Ministry of Transportation
and Infrastructure

Trip Data Submission Service API Documentation

for

Passenger Transportation Data Warehouse

PTDW

Published By:	Information Management Branch Ministry of Transportation and Infrastructure
Document Version:	1.2
Application Release:	1.0
Creation Date:	October 16, 2019
Last Updated Date:	Nov 20, 2019

Version Control

Version	*Date*	*Change Description*	
1.0	10/16/2019	Initial approved version	James Smith Sateesh Boggarapu Claus Gottschling
1.1	11/08/2019	Updated date formats	Sateesh Boggarapu Claus Gottschling
1.2	11/20/2019	Further guidance and enhancements to the Authorization Steps section and diagram. XML file size max 500 MB	James Smith

Contents

List of Tables

1 Introduction

This document describes the details of the Trip Data Submission API available for passenger transportation license holders. The API is part of the Passenger Transportation Data Warehouse (PTDW) solution and provides a means for system-to-system transfers of trip data. This is provided as an alternative to using the file upload feature on the *VehicleSafetyBC* portal to submit shift and trip data. The service API is a REST endpoint that is publicly accessible but secured through the use of a secure API token.

The portal and the API can be used together concurrently. This could be beneficial in the case of resubmissions to correct quality issues in specific data sets.

The primary document for reference is the *Trip Data Submission Guide and Specification*; the document outlines the requirements and process for submitting trip data in general. It explains the specifications of the data file in both accepted formats: CSV and XML. However, the API defined in this document only accepts XML as the trip data submission file.

The PTDW API is a REST based web service. The API has predictable resource-oriented URLs, accepts form-encoded request bodies, returns JSON-encoded responses, and uses standard HTTP response codes, authorization, and verbs.

The licensee will be required to structure the trip and shift data in XML and submit through this REST API. In order to transfer the data file, the licensee must first request an API token that will facilitate secure access. This is done through the *VehicleSafetyBC* portal and is a fully automated function. The API token should be stored securely by the licensee and used to generate a JSON Web Token (JWT) as defined in this document; for every connection attempt with the API.

The API will respond with a success or failure response code; if failed further details of the *API access denied* is provided. Once the API access has been authorized then the XML file will be submitted for processing. Because the process of ingesting the data goes through several stages of validation and quality assurance, further notifications of progress are sent through email to a designated licensee email recipient or can be viewed on the *VehicleSafetyBC* portal. The details of this are defined in the primary guide and specifications document.

Trip data submission can be made at any time of day and any day of the year. The data can be no smaller than a single day and no larger than one complete month. Any duration between this range is also acceptable as long as the range of dates does not cross into another month (only data from one month can be included in the submission). The XML file can be no larger than 500 MB.

The licensee may also attempt another submission if the previous attempt failed or was flagged for quality issues. The resubmitted XML would contain corrections made from the notifications given in the previous submission. If there are flagged issues in the submission that have been identified in the Quality Assurance stage of processing, then the licensee may re-submit a date range with corrections. In this case the entire data set will be overwritten and must be a complete set.

The rules for submission, resubmission and the processing of the submitted file are identical regardless of the means of submission; either through this API or the user portal. The API provides a means to automate the data file transfers between the licensee system and the Passenger Transportation Data Warehouse.

The URL for the production endpoint is:

https://analytics.th.gov.bc.ca/ptdw-api/api/ext/v1/submitdata/submitData

2 Authorization

Because the Ministry is providing a public facing web service that allows the upload of data into government data stores; significant security and oversight must be taken to ensure that the correct parties are accessing the API and that only verified and validated data will be uploaded. Any caller to the API must be authorized, but as a first measure all API requests must be made over HTTPS; any call made over plain HTTP will fail.

The REST API *Authorization* method that secures access to this endpoint is distinct from the user *Authentication* required for the *VehicleSafetyBC* portal access. The API is designed for System-to-System access and should be fully automated by the calling system. For this to happen the calling system will need to code the appropriate security measures defined here so that access can be authorized.

The approach is for the Ministry to generate a token in a secure manner which the organization will then receive and store. The primary value of the token is a UUID value that the Ministry generates. The organization will then use the token to gain access when it calls the API. The UUID and some additional claims are cryptographically signed and encrypted by the organization using a shared key and public/private key provided by the Ministry. The organization will request the UUID and keys through the VehicleSafetyBC portal by logging in with a BCeID account associated with the PT License. This is typically done by a System Admin who can make the request and provide the information to system developers / operators of their own dispatch system.

Following are the details of how the token is generated, stored, transported and authorized. The Ministry has adopted a simple API Token Generation approach; this is used across industry as a general standard.

2.1 Authorization Steps

There are several steps to request, receive and use the token. The first steps are manual (the SysAdmin through the portal), and then the remaining steps should be fully automated by the calling system.

a) A SysAdmin logs into the VehicleSafetyBC portal with a valid BCeID

b) The BCeID is passed to BCeID for Authorization

c) BCeID authenticates the user and provides the reference identifier for the associated business.

d) The SysAdmin is presented with the API Token generator screen

e) A request for a token for the PT Licensee holder is made (only one token per licensee PT number)

f) The API token request is verified and made to the MOTI system

g) A UUID (unique key) is generated and associated with this PT Number (and the shared secret key). Only the hash of the UUID is stored in the MOTI DB.

h) The UUID is returned to the end user in the portal

i) The licensee is informed MOTI does not retain the UUID; once logged out the UUID is not recoverable.

j) A UUID is provided along with a shared secret key (for signing) and a Public key (for encryption)

k) The Organization securely stores the token, shared key, and Public Key in their system

l) When the organization wishes to upload data through the API they must generate the token

   I. A JSON Web Token (JWT) is created with the appropriate claims (defined in the next section)

   II. The JWT is signed (to ensure it cannot be tampered with)

   III. The signed JWT is encrypted

IV. The JWT is used to access the API for trip data submission through HTTPS

m) The JWT is used in the API HTTP request, in the “Authorization” request header and passed to the REST API endpoint

I. The service will validate the authentication by decrypting, verifying the header and comparing the hash value of the UUID with the stored hash value in the MOTI DB

II. Access is granted, logged and the trip data now begins processing.

These steps are illustrated in the figure on the following page

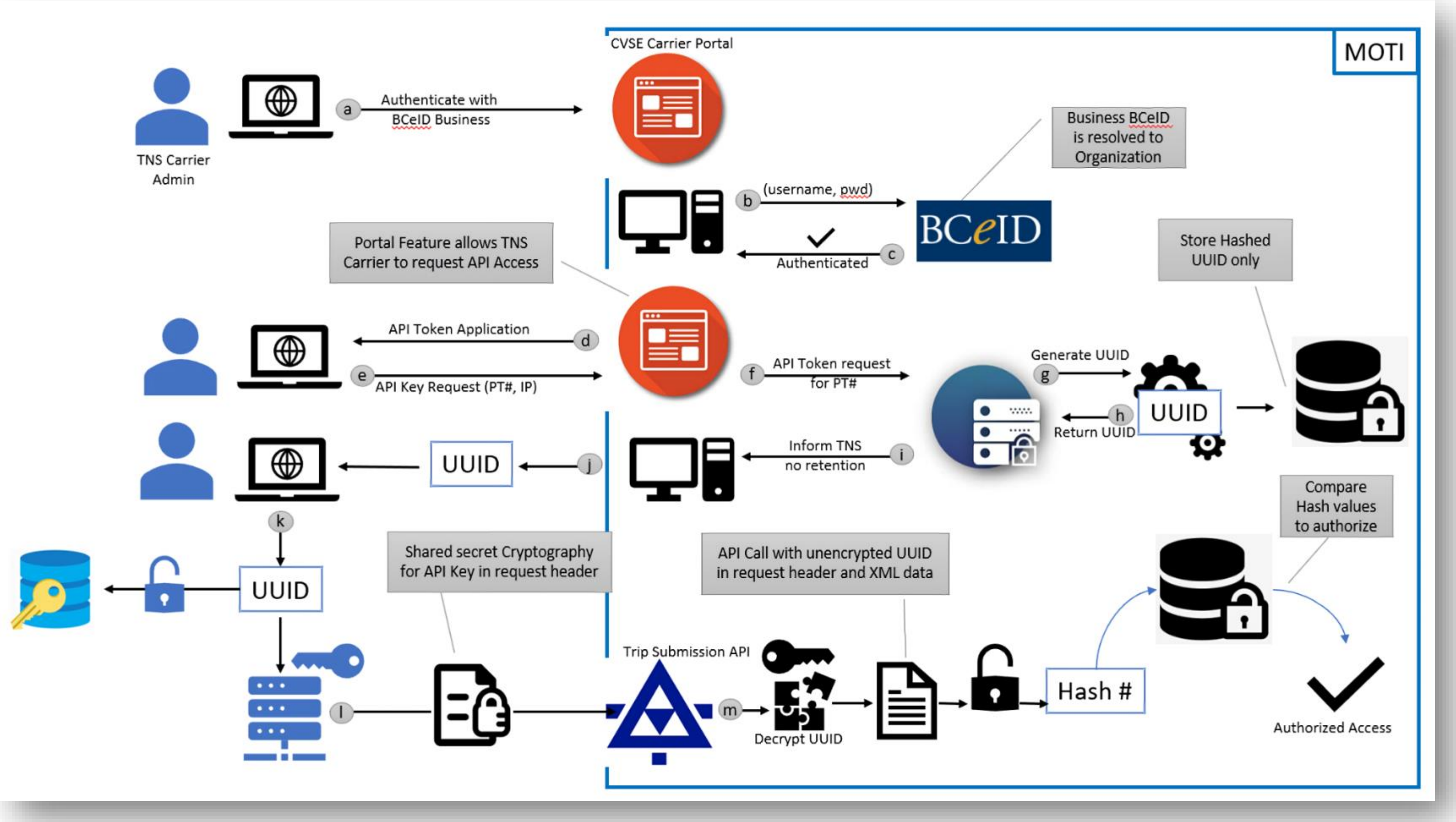
CVSE Carrier Portal
MOTI
TNS Carrier Admin
a
Authenticate with BCeID Business
Business BCeID is resolved to Organization
b
(username, pwd)
BCeID
c
Authenticated
Portal Feature allows TNS Carrier to request API Access
Store Hashed UUID only
API Token Application
d
e
API Key Request (PT#, IP)
f
API Token request for PT#
Generate UUID
g
h
Return UUID
UUID
UUID
j
Inform TNS no retention
i
k
UUID
Shared secret Cryptography for API Key in request header
API Call with unencrypted UUID in request header and XML data
Compare Hash values to authorize
Trip Submission API
l
m
Decrypt UUID
Hash #
Authorized Access

Figure 1: Authorization Process

Notes

- The JWT will include the standard *Header.Payload.Signature* structure.
- The header will include the type of token (JWT) and the signing algorithm (HS256).
- The JSON should be Base64Url encoded for the first part of the JWT.
- The second part is the payload which contains the claims. There are two types of claims: registered and private.
  - *Registered Claims:* Only the Issuer (iss) and Subject (sub) registered claims will be used.
    - The issuer should be the Organization name that is submitting the data (note this may be a different organization than the TNS and is only used for information purposes and not validated).
    - The subject should be "TripData".
  - *Private Claims:* There are 4 Private Claims:
    - the UUID. An example would be: "uuid": "4ea9c787-fb80-4d3b-b796-8142bb1c193e".
    - the PT licensee number: "ptnum": “123546”
    - the date of creation: “created”: "2019-07-22T16:54:27.014Z"
    - and a value provided by the calling party to uniquely identify this call to the API, an Organization Submission Value (OSV); "osv": "[any string of up to 12 characters]"
- The signature will be created using the header and payload; it should be signed with the Shared Key using the HS256 algorithm and encoded in Base64.
  - Note that this only ensures that the contents have not been tampered with. The entire JWT needs to be encrypted using a public/private key provided to ensure safe transport (in addition to TLS).
- By storing the hashed UUID, when we provide the key to the system owner they get it only once – we would have no record of it and could only verify it is correct after receiving it and hashing it for comparison in the Ministry DB.
  - This is so that if our DB is compromised none of the keys would be at risk since only the hashed values are stored.
  - The does mean that the system owner can never get the key from us again as we no longer have a record of it (a new key would need to be generated).

2.2 JWT Encoding example

In this example, the originator will create a JSON Web Token (JWT) and then encrypt it to become a JWE. The Originator will be supplied with a UUID, Shared Key, and Public Key. The JWT will be signed using the Shared Key then the entire JWT will be encrypted into a JWE using the public key.

2.2.1 Step 1: Compose JSON values for the JWT.

The JSON Web Token will have six claims. Below is a sample JWT:

```
{
```

```
"iss": "Name of Issuer",
"sub": "TripData",
"uuid": "4ea9c787-fb80-4d3b-b796-8142bb1c193e",
"ptnum": "123546",
"created": "2019-07-22T16:54:27.014Z",
"osv": "00001"
}
```

The claims inside the JWT:

- "iss": Originator name: This should be the organization that is accessing the API. For example, a software company that provides a dispatch service to license holders; the software company name should be used here.
- "sub": This must be set to "TripData"
- "uuid": Enter the UUID provided
- "ptnum": Enter your PT Licensee Number. Note that this PT License Number must match with the UUID provided and match the PT License Number provided in the API call.
- "created": Enter the current date and time in UTC. The token will expire within 5 minutes of this time to ensure that the token cannot be reused if it has somehow been accessed in transit.
- "osv": The Organization Submission Value. Enter a value to uniquely identify this call to the API for your own tracking purposes. This can be any alpha-numeric value up to 12 characters.

When composed with the header, the JWT will look like this:

```
{
"alg": "HS256",
"typ": "JWT"
}
{
"iss": "Name of Issuer",
"sub": "TripData",
"uuid": "4ea9c787-fb80-4d3b-b796-8142bb1c193e",
"ptnum": "123546",
"created": "2019-07-22T16:54:27.014Z",
"osv": "00001"
}
```

This JWT will be signed with the Shared Key using the HS256 algorithm and encoded in Base64.

2.2.2 Step 2: The base JWT will then be encrypted with the public key.

It will look like this:

```
{
  "alg": "RSA-OAEP",
  "enc": "A256GCM"
}
{
[The encrypted JWT payload is here]
}
```

The JWE will be encrypted using RSA-OAEP and A256GCM.

Sample encoding: Here is a sample encoded JWT. The JWT contains this value:

```
{
"iss": "John's Cloud Services",
"sub": "TripData",
"uuid": "4ea9c787-fb80-4d3b-b796-8142bb1c193e",
"ptnum": "70271",
"created": "2019-07-22T16:54:27.014Z",
"osv": "00001"
}
```

Using a Shared Key of "sample", the JWT is encoded and signed:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ew0KImlzcyI6ICJKb2huJ3MgQ2xvdWQgU2VydmljZX
MiLA0KInN1YiI6ICJUcmlwRGF0YSIsDQoidXVpZCI6ICI0ZWE5Yzc4Ny1mYjgwLTRkM2ItYjc5Ni04M
TQyYmIxYzE5M2UiLA0KInB0bnVtIjogIjcwMjcxIiwNCiJjcmVhdGVkIjogIjIwMTktMDctMjJUMTY6
NTQ6MjcuMDE0WiIsDQoib3N2IjogIjAwMDAxIg0KfQ.m9uOHOmSm0eAUwVLl8mW7TDgH1RKDwvqf3tZ
axm3pq8
```

Encrypting with a public key, the encoded JWT becomes:

```
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.ZxdimAddL_dc1-Zb2pz67aQROtS9U-
Apfdd-u3EBB4EybAIB-
v4j_K60topMUUm3lI7cvoudK4HaPm1J3YTOZNjhq6N0SN16QqpLFe5rsaRIS2X9E3MGIzHaIRRmcTgo
u651FAmZZAT7i6qkQf31qdzFCH-wbxAQe2sORGCmxhI.ABcBN__zZSgkN5N6.79sr-
vGHv7nSmHm0ocooGpkY3duGnLY5aRw-
fgRgVQF01RKiw3SdgABlSx2Vm7mJ4izrEpduNrrXIdfr2H5tL83lGtirQLYZQ-
8Ntlk4A_P3rXhznmKothwMNMWMQZw5x3tQWSVBqMWSPaFu7ZgLuyed3KEhLF4mC3-pzoDNijeVKODX-
lLoVRvrZI405EoCcz7fzi5zxRxgejM_urpRSHuy_TspxHLl_9z6jQw-
G36GdGD0cmy8Y0aUfQ4aMCiqsmrLYTCdYYmfZ53JcPpCysiI39CHU9hS9BtortI-
xeSybSqHDJVP2mCUmRfzOdJuabuIm1YHJpkbliRHG7kjl2psVGKQK5hXBt_A3zJ3hhTCQ9GFZs-D-
BMpO1cW_Sw3zBTlZFUwHEte-WQIwXNjTM4qB6eIQCYq4Rjkla0PlEMA-
Qo.oTXEqLmW6DoCOFDWB0kkGg
```

Notes:

To explore JWT encoding, you may use the website https://jwt.io/ which can be used to encode and decode tokens. (This website does not handle JWE).

On this page at https://jwt.io/, entering the encoded JWE shows the decoded header information indicating this is a valid JWE.

Entering the signed, encoded JWT reveals the readable text of the JSON. The signature, which is the key "sample", can be entered in the verify signature area. The page will then indicate that the signature of the JWT is valid.

2.2.3 Using the Encoded JWE in an API Submission

The encoded JWE should be placed in the HTTP request header in the Authorization. The format of this value is

```
Bearer <encoded jwe>
```

Which will look like this:

```
Bearer eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.ZxdimAddL_dc1-
Zb2pz67aQROtS9U-Apfdd-u3EBB4EybAIB-
v4j_K60topMUUm3lI7cvoudK4HaPm1J3YTOZNjhq6N0SN16QqpLFe5rsaRIS2X9E3MGIzHaIRRmcTgo
u651FAmZZAT7i6qkQf31qdzFCH-wbxAQe2sORGCmxhI.ABcBN__zZSgkN5N6.79sr-
vGHv7nSmHm0ocooGpkY3duGnLY5aRw-
fgRgVQF01RKiw3SdgABlSx2Vm7mJ4izrEpduNrrXIdfr2H5tL83lGtirQLYZQ-
8Ntlk4A_P3rXhznmKothwMNMWMQZw5x3tQWSVBqMWSPaFu7ZgLuyed3KEhLF4mC3-pzoDNijeVKODX-
lLoVRvrZI405EoCcz7fzi5zxRxgejM_urpRSHuy_TspxHLl_9z6jQw-
G36GdGD0cmy8Y0aUfQ4aMCiqsmrLYTCdYYmfZ53JcPpCysiI39CHU9hS9BtortI-
xeSybSqHDJVP2mCUmRfzOdJuabuIm1YHJpkbliRHG7kjl2psVGKQK5hXBt_A3zJ3hhTCQ9GFZs-D-
BMpO1cW_Sw3zBTlZFUwHEte-WQIwXNjTM4qB6eIQCYq4Rjkla0PlEMA-
Qo.oTXEqLmW6DoCOFDWB0kkGg
```

3 API Definition

The request accepts 4 parameters in the form of JSON object. All the parameters are mandatory. Refer section 8 for a sample request call.

- **PTNo** - Passenger Transportation Number of the carrier company (Ex: - 70277, 70242)
- **XMLData** - This data is the encoded format of the XML data. The XML should be encoded in base 64 format and the encoded data need to be send as the parameter. Refer to ***Error! Reference source not found. Error! Reference source not found.*** for more details.
- **StartDate** – UTC start date of the trip data submitted. Accepted date format is YYYY-MM-DD.
- **EndDate** – UTC end date of the trip data submitted. Accepted date format is YYYY-MM-DD. The start / end date cannot span more than 31 days and more than one calendar month.

3.1 API Responses

The API response is returned in the form of JSON.

<u>**JSON Object**</u>

```
{
       “Response Code”: <Code>
       “Response Body”: <Text>
}
```

Example: -

```
{
      “Response Code”: “406”,
      “Response Body”: “The XML uploaded may be a duplicate. (Same SHA-256 hash).”
}
```

Please refer to *Table 1: Response Codes* for the list of response codes and their descriptions.

3.2 Response Code Details

The below table lists down all the response code and its description.

Table 1: Response Codes

Response Code	*Details*
200	Successfully Processed the Request
406	Incorrect PT Number
406	Invalid PT Number: <PTNumber> in request
406	Incorrect Start Date
406	Incorrect End Date

Response Code	Details
406	Date is greater than the current date
406	Start Date is greater than the End Date
406	Start and End date cannot span over more than a month
406	The XML uploaded may be a duplicate. (Same SHA-256 hash).
406	Exception checking the SHA-256 hash.
406	Trip data provided is empty
406	File size crosses the limit range
406	Error adding database information for upload.
406	Incorrect Request

3.3 API Request Call

The below example will show how the API can be called using the curl command. Replace the Authorization header with the current value, and the XML Data with base64 encoded XML Data.

```
curl -X POST --header 'Authorization: Bearer
eyJhbGciOiJSU0EtT0FFUCIsImVuYyI6IkEyNTZHQ00ifQ.ZxdimAddL_dc1-Zb2pz67aQROtS9U-Apfdd-
u3EBB4EybAIB-
v4j_K60topMUUm3lI7cvoudK4HaPm1J3YTOZNjhq6N0SN16QqpLFe5rsaRIS2X9E3MGIzHaIRRmcTgou651FAm
ZZAT7i6qkQf31qdzFCH-wbxAQe2sORGCmxhI.ABcBN__zZSgkN5N6.79sr-
vGHv7nSmHm0ocooGpkY3duGnLY5aRw-
fgRgVQF01RKiw3SdgABlSx2Vm7mJ4izrEpduNrrXIdfr2H5tL83lGtirQLYZQ-
8Ntlk4A_P3rXhznmKothwMNMWMQZw5x3tQWSVBqMWSPaFu7ZgLuyed3KEhLF4mC3-pzoDNijeVKODX-
lLoVRvrZI405EoCcz7fzi5zxRxgejM_urpRSHuy_TspxHLl_9z6jQw-
G36GdGD0cmy8Y0aUfQ4aMCiqsmrLYTCdYYmfZ53JcPpCysiI39CHU9hS9BtortI-
xeSybSqHDJVP2mCUmRfzOdJuabuIm1YHJpkbliRHG7kjl2psVGKQK5hXBt_A3zJ3hhTCQ9GFZs-D-
BMpO1cW_Sw3zBTlZFUwHEte-WQIwXNjTM4qB6eIQCYq4Rjkla0PlEMA-Qo.oTXEqLmW6DoCOFDWB0kkGg' --
header 'Content-Type: application/json' --header 'Accept: text/plain' -d '{ "PTNo":
"70271",
"XMLData":"PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4NCjxQYXNzZW5nZXJUcmlwIHh
tbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hlbWEtaW5zdGFuY2UiIHhzaTpub05hbWVzc
GFjZVNjaGVtYUxvY2F0aW9uPSJQYXNzZW5nZXJUcmlwLnhzZCI+DQogICAgPEhlYWRlcj4NCiAgICAgICAgPFV
zZXJJRD5CaWdXYWxseTwvVXNlcklEPg0KICAgICAgICA8QXBwbGljYXRpb25JRD5CaWdEdW1iV2FsbEFwcDwvQ
XBwbGljYXRpb25JRD4NCiAgICAgICAgPFBUTm8+NzAyNzE8L1BUTm8+DQogICAgICAgIDxOU0NObz43MDI3MTw
vTlNDTm8+DQogICAgICAgIDxTdmNUeXBDZD5UQVhJPC9TdmNUeXBDZD4NCiAgICAgICAgPFN0YXJ0RHQ+MjAxO
S0wMS0wMVo8L1N0YXJ0RHQ+DQogICAgICAgIDxFbmREdD4yMDE5LTAxLTAyWjwvRW5kRHQ+DQogICAgICAgIDx
DcmVhdGVEVD4yMDE4LTEyLTMxVDA4OjAyOjAwLjkwN1o8L0NyZWF0ZURUPg0g==", "StartDate": "2019-
01-01", "EndDate": "2019-01-02" }' 'https://<prod>/ptdw-
api/api/ext/v1/submitdata/submitData'
```

Appendix A: Abbreviations

Table 2: Abbreviations

Abbreviation	*Details*
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
JSON	JavaScript Object Notation
JWT	Java Web Token
JWE	Java Web Encryption
MOTI	Ministry of Transportation and Infrastructure
PROD	Production
PTDW	Passenger Transportation Data Warehouse
PTMP	Passenger Transportation Modernization Project
REST	Representational State Transfer
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
XML	Extensible Markup Language
XSD	XML Schema Definition