS_EVENT_TCP:
    - GPS points to map on map
  - BITBREW_2023PILOT.FUEL_BY_TRIP_SEG:
    - Miles total
    - Miles breakdown
    - Fuel total
    - Fuel breakdown
- **Incentives tab:**
  - myMiles backend queries the Umbraco-hosted incentive tracker database with participant's email address to locate Umbraco database user ID, then queries and retrieves incentive information: incentive activities completed, incentive activities paid, dollar amounts related to incentive activities, total completed, and total paid.
- **Account tab:**
  - Vehicle Information – Make, model, year, VIN, type, fuel type: AGENTS.VEHICLE table.
  - Device Information – Serial #, device type, and status: AGENTS.MRO table; Last Seen: BITBREW_2023PILOT.LAST_SEEN (view).
  - Personal Information – First name, last name, email address, phone number, and address: AGENTS.PARTICIPANT table.

4.3 TCA

TCA serves as the account manager for TCA sub-pilot participants, and provides monthly road charge statements directly on their system. Other TCA sub-pilot participant account management information is hosted by WSP via the Umbraco TCA participant portal (more information can be found in the Participant Portal at Pilot Launch deliverable).

FROM OVERALL SYSTEM DATABASE (data collection, transaction processing, and/or administration subsystem data) TO TCA BACKEND SYSTEM:

Once an account is created, TCA grabs transaction-level data from WSP-generated Transaction Summary Message (TSM) posted to a WSP-hosted AWS S3 bucket to populate TCA sub-pilot participants' monthly road charge statements.

- Participant general information, such as participant name and address are populated directly by TCA via their existing customer database, associated by TCA Account Number indicated by WSP at initial enrollment (stored in a shared Microsoft Excel workbook, hosted on TCA SharePoint site).
- VIN, device serial number, miles, fuel gallons, gross road charges, gross fuel tax credits, and total net road charge balance: TSM generated and posted to S3 daily.

NOTE: TCA does not transmit any data directly to the overall pilot system and database.

5 Reporting

5.1 TRANSACTION SUMMARY MESSAGE

Transaction Summary Messages (TSM) are generated by the PRIME system and used in three ways. First, all transactions are stored in the overall pilot system database in the BITBREW_2023PILOT schema, in TSM tables respective for myMiles and TCA, and can be generated at any time for a reporting period. Second, daily TSMs are generated for TCA participants and posted to a WSP-hosted AWS S3 bucket for TCA to pull for generating TCA sub-pilot participant monthly road charge statements. Third, myMiles pulls transactions from the TSM_MYMILES table to generate myMiles participants' monthly statements in the myMiles platform (for both the current month and historical PDFs of statements).

The PRIME administrative portal also includes the option to generate TSMs on the fly. The functionality is presented to the administrative end user in each business partner's section of the portal. The user can select custom start and end timestamps (configured in Pacific Time), and the system will generate the appropriate TSM.

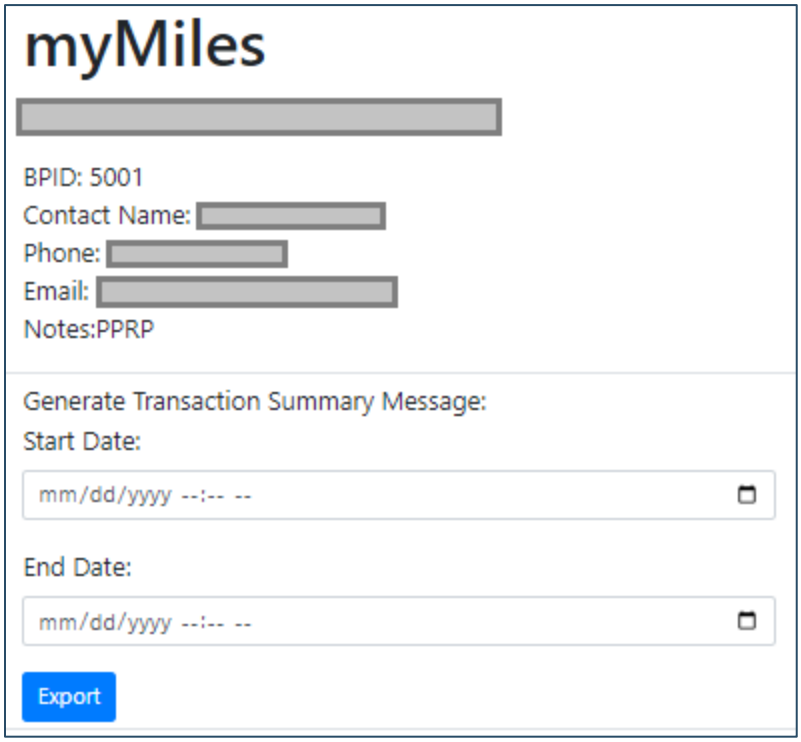


Figure 11: Transaction Summary Message Generator in PRIME Administrative Portal

5.2 MONTHLY REPORTING

The monthly summaries consist of 4 tables. Table 1 summarizes miles traveled and road charge amount by month. Tables 2 summarizes road charge amount and fuel tax credit amount by month by business partners. Table 3 summarizes the percentage of time each device stays active. And

Table 4 compares the fuel tax credit calculated from EPA fuel economy data and that based on BitBrew fuel consumption data for each device.

The script for generating the monthly summary tables is part of the core Python package. The script utilizes the Snowflake Connector in Python to query the Snowflake tables and retrieve information on miles traveled (under each RuleID and SubRuleID), fuel consumption, vehicle fuel economy, road charge rate, fuel tax credit rate, device connect time and device disconnect time for each device during given months. These pieces of information are then aggregated, synthesized, and organized into the above four tables.

6 PRIME Web Portal

The PRIME administrative portal and API provides access to System Administrators about the status of pilot operations without having to access the Snowflake database via SQL code. The web portal provides information about participants and their enrollment status, linkage to vehicles and MRO, as well as a location to monitor Danlaw device health and interactions with the business partners. Outside of the Snowflake database, the portal provides the most detailed look into pilot operations.

6.1 ADMINISTRATIVE PORTAL

The web portal includes administrative functionality for System Administrators to manage pilot operations, including:

- **Dashboards** *(accessible by System Administrator login credentials only)***:**
  - **VMT Summary (under development)**: Vehicle miles traveled by region, by county, and trips by day – dashboard level snapshots.
  - **Admin**: Report generation page – Ability to pull TSM or VIN Summary reports for each Business Partner, for any reporting period, formatted in JSON or CSV export.
- **Participants**: Add new participants, update existing participants, associate vehicles and devices, associate to a business partner, review general travel information (Figure 12).
- **Business Partners**: Add new business partners, update existing business partners, associate participants to a business partner, update existing participant associations, generate TSMs.
- **Vehicles**: List of active vehicles associated to pilot participants.
- **MROs**: List of mileage reporting options (MROs) assigned to participants and their current activity status (Figure 13).

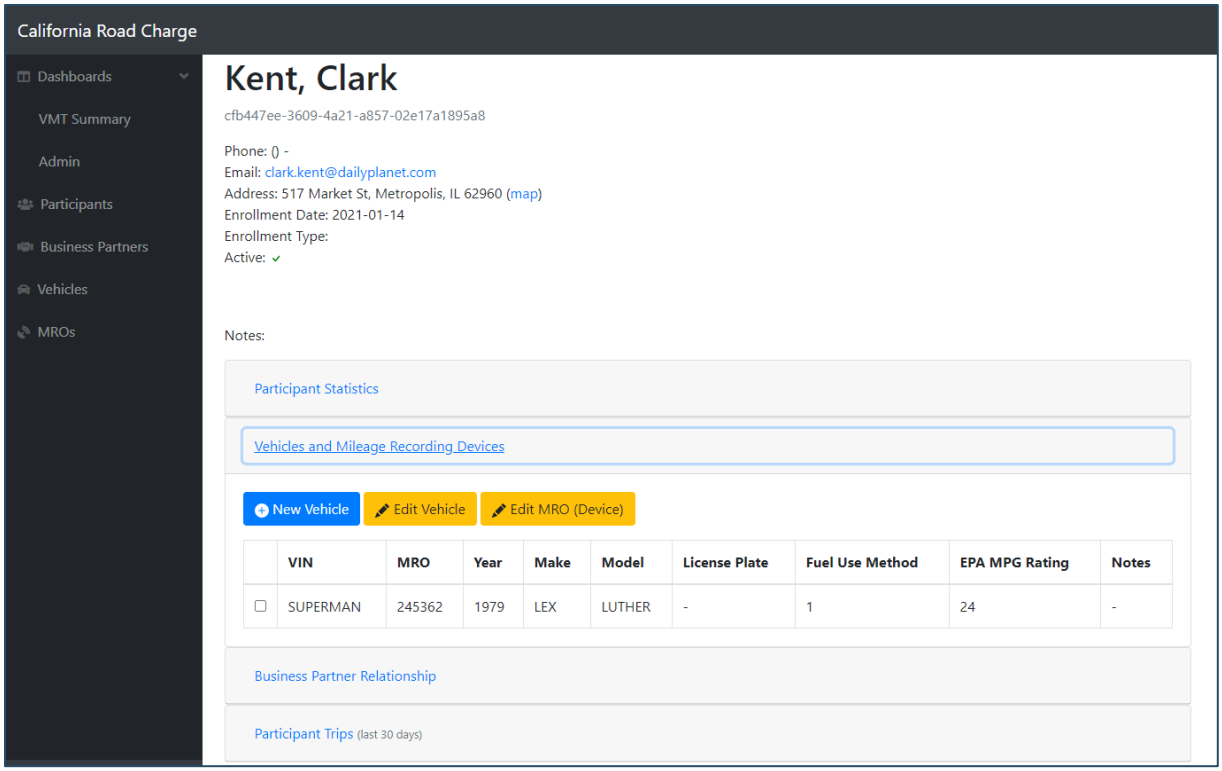


Figure 12: PRIME administrative portal - Participant Page (sample)

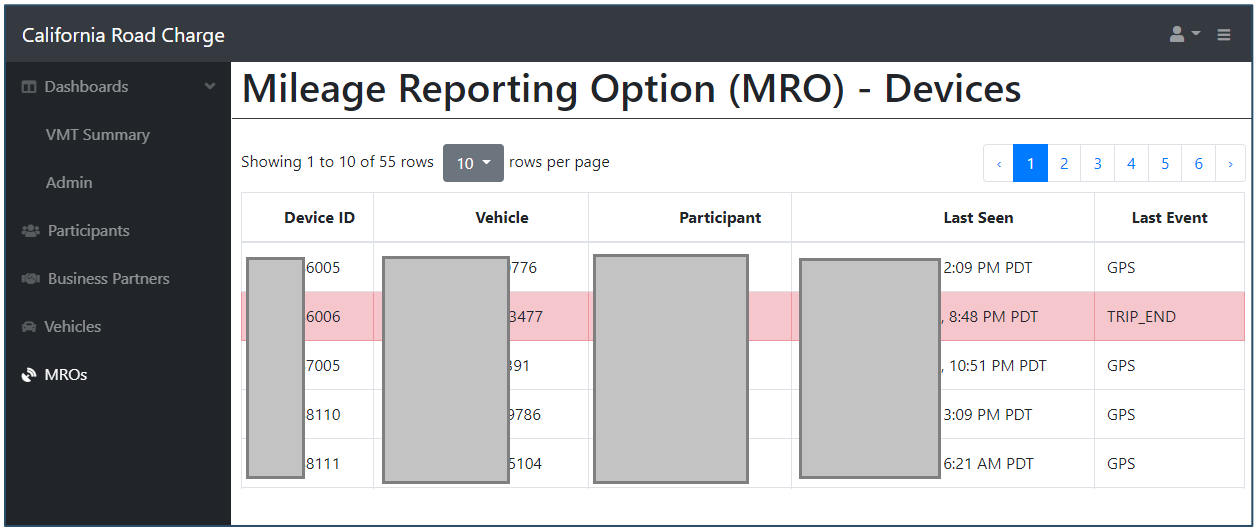


Figure 13: PRIME administrative portal - MRO management page (redacted)

6.2 ANALYTICS ENGINE

The PRIME system includes functionality to perform analytics on data for the 2017, 2021, and 2023 (current) California Road Charge pilots. For information on the PRIME system dashboards for the 2017 and 2021 pilots, please refer to the California Road Charge Four-Phase Demonstration Task 7 "End of Demonstration Database Summary Report" (v2.0; 10/27/21).

PRIME system dashboards and analytics for the 2023 Pilot (current) have not been developed and deployed to the PRIME platform at the time of writing this deliverable (PRIME Batch Updates thru Pilot Design and Testing). The Project Team will coordinate with Caltrans in Q3 2023 and Q4 to define, develop, and deploy dashboards and analytics related to this pilot.

6.3 UMBRACO WEB PORTAL

WSP hosts the https://caroadcharge.com website on the Umbraco platform. The Umbraco backend website management site, https://caroadcharge.com/umbraco/#/login/false?mculture=en-US, allows the project team to manage the program website itself (more information can be found in the Communications Plan deliverable), and the participant portal content. For this pilot, the participant portal content is used for TCA sub-pilot participants to manage enrollment information and participant agreement acknowledgement, as well as viewing monthly incentives information. Screenshots of what the Umbraco backend site content management and the frontend TCA sub-pilot participant portal look like are in the two figures below.

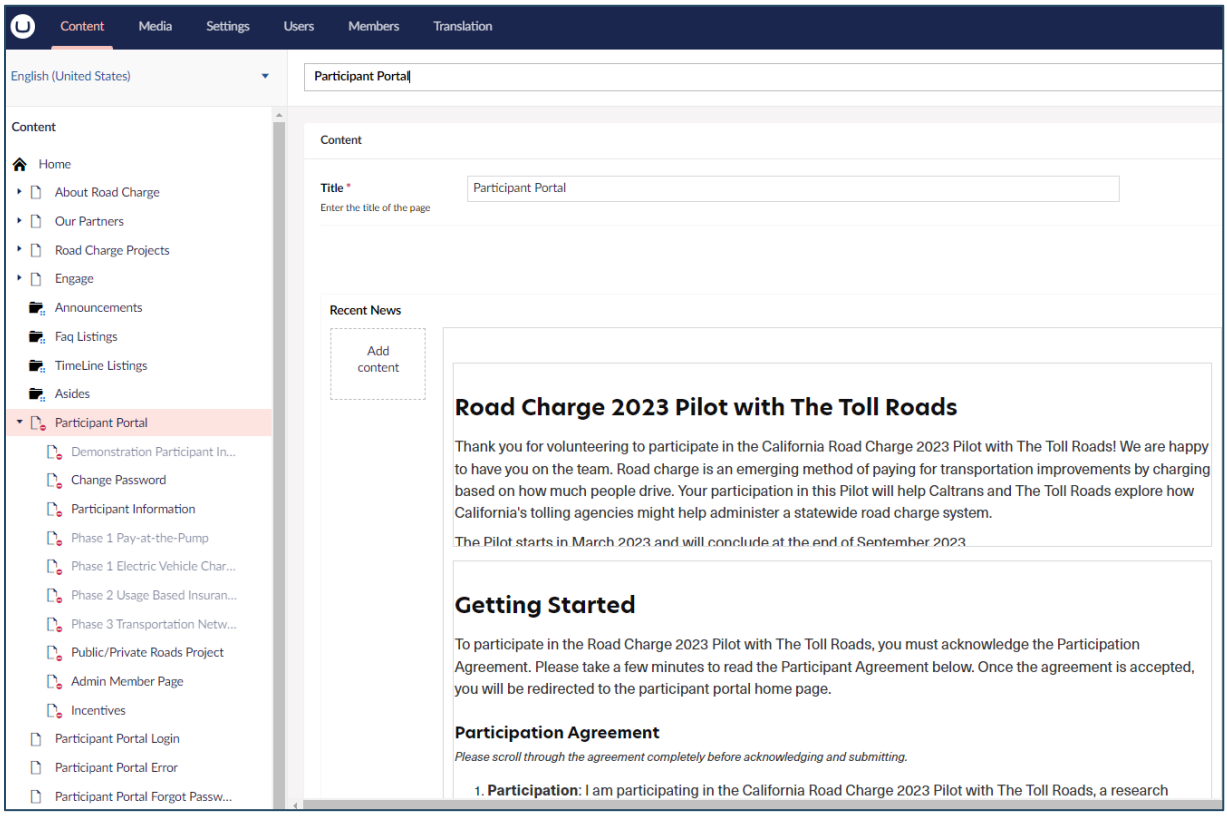


Figure 14: Umbraco backend management site - Content section

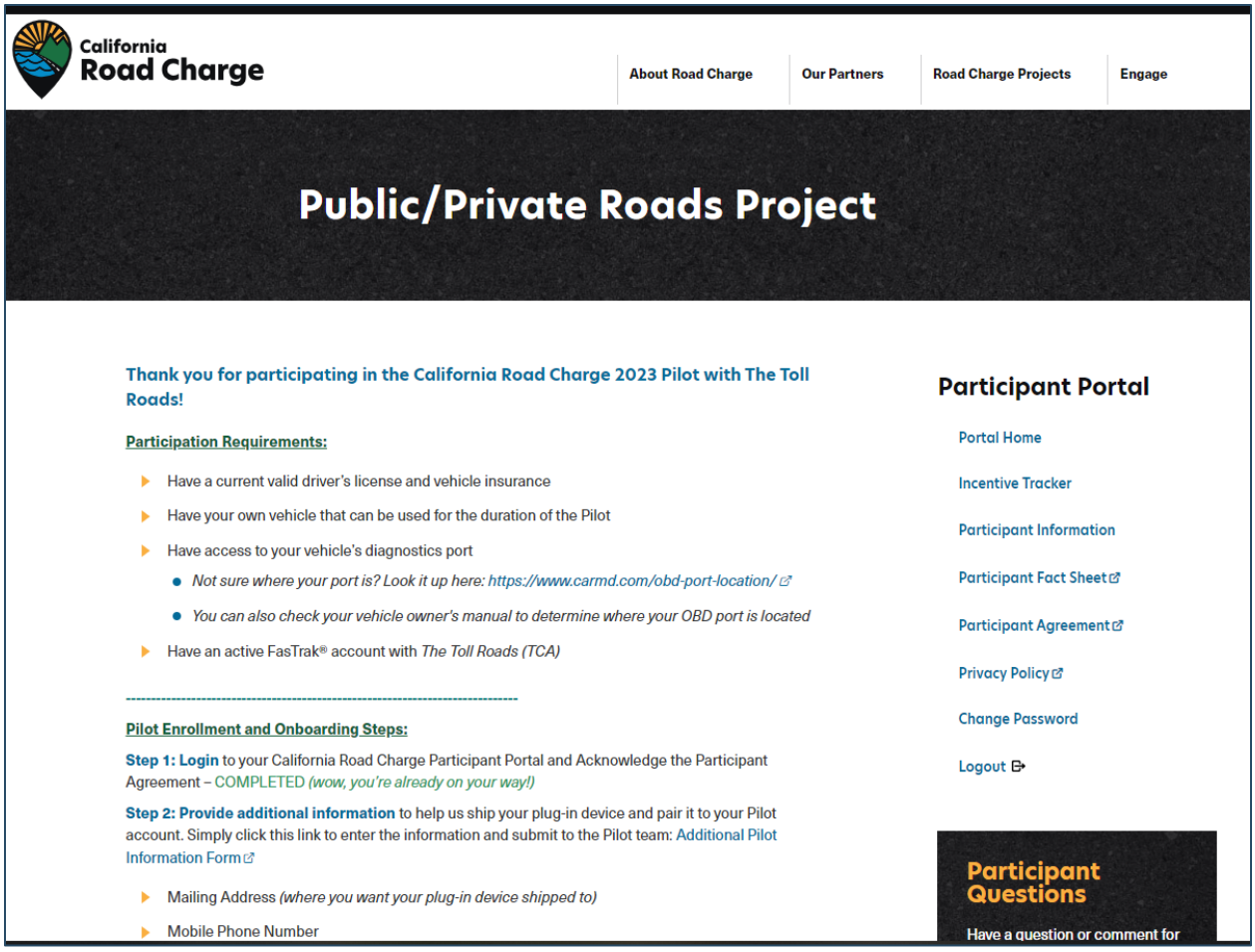
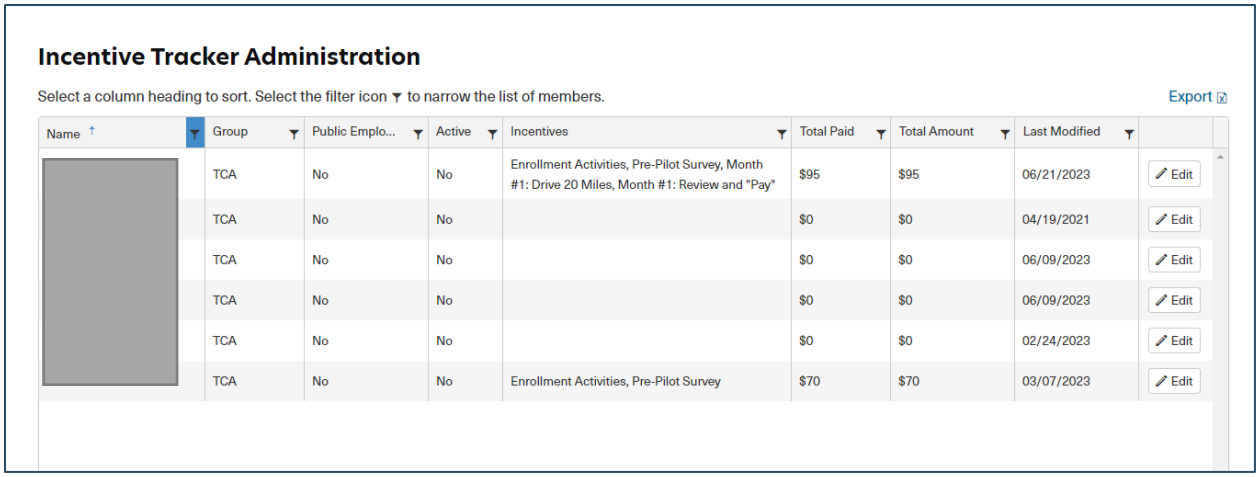


Figure 15: TCA sub-pilot participant portal

Also included in the Umbraco platform management of this website is the TCA sub-pilot participant portal (more details on this can be found in the Participant Portal at Pilot Launch deliverable) and the pilot incentives tracker administrative management page: https://caroadcharge.com/participant-portal/admin-member-page/

To configure access to the incentive tracker admin management page, one must have access to the Umbraco backend management site at https://caroadcharge.com/umbraco, then navigate to: Content > Home > Participant Portal > Admin Member Page > Actions > Restrict Public Access. From there, an Umbraco user can configure which Umbraco site users have access to the incentive tracker admin management page to manage pilot participants' incentives, including which incentives a participant has earned/completed and which incentives have been paid.

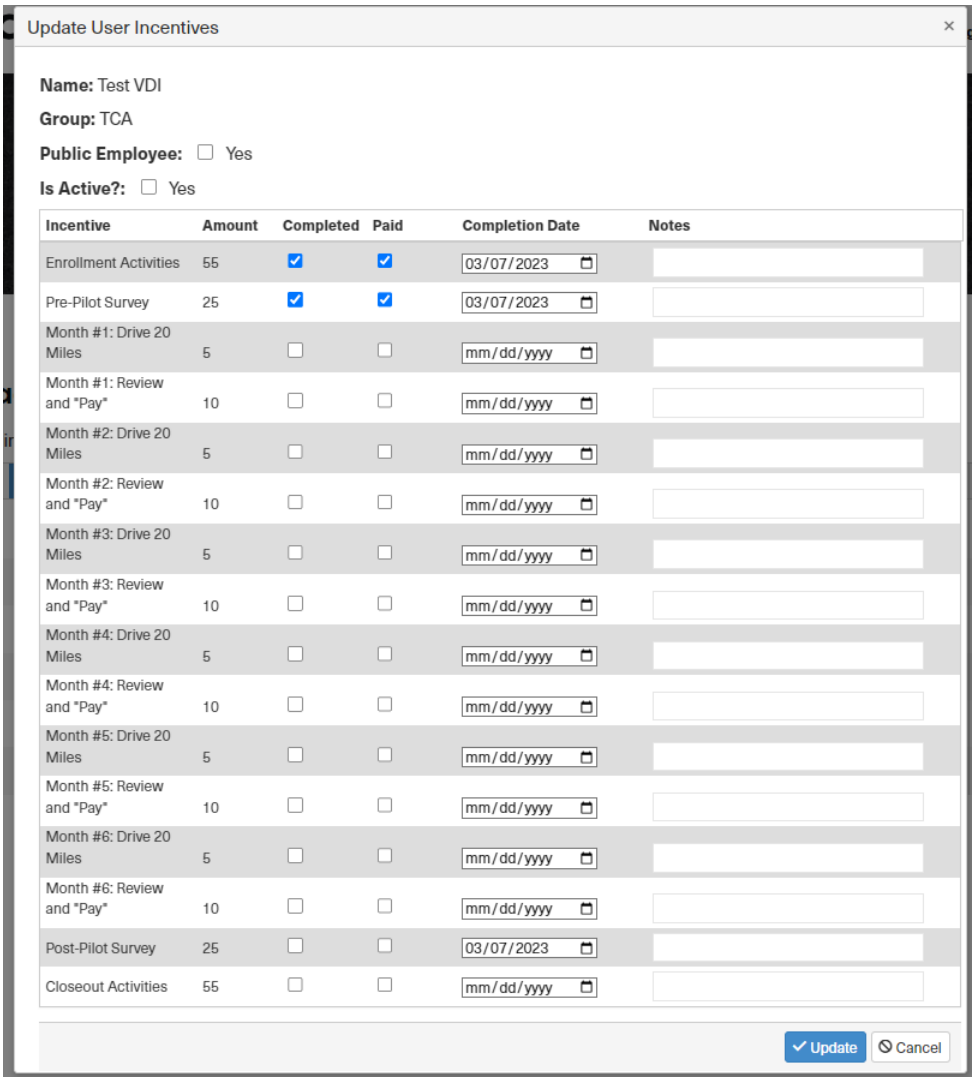


Incentive Tracker Administration

Select a column heading to sort. Select the filter icon to narrow the list of members. Export

Name	Group	Public Emplo...	Active	Incentives	Total Paid	Total Amount	Last Modified	
[redacted]	TCA	No	No	Enrollment Activities, Pre-Pilot Survey, Month #1: Drive 20 Miles, Month #1: Review and "Pay"	$95	$95	06/21/2023	Edit
	TCA	No	No		$0	$0	04/19/2021	Edit
	TCA	No	No		$0	$0	06/09/2023	Edit
	TCA	No	No		$0	$0	06/09/2023	Edit
	TCA	No	No		$0	$0	02/24/2023	Edit
	TCA	No	No	Enrollment Activities, Pre-Pilot Survey	$70	$70	03/07/2023	Edit

Figure 16: Pilot Incentives Tracker Admin Member/Management Page



Update User Incentives

Name: Test VDI

Group: TCA

Public Employee: ☐ Yes

Is Active?: ☐ Yes

Incentive	Amount	Completed	Paid	Completion Date	Notes
Enrollment Activities	55	☑	☑	03/07/2023	
Pre-Pilot Survey	25	☑	☑	03/07/2023	
Month #1: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #1: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Month #2: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #2: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Month #3: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #3: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Month #4: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #4: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Month #5: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #5: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Month #6: Drive 20 Miles	5	☐	☐	mm/dd/yyyy	
Month #6: Review and "Pay"	10	☐	☐	mm/dd/yyyy	
Post-Pilot Survey	25	☐	☐	03/07/2023	
Closeout Activities	55	☐	☐	mm/dd/yyyy	

Update Cancel

Figure 17: Pilot Incentives Tracker Admin Page: Editing a Participant's Incentives

7 Amazon Web Services

7.1 DATA SHARING (S3)

The PRIME system shares and stores backup data in several Amazon S3 buckets. Amazon S3 buckets are conceptually synonymous with traditional folders or directories in a Windows or Linux operating system.

7.1.1 BitBrew Backups

The PRIME system maintains a running backup of BitBrew records in Amazon S3 buckets. The type of records backed up by Amazon S3 include trip start, trip end, GPS, heartbeat, device connect/plug-in and device disconnect. Each type of record is stored in a separate S3 bucket. As soon as BitBrew streams a record, it will be sent to the corresponding S3 bucket for backup.

The S3 archiving buckets for BitBrew records provide two important features for the PRIME system. First, it provides a durable backup of the raw BitBrew event feeds. Though BitBrew stores the raw data record for the duration of the pilot project, the S3 buckets are intended for permanent data archiving. The archived data on S3 can be referenced after the current pilot ends or by future projects. Second, the S3 backup of the data records provides a handy data source for reprocessing the data. The need for reprocessing could be necessitated by a major update to the PRIME system (e.g., adding new features or bug fixing) or the detection of a processed trip as having incomplete data records. Loading data from S3 saves time and cost compared to re-streaming the data from BitBrew.

7.2 WEB HOSTING (ELASTIC BEANSTALK)

The PRIME administrative portal is hosted on an Amazon Elastic Beanstalk instance. Elastic Beanstalk is an all-in-one web server service that manages load balancing, web hosting, and application server in one package. The website and API are written in Python using the Flask application framework.

The application is deployed by packaging all the Python and supporting files into a ZIP file and uploading to the Elastic Beanstalk instance. In the background, Elastic Beanstalk unzips the application files into a Linux EC2 instance which serves the application.

The Elastic Beanstalk environment that hosts the PRIME administrative portal is configured to serve web pages and API calls via a secure connection only. Any attempts to access the site via HTTP (port 80) are redirected immediately to the HTTPS (port 440) endpoint. The HTTPS Certificate is managed and maintained by the Route 53 domain registration described in the next section.

The Elastic Beanstalk service is currently managed and secured via an AWS account managed by WSP.

7.3 DOMAIN REGISTRATION AND HTTPS (ROUTE 53)

The PRIME administrative portal is hosted on an Amazon Elastic Beanstalk instance as described in the previous section. For the site to run in a secure environment that is universally accessible across the web, the site needs a certified domain name registration. The domain name registration and HTTPS certificate management is handled through the AWS Route 53 service.

The Route 53 services are currently procured and managed via an AWS account managed by WSP. For the PRIME administrative portal, the domain name is https://caroadpilot.com

7.4 BITBREW REAL-TIME CONSUMER (DYNAMODB)

BitBrew records are received by AWS S3 and then copied to several staging tables on AWS DynamoDB, where these records are held until they are processed and transferred into the production Snowflake database. BitBrew is configured with rules that send records, formatted in JSON, to separate S3 buckets for each record type (e.g., trip start, trip end, and GPS point). When copying the records from S3 to DynamoDB, the PRIME system flattens the JSON format into a table format for storage in DynamoDB. DynamoDB tables are designed to be lightweight temporary storage containers for large volumes of streaming data, ideal for handling the BitBrew records until they can be processed. Unlike conventional databases, DynamoDB tables are key-value indexed, making reading and writing operations extremely fast, but limiting how data can be retrieved. Since BitBrew records are indexed by unique device and trip identifiers, records associated with each trip can be retrieved from DynamoDB efficiently.

DynamoDB tables are configured to trigger other AWS services when new records are inserted. This is leveraged to trigger the Lambda function for processing trip data as soon as the corresponding DynamoDB table receives a new trip end record (after the delay timer as mentioned earlier). This configuration provides visibility of trip events in the production Snowflake database shortly after each trip concludes, and thus enables timely information communication to the end users.

Once BitBrew records are processed, loaded into Snowflake, and archived to S3, they are deleted from DynamoDB. Storing data in DynamoDB for the long term could be costly. An AWS Lambda function is set up to regularly check if there is any records remaining in DynamoDB. Any remaining records in DynamoDB suggests that the corresponding trip had incomplete data when it was processed, and will trigger reprocessing of this trip, which is handled by the "sweep" function shown in Figure 10.

7.5 BITBREW PROCESSING (LAMBDA SERVICES)

AWS Lambda provides on-demand computing capacity for operations that are triggered by events. Rather than having a dedicated server that may have unused capacity during low-demand periods and insufficient capacity in peak periods, Lambda automatically scales the resources in response to the input event volume. The service is charged only for the capacity used. This is called a serverless application model.

A Lambda function named "process_trip_data" (as shown in Figure 10) is used for near-real-time processing of trip records streaming to DynamoDB from BitBrew and S3. The function is triggered whenever DynamoDB sees a new trip end event, which is inserted as a row to the trip end event table in the DynamoDB database. The Lambda function takes the event object as an argument, parses the device and trip identifiers, and queries associated trip data from other DynamoDB tables. All trip data are then parsed into the schema used by the Snowflake production database, inserted into staging Snowflake tables where geographic attributes are calculated, and then inserted into production Snowflake tables where the web application can access them. Finally, the Lambda function deletes the raw trip records from DynamoDB.

In addition to the "process_trip_data" function triggered by new trip end events, another Lambda function named "sweep" is triggered once daily at midnight to "sweep" BitBrew records that were delayed in the stream and therefore left unprocessed alongside other records for a given trip. The sweep function performs the following operations in sequence: 1) it scan for any record remaining in DynamoDB; 2) for a trip with remaining record(s) in DynamoDB, it deletes all records associated with this trip in Snowflake to avoid potential duplicates; 3) it reloads records associated with this trip from S3 to DynamoDB; and 4) once the trip end record is reloaded to DynamoDB, the "process_trip_data" function will be triggered to reprocess records associated with this trip.

Besides the above two core Lambda functions, six other Lambda functions are deployed to handle the data transfer between different data storage services. Among them, three transfer the trip start, trip end and GPS records from S3 to DynamoDB, respectively; and the other three transfer heartbeat, device connect and device disconnect records from S3 to Snowflake, respectively.

The computing environment for the Lambda function "process_trip_data" is provided by a Docker container, which contains all the software dependencies necessary for retrieving data from DynamoDB, processing trip records and inserting them into Snowflake, along with the Python code and input shapefiles (which are used to assign RuleID and SubRuleID to each GPS point) for the Lambda function itself. The container runs on Linux distribution from a base container image that is supplied by AWS, which is preconfigured with the Lambda runtime interface client. Instructions for how to build the Lambda container, including installation of the AWS base image, cloning the PRIME GitHub repository, installing it as a Python package, installing dependencies, and calling the Lambda function are provided in a Dockerfile, which Docker uses to build the container on a local workstation. The container is then uploaded to the Amazon Elastic Container Registry (ECR), which then serves the container to AWS Lambda. Updates to the Lambda function can be put into production by rebuilding the container with the Dockerfile, uploading it to ECR, and directing Lambda to the new container.

8 Security

The following sections describe how communications are secured between the PRIME system and the Snowflake backend. All communications between the systems described in the document occur over secure connections (usually HTTPS). Connection strings, usernames, passwords, and system URLs are all stored in a *credentials.YAML* file that is stored outside of the repositories described in Section 9.

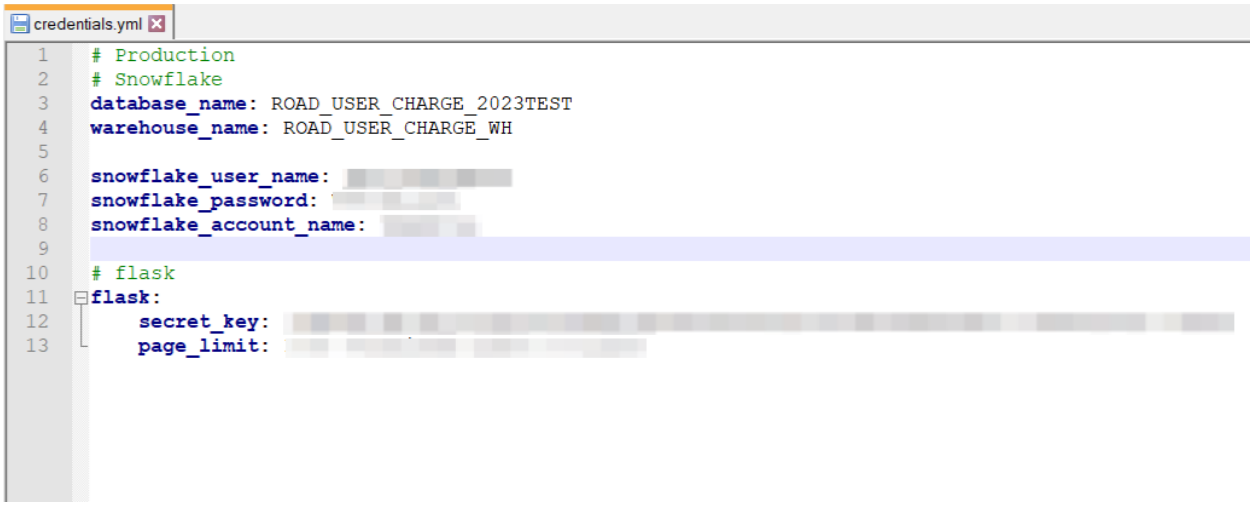


Figure 18: Credential YAML Layout (sample)

Figure 18 provides an overview of the *credentials.yaml* file used by the application. Maintaining the security credentials and connection strings in one centralized system allows the application to be deployed to testing, staging, and production environments with no software modifications while also reducing the potential opportunities for key security details to be compromised. The YAML file includes only credentials for Snowflake as it is required for the communication between AWS services (e.g., S3, DynamoDB and Lambda) and Snowflake. Communications among the AWS services are trusted and secured, and no additional credentials (e.g., AWS access keys) are required.

8.1 AMAZON WEB SERVICES

HTTPS	Exclude PII in Data Transfers	User Authentication
✓		✓

All the Amazon Web Services are managed and secured via an account managed by WSP. One single account provides access to all the services described in Section 337. WSP has also created several user accounts for WSP staff to interact, test, and deploy AWS services. Account

management and credentials will be transferred to Caltrans at the completion of the California Road Charge Demonstration project.

All interactions with AWS are completed over either a secure HTTPS connection to the AWS console or via secure connections via the Boto3 Python libraries via credential YAML file. The Lambda function also reads and writes to the Snowflake production database, and it needs secure access to credentials for the Snowflake API. Rather than storing these credentials within the container, the credentials are stored by AWS Secrets Manager and retrieved each time the function is invoked. Credentials can be rotated in Secrets Manager within any changes to the Lambda function.

8.2 BITBREW

HTTPS	Exclude PII in Data Transfers	User Authentication
✓		✓

Communication between BitBrew and the PRIME system is secured and protected using two techniques. First, all communication between BitBrew and the PRIME system happens across a secure socket layer or SSL connection. Additionally, the BitBrew connection is secured by a username and password generated by the BitBrew system specific to the feeds used.

While no direct PII is shared during the data transfers, indirect PII information is transferred. The risk, however, of a useful amount of PII being leaked or intercepted is low. Inferring an person's home, work, or other regular locations of travel requires piecing together a large amount of GPS points. BitBrew streams each record separately instead of in large chunks to PRIME, which reduces the risk of intercepting a sufficient amount of data for inferring PII. At no point, does BitBrew or the PRIME system transmit direct PII, like names and addresses.

8.3 PRIME WEB PORTAL

HTTPS	Exclude PII in Data Transfers	User Authentication
✓		✓

The PRIME administrative portal and API are protected using HTTPS, user authentication, and JSON Web Tokens. First, all communication occurs over a secure HTTPS connection managed by AWS Elastic Beanstalk and Route 53. If a request is made to the webpages or the API over HTTP (port 80), the requests are automatically redirected to HTTPS. All interactions with the site must include a valid JSON Web Token. For the website, a token is obtained by logging in to the site on the login screen. For the API, a request token call with a valid username and password must be provided. The JSON Web Tokens are stored as encrypted cookies and remain valid for 30 minutes.

8.4 SNOWFLAKE

HTTPS	Exclude PII in Data Transfers	User Authentication
✓		✓

The Snowflake database is accessed via two methods in the development and implementation of the PRIME system. First, for general development and ad hoc testing and monitoring, the Snowflake system is accessed via the online Snowflake browser. For the PRIME web application and PRIME data acquisition with partners, the database is accessed via the Snowflake Python connector. For each method, all communication occurs over a secure HTTPS connection with username and password verification.

The Snowflake online Integrated Development Environment (Figure 19) provides a framework for managing all aspects of the Snowflake system from creating users, roles, and databases to ad hoc SQL development. The WSP development team regularly conducts SQL queries within this environment to monitor data loads and trends within the data. Each user has their own account credentials, and different users are assigned different roles with different permissions. There is currently only one account administrator who has all the permissions including modifying other users' credentials. Most trusted users have the role of system administrator, who has the permission to read and write the Snowflake database. There also exist roles with only view permission to the database. The role configuration mitigates the risk of unauthorized access to the Snowflake database. Snowflake also provides a query archive where past queries can be reviewed for performance, errors, and other issues within the database. This SQL history provides a limited audit trail of users' interaction with the database as well.

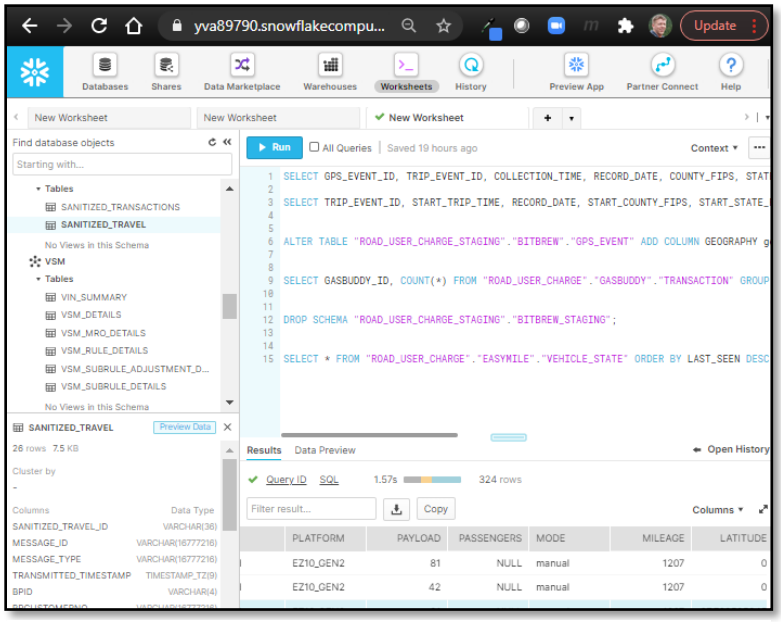


Figure 19 Snowflake IDE

The Python packages in the PRIME system always interact with the Snowflake database via the Snowflake Python connector. The Snowflake Python connector is a 100% pure Python database connector, and the communication between the PRIME system and Snowflake is secured via HTTPS. The PRIME system Python package uses a limited role account that only allows access to database functions and resources necessary to complete its tasks. The username and password are stored in the *credentials.yaml* file described above, and they are not committed to the GitHub repository.

9 Maintenance

9.1 GITHUB VERSION CONTROL

The PRIME system Python packages used to manage and orchestrate the processes described in this document are maintained and versioned in WSP-managed GitHub repositories (Figure 20). GitHub is a standard platform across software development projects to maintain a history of software changes while also supporting multiple contributors to the software. The repositories described below are all maintained by WSP as private repositories. Private repositories, unlike public repositories, limit the number of approved contributors and viewers who can access the PRIME Python packages. Finally, no user credentials (e.g., usernames, passwords, connections strings) are maintained or managed in the GitHub repositories.

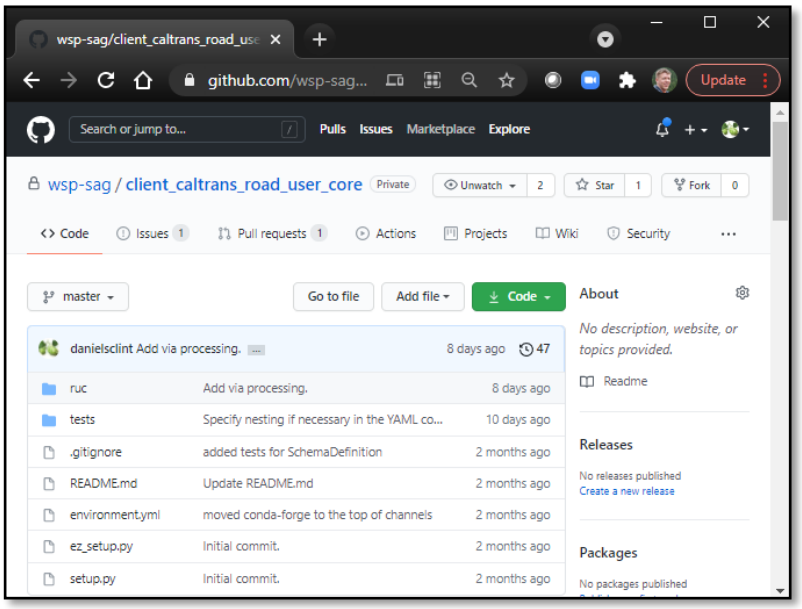


Figure 20 GitHub Repository

The PRIME system contains four packages:

- Core,
- Client,
- Web, and
- Working

The client, web, and working packages work independently from one another but all inherit functionality from the core package. (See Figure 21) The core and client packages include PIP enabled setups for easy installation on Python and SQL compatible machines. Each package also includes an Anaconda environment YAML file and directions for installing necessary package dependencies for each module.

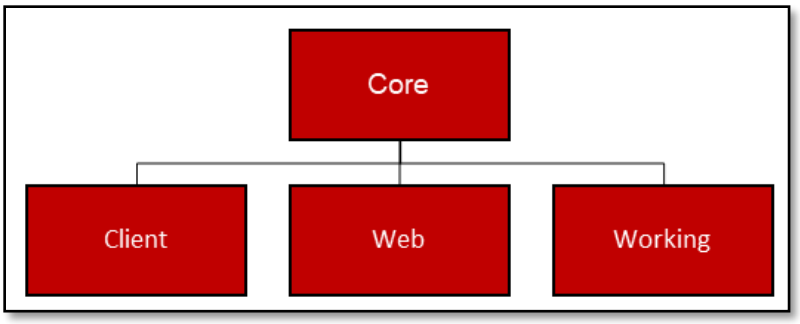


Figure 21: Python Package Hierarchy

9.1.1 Core Package

Location: https://github.com/wsp-sag/client_caltrans_road_user_core

As suggested by the name of the package, this Python repository contains most of the core software logic used by the PRIME system. This package is shared by the client, web, and working repositories to promote code reusability and modular software design principles.

The core package serves three main purposes. First, it contains the software and schema definitions to build the PRIME system from scratch in any SQL compliant database. Second, the core Python code includes all the JSON parsers necessary to restructure ICD-compliant JSON structures into CSV tables for data loading into the PRIME system (e.g., from AWS S3 to DynamoDB). Finally, it contains the core logic for intelligently interacting with the third-party data providers, AWS and Snowflake.

A guiding principle of the PRIME system is to avoid writing SQL syntax whenever possible. SQL is difficult to debug and test. The core package relies on a set of YAML expression files to build and interact with the PRIME Snowflake database system. The YAML files declare the structure of each data schema and unique characteristics like whether the table include geospatial information or whether data fields should be derived in process. When operating, the PRIME system interprets the YAML files, and creates the database structure and formats the input data records accordingly.

Part of the core package, which is responsible for taking in and processing data from BitBrew, is deployed on AWS Lambda. The rest of the core package contains modules that are utilized by the client, web and working packages. When updates are made to the core package, corresponding changes will be made to both AWS Lambda and the GitHub repository, so that the most recent version of the core package stays consistent between different platforms.

9.1.2 Client Package

Location: https://github.com/wsp-sag/client_caltrans_road_user_client

The client application library was initially developed for the previous pilot. It maintains and operates the ETL processes on the EC2 instance. This library largely drives the code of command-line executables to download and upload data from AWS and upload data into the Snowflake database.

This repository also contains the Windows Batch scripts that could be run as scheduled events on the EC2 instance.

9.1.3 Web Package

Location: https://github.com/wsp-sag/client_caltrans_road_user_web

The web application library maintains the code for the PRIME system administrative dashboards and external APIs. This code relies on the Python Flask and Dash libraries as related add-ons to provide a web-interfacing application and API for building the front-end of the PRIME administrative portal.

9.1.4 Working Package

Location: https://github.com/wsp-sag/client_caltrans_road_user_working

The working package largely contains scratch or test Jupyter notebooks used to isolate and test components of the PRIME system. These notebooks are often used as prototypes of features before they are added to one of the repositories described in the previous sections.

In many, but not all cases, the Jupyter notebooks maintain inline documentation explaining the test or procedure being implemented. Whenever possible, the notebooks use the core, client, or web packages to simulate actual operating conditions in the PRIME system.

9.2 SYSTEM ENVIRONMENTS

When a bug is found or a new feature is identified, code is written to support the system change, then deployed through 3 environments for code control and testing:

- **Testing Environment**: Allows developers, testers, and quality assurance (QA) validators to conduct unit, integration, and acceptance testing on the deployed code, how it interacts with existing code, and whether it meets project needs.
- **Staging Environment**: After code is validated in the Testing Environment, it is deployed to the Staging Environment which acts as a near copy of the Production Environment to complete one final stage of testing and validation in a production-like environment before deploying into Production.
- **Production Environment**: Where code is actually put into an operational setting and made available to users to manage the demonstration.

If maintenance is required to push a new feature or bug fix into production, the maintenance window is scheduled at a time that least impacts demonstration operations (such as the weekends).

Appendix A: Database Schema and Definitions

This appendix defines the tables in each schema, with field descriptions for each table.

Note: In the descriptions below, some data types include a number in parentheses after the data type (e.g., varchar(32), number(10,2)). In these cases, the data column has been constrained based on these parameters. For example, varchar(32) limits the data length to 32 characters. If no value is specified, the field defaults to its full capacity in Snowflake. However, Snowflake only uses the minimum space necessary to store each value in a table.

A.1 AGENTS

Table 2: Business Partners

Field Name	Data Type	Description
BPID	varchar	The business partner (BP) identifier number provided by the System Administrator.
BUSINESS_PARTNER_ID	varchar(36)	(primary key) Unique database-assigned identifier
CONTACT_EMAIL	varchar	Contact email address
CONTACT_NAME	varchar	Business partner representative / contact first and last name
CONTACT_NUMBER	varchar	Contact telephone number
DEMO_PHASE	number	Phase the business partner is associated with / operating
NAME	varchar	Business partner company name
NOTES	varchar	Freeform text for additional information
RECORD_DATE	date	Date when the record is added to Snowflake
STATUS	varchar	Status of business partner in demonstration
STATUS_DATE	date	Date of business partner status

Table 3: Certification

Field Name	Data Type	Description
CERTIFICATION_ID	varchar(36)	(primary key) Unique identifier assigned by the System Administrator denoting the associated mileage reporting option (MRO) has been certified
NOTES	varchar	Freeform text for additional information
RECORD_DATE	Timestamp_tz	Date when the record is added to Snowflake
STATUS	varchar	Status of the MRO's certification
STATUS_DATE	date	Date of MRO certification status

Table 4: Participant-Business Partner Client Map

Field Name	Data Type	Description
ACTIVE	boolean	Flag to indicate whether a participant is active with the associated business partner
BP_AUTHENTICATION_TOKEN	varchar	Unique authentication token for the participant, assigned by the business partner
BUSINESS_PARTNER_ID	varchar(36)	Unique database-assigned business partner identifier
CLIENT_MAP_ID	varchar(36)	(primary key)
ENROLLMENT_DATE	date	Date the participant enrolled with the business partner
NOTES	varchar	Freeform text for additional information
PARTICIPANT_BP_ID	varchar	Unique participant identifier assigned by the business partner
PARTICIPANT_ID	varchar(36)	Unique database-assigned participant identifier
RECORD_DATE	date	Date when the record is added to Snowflake

Table 5: MRO

Field Name	Data Type	Description
ACTIVE	boolean	Flag to indicate whether the MRO is active
ASSIGNED	boolean	Flag to indicate whether MRO has been assigned to a vehicle
CERTIFICATION_ID	varchar(36)	Identifier assigned by the System Administrator denoting the associated mileage reporting option has been certified
HARDWARE_VERSION	varchar	Major hardware (HW) Model number of the MRO
MAP_VERSION	varchar	Major map version number of the MRO
MRO_ID	varchar(36)	(primary key) Unique identifier for mileage reporting option (MRO) (such as serial number) associated to the account / vehicle
MRO_MODEL	varchar	Business partner-assigned model
MRO_VENDOR_ID	varchar	Vendors (e.g., Danlaw) Device ID
REPORTING_PERIOD_END	date	The end date and time of the period during which the miles being reported were traveled in.
REPORTING_PERIOD_START	date	The start date and time of the period during which the miles being reported were traveled in.
SOFTWARE_VERSION	varchar	Major software (SW) Model number of the MRO

Table 6: Participant

Field Name	Data Type	Description
CITY	varchar	City where the address is located
CONTACT_PHONE1	varchar	Participant contact telephone number
CONTACT_PHONE2	varchar	Additional participant contact telephone number (if provided)
EMAIL_ADDRESS	varchar	Email address of participant
ENROLLMENT_ACTIVE	boolean	Flag to indicate whether the participant is actively enrolled
ENROLLMENT_DATE	date	Date the participant enrolled
ENROLLMENT_TYPE	varchar	Indicator for targeted or public recruits
FIRST_NAME	varchar	First name of the participant
FORCE_CALIFORNIA	boolean	Flag to indicate whether participant's data should be "forced" to show as occurring in California or not (demonstration-specific field for targeted recruit use case)
LAST_NAME	varchar	Last name of the participant
NOTES	varchar	Free-form text field for providing additional notes/information
PARTICIPANT_ID	varchar(36)	(primary key) Unique database-assigned participant identifier
RECORD_DATE	date	Date when the record is added to Snowflake
STATE	varchar	State where the address is located
STREET_ADDRESS1	varchar	Street address, line 1, for participant
STREET_ADDRESS2	varchar	Street address, line 2, for participant
ZIP_CODE	varchar	Postal zip code where the address is located

Table 7: Recorder (reconciliation table)

Field Name	Data Type	Description
MRO_ID	varchar(36)	Unique identifier for mileage reporting option (MRO) (such as serial number) associated to the account / vehicle
PARTICIPANT_ID	varchar(36)	Unique database-assigned participant identifier
RECORDER_ID	varchar(36)	(primary key) Unique database-assigned identifier for reconciliation table record
RECORD_DATE	date	Date when the record is added to Snowflake
STATUS	varchar	Status of the participant / vehicle / MRO association
STATUS_DATE	date	Date of record status
UPDATED_BY	varchar(36)	Tool used to update this record
UPDATED_DATE	date	The most recent date when this record was updated
VEHICLE_ID	varchar(36)	Unique database-assigned identifier for vehicle record

Table 8: Vehicle

Field Name	Data Type	Description
ACTIVE	boolean	Flag to indicate whether the vehicle is still in use
EPA_MPG_RATING	number	Environmental Protection Agency (EPA) assigned combined, rounded miles per gallon (MPG) rating
FUEL_TYPE	number	A value indicating the type of fuel used by the vehicle • 1 = Gasoline • 2 = Diesel • 3 = Electric • 4 = Other
FUEL_USE_METHOD	number	A value indicating which fuel use method is used to calculate fuel usage • 1 = Actual fuel usage calculated • 2 = fuel usage calculated using EPA rules • 3 = fuel not taxable (e.g. electric vehicles)
LICENSE_PLATE	varchar	Vehicle license plate (if provided)
MAKE	varchar	Vehicle make
MODEL	varchar	Vehicle model
NOTES	varchar	Freeform text field for additional information
RECORD_DATE	date	Date when the record is added to Snowflake
REGISTRATION_STATE	varchar(2)	State vehicle is registered (if available)
TRIM	varchar	Vehicle trim (if available)
VEHICLE_ID	varchar(36)	(primary key) Unique database-assigned identifier for vehicle record
VIN	varchar	The vehicle identification number (VIN) of the vehicle for which the MRO is associated
YEAR	number	Vehicle year

A.2 BITBREW_2023PILOT

Table 9: Connect Messages

Field Name	Data Type	Description
CONNECT_DEVICE_TRIP_NUMBER	number	Device trip number when device connect event occurred
CONNECT_EVENT_TIME	timestamp_tz	Timestamp when device connect event occurred
CONNECT_ODOMETER	number	Odometer reading when device connect event occurred
DEVICE_CONNECT_ID	varchar(36)	(primary key) Unique identifier for each record

DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw

Table 10: Device Connect

Field Name	Data Type	Description
CONNECT_DEVICE_TRIP_NUMBER	number	Device trip number when device connect event occurred
CONNECT_EVENT_TIME	timestamp_tz	Timestamp when device connect event occurred
CONNECT_ODOMETER	number	Odometer reading when device connect event occurred
DEVICE_CONNECT_ID	varchar(36)	(primary key) Unique identifier for each record
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DISCONNECT_DEVICE_TRIP_NUMBER	number	Device trip number when device disconnect event occurred
DISCONNECT_EVENT_TIME	timestamp_tz	Device trip number when device disconnect event occurred
DISCONNECT_ODOMETER	number	Odometer reading when device disconnect event occurred
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw

Table 11: Event Load Log

Field Name	Data Type	Description
CHANGE_DATE	timestamp_tz	Timestamp when record was last updated
CHANGE_ROWS	number	Number of rows that were changed in the corresponding Snowflake table
CHANGE_TYPE	varchar(36)	Type of operation performed to the corresponding Snowflake table
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Identifier for each trip made by a given device
EVENT_DATE	timestamp_tz	Timestamp when the event happened
EVENT_LOAD_LOG_ID	varchar(36)	Unique identifier for each record in this table
EVENT_TYPE	varchar(15)	Type of the event, as defined by BitBrew
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
TRIP_EVENT_ID	varchar(36)	Unique identifier of trip the record is associated with

Table 12: GPS Event

Field Name	Data Type	Description
COLLECTION_TIME	timestamp_tz	Timestamp of the GPS Event
COUNTY_FIPS	varchar(3)	County FIPS of the GPS location
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Identifier for each trip made by a given device
GEOGRAPHY	geography	Well-known text (WKT) geographical point
GPS_EVENT_ID	varchar(36)	(primary key) Unique identifier for each GPS record
HEADING	number	Travel direction at the GPS location
INVALID_GPS	varchar(4)	Indicator of whether the device received sufficient GPS signal for positioning this point
IN_PUBLIC_LAND	varchar(3)	Indicator of whether the GPS point is located in public land
IN_ROAD_NETWORK	varchar(3)	Indicator of whether the GPS point suggests traveling on road or off road
IN_TRIBAL_LAND	varchar(3)	Indicator of whether the GPS point is located in tribal lands
LATITUDE	number(8,5)	Geographical latitude position
LONGITUDE	number(8,5)	Geographical longitude position
NONTRIP_GPS	varchar(4)	Indicator of whether the GPS record was collected not during a trip
ON_PRIVATE_ROAD	varchar(3)	Indicator of whether the GPS point suggests traveling on private road
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
ROAD_NAME	varchar(50)	Name of the road where the device was traveling along, if it was traveling on road
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
STATE_FIPS	varchar(3)	State FIPS of the GPS location
SUB_RULE_ID	varchar(5)	Unique code for a set of (one or more) business rules that apply to a given STATE_FIPS (e.g., road type and land type within a STATE_FIPS).
TRIBAL_BUFFER	varchar(3)	Indicator of whether the GPS point is near tribal land boundaries
TRIP_EVENT_ID	varchar(36)	Unique identifier for each trip
VIN	varchar(20)	The vehicle identification number (VIN) of the vehicle associated with the device that sent the GPS record

Table 13: Heartbeat

Field Name	Data Type	Description
COLLECTION_TIME	timestamp_tz	Timestamp of heartbeat event

Field Name	Data Type	Description
COUNTY_FIPS	varchar(3)	County FIPS of heartbeat event
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Identifier for each trip made by a given device
GEOGRAPHY	geography	WKT Point
HEARTBEAT_ID	varchar(36)	(primary key) Unique identifier for each heartbeat record
LATITUDE	number(8,5)	Geographical latitude position
LONGITUDE	number(8,5)	Geographical longitude position
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
SPEED	number(8,5)	Speed, if device is moving
STATE_FIPS	varchar(2)	State FIPS of the heartbeat event

Table 14; Lost Device

Field Name	Data Type	Description
COLLECTION_TIME	timestamp_tz	Last seen time of the device
DEVICE_ID	varchar(38)	Device ID of the missing device
DEVICE_TRIP_NUMBER	number	Last seen trip number of the missing device
HOURS_LOST	number	How long has the device been missing?
LATITUDE	number(8,5)	Last seen latitude
LONGITUDE	number(8,5)	Last seen longitude
LOST_DEVICE_ID	varchar(36)	(primary_key) Unique identifier for each lost device record
RECORD_DATE	timestamp_tz	Timestamp when records was inserted into Snowflake
RECORD_SOURCE	varchar(20)	Type of event when device was last seen (e.g., GPS, heartbeat)
SERIAL_NUMBER	varchar(12)	Serial number of the missing device

Table 15: Trip Event

Field Name	Data Type	Description
AIR_CONSUMED	number(11,3)	Amount of air consumed during the trip in grams (totaled from the Mass Airflow Sensor)
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Unique identifier for each trip made by a given device
DISTANCE_TRAVELLED	number(11,3)	Distance, in kilometers, traveled during the trip, as recorded by BitBrew
DISTANCE_TRAVELLED_MI	number(11,3)	Distance, in miles, traveled during the trip, as recorded by BitBrew
END_COUNTY_FIPS	varchar(3)	County FIPS of the trip end location

Field Name	Data Type	Description
END_GEOGRAPHY	geography	WKT point of the trip end location
END_LATITUDE	number(8,5)	Geographic latitude at the end of the trip
END_LONGITUDE	number(8,5)	Geographic longitude at the end of the trip
END_ODOMETER	number	Odometer reading provided by the device at the end of the trip
END_STATE_FIPS	varchar(3)	State FIPS at the trip end location
END_TRIP_TIME	timestamp_tz	Date/time when the trip ended
FUEL_CONSUMED	number(11,3)	Fuel, in gallons, consumed during the trip, as recorded by BitBrew
MIL_STATUS	number	Malfunction indicator light (MIL) status ("check engine light")
RECORD_DATE	timestamp_tz	Timestamp when records was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
START_COUNTY_FIPS	varchar(3)	County FIPS of at the trip start location
START_GEOGRAPHY	geography	WKT point of the trip start Location
START_LATITUDE	number(8,5)	Geographic latitude at the start of the trip
START_LONGITUDE	number(8,5)	Geographic longitude at the start of the trip
START_ODOMETER	number	Odometer reading provided by the device at the start of the trip
START_STATE_FIPS	varchar(3)	State FIPS at the trip start location
START_TRIP_TIME	timestamp_tz	Date/time when the trip started
TRIP_DURATION	number(8,2)	Duration of the trip in seconds
TRIP_EVENT_ID	varchar(36)	(primary key) Unique identifier for each trip
VIN	varchar(20)	The vehicle identification number (VIN) of the vehicle making the trip

Table 16: Trip Segment Event

Field Name	Data Type	Description
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Unique identifier for each trip made by a given device
DISTANCE_TRAVELLED	number(11,3)	Distance, in miles, traveled within the trip segment, as calculated from the trip segment's GPS trajectory
DISTANCE_TRAVELLED_BB	number(11,3)	Distance, in miles, traveled within the trip segment, as adjusted from DISTANCE_TRAVELLED to be consistent with BitBrew recorded travel distance
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
STATE_FIPS	varchar(4)	State FIPS where the trip segment occurred
SUB_RULE_ID	varchar(4)	Unique code for a set of (one or more) business rules that apply to a given STATE_FIPS (e.g., geographical region within a STATE_FIPS)

The two tables, "TSM_MYMILES_TABLE" and "TSM_TCA_TABLE", are exact replicas of the two views, "TSM_myMiles" and "TSM_TCA", respectively. The two views are described later in this section.

Table 17: Events (View)

Field Name	Data Type	Description
COLLECTION_TIME	timestamp_tz	Time of event
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Unique identifier for each trip made by a given device
EVENT_ID	varchar(36)	(primary key) Unique identifier for the event record
EVENT_TYPE	varchar(10)	Type of event (e.g., GPS, heartbeat, trip start, device connect)
LATITUDE	number(8,5)	Latitude of the device when the event happened
LONGITUDE	number(8,5)	Longitude of the device when the event happened
RECORD_DATE	timestamp_tz	Timestamp when records was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw

Table 18: Fuel by Trip (View)

Field Name	Data Type	Description
AIR_CONSUMED	number(11,3)	Amount of air consumed during the trip in grams (totaled from the Mass Airflow Sensor)
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Unique identifier for each trip made by a given device
DISTANCE_TRAVELLED	number(11,3)	Distance, in kilometers, traveled during the trip, as recorded by BitBrew
DISTANCE_TRAVELLED_MI	number(11,3)	Distance, in miles, traveled during the trip, as recorded by BitBrew
END_COUNTY_FIPS	varchar(3)	County FIPS of the trip end location
END_GEOGRAPHY	geography	WKT point of the trip end location
END_LATITUDE	number(8,5)	Geographic latitude at the end of the trip
END_LONGITUDE	number(8,5)	Geographic longitude at the end of the trip
END_ODOMETER	number	Odometer reading provided by the device at the end of the trip
END_STATE_FIPS	varchar(3)	State FIPS at the trip end location
END_TRIP_TIME	timestamp_tz	Date/time when the trip ended
FUEL_CONSUMED	number(11,3)	Fuel, in gallons, consumed during the trip, as recorded by BitBrew
FUEL_CONSUMPTION_EPA	number(21,12)	Fuel, in gallons, consumed during the trip, as calculated by BitBrew recorded travel distance divided by the EPA mpg of the traveling vehicle

Field Name	Data Type	Description
MIL_STATUS	number	Malfunction indicator light (MIL) status ("check engine light")
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
START_COUNTY_FIPS	varchar(3)	County FIPS of at the trip start location
START_GEOGRAPHY	geography	WKT point of the trip start Location
START_LATITUDE	number(8,5)	Geographic latitude at the start of the trip
START_LONGITUDE	number(8,5)	Geographic longitude at the start of the trip
START_ODOMETER	number	Odometer reading provided by the device at the start of the trip
START_STATE_FIPS	varchar(3)	State FIPS at the trip start location
START_TRIP_TIME	timestamp_tz	Date/time when the trip started
TRIP_DURATION	number(8,2)	Duration of the trip in seconds
TRIP_EVENT_ID	varchar(36)	Unique identifier for each trip
VIN	varchar(20)	The vehicle identification number (VIN) of the vehicle making the trip

Table 19: Fuel by Trip Seg (View)

Field Name	Data Type	Description
DEVICE_ID	varchar(38)	Unique identifier for each device, generated by BitBrew
DEVICE_TRIP_NUMBER	number	Unique identifier for each trip made by a given device
DISTANCE_TRAVELLED	number(11,3)	Distance, in miles, traveled within the trip segment, as calculated from the trip segment's GPS trajectory
DISTANCE_TRAVELLED_BB	number(11,3)	Distance, in miles, traveled within the trip segment, as adjusted from DISTANCE_TRAVELLED to be consistent with BitBrew recorded travel distance
FUEL_CONSUMPTION_BB	number(31,12)	Fuel, in gallons, consumed within the trip segment, as calculated by partitioning BitBrew recorded fuel consumption for the trip to the constituent trip segments in proportion to the trip segment lengths
FUEL_CONSUMPTION_EPA	number(17,9)	Fuel, in gallons, consumed within the trip segment, as calculated by dividing the DISTANCE_TRAVELLED by the EPA mpg of the traveling vehicle
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw
STATE_FIPS	varchar(4)	State FIPS where the trip segment occurred
SUB_RULE_ID	varchar(4)	Unique code for a set of (one or more) business rules that apply to a given STATE_FIPS (e.g., geographical region within a STATE_FIPS).
TRIP_EVENT_ID	varchar(36)	The TRIP_EVENT_ID of the trip that the trip segment belongs to

Table 20: Last Seen (View)

Field Name	Data Type	Description
COLLECTION_TIME	timestamp_tz	Time of the last seen event
DEVICE_ID	varchar(38)	Device ID of the last seen device
DEVICE_TRIP_NUMBER	number	Last seen trip number, id the last seen event is a trip
EVENT_ID	varchar(36)	Unique identifier for each last seen event
EVENT_TYPE	varchar(10)	Type of the last seen event (e.g., GPS, heartbeat, trip start, device connect).
LATITUDE	number(8,5)	Latitude where the device was last seen
LONGITUDE	number(8,5)	Longitude where the device was last seen
RECORD_DATE	timestamp_tz	Timestamp when record was inserted into Snowflake
SERIAL_NUMBER	varchar(12)	Unique identifier for each device, provided by Danlaw

Table 21: Participant Travel by Month (View)

Field Name	Data Type	Description
BUSINESS_PARTNER	varchar	Business partner the participant is associated with
EMAIL_ADDRESS	varchar	Email address of the participant
FIRST_NAME	varchar	Participant's first name
LAST_NAME	varchar	Participant's last name
MONTH	number(2,0)	Month for which to summarize the travel distance
PARTICIPANT_ID	varchar(36)	Unique database-assigned participant identifier
TOTAL_TRAVEL_MI	number(30,9)	Distance, in miles, traveled in the given month by the participant

Table 22: Trip by BP (View)

Field Name	Data Type	Description
BP_ID	varchar(36)	Business partner (BP) identifier provided by the System Administrator.
BP_NAME	varchar	Business partner (BP) name
SERIAL_NUMBER	varchar(12)	Serial number of the device making the trip
START_TRIP_TIME	timestamp_tz	Timestamp when the trip started
TRIP_EVENT_ID		Trip event ID of the device making the trip

Table 23: TSM TCA (View)

Field Name	Data Type	Description

BPCUSTOMERNO	varchar	Unique identifier for each participant associated with a given business partner
BPID	varchar	Unique identifier for each business partner
CERTID	varchar(36)	Certification ID number assigned by the System Administrator
FUELTYPE	number	A value indicating the type of fuel used by the vehicle • 1 = Gasoline • 2 = Diesel • 3 = Electric • 4 = Other
FUELUSEMETHOD	number	A value indicating which fuel use method is used to calculate fuel usage • 1 = Actual fuel usage calculated • 2 = fuel usage calculated using EPA rules • 3 = fuel not taxable (e.g. electric vehicles)
FUEL_CREDIT_RATE	number	Fuel tax credit rate in dollars per gallon
MROADJBALANCEINSUBRULEID	number	Balance adjusted for this account/VIN/MRO in given Sub Rule ID for given period
MROADJFUELTAXCREDITINSUBRULEID	number	Fuel Tax Credit adjusted for this account/VIN/MRO in given Sub Rule ID for given period
MROADJFUELUSAGEINSUBRULEID	number	Fuel Usage adjusted for this account/VIN/MRO in given Sub Rule ID for given period
MROADJMILEAGEINSUBRULEID	number	Miles adjusted for this account/VIN/MRO in given Sub Rule ID for given period
MROADJROADCHARGEINSUBRULEID	number	Gross Road Charge adjusted for this account/VIN/MRO in given Sub Rule ID for given period
MROBALANCEINSUBRULEID	number	Amount to invoice/credit Participant (dollars) in the given SubRuleID for the given transaction
MROFUELTAXCREDITINSUBRULEID	number	Fuel tax credit amount (dollars) in the given SubRuleID for the given transaction
MROFUELUSAGEINSUBRULEID	number	Fuel consumption (gallons) in the given SubRuleID for the given transaction
MROID	varchar	Serial number of the device
MROMILEAGEINSUBRULEID	number	Distance traveled (miles) in the given SubRuleID for the given transaction

MROROADCHARGEINSUBRULEID	number	Gross road use charge amount (dollars) in the given SubRuleID for the given transaction
ROAD_CHARGE_RATE	number	Road use charge rate in dollars per mile
RULEID	varchar(4)	State FIPS for each trip segment in the given transaction
SUBRULEID	varchar(4)	Unique code for a set of (one or more) business rules that apply to a given RuleID (e.g., geographical region within a RuleID).
TRANSACTIONID	varchar(36)	Identifier of the trip associated with the transaction
TRANSACTIONSDATERANGEEND	varchar	Date and time when the transaction ended
TRANSACTIONSDATERANGESTART	varchar	Date and time when the transaction started
TRANSMITTEDTIMESTAMP	varchar	Date and time when the view was generated
VIN	varchar	Vehicle identification number (VIN) of the vehicle used for the transaction
VINSTATUS	number(1,0)	An integer value indicating the status of a given VIN at the period end date of the report

The "TSM_myMiles" view has the exact same field definitions as the "TSM_TCA" view, and thus will not be described separately.

A.3 FINANCE

Table 20: Road_User_Charge_Rates

Field Name	Data Type	Description
ROAD_USER_CHARGE_RATES_ID	varchar(36)	(primary key)
RULE_ID	varchar(2)	Unique code for a fixed-rate region, such as a U.S. State or Canadian territory.
SUBRULE_ID	number	Unique code for a set of (one or more) business rules that apply to a given RuleID (e.g. geographical region within a RuleID).
ROAD_CHARGE_FLAG	boolean	Yes/No flag to indicate whether a road charge rate is applicable in a given RuleID/SubRuleID.
ROAD_CHARGE_RATE	number(5,3)	Road charge rate amount in dollars
FTC_FLAG	Boolean	Yes/No flag to indicate whether a fuels tax credit rate is applicable in a given RuleID/SubRuleID.
FTC_RATE	number(5,3)	Fuel tax credit rate amount in dollars
PRIORITY	number	Integer used to indicate the processing order for SubRuleID. Each set of SubRuleID and the associated

Field Name	Data Type	Description
		business rules under a RuleID shall be processed in sequential order.
STATE	varchar	Name of U.S. state associated with RuleID
START_EFFECTIVE_DATE	date	Effective start date in which RuleID/SubRuleID combination and associates rates and business rules are in effect.
END_EFFECTIVE_DATE	date	Effective end date in which RuleID/SubRuleID combination and associates rates and business rules are in effect.

A.4 SECURITY

Table 21: Users

Field Name	Data Type	Description
USER_ID	varchar(36)	(primary key)
USERNAME	varchar(40)	Username for user granted access
PASSWORD	varchar	(masked) password for associated user
ROLE	varchar	Role(s) user has been granted access to
FIRM	varchar	Company user associated with
TOKEN	varchar	User authentication token
TOKEN_EXPIRATION	timestamp_tz	Date/timestamp authentication token expires

A.5 WEB

Table 22: MRO Last Seen (View)

Field Name	Data Type	Description
MRO_ID	varchar(36)	(primary key) Unique identifier for mileage reporting option (MRO) (such as serial number) associated to the account / vehicle
MRO_VENDOR_ID	varchar	MRO vendor database ID
MRO_MODEL	varchar	Vendor-assigned MRO model
ACTIVE	boolean	True/false – is the device active?
PARTICIPANT_ID	varchar(36)	Unique database-assigned participant identifier
LAST_NAME	varchar	Last name of the participant
FIRST_NAME	varchar	First name of the participant
VEHICLE_ID	varchar(36)	Unique database-assigned identifier for vehicle record
VIN	varchar	The vehicle identification number (VIN) of the vehicle for which the MRO is associated
LAST_SEEN_EVENT	varchar(10)	Event (GPS, trip start or end, heartbeat) that was last reported by the vehicle / device

Field Name	Data Type	Description
LAST_SEEN	timestamp_tz	Last time vehicle / device was seen / reported

Appendix B: Schema Definition File Layout

The schema definition YAML files provide a linkage between third-party file formats and the PRIME Snowflake database. They also provide a convenient mechanism to build and query the database through the metadata stored in the YAML files. The schema is composed of four key sections which are outlined below.

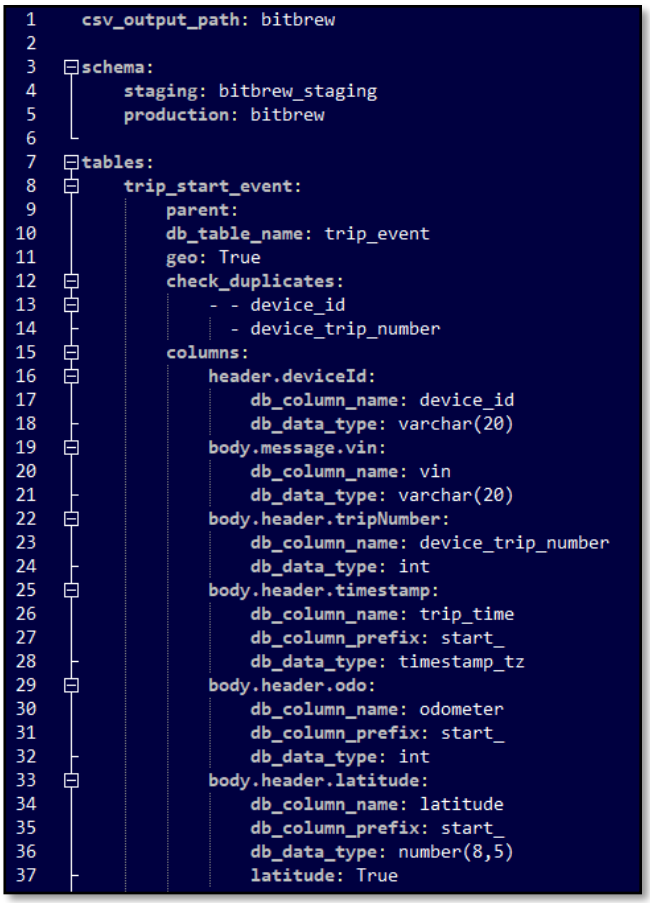


Figure 22: Excerpt of YAML Schema

csv_output_path

Path or folder to search in the working directory for intermediate files associated with this definition.

schema

The schema names for production and staging where the tables and view described in this file reside.

staging: <Staging Schema Name>p

production: <Production Schema Name>

tables / view

parent: <Name of parent table, if needed>

db_table_name: <Table name in the database>

geo: <Is the table geo-enabled?>

check_duplicates: <During the load validation, which columns should be used to check for duplicates>

columns: <Column metadata, see components below>

The column metadata contains the following attributes:

- db_column_name: <column name in the database>
- db_data_type: <database data type>
- prefix: <whether a prefix should be appended to the column name>
- id: <field to use to uniquely identify records, otherwise primary key used>
- latitude: <latitude column for geo enabled tables>
- longitude: <longitude column for geo enabled tables>
- geography: <identifies the geography column where the WKT object is stored>
- calculated: <whether the field is calculated on the fly in Python>