VCC Public API

Virginia Tech Transportation Institute

Version 3.1, September 2024

VCC - Public API - Overview

Overview

The Virginia Connected Corridor (VCC) project provides a REST interface which allows external applications to interact with the system. It is available at:

```
https://vcc.vtti.vt.edu
```

The VCC Public API provides access to the messages flowing on the VCC, such as the SAE J2735 MapData and SPAT messages.

The REST endpoints that are available within the VCC Public API are grouped and described by function. The Data Transfer Objects (DTOs) used by the REST endpoints to communicate data are described in the sections on DTOs. Links to the appropriate DTO are included in the documentation for the corresponding REST endpoint.

Decoding SAE J2735 Messages

The SAE J2735 messages are encoded using the specification from March, 2016. Documentation describing this specification is available from SAE. All message types are encoded as a Message Frame. Since the J2735 messages are binary, within this Pubic API, the binary messages are typically base64-encoded into a string for ease of transfer.

The following steps can be used to decode the data in a base64-encoded message, such as MAP, SPAT, or BSM:

1. Convert from base64 to hex using https://www.asciitohex.com or similar
2. Decode the hex as an SAE International J2735 2016-03 (r70)/DSRC Message Frame on https://www.marben-products.com/decoder-asn1-automotive

JSON Messages

To allow client applications to easily utilize the data within the various J2735 messages, a JSON version of the messages is also available. The JSON version of the messages is quite similar to the corresponding J2735 message, but is sometimes simplified. The JSON structures are defined in this document.

VCC - Public API - Authorization

Since the REST interface is stateless, login is not used for authorization purposes. Instead, a special token is used to provide access to the API. There are two types of tokens that may be used for authorization. Only one will be used per client.

Authorization using JSON Web Tokens (JWT)

New clients that wish to access the VCC Public API will be given a client_id and a client_secret specifically generated for that client. These are essentially the username and password for the new client application. The client_id and client_secret are used to request a JSON Web Token (JWT). After receiving a JWT, the client includes the JWT in the Authorization header of all additional HTTP requests. The JWT will expire after a fixed amount of time, specified within the JWT, and a new JWT will need to be requested. If a request from a client does not include the JWT or the JWT is invalid or expired, the endpoints will return a 401 (Unauthorized) error. The following request is used to request a JWT.

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
POST	api/auth/client		Request a JWT		ClientCredentials	JWT	400

This is a python example that illustrates the use of JSON Web Tokens for authentication.

Example using Python 3

```python
# python3

import asyncio
import websockets
import requests

token_url = "https://vcc.vtti.vt.edu/api/auth/client"
client_id = 'your client_id' # TODO: replace with your client_id
client_secret = 'your client_secret' # TODO: replace with your client_secret

# Step A - use your client credentials to obtain a JWT
data = {'client_id': client_id, 'client_secret': client_secret}

access_token_response = requests.post(token_url, data=data, allow_redirects=False)
json_response = access_token_response.json()
access_token = json_response.get('access_token')
print("access token: " + access_token)

# Step B - use the access token within the JWT to make desired api calls
api_call_headers = {'Authorization': 'Bearer ' + access_token}
r = requests.get('https://vcc.vtti.vt.edu/api/bsm/key', headers=api_call_headers)      # for example
r = requests.get('https://vcc.vtti.vt.edu/api/mapdata/152', headers=api_call_headers)  # for example
```

Authorization using Application Tokens

This method of authorization has been **deprecated**, but continues to be supported for existing clients. New clients should use JWT, described above.

Clients that use Application Tokens for authorization will be given an appToken that is generated specifically for that client. This appToken is used as the username for Basic Authorization. Basic Authorization also requires a password, but the password is ignored.

This is a Python example that illustrates the use of the application token as a username.

AppToken Authentication

```python
import requests

r = requests.get('https://vcc.vtti.vt.edu/api/mapdata/152', auth=('MY_APP_TOKEN','password_is_ignored'))

print(r.status_code)
print(r.content)
```

If using Postman to make the request, the Auth tab can be used the specify the authentication. The option "Basic Auth" should be selected from the drop-down list and the appToken should be entered in the username field. The password field can be left blank.

VCC - Public API - BSM

The REST endpoints for accessing BSM (Basic Safety Message) messages from the VCC are described below.

Authentication is required for any endpoint with a Role specified.

<table>
<tr><th>Method</th><th>URI</th><th>Parameters</th><th>Description</th><th>Role</th><th>Data In</th><th>Data Out</th><th>Error Code</th></tr>
<tr><td>GET</td><td>api/bsm/current</td><td></td><td>Get a list of the current BSM.</td><td>PUBLIC_API</td><td>none</td><td>list of BsmWithMetadata</td><td></td></tr>
<tr><td>POST</td><td>api/bsm/base64</td><td></td><td>Post a binary J2735 BSM that has been base64 encoded into a string.</td><td>PUBLIC_API</td><td>String</td><td>BsmReceived</td><td>400</td></tr>
<tr><td>POST</td><td>api/bsm/binary</td><td></td><td>Post a binary J2735 BSM.</td><td>PUBLIC_API</td><td>binary</td><td>BsmReceived</td><td>400</td></tr>
<tr><td>POST</td><td>api/bsm/json</td><td></td><td>Post a BSM in JSON format.</td><td>PUBLIC_API</td><td>Bsm</td><td>BsmReceived</td><td>400</td></tr>
<tr><td>GET</td><td>api/bsm/key</td><td></td><td>Get a key (a string) to use when opening a WebSocket for BSM. It is the string to use for PROVIDED_TOKEN. This key can only be used once and is only valid for a short period of time. This key is valid for both WebSocket urls.</td><td>PUBLIC_API</td><td>none</td><td>String</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>WebSocket (wss)</td><td>ws/bsm</td><td>• key=PROVIDED_TOKEN
• rsu=RSU_NAME_FILTER (optional)</td><td>This url is used to open a WebSocket over which lists of BSM messages are sent from the server at 2 Hz. If no BSM have been received recently, an empty list will be sent over the WebSocket. This helps keep the WebSocket open and allows the client to confirm that the connection is working properly. Although BSM are typically generated at 10 Hz, to reduce the amount of data that is sent over the WebSocket, the BSM are thinned to 2 Hz.

This WebSocket is bidirectional and can be used by the client to upload BSM to the server. Each BSM sent from the client is sent as a string and the BSM are sent one at a time, rather than in a list. The BSM from the client do not need to be sent at 2 Hz.

The PROVIDED_TOKEN is generated by the /api/bsm/key endpoint. The RSU_NAME_FILTER is a regular expression. If specified, only BSM that were sent from an RSU that matches one of the RSU_NAME_FILTERs will be sent over the WebSocket. If the rsu parameter is not specified, all BSM will be returned. If multiple filters are desired, the rsu parameter may be specified multiple times.

Some examples of RSU_NAME_FILTER:
<table><tr><th>RSU_NAME_FILTER</th><th>Matches on</th></tr><tr><td>VA7_TycoRd</td><td>The one RSU named VA7_TycoRd</td></tr><tr><td>VA7_.*</td><td>All six RSUs with names that start "VA7_"</td></tr><tr><td>I-66_.*</td><td>All RSUs with names that start "I-66_"</td></tr><tr><td>.*Prosperity.*</td><td>All RSUs with "Prosperity" in their names</td></tr><tr><td>From_WebSocket</td><td>BSM that are uploaded through a WebSocket rather than through an RSU.</td></tr></table></td><td>none</td><td>Optional series of Strings, where each string is a BSM (either a Base64-encoded J2735 BSM or a JSON Bsm)</td><td>A series of lists of BsmWithMetadata

*Note that this format is not backwards compatible with the old Public API.</td><td></td></tr>
</table>

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
			<table><tr><th>RSU_NAME_FILTER</th><th>Matches on</th></tr><tr><td>From_Post</td><td>BSM that are uploaded through an HTTP POST request rather than through an RSU.</td></tr></table>				
WebSocket (wss)	ws/bsm/upload	• key=PROVIDED_TOKEN	This url is used to open a WebSocket for uploading BSM, that can optionally include additional metadata, to the server. The PROVIDED_TOKEN is generated by the /api/bsm/key endpoint. This WebSocket is bidirectional. The server will periodically (every 5 seconds) send counts of the number of BSM received over the WebSocket that were valid and invalid. This will keep the WebSocket open, even during long stretches when the client does not upload a BSM. Also, it provides a way for the client to verify that all BSM were received by the server and were in an expected format.	none	BsmWithMetadataUpload	A series of lists, each with two elements: the number of valid BSM received over the WebSocket and the number of invalid BSM received over the WebSocket. example for five valid and one invalid BSM received: [5, 1]	

Examples

Open a WebSocket to receive BSM

receiveBsmOverWebSocket.py

```
# python3

import asyncio
import websockets
import requests

token_url = "https://vcc.vtti.vt.edu/api/auth/client"
client_id = 'your client_id'             # TODO:  replace with your client_id
client_secret = 'your client_secret'     # TODO:  replace with your client_secret

# Step A - request JWT using your client credentials
data = {'client_id': client_id, 'client_secret': client_secret}

access_token_response = requests.post(token_url, data=data, allow_redirects=False)
json_response = access_token_response.json()
access_token = json_response.get('access_token')

# Step B - use the JWT access token to request a key for a BSM WebSocket
api_call_headers = {'Authorization': 'Bearer ' + access_token}

r = requests.get('https://vcc.vtti.vt.edu/api/bsm/key', headers=api_call_headers)

print(r.status_code)
bsmKey = r.content.decode('utf-8')
print(bsmKey)

# Step C - use the bsmKey to open a WebSocket to receive BSM.
async def listen_to_websocket():
    uri = "wss://vcc.vtti.vt.edu/ws/bsm?key=" + bsmKey
    async with websockets.connect(uri) as websocket:
        for i in range (10):
            message = await websocket.recv()
            print(f"Received: {message}")

asyncio.get_event_loop().run_until_complete(listen_to_websocket())



# The examples below illustrate a few options for controlling which BSM are
# returned over the WebSocket.  These lines would replace the uri line above.
#-------------------------------------------------

# Example 1:  Open a WebSocket that will return BSM from all sources since
```

```
# the parameter "rsu" is not specified.  This line is used above.
# uri = "wss://vcc.vtti.vt.edu/ws/bsm?key=" + bsmKey

# Example 2:  Open a WebSocket that will return BSM from all RSUs with names
# that start VA7_.
# uri = "wss://vcc.vtti.vt.edu/ws/bsm?rsu=VA7_.*&key=" + bsmKey

# Example 3:  Open a WebSocket that will return BSM from either of two specific RSU:
# VA7_TycoRd and VA7_Pike7Plaza.
# uri = "wss://vcc.vtti.vt.edu/ws/bsm?rsu=VA7_TycoRd&rsu=VA7_Pike7Plaza&key=" + bsmKey

# Example 4:  Open a WebSocket that will return BSM that are from sources
# other than RSUs.  If you have an application that is POSTing BSM, this is
# a handy way to receive those BSM for testing purposes.
# uri = "wss://vcc.vtti.vt.edu/ws/bsm?rsu=From_WebSocket&rsu=From_Post&key=" + bsmKey
```

Upload a BSM over a WebSocket

uploadBsmOverWebSocket.py

```
# A BSM can be uploaded in two separate string formats:  either a Base64-encoded
# J2735 binary BSM or a JSON BSM.  Each BSM is sent separately.
# Here is an example of each:

  ws.send("ABQlGlFAAAAPCiXnuj6drx8/E9MGBgAAcACfWv36H6EAf/+AAPAPAA==")
  ws.send("{ \"coreData\": { \"msgCnt\": 1, \"id\": 42, \"lat\": 38.8732164, \"lon\": -77.2468636 } }")
```

VCC - Public API - MapData

The REST endpoints for accessing MapData messages from the VCC are described below.

Authentication is required for any endpoint with a Role specified.

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
GET	api/mapdata	• changedSinceTimestamp=LONG (optional) • receivedSinceTimestamp=LONG (optional, defaults to changedSinceTimestamp or one day ago)	Get the latest MapData message for each intersection in J2735 format, optionally limiting the list based on when the message was last changed and/or last received from an RSU. This endpoint is backwards compatible with Public API v1.	PUBLIC_API	none	List of MapDataSummary	
GET	api/mapdata/decoded	• changedSinceTimestamp=LONG (optional) • receivedSinceTimestamp=LONG (optional, defaults to changedSinceTimestamp or one day ago)	Get the latest MapData message for each intersection in JSON format, optionally limiting the list based on when the message was last changed and/or last received from an RSU.	PUBLIC_API	none	List of MapData	
GET	api/mapdata/{id}		Get a single MapData message in J2735 format. This endpoint is backwards compatible with Public API v1.	PUBLIC_API	integer intersection ID	MapDataSummary	404
GET	api/mapdata/{id}/decoded		Get a single MapData message in JSON format.	PUBLIC_API	integer intersection ID	MapData	404

VCC - Public API - SPAT

The REST endpoints for accessing SPAT (Signal Phase and Timing) messages from the VCC are described below.

Authentication is required for any endpoint with a Role specified.

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
GET	api/spat	• format= [JSON_COMPLETE | UPER | JSON] (optional, defaults to UPER)	Get latest SPAT for each intersection. This endpoint is backwards compatible with the Public API v1. • If format = JSON_COMPLETE, each message string is a SPAT (New JSON Format) • If format = JSON, each message string is a SPAT (Legacy JSON Format) This format has been deprecated. • If format = UPER, each message string is a Base64-encoded J2735 SPAT message	PUBLIC_API	none	List of strings in the requested format	400
GET	api/spat/{id} api/spat/all	• format= [JSON_COMPLETE | UPER | JSON | JSON_METADATA] (optional, defaults to UPER)	Get a URL that can be used to open a WebSocket and receive SPAT for the one specified intersection or for all intersections. The token returned from this url is only valid for a few seconds and can only be used a single time. This endpoint is backwards compatible with Public API v1. For backwards compatibility, a format of JSON is the old simplified JSON. The options of JSON_COMPLETE and JSON_METADATA provide a more complete JSON message. JSON_COMPLETE provides the complete J2735 message. JSON_METADATA adds some metadata to the JSON_COMPETE.	PUBLIC_API	integer intersection ID	SpatWebSocketUrl	400, 404
WebSocket (wss)	ws/spat	• key=PROVIDED_TOKEN	This url is used to open a WebSocket over which SPAT messages are sent each time they change. The url to use, complete with the required token is provided by the /api/spat/{id} endpoints. Each message sent over the WebSocket is a string. The format of the string is determined by the format specified when the url is requested. • If format = JSON_METADATA, each message string is a SpatWithMetadata. • If format = JSON_COMPLETE, each message string is a SPAT (New JSON Format) • If format = JSON, each message string is a SPAT (Legacy JSON Format) This format has been deprecated. • If format = UPER, each message string is a Base64-encoded J2735 SPAT message	none		A series of strings in the requested format	

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
			When a WebSocket is opened for a single intersection, the most recent SPAT will be sent immediately, but only if it is not too old and is therefore likely currently valid.				

VCC - Public API - PSM

The REST endpoints for accessing PSM (Personal Safety Message) messages from the VCC are described below.

Authentication is required for any endpoint with a Role specified.

Method	URI	Parameters	Description	Role	Data In	Data Out	Error Code
GET	api/psm/current		Get a list of the current PSM.	PUBLIC_API	none	list of PsmWithMetadata	
POST	api/psm/base64		Post a binary J2735 PSM that has been base64 encoded into a string.	PUBLIC_API	String	PsmReceived	400
POST	api/psm/binary		Post a binary J2735 PSM.	PUBLIC_API	binary	PsmReceived	400
POST	api/psm/json		Post a PSM in JSON format.	PUBLIC_API	Psm	PsmReceived	400
GET	api/psm/key		Get a key (a string) to use when opening a WebSocket for PSM. It is the string to use for PROVIDED_TOKEN. This key can only be used once and is only valid for a short period of time.	PUBLIC_API	none	String	
WebSocket (wss)	ws/psm	• key=PROVIDED_TOKEN • rsu=RSU_NAME_FILTER (optional) • format=[JSON | V2016 | BOTH] (optional, defaults to JSON)	This url is used to open a WebSocket over which lists of PSM messages are sent from the server at 2 Hz. If no PSM have been received recently, an empty list will be sent over the WebSocket. This helps keep the WebSocket open and allows the client to confirm that the connection is working properly. If PSM are generated faster than 2 Hz, to reduce the amount of data that is sent over the	none	Optional series of Strings, where each string is a PSM (either a Base64-encoded J2735 PSM or a JSON Psm)	A series of lists of PsmWithMetadata	

			WebSocket, the PSM are thinned to 2 Hz. This WebSocket is bidirectional and can be used by the client to upload PSM to the server. Each PSM sent from the client is sent as a string and the PSM are sent one at a time, rather than in a list. The PSM from the client do not need to be sent at 2 Hz. The PROVIDED_TOKEN is generated by the /api/psm/key endpoint. The RSU_NAME_FILTER is a regular expression. If specified, only PSM that were sent from an RSU that matches one of the RSU_NAME_FILTERs will be sent over the WebSocket. If the rsu parameter is not specified, all PSM will be returned. If multiple filters are desired, the rsu parameter may be specified multiple times. See the table later in the documentation for some examples of RSU_NAME_FILTER. The format parameter specifies the desired format of messages sent from the server. The PsmWithMetadata JSON object will be populated based on the format parameter. The format parameter does not specify the format used by the client. The client can send either a JSON PSM or Base64-encoded J2735 PSM.				
Web Sock	ws/psm/upload	• key=PROVIDED_TOKEN	This url is used to open a WebSocket for uploading PSM, that can optionally	none	PsmWithMetadataUpload	A series of lists, each with two elements: the	

et (wss)			include additional metadata, to the server. The PROVIDED_TOKEN is generated by the /api/psm/key endpoint. This WebSocket is bidirectional. The server will periodically (every 10 seconds) send counts of the number of PSM received over the WebSocket that were valid and invalid. This will keep the WebSocket open, even during long stretches when the client does not upload a PSM. Also, it provides a way for the client to verify that all PSM were received by the server and were in an expected format.			number of valid PSM received over the WebSocket and the number of invalid PSM received over the WebSocket. example for five valid and one invalid PSM received: [5, 1]	

Examples of RSU_NAME_FILTER

RSU_NAME_FILTER	Matches on
VA7_TycoRd	The one RSU named VA7_TycoRd
VA7_.*	All six RSUs with names that start "VA7_"
I-66_.*	All RSUs with names that start "I-66_"
.*Prosperity.*	All RSUs with "Prosperity" in their names
From_WebSocket	PSM that are uploaded through a WebSocket rather than through an RSU.
From_Post	PSM that are uploaded through an HTTP POST request rather than through an RSU.

Examples

Open a WebSocket to receive PSM

receivePsmOverWebSocket.py

```python
# python3

import asyncio
import websockets
import requests
```

```
6
7  token_url = "https://vcc.vtti.vt.edu/api/auth/client"
8  client_id = 'your client_id'             # TODO:  replace with your client_id
9  client_secret = 'your client_secret'     # TODO:  replace with your client_secret
10
11 # Step A - request JWT using your client credentials
12 data = {'client_id': client_id, 'client_secret': client_secret}
13 access_token_response = requests.post(token_url, data=data, allow_redirects=False)
14 json_response = access_token_response.json()
15 access_token = json_response.get('access_token')
16
17 # Step B - use the JWT to request a key for a PSM WebSocket
18 api_call_headers = {'Authorization': 'Bearer ' + access_token}
19 r = requests.get('https://vcc.vtti.vt.edu/api/psm/key', headers=api_call_headers)
20
21 print(r.status_code)
22 psmKey = r.content.decode('utf-8')
23 print(psmKey)
24
25 # Step C - use the psmKey to open a WebSocket to receive PSM.
26 async def listen_to_websocket():
27     uri = "wss://vcc.vtti.vt.edu/ws/psm?key=" + psmKey
28     async with websockets.connect(uri) as websocket:
29         for i in range (10):
30             message = await websocket.recv()
31             print(f"Received: {message}")
32
33 asyncio.get_event_loop().run_until_complete(listen_to_websocket())
34
35
36 # The examples below illustrate a few options for controlling which PSM are
37 # returned over the WebSocket.
38 #-----------------------------------------------
39
40 # Example 1:  Open a WebSocket that will return PSM from all sources since
41 # the parameter "rsu" is not specified.  This line is used above
42 # uri = "wss://vcc.vtti.vt.edu/ws/psm?key=" + psmKey
43
44 # Example 2:  Open a WebSocket that will return PSM from all RSUs with names
45 # that start VA7_.
46 # uri = "wss://vcc.vtti.vt.edu/ws/psm?rsu=VA7_.*&key=" + psmKey
47
48 # Example 3:  Open a WebSocket that will return PSM from either of two RSU:
49 # VA7_TycoRd and VA7_Pike7Plaza.
50 # uri = "wss://vcc.vtti.vt.edu/ws/psm?rsu=VA7_TycoRd&rsu=VA7_Pike7Plaza&key=" + psmKey
51
52 # Example 4:  Open a WebSocket that will return PSM that are from sources
53 # other than RSUs.  If you have an application that is POSTing PSM, this is
54 # a handy way to receive those PSM for testing purposes.
55 # uri = "wss://vcc.vtti.vt.edu/ws/psm?rsu=From_WebSocket&rsu=From_Post&key=" + psmKey
```

Upload a PSM over a WebSocket

uploadPsmOverWebSocket.py

```
# A PSM can be uploaded in two separate string formats:  either a Base64-encoded
# J2735 binary PSM or a JSON PSM.  Each PSM is sent separately.
# Here is an example of each:

  ws.send("ACAaAAAAkhjEAAAAAEvZYj07XZhvegBm3gAAAAA=")
  ws.send("{ \"msgCnt\": 1, \"id\": 42, \"position\": { \"lat\": 38.8732164, \"lon\": -77.2468636 } }")


```

VCC - DTO - Authentication

DTOs Used for Authentication

There are no items with the selected labels at this time.

VCC - DTO - Auth - ClientCredentials

Authentication is being handled by OAuth2 and JSON Web Tokens (JWT). There are a variety of OAuth2 client libraries that can be used to simplify this process on the clients.

Client credentials are used to request a JWT. The client Information must be provided as x-www-form-urlencoded parameters and the corresponding header must be set accordingly.

The required HTTP request header:

- Content Type - application/x-www-form-urlencoded

The required request parameters are:

- client_id = ClientID. The client_id provided for this client, typically the application name.
- client_secret = ClientSecret. The client_secret for this ClientID, typically a long string of alphanumeric characters with a few dashes separating groups of characters.

Since the content-type is application/x-www-form-urlencoded, these values must be sent as form-encoded values and not a JSON object.

VCC - DTO - Auth - Credentials

Overview

An overview of the authentication process is provided in the Authentication Service.

Authentication for Applications That Require User Login

Authentication is being handled by OAuth2 JSON Web Tokens (JWT). Hopefully, by using OAuth2, there will be libraries that help with the implementation of JWT on the clients. The credentials needed to request a JWT include information about both the client and the user and are specified as a combination of HTTP request headers and request parameters.

Two HTTP request headers are needed:

- Authorization - Basic authorization using the ClientID and the ClientSecret
- Content Type - application/x-www-form-urlencoded

The Authorization header uses the standard Basic format, except it uses client information instead of the user information. In this case ClientID:ClientSecret where ClientID will be assigned to each client/application, examples are `uma` and `wzb`. The ClientSecret is the app token generated by the server for each app. If specifying the header directly, the string needs to be base64 encoded. For example, for a ClientID of "uma" and a ClientSecret of "secret", the header would be:

Authorization Header

```
Authorization: Basic dW1hOnNlY3JldA==
```

Additionally, the following request parameters are required:

- grant_type = password. The actual string "password".
- username = the user's username, typically an email address
- password = the user's password

Since the content-type is application/x-www-form-urlencoded, these values should be sent as form-encoded values and not a JSON object.

Authentication for Applications That Do Not Require User Login

For applications that do not require user login, see the section on ClientCredentials.

VCC - DTO - Auth - JWT (JSON Web Token)

JWT

The response for a token is the following JSON body:

Example token JSON response


```
{    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTQ4MTc2NTgsImlhdCI6MTYxNDgxNDA1OCwianRpIjoi
    "expires_in": 3600,
    "refresh_expires_in": 1800,
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9....",
    "token_type": "bearer",
    "not-before-policy": 0,
    "scope": "tbd"
}
```


The field expires_in is specified in seconds.

The value for access_token is the JWT. That is the token that needs to be sent in the HTTP Authorization header as a Bearer token in all future requests, such as:

Example authorization header


```
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2MTQ4MTc2NTgsImlhdCI6MTYxNDgxNDA1OCwianRpIjoi
```


Additional Information

The client application does not need to parse the JWT or access the information within it. The following provide additional information about what is contained in the JWT for informational purposes only.

The JWT consists of three pieces separated by periods in this order: Header.Payload.Signature. Each piece is Base64 encoded. The first two sections from this example are shown decoded below.

Example JWT section 1


```
{
    "alg": "HS256",
    "typ": "JWT"
}
```


Example JWT section 2


```
{
    "exp": 1614817658,
    "iat": 1614814058,
    "jti": "0628686c-e278-46dd-8813-a289e17ef8ee",
    "iss": "https://vcc.vtti.vt.edu/auth/realms/VCC",
    "sub": "6016cf83-2922-4ee3-b793-99f10bacae7d",
    "typ": "Bearer",
    "realm_access": {
        "roles": ["ROLE_PUBLIC_API"]
    },
    "clientId": "vcc-test"
}
```


The fields within the payload can be updated, as needed.

VCC - DTO - BSM

DTOs For BSM

There are no items with the selected labels at this time.

VCC - DTO - BSM - Bsm

BasicSafetyMessage (BSM) JSON

This BSM JSON uses the J2735 structure and field names. Fields that have a scaling factor described in the J2735 documentation have the scaling factor applied in this JSON object. For example, rather than expressing latitude as an integer in 1/10 micro degrees, it is scaled to degrees and stored as a double. When scaling, fields with a special value that indicates unavailable are converted to null.

This JSON structure is used for POSTing a BSM.

Bsm JSON

```
Bsm = {
  "coreData": BsmCoreData
  // Note:  Part II data could be added here in the future
}

BsmCoreData = {
  "msgCnt": Int,        // values 0-127
  "id": Int,            // converted from 4 byte octet string, i.e TemporaryID,
  "secMark": Int?,      // values 0-65_535, in milliseconds, 65535 = unavailable
  "lat": Double?,       // in degrees, converted from integer in 1/10 micro degrees
  "lon": Double?,       // in degrees, converted from integer in 1/10 micro degrees
  "elev": Float?,       // in meters, converted from integer in 10 cm, -4096 = unknown
  "accuracy": PositionalAccuracy?,
  "transmission": Int?, // enum with values 0-7, i.e.  TransmissionState, 7 = unavailable
  "speed": Float?,      // in m/s, converted from integer in 0.02 m/s units, 8191 = unavailable, input 0-8191
  "heading": Float?,    // in degrees, converted from integer in 0.0125 degrees, input 0-28_800, 28_800 = unavailab
  "angle": Float?,      // in degrees, converted from integer in 1.5 degrees, input -126-127, 127 = unavailable
  "accelSet": AccelerationSet4Way?,   // to support nullable, the yaw acceleration will default to 0
  "brakes": BrakeSystemStatus?,
  "size": VehicleSize?
}

PositionalAccuracy = {
  "semiMajor": Float?,   // in meters, converted from values 0-255, i.e. SemiMajorAxisAccuracy, 255 = unavailable
  "semiMinor": Float?,   // in meters, converted from values 0-255, i.e. SemiMinorAxisAccuracy, 255 = unavailable
  "orientation": Float?  // in degrees, converted from values 0-65_535, i.e.  SemiMajorAxisOrientation, 65_535 = un
}

AccelerationSet4Way = {
  "lon": Float?,  // in m/s^2, converted from Acceleration, values -2000-+2001, in 0.01 m/s^2, 2001=unavailable
  "lat": Float?,  // in m/s^2, converted from Acceleration, values -2000-+2001, in 0.01 m/s^2, 2001=unavailable
  "vert": Float?, // in G, converted from VerticalAcceleration, values -127-+127, in 0.02 G, -127=unavailable
  "yaw": Float    // in degrees/second, converted from YawRate, values -32767-+32767, in 0.01 degrees/s, not nullab
}

BrakeSystemStatus = {
  "wheelBrakes": Int,   // convert from BitString BrakeAppliedStatus, bit 0 = unavailable (bit 0 = 0x10 = 16)
  "traction": Int,      // enum TractionControlStatus, 0 = unavailable
  "abs": Int,           // enum AntiLockBrakeStatus, 0 = unavailable
  "scs": Int,           // enum StabilityControlStatus, 0 = unavailable
  "brakeBoost": Int,    // enum BrakeBoostApplied, 0 = unavailable
  "auxBrakes": Int      // enum AuxiliaryBrakeStatus, 0 = unavailable
}

VehicleSize = {
  "width": Int?,        // in cm, values 0-1023, 0 = unavailable
  "length": Int?        // in cm, values 0-4095, 0 = unavailable
}
```

Example BSM JSON

This is an example of JSON that can be used for POSTing a BSM.

Example BSM JSON


```
{
  "coreData: {
    "msgCnt": 1,
    "id": 42,
    "secMark": 12345,
    "lat": 38.8732164,
    "lon": -77.2468636,
    "elev": 84.1,
    "accuracy": {
        "semiMajor": 0.35,
        "semiMinor": 0.35,
        "orientation": 0.0
    },
    "transmission": 2,
    "speed": 10.4,
    "heading": 92.7,
    "angle": 4.5,
    "accelSet": {
        "lon": 0.1,
        "lat": 0.2,
        "vert": 0.003,
        "yaw": 0.4
    },
    "brakes": {
        "wheelBrakes": 0,
        "traction": 1,
        "abs": 1,
        "scs": 1,
        "brakeBoost": 1,
        "auxBrakes": 1
    },
    "size": {
        "width": 162,
        "length": 473
    }
  }
}
```


Simplest Example for Posting

Any field not specified, except yaw accel, will be set to unavailable. Since yaw accel doesn't have a value for unavailable, it will be set to 0. In this example, all optional fields have been omitted.

Simplest Example for Posting a BSM


```
{
  "coreData": {
    "msgCnt": 1,
    "id": 42,
    "lat": 38.8732164,
    "lon": -77.2468636
  }
}
```

VCC - DTO - BSM - BsmReceived

This is a small JSON object that is returned in response to a BSM being successfully POSTed. It contains a couple fields from the POSTed BSM.

BsmReceived JSON

```
BsmReceived = {
  "msgCnt": Int,
  "id": Int
}
```

Example BsmReceived JSON

BsmReceived Example

```
{
    "msgCnt": 1,
    "id": 42
}
```

VCC - DTO - BSM - BsmWithMetadata

This JSON structure is used when the server provides a list of BSM. In addition to containing the data from the J2735 BSM message, it contains additional metadata about the receipt of the BSM. Note that null metadata fields may be omitted to reduce the size of the JSON. See the Bsm JSON format for details on the bsmJson field.

BsmWithMetadata JSON

```
{
  "timestamp": Long,        // epoch timestamp in milliseconds at which the BSM was received by VCC Cloud
  "rsuName": String,        // The name of the RSU that sent the BSM or "From_WebSocket" or "From_Post"
  "format": String,         // The format in which the BSM was received, includes:  V2016, V2009, JSON, JSON_PLUS, V2016_PLUS
  "clientId": String?,      // The client_id of the application that POSTed a BSM or null if received from an RSU
  "locationName": String?,  // Client-provided location, such as: intersection, work zone, street, school, etc.
  "sourceType": String?,    // Client-provided source that generated BSM, such as RSU, OBU, Phone, Tablet, Smart Intersection
  "vendorName": String?,    // Client-provided name of vendor of source, such as:  Bluecity, Iteris-Conti, DERQ, Commsignia
  "vendorVersion": String?, // Client-provided versioning info of software and/or hardware that generated BSM
  "misc": String?,          // Client-provided extra field for project-specific uses
  "bsmJ2735": String?,      // Optionally included.  The Base64-encoded J2735 BSM, only populated if the BSM was received in this format
  "bsmJson": Bsm            // The JSON version of the BSM data
}
```

Example

Example BsmWithMetadata

```
{
    "timestamp": 1617220157230,
    "rsuName": "From_WebSocket",
    "format": "JSON",
    "clientId": "test-client",
    "locationName": "US-29 & Nutley",
    "sourceType": "Smart Intersection",
    "vendorName": "DERQ",
    "vendorVersion": "rev1",
    "misc": "red light event 123",
    "bsmJson": {
        "coreData": {
            "msgCnt": 1,
            "id": 42,
            "secMark": 12345,
            "lat": 38.8732164,
            "lon": -77.2468636,
            "elev": 84.1,
            "accuracy": {
                "semiMajor": 3,
                "semiMinor": 4,
                "orientation": 5
            },
```

```
            "transmission": 2,
            "speed": 10.4,
            "heading": 92.7,
            "angle": 4.5,
            "accelSet": {
                "lon": 0.1,
                "lat": 0.2,
                "vert": 0.003,
                "yaw": 0.4
            },
            "brakes": {
                "wheelBrakes": 0,
                "traction": 1,
                "abs": 1,
                "scs": 1,
                "brakeBoost": 1,
                "auxBrakes": 1
            },
            "size": {
                "width": 162,
                "length": 473
            }
        }
    }
}
```

VCC - DTO - BSM - BsmWithMetadataUpload

This JSON structure is used by a client to include metadata with a BSM that is being uploaded to the server. Either the field `bsmJ2735` or the field `bsmJson` is required. Only one of those fields should be populated. If both are specified, `bsmJ2735` is used and `bsmJson` is ignored. All other fields are optional and can be omitted if null.

See the Bsm JSON format for details on the bsmJson field.

BsmWithMetadataUpload JSON

```
{
  "locationName": String?,  // Location, such as: intersection, work zone, street, school, etc.
  "sourceType": String?,    // The source that generated BSM, such as RSU, OBU, Phone, Tablet, Smart Intersection
  "vendorName": String?,    // The name of vendor of source, such as:  Bluecity, Iteris-Conti, DERQ, Commsignia
  "vendorVersion": String?, // Versioning info of software and/or hardware that generated BSM
  "misc": String?,          // Extra field for project-specific uses
  "bsmJ2735": String?,      // The Base64-encoded J2735 BSM.  One of bsmJ2735 or bsmJson is required.
  "bsmJson": Bsm?           // The JSON version of the BSM data.  One of bsmJ2735 or bsmJson is required.
}
```

Example

Example BsmWithMetadataUpload

```
{
    "locationName": "US-29 & Nutley",
    "sourceType": "Smart Intersection",
    "vendorName": "DERQ",
    "vendorVersion": "rev1",
    "misc": "red light event 123",
    "bsmJson": {
        "coreData": {
            "msgCnt": 1,
            "id": 42,
            "secMark": 12345,
            "lat": 38.8732164,
            "lon": -77.2468636,
            "elev": 84.1,
            "accuracy": {
                "semiMajor": 3,
                "semiMinor": 4,
                "orientation": 5
            },
            "transmission": 2,
            "speed": 10.4,
            "heading": 92.7,
            "angle": 4.5,
            "accelSet": {
                "lon": 0.1,
                "lat": 0.2,
                "vert": 0.003,
                "yaw": 0.4
            },
            "brakes": {
```

```
                "wheelBrakes": 0,
                "traction": 1,
                "abs": 1,
                "scs": 1,
                "brakeBoost": 1,
                "auxBrakes": 1
            },
            "size": {
                "width": 162,
                "length": 473
            }
        }
    }
}
```

VCC - DTO - BSM - Bsm (Legacy JSON Format)

This format has been deprecated.

Legacy JSON Example

Legacy JSON


```
{
  "rseID" : 163,
  "tempVehicleID" : "81",
  "timestamp" : 1616095279814,
  "latitude" : 38.123456,
  "longitude" : -77.123456,
  "speed" : 0.02,
  "heading" : 47.1125,
  "elevation" : 74.0,
  "airTempDegC": null,
  "roadTempDegC": null,
  "brake" : 0,
  "format": "UPER"
}
```

VCC - DTO - MapData

DTOs For MapData or SPAT

There are no items with the selected labels at this time.

VCC - DTO - MapData - MapData

Overview

The MapData message is part of the standard message set defined in the SAE J2735 standard.

From the standard:

> “The MapData message is used to convey many types of geographic road information. At the current time its primary use is to convey one or more intersection lane geometry maps within a single message. The map message content includes such items as complex intersection descriptions, road segment descriptions, high speed curve outlines (used in curve safety messages), and segments of roadway (used in some safety applications). A given single MapData message may convey descriptions of one or more geographic areas or intersections. The contents of this message involve defining the details of indexing systems that are in turn used by other messages to relate additional information (for example, the signal phase and timing via the SPAT message) to events at specific geographic locations on the roadway.
> ”

Note that the following JSON represents a substantially smaller subset of the fields that exist in the J2735 MapData message. Most of the optional fields have been stripped out. But, in general, this JSON structure follows the structure of the J2735 MapData message very closely.

JSON Definition

The JSON format for the MapData message is shown below. Types annotated with a question mark "?" are marked as optional in the specification. They will always be present in the JSON but may be null if they were not provided in the original ASN.1 message.

JSON Definition

```
MapData = {
  "timeStamp": Int?,        // minute of year, 0..527040, 527040 = invalid
  "msgIssueRevision": Int,  // 0..127
  "intersections": IntersectionGeometry[] // Unordered list of intersections
}

IntersectionGeometry = {
  "name": String?,
  "id": IntersectionReferenceID,
  "revision": Int,             // 0..127
  "refPoint": Position3D,
  "laneWidth": Double?,        // meters
  "speedLimits": RegulatorySpeedLimit[]?, // Unordered list of speed limits
  "laneSet": GenericLane[] // Unordered list of lanes
}

IntersectionReferenceID = {
  "region": Int?,
  "id": Int       // (0..65535)
}

RegulatorySpeedLimit = {
  "type": Int, // Enum: 0 = "unknown"
               //       1 = "maxSpeedInSchoolZone"
               //       2 = "maxSpeedInSchoolZoneWhenChildrenArePresent"
               //       3 = "maxSpeedInConstructionZone"
               //       4 = "vehicleMinSpeed"
               //       5 = "vehicleMaxSpeed"
               //       6 = "vehicleNightMaxSpeed"
               //       7 = "truckMinSpeed"
               //       8 = "truckMaxSpeed"
               //       9 = "truckNightMaxSpeed"
               //      10 = "vehiclesWithTrailersMinSpeed"
               //      11 = "vehiclesWithTrailersMaxSpeed"
               //      12 = "vehiclesWithTrailersNightMaxSpeed"
  "speed": Double  // m/s (0..163.8)
}

Position3D = {
  "lat": Double,   // decimal degrees,
```

```
  "lon": Double,   // decimal degrees,
  "elevation": Double? // Range -409.5m to + 6143.9m
}

GenericLane = {
  "laneID": Int,            // (0..255)
  "name": String?,          // max length 63
  "ingressApproach": Int?,  // approach ID, (0..15)
  "egressApproach": Int?,   // approach ID, (0..15)
  "laneAttributes": LaneAttributes,
  "maneuvers": Int?,        // 12-bit Bit field (see AllowedManeuvers in J2735 spec) Note: per ASN.1 bits are left
  "nodesList": NodeListXY,
  "connectsTo": Connection[]? // Ordered list of outward lane connections, starting on the left and continuing cloc
}

LaneAttributes = {
  "directionalUse": Int,    // 2-bit Bit field (see LaneDirection in J2735 spec) Note: per ASN.1 bits are left to r
  "sharedWith": Int,        // 10-bit Bit field (see LaneSharing in J2735 spec) Note: per ASN.1 bits are left to ri
  "laneType": LaneTypeAttributes
}

// This is a CHOICE struct, only one of the following properties will be non-null
LaneTypeAttributes = {
  "vehicle": Int?,
  "crosswalk": Int?,
  "bikeLane": Int?,
  "sidewalk": Int?,
  "median": Int?,
  "striping": Int?,
  "trackedVehicle": Int?,
  "parking": Int?
}

// This is a CHOICE struct, only one of the following properties will be non-null
NodeListXY = {
  "nodes": NodeXY[]?,         // Ordered list of lane center offsets. min length = 2, max length = 63
  "computed": ComputedLane?
}

NodeXY = {
  "delta": NodeOffsetPointXY,
  "attributes": NodeAttributeSetXY?
}

NodeOffsetPointXY = {
  "lat": Double  // decimal degrees,
  "lon": Double // decimal degrees
}

Connection = {
  "connectingLane": ConnectingLane,
  "signalGroup": Int?   // (0..255), 0 = N/A or unavailable, 255 = permanent green
}

ConnectingLane = {
  "lane": Int,       // (0..255)
  "maneuver": Int?  // 12-bit Bit field (see AllowedManeuvers in J2735 spec) Note: per ASN.1 bits are left to right
}
```

Example for intersection 152:

The following example is a shortened version of the real data for intersection 152. It only contains a few lanes and only a few of the nodes per lane. The additional lanes and nodes were removed to allow the example to better fit in the documentation.

Click here to expand...

```
{
    "timeStamp": null,
    "msgIssueRevision": 2,
```

```
"intersections": [
    {
        "name": null,
        "id": {
            "region": null,
            "id": 152
        },
        "revision": 2,
        "refPoint": {
            "lat": 38.8629807,
            "lon": -77.22769559999999,
            "elevation": null
        },
        "laneWidth": 3.66,
        "speedLimits": null,
        "laneSet": [
            {
                "laneID": 1,
                "name": null,
                "ingressApproach": null,
                "egressApproach": null,
                "laneAttributes": {
                    "directionalUse": 2,
                    "sharedWith": 0,
                    "laneType": {
                        "vehicle": 0,
                        "crosswalk": null,
                        "bikeLane": null,
                        "sidewalk": null,
                        "median": null,
                        "striping": null,
                        "trackedVehicle": null,
                        "parking": null
                    }
                },
                "maneuvers": 512,
                "nodesList": {
                    "nodes": [
                        {
                            "delta": {
                                "lat": 38.86315363954482,
                                "lon": -77.22788661590909
                            }
                        },
                        {
                            "delta": {
                                "lat": 38.863279454637485,
                                "lon": -77.22786638117296
                            }
                        },
                        {
                            "delta": {
                                "lat": 38.86489778386733,
                                "lon": -77.22787033818803
                            }
                        },
                        {
                            "delta": {
                                "lat": 38.86502054126654,
                                "lon": -77.22790999827085
                            }
                        }
                    ]
                },
                "connectsTo": [
                    {
                        "connectingLane": {
                            "lane": 23,
                            "maneuver": 512
                        },
```

```
            "signalGroup": null
        }
    ]
},
{
    "laneID": 2,
    "name": null,
    "ingressApproach": null,
    "egressApproach": null,
    "laneAttributes": {
        "directionalUse": 2,
        "sharedWith": 0,
        "laneType": {
            "vehicle": 0,
            "crosswalk": null,
            "bikeLane": null,
            "sidewalk": null,
            "median": null,
            "striping": null,
            "trackedVehicle": null,
            "parking": null
        }
    },
    "maneuvers": 2048,
    "nodesList": {
        "nodes": [
            {
                "delta": {
                    "lat": 38.86310174868816,
                    "lon": -77.22781341113037
                }
            },
            {
                "delta": {
                    "lat": 38.86328341165255,
                    "lon": -77.22783130763032
                }
            },
            {
                "delta": {
                    "lat": 38.86489472617388,
                    "lon": -77.22783598410268
                }
            },
            {
                "delta": {
                    "lat": 38.865223158424364,
                    "lon": -77.22782591170069
                }
            },
            {
                "delta": {
                    "lat": 38.86594000767644,
                    "lon": -77.22779866225603
                }
            }
        ]
    },
    "connectsTo": [
        {
            "connectingLane": {
                "lane": 17,
                "maneuver": 2048
            },
            "signalGroup": 2
        }
    ]
},
{
    "laneID": 17,
```

```
            "name": null,
            "ingressApproach": null,
            "egressApproach": null,
            "laneAttributes": {
                "directionalUse": 1,
                "sharedWith": 0,
                "laneType": {
                    "vehicle": 0,
                    "crosswalk": null,
                    "bikeLane": null,
                    "sidewalk": null,
                    "median": null,
                    "striping": null,
                    "trackedVehicle": null,
                    "parking": null
                }
            },
            "maneuvers": 0,
            "nodesList": {
                "nodes": [
                    {
                        "delta": {
                            "lat": 38.862778262706506,
                            "lon": -77.2277822946028
                        }
                    },
                    {
                        "delta": {
                            "lat": 38.86256161613164,
                            "lon": -77.2277652074923
                        }
                    },
                    {
                        "delta": {
                            "lat": 38.860111684212804,
                            "lon": -77.2268032931026
                        }
                    }
                ]
            },
            "connectsTo": null
        },
        {
            "laneID": 23,
            "name": null,
            "ingressApproach": null,
            "egressApproach": null,
            "laneAttributes": {
                "directionalUse": 1,
                "sharedWith": 0,
                "laneType": {
                    "vehicle": 0,
                    "crosswalk": null,
                    "bikeLane": null,
                    "sidewalk": null,
                    "median": null,
                    "striping": null,
                    "trackedVehicle": null,
                    "parking": null
                }
            },
            "maneuvers": 0,
            "nodesList": {
                "nodes": [
                    {
                        "delta": {
                            "lat": 38.86305579335409,
                            "lon": -77.22789668831108
                        }
                    },
```

```
                            {
                                "delta": {
                                    "lat": 38.8630477893918,
                                    "lon": -77.22813096158941
                                }
                            },
                            {
                                "delta": {
                                    "lat": 38.863149682529745,
                                    "lon": -77.22882433854758
                                }
                            }
                        ]
                    },
                    "connectsTo": null
                }
            ]
        }
    ]
}
```


Notes:

- A MapData message can contain no more than 32 intersections
- At this time we are not supporting the description of how to request preemption and priority services listed in the preemptPriorityData element
- At this time we are not supporting the regional extension block
- ComputedLane and NodeAttributeSetXY are listed for completeness and could be implemented in the future, but will not be supported initially.
- Note: per ASN.1 bits are left to right. Bit 0 set is a large value so you must know the bit count to decode the decimal values used in JSON.

VCC - DTO -MapData - MapData Additional Examples

The following examples do not contain data from real intersections, but are useful in understanding the structure of the JSON.

Example JSON

```
// MapData
{
        "mapData": {
                "timeStamp": 527040,
                "msgIssueRevision": 0,
                "intersections": [{
                        "name": "bob",
                        "id": {
                                "region": 6,
                                "id": 6
                        },
                        "revision": 5,
                        "refPoint": {
                                "lat": 2.3,
                                "lon": 3.4,
                                "elevation": 6.3
                        },
                        "laneWidth": 1.2,
                        "speedLimits": [{
                                "type": 3,
                                "speed": 6.32
                        }],
                        "laneSet": [{
                                "laneID": 2,
                                "name": "bob",
                                "ingressApproach": 2,
                                "egressApproach": 3,
                                "laneAttributes": {
                                        "directionalUse": 5,
                                        "sharedWith": 6,
                                        "laneType": {
                                                "crosswalk": 10101
                                        }
                                },
                                "maneuvers": 3,
                                "nodesList": {
                                        "nodes": [
                                {
                                "delta": {
                                    "lat": 1.2,
                                        "lon": 3.4
                                },
                                    "attributes": null
                                                },
                                                {
                                    "delta": {
                                                                        "lat": 7.6,
                                        "lon": 7.5
                                    },
                                        "attributes": null
                                                }
                                        ]
                                },
                                "connectsTo": [{
                                        "connectingLane": {
                                                "lane": "0",
                                                "maneuver": "4"
                                        },
                                        "signalGroup": "3"
                                }]
                        }]
```

```
                }]
        }
}

// IntersectionGeometry
{
        "intersectionGeometry": {
                "name": "bob",
                "id": {
                        "region": 6,
                        "id": 6
                },
                "revision": 5,
                "refPoint": {
                        "lat": 2.3,
                        "lon": 3.4,
                        "elevation": 6.3
                },
                "laneWidth": 1.2,
                "speedLimits": [{
                        "type": 3,
                        "speed": 6.32
                }],
                "laneSet": [{
                        "laneID": 2,
                        "name": "bob",
                        "ingressApproach": 2,
                        "egressApproach": 3,
                        "laneAttributes": {
                                "directionalUse": 5,
                                "sharedWith": 6,
                                "laneType": {
                                        "crosswalk": 10101
                                }
                        },
                        "maneuvers": 3,
                        "nodesList": {
                                "nodes": [
                        {
                        "delta": {
                                "lat": 1.2,
                                "lon": 3.4
                        },
                            "attributes": null
                                        },
                        {
                                "delta": {
                                                        "lat": 7.6,
                                "lon": 7.5
                                },
                                "attributes": null
                                        }
                                ]
                        },
                        "connectsTo": [{
                                "connectingLane": {
                                        "lane": "0",
                                        "maneuver": "4"
                                },
                                "signalGroup": "3"
                        }]
                }]
        }
}

// IntersectionReferenceID
{
    "intersectionReferenceID": {
        "region": 3,
        "id": 6
```

```
    }
}

// RegulatorySpeedLimit
{
        "regulatorySpeedLimit": {
                "type": 4,
                "speed": 1.2
        }
}

// Position3D
{
        "position3D": {
                "lat": 2.3,
                "lon": 3.4,
                "elevation": 6.3
        }
}

// GenericLane
{
        "genericLane": {
                "laneID": 2,
                "name": "bob",
                "ingressApproach": 2,
                "egressApproach": 3,
                "laneAttributes": {
                        "directionalUse": 5,
                        "sharedWith": 6,
                        "laneType": {
                                "vehicle": 10101
                        }
                },
                "maneuvers": 10101,
                "nodesList": {
                        "nodes": [
                        {
                        "delta": {
                                "lat": 1.2,
                                "lon": 3.4
                        },
                    "attributes": null
                                },
                {
                        "delta": {
                                                "lat": 7.6,
                        "lon": 7.5
                        },
                        "attributes": null
                                }
                        ]
                },
                "connectsTo": [{
                        "connectingLane": {
                                "lane": 0,
                                "maneuver": 4
                        },
                        "signalGroup": 3
                }]
        }
}

// LaneAttributes
{
        "laneAttributes": {
                "directionalUse": 10101,
                "sharedWith": 10101,
                "laneType": {
           "crosswalk": 10101
```

```
                }
        }
}


// LaneTypeAttributes - Choice of only one
{
        "laneTypeAttributes": {
       "vehicle": 10101,
       "crosswalk": 10101,
       "bikeLane": 10101,
       "sidewalk": 10101,
       "median": 10101,
       "striping": 10101,
       "trackedVehicle": 10101,
       "parking": 10101
    }
}
// LaneTypeAttributes - one choice example
{
        "laneTypeAttributes": {
       "vehicle": 10101
    }
}

// NodeListXY - choice 1
{
    "nodeListXY": {
        "nodes": [
            {
                "delta": {
                    "lat": 5.6,
                    "lon": 5.5
                },
                "attributes": null
            },
            {
                "delta": {
                    "lat": 7.6,
                    "lon": 7.5
                },
                "attributes": null
            }
        ]
    }
}
// NodeListXY - choice 2
{
    "nodeListXY": {
        "computed": {
            "tbd": "abc"
        }
    }
}

// NodeXY
{
        "nodeXY": {
                "delta": {
                        "lat": 5.6,
                        "lon": 5.5
                },
                "attributes": [{
            "tbd_1": "attr_1",
            "tbd_2": "attr_2"
                }]
        }
}

// NodeAttributeSetXY
```

```
{
    "nodeAttributeSetXY": {
        "tbd_1": "attr_1"
    }
}

// NodeOffsetPointXY
{
        "nodeOffsetPointXY": {
                "lat": 5.6,
                "lon": 7.8
        }
}

// Connection
{
        "connection": {
                "connectingLane": {
                        "lane": 0,
                        "maneuver": 4
                },
                "signalGroup": 3
        }
}

// ConnectingLane
{
        "connectingLane": {
                "lane": 0,
                "maneuver": 4
        }
}
```

VCC - DTO - MapData - MapData (Legacy JSON Format)

Overview

The MapData message is part of the standard message set defined in the SAE J2735 standard. This format of the JSON data is a vastly simplified structure compared to the J2735 definition. It is now considered legacy and was only used in the VCC Monitor web application.

From the standard:

> “The MapData message is used to convey many types of geographic road information. At the current time its primary use is to convey one or more intersection lane geometry maps within a single message. The map message content includes such items as complex intersection descriptions, road segment descriptions, high speed curve outlines (used in curve safety messages), and segments of roadway (used in some safety applications). A given single MapData message may convey descriptions of one or more geographic areas or intersections. The contents of this message involve defining the details of indexing systems that are in turn used by other messages to relate additional information (for example, the signal phase and timing via the SPAT message) to events at specific geographic locations on the roadway.
> ”

Note that the following JSON represents a substantially smaller subset of the fields that exist in the J2735 MapData message. Most of the optional fields have been stripped out and some of the field names have been shortened (e.g. "lanes" instead of "laneSet") while others have been expanded for clarity (e.g. revisionMsgCount instead of revision of type MsgCount).

JSON Definition

The JSON format for the MapData message is shown below.

JSON Definition

```
{
  "id": Int,
  "revisionMsgCount": Int,
  "latitude": Double,  // decimal degrees
  "longitude": Double, // decimal degrees
  "notes": String?,
  "lanes": [
    {
      "laneId": Int,
      "heading": Float,
      "connections": [
        {
          "laneId": Int,
          "maneuver": Int?,
          "signalGroup": Int?
        }
      ],  // end "connections"
      "nodes": [
        {
          "nodeIndex": Int,
          "latitude": Double  // decimal degrees,
          "longitude": Double // decimal degrees
        }
      ],  // end "nodes"
    }
  ]
}
```

The code below shows an actual example of the JSON definition for the MapData message.

JSON Example

```
{
  "id": 115,
  "revisionMsgCount": 1,
  "latitude": 38.8652467,
  "longitude": -77.2517808,
  "notes": null,
  "lanes": [
    {
      "laneId": 1,
      "heading": 181.56,
      "connections": [
```

```
        {
          "laneId": 18,
          "maneuver": 512,
          "signalGroup": 4
        }
      ],
      "nodes": [
        {
          "nodeIndex": 1,
          "latitude": 38.86544230244929,
          "longitude": -77.25182596105233
        },
        {
          "nodeIndex": 2,
          "latitude": 38.86559752535847,
          "longitude": -77.25182053248594
        }
      ]
    },
    {
      "laneId": 2,
      // ... etc. etc.
    }
  ]
}
```

VCC - DTO - MapData - MapDataSummary

This format was originally provided by the Public API. It is still supported for backwards compatibility.

Description of Fields

- `id`: Int. The intersection ID contained within the MapData
- `revision`: Int. The message revision contained within the MapData
- `msg`: String. The binary 2016 J2735 MapData message base64 encoded into a string.

Example


```
{
    "id": 167,
    "revision": 3,
    "msg": "ABKCCggDAQAKcGJmjUO56gr6qBbhAMAUAAAAgACK/I0OtlgLqFhiwGjCfZX50g+UEiJAAIGASAAABgQBFgBeHFiwFHDcRYB0hVgr90w
}
```

VCC - DTO - PSM

DTOs For PSM

There are no items with the selected labels at this time.

VCC - DTO - PSM - Psm

PersonalSafetyMessage (PSM) JSON

This PSM JSON uses the J2735 structure and field names. Fields that have a scaling factor described in the J2735 documentation have the scaling factor applied in this JSON object. For example, rather than expressing latitude as an integer in 1/10 micro degrees, it is scaled to degrees and stored as a double. When scaling, fields with a special value that indicates unavailable are converted to null.

This JSON structure is used for POSTing a PSM.

Psm JSON

```
Psm = {
  "basicType": Int,     // enum with values 0-4, 0 = unavailable
  "secMark": Int?,      // values 0-65_535, in milliseconds, 65535 = unavailable
  "msgCnt": Int,        // values 0-127
  "id": Int,            // converted from 4 byte octet string, i.e TemporaryID,
  "position": Position3D,
  "accuracy": PositionalAccuracy?,
  "speed": Float?,      // in m/s, converted from integer in 0.02 m/s units, 8191 = unavailable, input 0-8191
  "heading": Float?,    // in degrees, converted from integer in 0.0125 degrees, input 0-28_800, 28_800 = unavailab
  "accelSet": AccelerationSet4Way?,   // to support nullable, the yaw acceleration will default to 0
  "eventResponderType": Int?,   // convert from enum PublicSafetyEventResponderWorkerType, 0 = unavailable
  "activityType": Int?,         // convert from bit string PublicSafetyAndRoadWorkerActivity (bit 0 = 0x20 = 32)
  "activitySubType": Int?       // convert from bit string PublicSafetyDirectingTrafficSubType (bit 0 = 0x40 = 64)

  // Additional optional fields may be added in the future
}

Position3D = {
  "lat": Double?,       // in degrees, converted from integer in 1/10 micro degrees
  "lon": Double?,       // in degrees, converted from integer in 1/10 micro degrees
  "elev": Float?        // in meters, converted from integer in 10 cm, -4096 = unknown
}

PositionalAccuracy = {
  "semiMajor": Float?,   // in meters, converted from values 0-255, i.e. SemiMajorAxisAccuracy, 255 = unavailable
  "semiMinor": Float?,   // in meters, converted from values 0-255, i.e. SemiMinorAxisAccuracy, 255 = unavailable
  "orientation": Float?  // in degrees, converted from values 0-65_535, i.e.  SemiMajorAxisOrientation, 65_535 = un
}

AccelerationSet4Way = {
  "lon": Float?,  // in m/s^2, converted from Acceleration, values -2000-+2001, in 0.01 m/s^2, 2001=unavailable
  "lat": Float?,  // in m/s^2, converted from Acceleration, values -2000-+2001, in 0.01 m/s^2, 2001=unavailable
  "vert": Float?, // in G, converted from VerticalAcceleration, values -127-+127, in 0.02 G, -127=unavailable
  "yaw": Float    // in degrees/second, converted from YawRate, values -32767-+32767, in 0.01 degrees/s, not nullab
}
```

Example PSM JSON

This is an example of JSON that can be used for POSTing a PSM.

Example PSM JSON

```
{
    "basicType": 2,
    "secMark": 12345,
    "msgCnt": 18,
    "id": 567,
    "position": {
        "lat": 37.2537661,
        "lon": -80.4010384
    },
    "accuracy": {
        "semiMajor": 122,
```

```
        "semiMinor": 0,
        "orientation": 26334
    },
    "speed": 23.45,
    "heading": 18.72,
    "accelSet": {
      "lon": 0.02,
      "lat": 0.04,
      "vert": 0.06,
      "yaw": 0.08
    },
    "eventResponderType": 3,
    "activityType": 5,
    "activitySubType": 1
}
```

Simplest Example for Posting

Any field not specified, except yaw accel, will be set to unavailable. Since yaw accel doesn't have a value for unavailable, it will be set to 0. In this example, all optional fields have been omitted.

Simplest Example for Posting a PSM

```
{
    "msgCnt": 18,
    "id": 567,
    "position": {
        "lat": 37.2537661,
        "lon": -80.4010384
    }
}
```

VCC - DTO - PSM - PsmReceived

This is a small JSON object that is returned in response to a PSM being successfully POSTed. It contains a couple fields from the POSTed PSM.

BsmReceived JSON

```
PsmReceived = {
  "msgCnt": Int,
  "id": Int
}
```

Example PsmReceived JSON

BsmReceived Example

```
{
    "msgCnt": 1,
    "id": 42
}
```

VCC - DTO - PSM - PsmWithMetadata

This JSON structure is used when the server provides a list of PSM. In addition to containing the data from the J2735 PSM message, it contains additional metadata about the receipt of the PSM. Note that null metadata fields may be omitted to reduce the size of the JSON. See the PSM JSON format for details on the psmJson field.

PsmWithMetadata JSON

```
{
  "timestamp": Long,        // Epoch timestamp in milliseconds at which the PSM was received by VCC Cloud
  "rsuName": String,        // The name of the RSU that sent the PSM, the string "From_Post" or the string "From_WebSocket"
  "format": String,         // The format in which the PSM was received, includes:  v2016, JSON, v2016_PLUS, JSON_PLUS
  "clientId": String?,      // The client_id of the application that uploaded the PSM (if POST or WebSocket) or null if received from an RSU
  "locationName": String?,  // Client-provided location, such as: intersection, work zone, street, school, etc.
  "sourceType": String?,    // Client-provided source that generated PSM, such as RSU, OBU, Phone, Tablet, Smart Intersection
  "vendorName": String?,    // Client-provided name of vendor of source, such as:  Bluecity, Iteris-Conti, DERQ, Commsignia
  "vendorVersion": String?, // Client-provided versioning info of software and/or hardware that generated PSM
  "misc": String?,          // Client-provided extra field for project-specific uses
  "psmJ2735": String?,      // Optionally included.  The Base64-encoded J2735 PSM, only populated if the PSM was received in this format
  "psmJson": Psm            // The JSON version of the PSM data
}
```

Example

Example PsmWithMetadata

```
{
    "timestamp": 1623690664522,
    "rsuName": "From_WebSocket",
    "format": "V2016_PLUS",
    "clientId": "vcctest",
    "locationName": "US-29 & Nutley",
    "sourceType": "Smart Intersection",
    "vendorName": "DERQ",
    "vendorVersion": "rev1",
    "misc": "red light event 123",
    "psmJ2735": "ACAaAAAAkhjEAAAAAEvZYj07XZhvegBm3gAAAAA=",
    "psmJson": {
        "basicType": 0,
        "secMark": 18700,
        "msgCnt": 49,
        "id": 0,
        "position": {
            "lat": 37.2537661,
            "lon": -80.4010384,
            "elev": null
        },
        "accuracy": {
```

```
            "semiMajor": 61.0,
            "semiMinor": 0.0,
            "orientation": 144.6592
        },
        "speed": 0.0,
        "heading": 0.0,
        "accelSet": null
    }
}
```

VCC - DTO - PSM - PsmWithMetadataUpload

This JSON structure is used by a client to include metadata with a PSM that is being uploaded to the server. Either the field `psmJ2735` or the field `psmJson` is required. Only one of those fields should be populated. If both are specified, `psmJ2735` is used and `psmJson` is ignored. All other fields are optional and can be omitted if null.

See the PSM JSON format for details on the psmJson field.

PsmWithMetadataUpload JSON

```
{
  "locationName": String?,  // Client-provided location, such as: intersection, work zone, street, school, etc.
  "sourceType": String?,    // Client-provided source that generated PSM, such as RSU, OBU, Phone, Tablet, Smart Intersection
  "vendorName": String?,    // Client-provided name of vendor of source, such as:  Bluecity, Iteris-Conti, DERQ, Commsignia
  "vendorVersion": String?, // Client-provided versioning info of software and/or hardware that generated PSM
  "misc": String?,          // Client-provided extra field for project-specific uses
  "psmJ2735": String?,      // The Base64-encoded J2735 PSM.  One of psmJ2735 or psmJson is required.
  "psmJson": Psm?           // The JSON version of the PSM data.  One of psmJ2735 or psmJson is required.
}
```

Example

Example PsmWithMetadataUpload

```
{
    "locationName": "US-29 & Nutley",
    "sourceType": "Smart Intersection",
    "vendorName": "DERQ",
    "vendorVersion": "rev1",
    "misc": "red light event 123",
    "psmJ2735": "ACAaAAAAkhjEAAAAAEvZYj07XZhvegBm3gAAAAA="
}
```

VCC - DTO - SPAT

DTOs For MapData or SPAT

There are no items with the selected labels at this time.

VCC - DTO - SPAT - SPAT

Overview

A JSON representation of the standard J2735 SPAT (Signal Phase And Timing) data message.

JSON Definition

The code below specifies the format of the SPAT message in JSON. Types annotated with a question mark "?" are optional and may be set to null in the JSON data.

JSON Definition

```
Spat = {
  "timeStamp": Int?,    // minute of year, (0..527040), 527040 = invalid
  "name": String?,      // max len = 63
  "intersections": IntersectionState[],
}

IntersectionState = {
  "name": String?,         // max len = 63
  "id": IntersectionReferenceID,
  "revision": Int,         // (0..127)
  "status": Int,           // 16-bit Bit field (see IntersectionStatusObject in J2735 spec) Note: per ASN.1 bits ar
  "moy": Int?,             // minute of year, (0..527040), 527040 = invalid
  "timeStamp": Int?,       // milliseconds, (0..65535)
  "enabledLanes": Int[]?,  // List of lane IDs, (0..255) each, 0 = unavailable, 255 = reserved
  "states": MovementState[]
  // "maneuverAssistList": ConnectionManeuverAssist[]?   - not currently being used
}

IntersectionReferenceID = {
  region: Int?,  // (0..65535)
  id: Int,      // (0..65535)
}

MovementState = {
  "movementName": String?,  // max len = 63
  "signalGroup": Int,       // (0..255), 0 = unavailable, 255 = permanent green
  "stateTimeSpeed": MovementEvent[]
  // "maneuverAssistList": ConnectionManeuverAssist[]?   - not currently being used
}

MovementEvent = {
  "eventState": Int,  // Enum: 0 = "unavailable"
                      //       1 = "dark"
                      //       2 = "stop-Then-Proceed"           (reds)
                      //       3 = "stop-And-Remain"
                      //       4 = "pre-Movement"                (greens)
                      //       5 = "permissive-Movement-Allowed"
                      //       6 = "protected-Movement-Allowed"
                      //       7 = "permissive-Clearance"        (yellows/ambers)
                      //       8 = "protected-Clearance"
                      //       9 = "caution-Conflicting-Traffic"
  "timing": TimeChangeDetailsUtc?,
  "speeds": AdvisorySpeed[]?,
}

TimeChangeDetailsUtc = {
  "startTimeStamp": Long?,   // milliseconds since epoch or magic value of 36,000 or 36,001
  "minEndTimeStamp": Long?,  // milliseconds since epoch or magic value of 36,000 or 36,001
  "maxEndTimeStamp": Long?,  // milliseconds since epoch or magic value of 36,000 or 36,001
  "likelyTimeStamp": Long?,  // milliseconds since epoch or magic value of 36,000 or 36,001
  "confidence": Int?, // (0..15), 0 = 21%, 15 = 100%, see TimeIntervalConfidence in J2735 spec
}
```

```
TimeChangeDetails = {
  "startTime": Int?,  // (0..36001), see TimeMark in J2735 spec
  "minEndTime": Int,  // (0..36001), see TimeMark in J2735 spec
  "maxEndTime": Int?, // (0..36001), see TimeMark in J2735 spec
  "likelyTime": Int?, // (0..36001), see TimeMark in J2735 spec
  "confidence": Int?, // (0..15), 0 = 21%, 15 = 100%, see TimeIntervalConfidence in J2735 spec
}

AdvisorySpeed = {
  "type": Int,         // Enum: 0 = none, 1 = greenwave, 2 = ecoDrive, 3 = transit
  "speed": Int?,       // (0..500), in decimeters/second,
                       //   500 = unavailable,
                       //   499 = greater than or equal to 49.9 m/s
  "confidence": Int?,  // Enum: 0 = unavailable,
                       //       1 = 100 m/s precision
                       //       2 = 10 m/s precision
                       //       3 = 5 m/s precision
                       //       4 = 1 m/s precision
                       //       5 = .1 m/s precision
                       //       6 = .05 m/s precision
                       //       7 = .01 m/s precision
  "distance": Int?,    // (0..10000), meters, 0 = unknown, 10000 used for >= 10000 meters
  "class": Int?,       // (0..255), see RestrictionClassID in J2735 spec
}
```

Notes:

- We do not support regional extensions at this time.
- We do not support TimeChangeDetails.nextTime at this time.
- We do not support maneuverAssistList at this time.
- TimeMark values are converted to timestamps for convenience.

JSON Example

The example below is from a SPAT received from intersection 152. The original, full JSON contains 32 entries in the states array. Some of the entries have been removed to allow the shortened example to better fit in the documentation.


```
{
    "timeStamp": null,
    "name": null,
    "intersections": [{
        "name": null,
        "id": {
            "region": null,
            "id": 152
        },
        "revision": 108,
        "status": 0,
        "moy": 89316,
        "timeStamp": null,
        "enabledLanes": null,
        "states": [{
            "movementName": null,
            "signalGroup": 1,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 1614818291600,
                    "maxEndTimeStamp": 36001,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 2,
            "stateTimeSpeed": [{
                "eventState": 5,
```

```
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1614818286500,
                "maxEndTimeStamp": 36001,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 3,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36001,
                "maxEndTimeStamp": 1614821771600,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 4,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1614818291600,
                "maxEndTimeStamp": 36001,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 5,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36001,
                "maxEndTimeStamp": 1614821771600,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 6,
        "stateTimeSpeed": [{
            "eventState": 5,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1614818286800,
                "maxEndTimeStamp": 36001,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 7,
        "stateTimeSpeed": [{
            "eventState": 3,
```

```
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36001,
                    "maxEndTimeStamp": 1614821771600,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 8,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36001,
                    "maxEndTimeStamp": 1614821771600,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 32,
            "stateTimeSpeed": [{
                "eventState": 1,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36001,
                    "maxEndTimeStamp": 1614821771600,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }]
    }]
}
```

VCC - DTO - SPAT - SPAT (Legacy JSON Format)

Overview

A JSON representation of the standard J2735 SPAT data message. This format of the JSON data is a vastly simplified structure compared to the J2735 definition. It is now considered legacy and is only used in the VCC Monitor web application and the Signal Aware application that calculates likelyTime.

JSON Definition

The code below specifies the format of the SPAT message in JSON.

JSON Definition

```
{
  "intersectionId": Int  // same as "id" in MapData Json
  "revision": Int,
  "status": Int,
  "movements": [
    {
      "maxEndTimestamp": Long - milliseconds since Jan 1, 1970, magic values (36,000 and 36,001) are converted to n
      "minEndTimestamp": Long - milliseconds since Jan 1, 1970, magic values (36,000 and 36,001) are converted to n
      "phase": Int,
      "signalGroup": Int
    },
 ],
}
```

JSON Example

```
{
    "intersectionId": 127,
    "revision": 32,
    "status": 0,
    "movements": [
    {
        "maxEndTimestamp": 1556827127400,
        "minEndTimestamp": 1556827127400,
        "phase": 5,
        "signalGroup": 1
    },
    {
        "maxEndTimestamp": 1556827133500,
        "minEndTimestamp": 1556827127600,
        "phase": 3,
        "signalGroup": 2
    },
    {
        "maxEndTimestamp": 1556827218500,
        "minEndTimestamp": 1556827218500,
        "phase": 3,
        "signalGroup": 3
    },
    {
        "maxEndTimestamp": 1556827240500,
        "minEndTimestamp": 1556827232900,
        "phase": 3,
        "signalGroup": 4
    },
    {
        "maxEndTimestamp": 1556827132100,
        "minEndTimestamp": 1556827132100,
        "phase": 5,
        "signalGroup": 5
    },
    {
```

```
        "maxEndTimestamp": 1556827138500,
        "minEndTimestamp": 1556827127900,
        "phase": 3,
        "signalGroup": 6
    },
}
```

VCC - DTO - SPAT - SpatTimeStamp

This JSON object is used for reporting the timestamp at which the most recent SPAT was received for an intersection.

Description of Fields

- `intersectionId`: Int. The ID of the intersection.
- `latestTimestamp`: Long. The most recent timestamp at which SPAT was received for this intersection.

JSON Example

```
{
    "intersectionId": 152,
    "latestTimestamp": 1597859649469
}
```

VCC - DTO - SPAT - SpatWebSocketUrl

This URL is used for opening a WebSocket for streaming SPAT.

Description of Fields

- `url`: String. The URL for opening a WebSocket for streaming SPAT. The URL includes a unique token that can only be used one time and is only valid for a short amount of time.

JSON Example

```
{
  "url":"/ws/spat?key=TjDjM9WH8sAMwDDeimrE8II3"
}
```

VCC - DTO - SPAT - SpatWithMetadata

This JSON structure can be used when the server provides a SPAT over a WebSocket. In addition to containing the data from the J2735 SPAT message, it contains additional metadata about the SPAT. See the SPAT JSON Format for details on the spatJson field.

SpatWithMetadata JSON

```
{
  "receivedTimestamp": Long,    // Epoch timestamp in milliseconds of when the SPAT was received by VCC Cloud
  "rsuName": String,            // The name of the RSU that sent the SPAT
  "spatTimestamp": Long,        // A temporary field created by combining the bytes from minEndTime for SignalGroup
  "spatCounter": Int,           // A temporary field that equals the minEndTime for SignalGroup 17
  "spat": Spat                  // The JSON version of the SPAT data
}
```

Example:

Example


```
{
    "receivedTimestamp": 1643065912980,
    "rsuName": "Site7_CV2X",
    "spatTimestamp": 1643065912975,
    "spatCounter": 143,
    "spat": {
        "timeStamp": null,
        "name": null,
        "intersections": [{
            "name": "VTTI SPaT",
            "id": {
                "region": null,
                "id": 180
            },
            "revision": 122,
            "status": 16384,
            "moy": 2020,
            "timeStamp": 52976,
            "enabledLanes": null,
            "states": [{
                "movementName": null,
                "signalGroup": 1,
                "stateTimeSpeed": [{
                    "eventState": 3,
                    "timing": {
                        "startTimeStamp": null,
                        "minEndTimeStamp": 36000,
                        "maxEndTimeStamp": 1643069512900,
                        "likelyTimeStamp": null,
                        "confidence": null
                    },
                    "speeds": null
                }]
            }, {
                "movementName": null,
                "signalGroup": 2,
                "stateTimeSpeed": [{
                    "eventState": 3,
                    "timing": {
                        "startTimeStamp": null,
                        "minEndTimeStamp": 1643065924900,
                        "maxEndTimeStamp": 1643065949900,
                        "likelyTimeStamp": null,
                        "confidence": null
                    },
                    "speeds": null
                }]
            }, {
```

```
            "movementName": null,
            "signalGroup": 3,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 4,
            "stateTimeSpeed": [{
                "eventState": 6,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 1643065917900,
                    "maxEndTimeStamp": 1643065942900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 5,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 6,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 1643065924900,
                    "maxEndTimeStamp": 1643065949900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 7,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
```

```
            "movementName": null,
            "signalGroup": 8,
            "stateTimeSpeed": [{
                "eventState": 6,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 1643065917900,
                    "maxEndTimeStamp": 1643065942900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 9,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 10,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 11,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
            "movementName": null,
            "signalGroup": 12,
            "stateTimeSpeed": [{
                "eventState": 3,
                "timing": {
                    "startTimeStamp": null,
                    "minEndTimeStamp": 36000,
                    "maxEndTimeStamp": 1643069512900,
                    "likelyTimeStamp": null,
                    "confidence": null
                },
                "speeds": null
            }]
        }, {
```

```
        "movementName": null,
        "signalGroup": 13,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36000,
                "maxEndTimeStamp": 1643069512900,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 14,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36000,
                "maxEndTimeStamp": 1643069512900,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 15,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36000,
                "maxEndTimeStamp": 1643069512900,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 16,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 36000,
                "maxEndTimeStamp": 1643069512900,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 17,
        "stateTimeSpeed": [{
            "eventState": 3,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068814300,
                "maxEndTimeStamp": 1643068814300,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
```

```
        "movementName": null,
        "signalGroup": 18,
        "stateTimeSpeed": [{
            "eventState": 1,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068814300,
                "maxEndTimeStamp": 1643068804600,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 19,
        "stateTimeSpeed": [{
            "eventState": 1,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068804600,
                "maxEndTimeStamp": 1643068814200,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 20,
        "stateTimeSpeed": [{
            "eventState": 1,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068809200,
                "maxEndTimeStamp": 1643068800100,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 21,
        "stateTimeSpeed": [{
            "eventState": 1,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068814200,
                "maxEndTimeStamp": 1643068800000,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
        "movementName": null,
        "signalGroup": 21,
        "stateTimeSpeed": [{
            "eventState": 1,
            "timing": {
                "startTimeStamp": null,
                "minEndTimeStamp": 1643068812600,
                "maxEndTimeStamp": 1643068800000,
                "likelyTimeStamp": null,
                "confidence": null
            },
            "speeds": null
        }]
    }, {
```

```
                "movementName": null,
                "signalGroup": 21,
                "stateTimeSpeed": [{
                    "eventState": 1,
                    "timing": {
                        "startTimeStamp": null,
                        "minEndTimeStamp": 1643068800100,
                        "maxEndTimeStamp": 1643068800000,
                        "likelyTimeStamp": null,
                        "confidence": null
                    },
                    "speeds": null
                }]
            }, {
                "movementName": null,
                "signalGroup": 21,
                "stateTimeSpeed": [{
                    "eventState": 1,
                    "timing": {
                        "startTimeStamp": null,
                        "minEndTimeStamp": 1643068800000,
                        "maxEndTimeStamp": 1643068800000,
                        "likelyTimeStamp": null,
                        "confidence": null
                    },
                    "speeds": null
                }]
            }, {
                "movementName": null,
                "signalGroup": 21,
                "stateTimeSpeed": [{
                    "eventState": 1,
                    "timing": {
                        "startTimeStamp": null,
                        "minEndTimeStamp": 1643068800000,
                        "maxEndTimeStamp": 1643068800000,
                        "likelyTimeStamp": null,
                        "confidence": null
                    },
                    "speeds": null
                }]
            }]
        }]
    }
}
```